

RI600PX

リアルタイム・オペレーティング・システム

ユーザーズマニュアル コーディング編

対象デバイス

MPU (Memory Protection Unit) 搭載RXファミリ

本資料に記載の全ての情報は発行時点のものであり、ルネサス エレクトロニクスは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システム的设计において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したのですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置等
当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社とその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

このマニュアルの使い方

対象者	このマニュアルは、RX MCU ファミリの各製品の応用システムを設計、開発するユーザを対象としています。
目的	このマニュアルは、次の構成に示すルネサス エレクトロニクス製リアルタイム OS RI600PX の機能をユーザに理解していただくことを目的としています。
構成	このマニュアルは、大きく分けて次の内容で構成しています。 第 1 章 概 説 第 2 章 システム構築 第 3 章 メモリ保護機能 第 4 章 タスク管理機能 第 5 章 タスク付属同期機能 第 6 章 タスク例外処理機能 第 7 章 同期通信機能 第 8 章 拡張同期通信機能 第 9 章 メモリ・プール管理機能 第 10 章 時間管理機能 第 11 章 システム状態管理機能 第 12 章 割り込み管理機能 第 13 章 システム構成管理機能 第 14 章 オブジェクト・リセット機能 第 15 章 システム・ダウン 第 16 章 スケジューリング機能 第 17 章 システム初期化処理 第 18 章 データ・タイプとマクロ 第 19 章 サービス・コール 第 20 章 システム・コンフィギュレーション・ファイル 第 21 章 コンフィギュレータ cfg600px 第 22 章 テーブル生成ユーティリティ mkritblpx 付録 A ウィンドウ・リファレンス 付録 B 浮動小数点演算機能 付録 C DSP 機能 付録 D スタック使用量の算出
読み方	このマニュアルの読者には、電気、論理回路、マイクロコンピュータ、C 言語、アセンブラの一般知識を必要とします。

RX MCU のハードウェア機能を知りたいとき

→各製品のユーザズマニュアルを参照してください。

凡 例

データ表記の重み:	左が上位桁, 右が下位桁
注	: 本文中につけた注の説明
注意	: 気をつけて読んでいただきたい内容
備考	: 本文中の補足説明
数の表記	: 10 進数 ... XXXX 16 進数 ... 0xXXXX
2 のべき乗を示す接頭辞 (アドレス空間, メモリ容量):	K (キロ) $2^{10} = 1024$ M (メガ) $2^{20} = 1024^2$
up4(data)	: data を 4 の倍数に切り上げた値
down(data)	: data の整数部

関連資料 関連資料は暫定版の場合がありますが、この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

資料名		資料番号	
		和文	英文
RI シリーズ	起動編	R20UT0751J	R20UT0751E
	メッセージ編	R20UT0756J	R20UT0756E
RI600PX	コーディング編	このマニュアル	R20UT0964E
	デバッグ編	R20UT0950J	R20UT0950E

注意 上記関連資料は、予告なしに内容を変更することがあります。設計などには、必ず最新の資料を使用してください。

この資料に記載されている会社名、製品名などは、各社の商標または登録商標です。

目 次

第1章 概 説 … 13

- 1.1 概 要 … 13
 - 1.1.1 リアルタイム OS … 13
 - 1.1.2 マルチタスク OS … 13
 - 1.1.3 メモリ保護機能 … 13

第2章 システム構築 … 14

- 2.1 概 要 … 14
- 2.2 処理プログラムの記述 … 15
- 2.3 システム・コンフィギュレーション・ファイルの記述 … 15
- 2.4 ユーザ・OWN・コーディング部の記述 … 16
- 2.5 ロード・モジュールの生成 … 16
- 2.6 ビルド・オプション … 21
 - 2.6.1 サービス・コール情報ファイルと“-ri600_preinit_mrc”コンパイラ・オプション … 21
 - 2.6.2 ブート処理ファイルのコンパイラ・オプション … 22
 - 2.6.3 カーネル・ライブラリ … 23
 - 2.6.4 セクション配置 … 24
 - 2.6.5 初期化データ・セクション … 27

第3章 メモリ保護機能 … 28

- 3.1 概 要 … 28
- 3.2 ドメイン, メモリオブジェクト, アクセス許可ベクタ … 28
- 3.3 メモリ・オブジェクト数の制約 … 30
- 3.4 信頼されたドメイン … 30
- 3.5 アクセス許可の変更 … 30
- 3.6 ユーザ・スタック保護 … 31
- 3.7 アクセス許可のチェック … 31
- 3.8 プロセッサのモード … 31
- 3.9 MPU (Memory Protection Unit) の有効化 … 31
- 3.10 アクセス例外ハンドラ (`_RI_sys_access_exception()`) … 32
 - 3.10.1 ユーザ・OWN・コーディング部 … 32
- 3.11 メモリ・マップ設計 … 34
 - 3.11.1 メモリ・オブジェクトのアドレス制約 … 34
 - 3.11.2 メモリ・オブジェクト化すべき領域 … 35
 - 3.11.3 メモリ・オブジェクトにしてはならない領域 … 36

第4章 タスク管理機能 … 37

- 4.1 概 要 … 37
- 4.2 タスク … 37
 - 4.2.1 タスクの状態 … 37
 - 4.2.2 タスクの優先度 … 39

4.2.3	タスクの基本型	…	40
4.2.4	タスク内での処理	…	41
4.2.5	タスク実行時のプロセッサ・モード	…	41
4.3	タスクの生成	…	42
4.4	タスクの削除	…	44
4.5	タスクの起動	…	45
4.5.1	起動要求をキューイングする起動	…	45
4.5.2	起動要求をキューイングしない起動	…	46
4.6	起動要求のキューイング解除	…	47
4.7	タスクの終了	…	48
4.7.1	自タスクの終了	…	48
4.7.2	他タスクの強制終了	…	49
4.8	タスク優先度の変更	…	50
4.9	タスク優先度の参照	…	51
4.10	タスク状態の参照	…	52
4.10.1	タスク詳細情報の参照	…	52
4.10.2	タスク基本情報の参照	…	53

第 5 章 タスク付属同期機能 … 54

5.1	概 要	…	54
5.2	起床待ち状態への移行	…	54
5.2.1	永久待ち	…	54
5.2.2	タイムアウト付き	…	55
5.3	タスクの起床	…	56
5.4	起床要求の解除	…	57
5.5	WAITING 状態の強制解除	…	58
5.6	SUSPENDED 状態への移行	…	59
5.7	SUSPENDED 状態の解除	…	60
5.7.1	SUSPENDED 状態の解除	…	60
5.7.2	SUSPENDED 状態の強制解除	…	61
5.8	時間経過待ち状態への移行	…	62
5.9	タイムアウト付き起床待ちと時間経過待ちの違い	…	63

第 6 章 タスク例外処理機能 … 64

6.1	概 要	…	64
6.2	タスク例外処理ルーチン	…	65
6.2.1	タスク例外処理ルーチンの基本型	…	65
6.2.2	タスク例外処理ルーチン内での処理	…	66
6.2.3	タスク例外処理ルーチンの起動条件	…	67
6.3	タスク例外処理ルーチンの定義	…	68
6.4	タスク例外処理ルーチンの定義解除	…	69
6.5	タスク例外処理の要求	…	70
6.6	タスク例外処理の禁止と許可	…	71
6.7	タスク例外処理禁止状態の参照	…	72
6.8	タスク例外処理の詳細情報の参照	…	73

第 7 章 同期通信機能 … 74

7.1	概 要	…	74
7.2	セマフォ	…	74
7.2.1	セマフォの生成	…	75
7.2.2	セマフォの削除	…	76
7.2.3	資源の獲得	…	77
7.2.4	資源の返却	…	80
7.2.5	セマフォ詳細情報の参照	…	81
7.3	イベントフラグ	…	82
7.3.1	イベントフラグの生成	…	83
7.3.2	イベントフラグの削除	…	84
7.3.3	ビット・パターンのセット	…	85
7.3.4	ビット・パターンのクリア	…	86
7.3.5	ビット・パターンのチェック	…	87
7.3.6	イベントフラグ詳細情報の参照	…	92
7.4	データ・キュー	…	93
7.4.1	データ・キューの生成	…	94
7.4.2	データ・キューの削除	…	95
7.4.3	データの送信	…	96
7.4.4	データの強制送信	…	101
7.4.5	データの受信	…	102
7.4.6	データ・キュー詳細情報の参照	…	107
7.5	メールボックス	…	108
7.5.1	メッセージ	…	108
7.5.2	メールボックスの生成	…	110
7.5.3	メールボックスの削除	…	111
7.5.4	メッセージの送信	…	112
7.5.5	メッセージの受信	…	113
7.5.6	メールボックス詳細情報の参照	…	116

第 8 章 拡張同期通信機能 … 117

8.1	概 要	…	117
8.2	ミューテックス	…	117
8.2.1	優先度逆転問題	…	118
8.2.2	現在優先度とベース優先度	…	118
8.2.3	簡略化した優先度上限プロトコル	…	119
8.2.4	セマフォとの相違点	…	119
8.2.5	ミューテックスの生成	…	120
8.2.6	ミューテックスの削除	…	121
8.2.7	ミューテックスのロック	…	122
8.2.8	ミューテックスのロック解除	…	126
8.2.9	ミューテックス詳細情報の参照	…	127
8.3	メッセージ・バッファ	…	128
8.3.1	メッセージ・バッファの生成	…	129
8.3.2	メッセージ・バッファの削除	…	130
8.3.3	メッセージの送信	…	131
8.3.4	メッセージの受信	…	136
8.3.5	メッセージ・バッファ詳細情報の参照	…	141

第 9 章 メモリ・プール管理機能 … 142

- 9.1 概 要 … 142
- 9.2 固定長メモリ・プール … 143
 - 9.2.1 固定長メモリ・プールの生成 … 143
 - 9.2.2 固定長メモリ・プールの削除 … 145
 - 9.2.3 固定長メモリ・ブロックの獲得 … 146
 - 9.2.4 固定長メモリ・ブロックの返却 … 151
 - 9.2.5 固定長メモリ・プール詳細情報の参照 … 152
- 9.3 可変長メモリ・プール … 153
 - 9.3.1 可変長メモリ・ブロックのサイズ … 153
 - 9.3.2 可変長メモリ・プールの生成 … 154
 - 9.3.3 可変長メモリ・プールの削除 … 156
 - 9.3.4 可変長メモリ・ブロックの獲得 … 157
 - 9.3.5 可変長メモリ・ブロックの返却 … 162
 - 9.3.6 可変長メモリ・プール詳細情報の参照 … 163

第 10 章 時間管理機能 … 164

- 10.1 概 要 … 164
- 10.2 システム時刻 … 164
 - 10.2.1 基本クロック用タイマ割り込み … 164
 - 10.2.2 基本クロック周期 … 164
- 10.3 タイマ・オペレーション機能 … 165
- 10.4 タスクの遅延 … 165
- 10.5 タイムアウト … 165
- 10.6 周期ハンドラ … 166
 - 10.6.1 周期ハンドラの基本形 … 166
 - 10.6.2 周期ハンドラ内での処理 … 167
 - 10.6.3 周期ハンドラの生成 … 168
 - 10.6.4 周期ハンドラの削除 … 169
 - 10.6.5 周期ハンドラの動作開始 … 170
 - 10.6.6 周期ハンドラの動作停止 … 172
 - 10.6.7 周期ハンドラ詳細情報の参照 … 173
- 10.7 アラーム・ハンドラ … 174
 - 10.7.1 アラーム・ハンドラの基本形 … 174
 - 10.7.2 アラーム・ハンドラ内での処理 … 175
 - 10.7.3 アラーム・ハンドラの生成 … 176
 - 10.7.4 アラーム・ハンドラの削除 … 177
 - 10.7.5 アラーム・ハンドラの動作開始 … 178
 - 10.7.6 アラーム・ハンドラの動作停止 … 179
 - 10.7.7 アラーム・ハンドラ詳細情報の参照 … 180
- 10.8 システム時刻 … 181
 - 10.8.1 システム時刻の設定 … 181
 - 10.8.2 システム時刻の参照 … 182
- 10.9 基本クロック用タイマ初期化ルーチン (`_RI_init_cmt_knl()`) … 183
 - 10.9.1 ユーザ・オウン・コーディング部 … 183

第 11 章 システム状態管理機能 … 185

- 11.1 概 要 … 185
- 11.2 タスクの優先順位の回転 … 185
- 11.3 RUNNING 状態のタスクの参照 … 187
- 11.4 CPU ロック状態への移行と解除 … 188
- 11.5 CPU ロック状態の参照 … 190
- 11.6 ディスパッチ禁止状態への移行と解除 … 191
- 11.7 ディスパッチ禁止状態の参照 … 193
- 11.8 コンテキスト種別の参照 … 194
- 11.9 ディスパッチ保留状態の参照 … 195

第 12 章 割り込み管理機能 … 196

- 12.1 割り込みの種類 … 196
- 12.2 RX MCU の高速割り込み … 196
- 12.3 CPU 例外の扱い … 196
- 12.4 基本クロック用タイマ割り込み … 197
- 12.5 多重割り込み … 197
- 12.6 割り込みハンドラ … 198
 - 12.6.1 割り込みハンドラの基本型 … 198
 - 12.6.2 割り込みハンドラの登録 … 199
- 12.7 処理プログラム内におけるマスカブル割り込み受け付け状態 … 199
- 12.8 マスカブル割り込みの受け付け禁止 … 200
 - 12.8.1 loc_cpu, iloc_cpu を使用して CPU ロック状態にする … 200
 - 12.8.2 chg_ims, ichg_ims を使用して PSW.IPL を変更する … 200
 - 12.8.3 PSW.I, PSW.IPL を直接変更する (ハンドラ限定) … 200

第 13 章 システム構成管理機能 … 201

- 13.1 概 要 … 201
- 13.2 バージョン情報の参照 … 201

第 14 章 オブジェクト・リセット機能 … 202

- 14.1 概 要 … 202
- 14.2 データ・キューのリセット … 202
- 14.3 メールボックスのリセット … 203
- 14.4 メッセージ・バッファのリセット … 204
- 14.5 固定長メモリ・プールのリセット … 205
- 14.6 可変長メモリ・プールのリセット … 206

第 15 章 システム・ダウン … 207

- 15.1 概要 … 207
- 15.2 ユーザ・OWN・コーディング部 … 207
 - 15.2.1 システム・ダウン・ルーチン (`_RI_sys_dwn__()`) … 207
 - 15.2.2 システム・ダウン・ルーチンのパラメータ … 208

第 16 章 スケジューリング機能 … 212

16.1	概 要	…	212
16.2	処理の単位と優先順位	…	212
16.3	タスクの駆動方式	…	212
16.4	タスクのスケジューリング方式	…	213
16.4.1	レディ・キュー	…	213
16.5	タスク・スケジューリングのロック機能	…	214
16.6	アイドルリング	…	215
16.7	非タスク内におけるタスク・スケジューリング処理	…	215

第 17 章 システム初期化処理 … 216

17.1	概 要	…	216
17.2	ブート処理ファイル (ユーザ・OWN・コーディング部)	…	217
17.2.1	ブート処理関数 (PowerON_Reset_PC())	…	217
17.2.2	kernel_ram.h および kernel_rom.h の取り込み	…	218
17.2.3	ブート処理ファイルのコンパイラ・オプション	…	218
17.2.4	ブート処理ファイルの例	…	219
17.3	カーネル初期化部 (vsta_knl, ivsta_knl)	…	221
17.4	セクション初期化関数 (_INITSCT())	…	222
17.4.1	セクション情報ファイル (ユーザ・OWN・コーディング部)	…	223
17.5	固定ベクタ・テーブル/例外ベクタ・テーブルのレジスタなど	…	225

第 18 章 データ・タイプとマクロ … 226

18.1	データ・タイプ	…	226
18.2	定数マクロ	…	227
18.3	関数マクロ	…	232
18.3.1	エラー・コード関連	…	232
18.3.2	データ・キュー関連	…	232
18.3.3	固定長メモリ・プール関連	…	232
18.3.4	ドメイン関連	…	232
18.3.5	アクセス許可関連	…	233

第 19 章 サービス・コール … 234

19.1	概 要	…	234
19.1.1	サービス・コールの呼び出し方法	…	235
19.2	サービス・コール解説	…	236
19.2.1	タスク管理機能	…	238
19.2.2	タスク付属同期機能	…	262
19.2.3	タスク例外処理機能	…	280
19.2.4	同期通信機能 (セマフォ)	…	290
19.2.5	同期通信機能 (イベントフラグ)	…	303
19.2.6	同期通信機能 (データ・キュー)	…	320
19.2.7	同期通信機能 (メールボックス)	…	341
19.2.8	拡張同期通信機能 (ミューテックス)	…	355
19.2.9	拡張同期通信機能 (メッセージ・バッファ)	…	370
19.2.10	メモリ・プール管理機能 (固定長メモリ・プール)	…	391
19.2.11	メモリ・プール管理機能 (可変長メモリ・プール)	…	406

- 19.2.12 時間管理機能 … 421
- 19.2.13 システム状態管理機能 … 444
- 19.2.14 割り込み管理機能 … 461
- 19.2.15 システム構成管理機能 … 465
- 19.2.16 オブジェクト・リセット機能 … 468
- 19.2.17 メモリ・オブジェクト保護機能 … 474

第 20 章 システム・コンフィギュレーション・ファイル … 485

- 20.1 概 要 … 485
- 20.2 デフォルト・システム・コンフィギュレーション・ファイル … 486
- 20.3 コンフィギュレーション情報（静的 API） … 486
- 20.4 システム情報（system） … 487
- 20.5 system.context の注意事項 … 490
 - 20.5.1 FPU, DSP に関する注意事項 … 490
 - 20.5.2 コンパイラの “fint_register”, “base”, および “pid” オプションとの関係 … 492
- 20.6 基本クロック割り込み情報（clock） … 493
- 20.7 最大 ID（maxdefine） … 495
- 20.8 ドメイン定義（domain） … 501
- 20.9 メモリ・オブジェクト定義（memory_object[]） … 502
- 20.10 タスク情報（task[]） … 504
- 20.11 セマフォ情報（semaphore[]） … 507
- 20.12 イベントフラグ情報（flag[]） … 509
- 20.13 データ・キュー情報（dataqueue[]） … 511
- 20.14 メールボックス情報（mailbox[]） … 513
- 20.15 ミューテックス情報（mutex[]） … 515
- 20.16 メッセージ・バッファ情報（message_buffer[]） … 516
- 20.17 固定長メモリ・プール情報（memorypool[]） … 518
- 20.18 可変長メモリ・プール情報（variable_memorypool[]） … 520
- 20.19 周期ハンドラ情報（cyclic_hand[]） … 522
- 20.20 アラーム・ハンドラ情報（alarm_hand[]） … 525
- 20.21 可変ベクタ情報（interrupt_vector[]） … 527
- 20.22 固定ベクタ／例外ベクタ情報（interrupt_fvector[]） … 529
- 20.23 RAM 使用量の算出 … 532
 - 20.23.1 BRI_RAM, RRI_RAM セクション … 532
 - 20.23.2 BURI_HEAP セクション … 535
 - 20.23.3 SURI_STACK セクション … 535
 - 20.23.4 SI セクション … 535
- 20.24 記述例 … 536

第 21 章 コンフィギュレータ cfg600px … 541

- 21.1 概 要 … 541
- 21.2 cfg600px の起動 … 542
 - 21.2.1 コマンド・ラインからの起動 … 542
 - 21.2.2 CubeSuite+ からの起動 … 542

第 22 章 テーブル生成ユーティリティ mkritblpx … 543

- 22.1 概 要 … 543
- 22.2 mkritblpx の起動 … 544
 - 22.2.1 コマンド・ラインからの起動 … 544
 - 22.2.2 CubeSuite+ からの起動 … 544
- 22.3 注意事項 … 544

付録 A ウィンドウ・リファレンス … 545

- A.1 説 明 … 545

付録 B 浮動小数点演算機能 … 559

- B.1 タスクおよびタスク例外処理ルーチンで浮動小数点演算命令を使用する場合 … 559
- B.2 ハンドラで浮動小数点演算命令を使用する場合 … 559

付録 C DSP 機能 … 560

- C.1 タスクおよびタスク例外処理ルーチンで DSP 機能命令を使用する場合 … 560
- C.2 割り込みハンドラでの ACC レジスタの保証 … 560

付録 D スタック使用量の算出 … 561

- D.1 スタックの種類 … 561
- D.2 Call Walker … 561
- D.3 ユーザ・スタック使用量の算出 … 562
- D.4 システム・スタック使用量の算出 … 563

第1章 概 説

1.1 概 要

RI600PXは、効率のよいリアルタイム処理環境、および、マルチタスク処理環境を提供するとともに、対象CPUの制御機器分野における応用範囲を拡大することを目的として開発された“リアルタイム・マルチタスクOS”です。

また、実行環境に組み込んで使用することを前提として開発されているため、ROM化を意識し、コンパクトな設計が行われています。

RI600PXは、“リアルタイム・マルチタスクOS”として普及している μ ITRON4.0仕様に準拠しており、さらにITRON4.0仕様保護機能拡張のメモリ保護機能をサポートしています。

1.1.1 リアルタイム OS

制御機器分野におけるシステムでは、内外の事象変化に対するリアルタイム性が要求されます。しかし、従来のシステムでは、このような要求をユーザが用意した単純な割り込み処理で対処してきたため、制御機器が高性能化、多様化するにつれ、単純な割り込み処理だけではの対処が困難になってきています。

つまり、処理プログラム量の増大、システムの複雑化により、内外の事象変化に対する処理を“どのような順序で実行させるのか”を管理することが煩雑になってきたといえます。

そこで、このような問題を解決するために考えられたのが“リアルタイムOS”です。

リアルタイムOSは、内外の事象変化に対するリアルタイム性を保証するとともに、最適な処理プログラムを最適な順序で実行させることを主な目的（仕事）としています。

1.1.2 マルチタスク OS

OSの世界では、OSの管理下で実行する処理プログラムを“タスク”、1つのプロセッサ上で複数のタスクを同時実行させることを“マルチタスキング”と呼んでいます。

しかし、厳密にはプロセッサ自体は一度に1つのタスク（命令）しか実行することができないため、タスクの実行を何らかの基準（きっかけ）を利用して非常に短い間隔で切り替えることにより、疑似的に複数のタスクが同時実行しているかのように見せています。

このように、システム内で規定されている何らかの基準を利用してタスクを切り替え、タスクの並列処理を可能としたのが“マルチタスクOS”です。

マルチタスクOSは、複数のタスクを並列実行させることにより、システム全体の処理能力を向上させることを主な目的（仕事）としています。

1.1.3 メモリ保護機能

1) 高信頼性システム

プログラム・ミスプログラム・デバッグ時に検出できず、システムが市場に出荷された後で不具合が発生する可能性に対して、RI600PXはシステムの高信頼性化を実現します。メモリ・データ破壊が、特にOSなどを格納しているメモリ領域で発生すると、システムの動作が異常になる、などの危険な状態を招きます。しかし本OSではメモリ・データが破壊されることはないため、システムは正常な動作を継続でき、信頼性の高いシステムの実現に貢献します。

2) デバッグ支援

メモリ保護機能のないシステムでは、一般にはポインタ不正などでメモリ内容が破壊されても、実際に不具合現象となってはじめて気付きます。不具合の原因を特定するには、エミュレータのトレース機能などを元に解析するしかなく、それには多くの時間を費やす必要がありました。RI600PXでは、不正メモリ・アクセスの時点でそれが検出されるので、デバッグ効率が飛躍的に高まります。

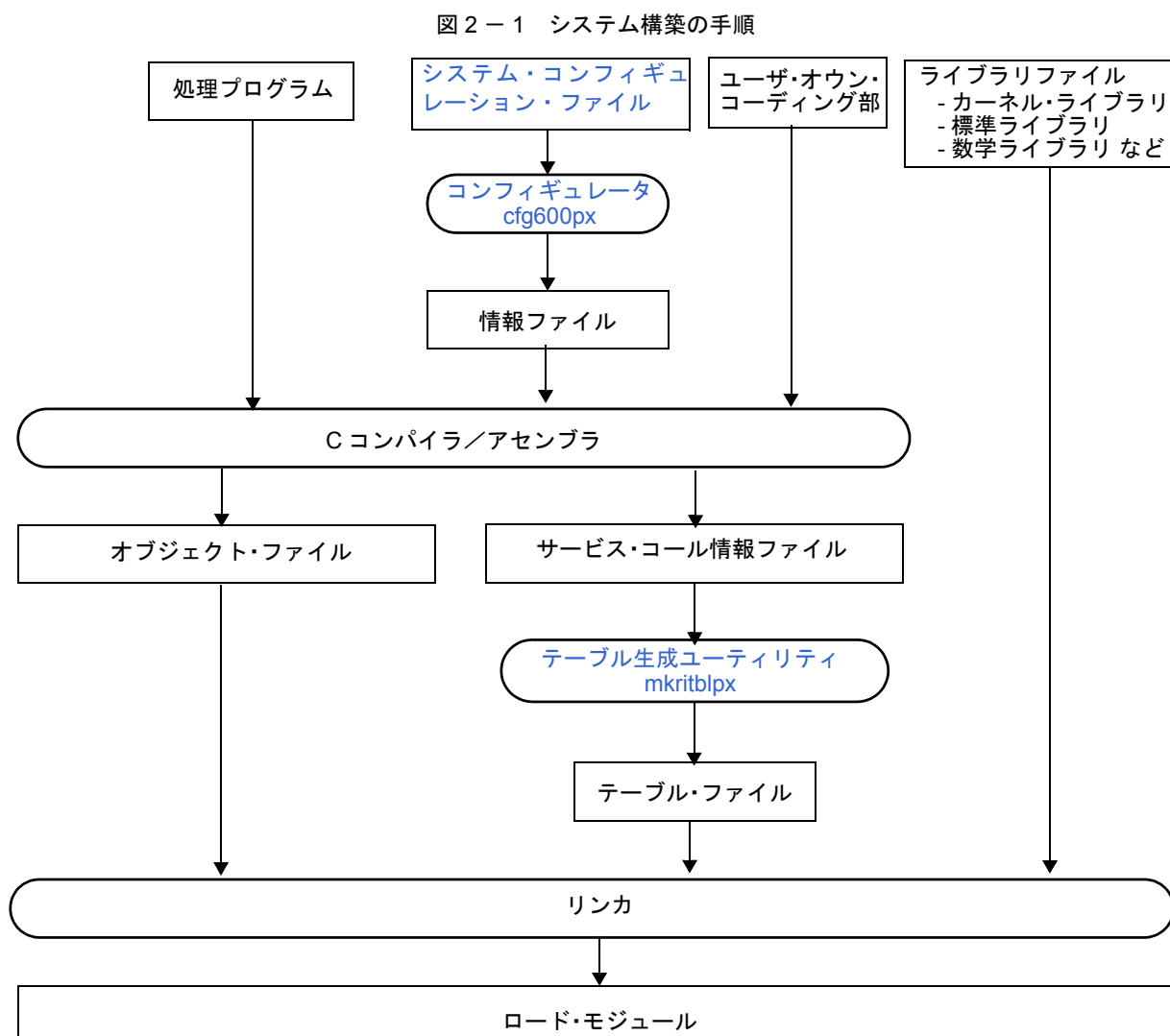
第2章 システム構築

本章では、RI600PXが提供している機能を利用したシステム(ロード・モジュール)の構築手順について解説しています。

2.1 概要

システム構築とは、RI600PX の提供媒体からユーザの開発環境（ホスト・マシン）上にインストールされたファイル群（カーネル・ライブラリなど）を用いてロード・モジュールを生成することです。

図2-1に、システム構築の手順を示します。



RI600PX では、ロード・モジュールを生成する際に必要となるファイル群のサンプル・プログラムを提供しています。サンプル・プログラムは、以下のフォルダに格納されています。ソース・ファイルは、appli サブ・フォルダに格納されています。

<ri_sample> = <CubeSuite+_root>%SampleProjects%RX% デバイス名_RI600PX

- <CubeSuite+_root>

CubeSuite+ のインストール・フォルダを表します。

デフォルトでは、“C:%Program Files%Renesas Electronics%CubeSuite+” となります。

- SampleProjects
CubeSuite+ のサンプル・プロジェクト・フォルダです。
- RX
RX MCU 用のサンプル・プロジェクト・フォルダです。
- デバイス名_RI600PX
RI600PX のサンプル・プロジェクト・フォルダです。このフォルダに、プロジェクト・ファイルがあります。
デバイス名： サンプルを提供しているデバイス名を表しています。

2.2 処理プログラムの記述

システムとして実現すべき処理を記述します。

なお、RI600PX では、処理プログラムを実現すべき処理の種類、および、用途にあわせて以下に示した 5 種類に分類しています。

- タスク

他処理プログラム（割り込みハンドラ、周期ハンドラ、およびアラーム・ハンドラ）とは異なり、RI600PX が提供するサービス・コールを使用して明示的に操作しないかぎり実行されることのない処理プログラムです。

- タスク例外処理ルーチン

タスクに対してタスク例外を要求すると、タスク毎に定義されたタスク例外処理ルーチンが起動されます。タスク例外処理ルーチンには、要求された例外要因が渡されます。

- 周期ハンドラ

指定された周期時間毎に起動されるルーチンです。

なお、RI600PX では、周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため、一定の時間が経過した際には、システム内で最高優先度を持つタスクが処理を実行中であっても、その処理は中断され、周期ハンドラに制御が移ります。

- アラーム・ハンドラ

指定した時間後に一度だけ起動されるルーチンです。

なお、RI600PX では、アラーム・ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため、一定の時間が経過した際には、システム内で最高優先度を持つタスクが処理を実行中であっても、その処理は中断され、アラーム・ハンドラに制御が移ります。

- 割り込みハンドラ

割り込みが発生した際に起動されるルーチンです。

なお、RI600PX では、割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため、割り込みが発生した際には、システム内で最高優先度を持つタスクが処理を実行中であっても、その処理は中断され、割り込みハンドラに制御が移ります。

備考 処理プログラムについての詳細は、「[第 4 章 タスク管理機能](#)」、「[第 6 章 タスク例外処理機能](#)」、「[第 10 章 時間管理機能](#)」、「[第 12 章 割り込み管理機能](#)」を参照してください。

2.3 システム・コンフィギュレーション・ファイルの記述

RI600PX に提供するデータを保持した情報ファイルを生成する際に必要となるシステム・コンフィギュレーション・ファイルを記述します。

備考 システム・コンフィギュレーション・ファイルについての詳細は、「[第 20 章 システム・コンフィギュレーション・ファイル](#)」を参照してください。

2.4 ユーザ・オウン・コーディング部の記述

- メモリ保護機能
 - アクセス例外ハンドラ (`_RI_sys_access_exception()`)
アクセス例外ハンドラは、タスクまたはタスク例外処理ルーチンが許可されていないメモリ・アクセスを行ったときに起動されます。
- 時間管理機能
 - 基本クロック用タイマ初期化ルーチン (`_RI_init_cmt_knl()`)
基本クロック用タイマ初期化ルーチンは、RI600PX 起動時に呼び出されます。
- システム・ダウン
 - システム・ダウン・ルーチン (`_RI_sys_dwn_()`)
システム・ダウン・ルーチンは、システム・ダウンが発生したときに呼び出されます。
- システム初期化処理
 - ブート処理ファイル
詳細は、「17.2 ブート処理ファイル (ユーザ・オウン・コーディング部)」を参照してください。
 - セクション情報ファイル (ユーザ・オウン・コーディング部)
未初期化データ・セクションおよび初期化データ・セクションの情報を定義します。

2.5 ロード・モジュールの生成

「2.2 処理プログラムの記述」から「2.4 ユーザ・オウン・コーディング部の記述」で作成されたファイル群、および、RI600PX、C コンパイラ・パッケージが提供しているライブラリ・ファイルに対して、CubeSuite+ 上でビルドを実行し、ロード・モジュールを生成します。

1) プロジェクトの作成／読み込み

プロジェクトの新規作成、または既存のプロジェクトの読み込みを行います。

備考 プロジェクトの新規作成、および既存のプロジェクトの読み込みについての詳細は、「RI シリーズ リアルタイム・オペレーティング・システム ユーザーズマニュアル 起動編」, 「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」, および本製品のリリースノートを参照してください。

2) ビルド対象プロジェクトの設定

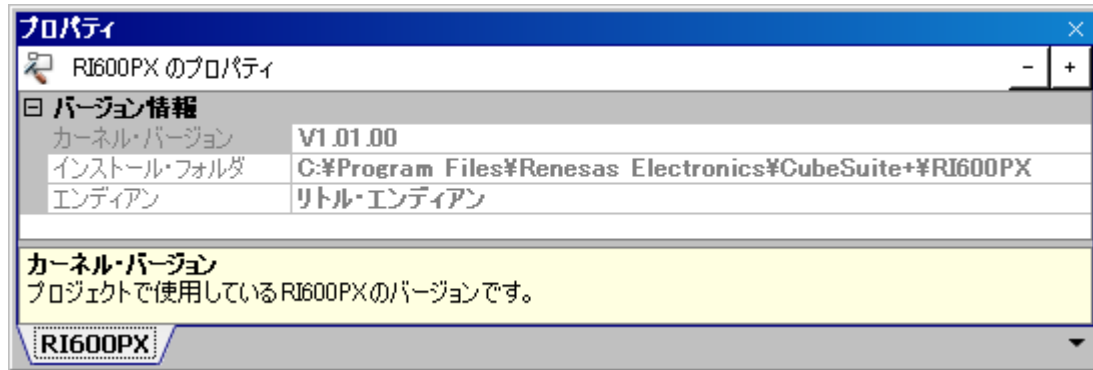
ビルドの設定や実行を行う場合は、アクティブ・プロジェクトを設定します。
なお、サブプロジェクトがない場合、プロジェクトは常にアクティブになります。

備考 アクティブ・プロジェクトの設定についての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

3) バージョンの確認

プロジェクト・ツリーでリアルタイム OS ノードを選択し、**プロパティ パネル**をオープンします。
[RI600PX] タブの**[カーネル・バージョン]**プロパティにおいて、使用する RI600PX のバージョンを確認します。

図 2-2 プロパティ パネル : [RI600PX] タブ



4) ビルド対象ファイルの設定

プロジェクトへのビルド対象ファイルの追加/削除、依存関係の更新などを行います。

備考 プロジェクトへのビルド対象ファイルの追加/削除、依存関係の更新についての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

以下に、ロード・モジュールを生成する際に必要となるファイル群の一覧を示します。

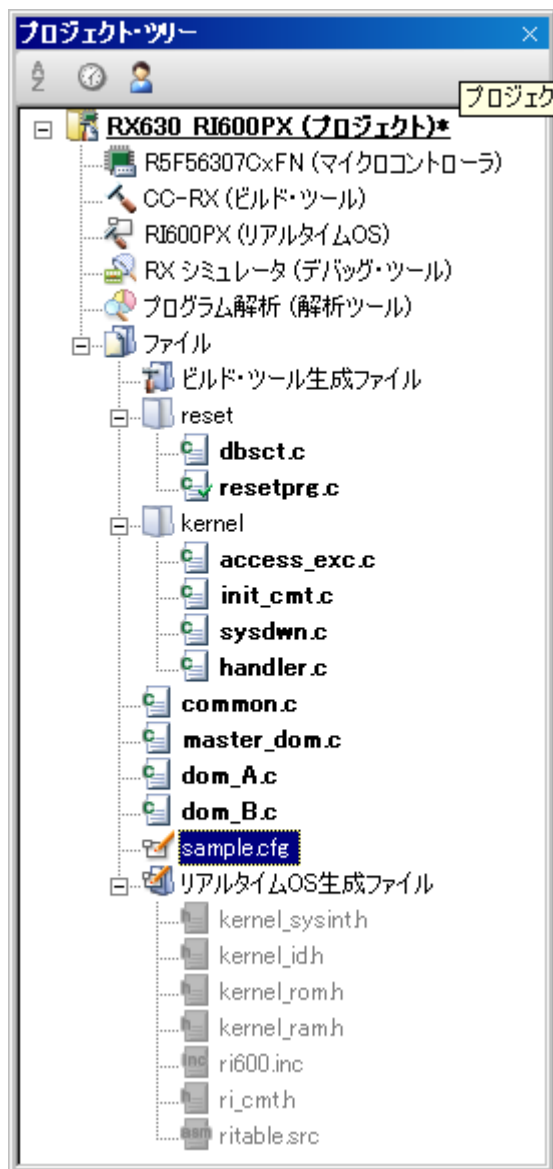
- 「[2.2 処理プログラムの記述](#)」で作成されたソース・ファイル
 - 処理プログラム (タスク, 周期ハンドラ, アラーム・ハンドラ, 割り込みハンドラ)
 - 「[2.3 システム・コンフィギュレーション・ファイルの記述](#)」で作成されたシステム・コンフィギュレーション・ファイル
 - システム・コンフィギュレーション・ファイル
- 備考** システム・コンフィギュレーション・ファイル名の拡張子は、“cfg”を指定してください。
 拡張子が異なる場合は、“cfg”が自動的に付加されます (例えば、ファイル名に“aaa.c”を指定した場合、“aaa.c.cfg”となります)。
- 「[2.4 ユーザ・OWN・コーディング部の記述](#)」で作成されたソース・ファイル
 - ユーザ・OWN・コーディング部 (システム・ダウン・ルーチン, ブート処理)
 - RI600PX が提供しているライブラリ・ファイル
 - カーネル・ライブラリ ([2.6.3 カーネル・ライブラリ](#)参照)
 - C コンパイラ・パッケージが提供しているライブラリ・ファイル
 - 標準ライブラリ, 数学ライブラリなど

備考 1 **プロジェクト・ツリー パネル**にシステム・コンフィギュレーション・ファイルを追加すると、リアルタイム OS 生成ファイル・ノードが表示されます。
 リアルタイム OS 生成ファイル・ノードには、以下の情報ファイルが表示されます。ただし、この時点では、これらのファイルは生成されません。

- システム情報ヘッダ・ファイル (kernel_id.h)
- サービス・コール定義ファイル (kernel_sysint.h)

- ROM 定義ファイル (kernel_rom.h)
- RAM 定義ファイル (kernel_ram.h)
- システム定義ファイル (ri600.inc)
- CMT タイマ定義ファイル (ri_cmt.h)
- テーブル・ファイル (ritable.src)

図 2-3 プロジェクト・ツリー パネル



備考 2 システム・コンフィギュレーション・ファイルを差し替える場合は、追加しているシステム・コンフィギュレーション・ファイルを一旦プロジェクトから外したのち、再度ファイルを追加してください。

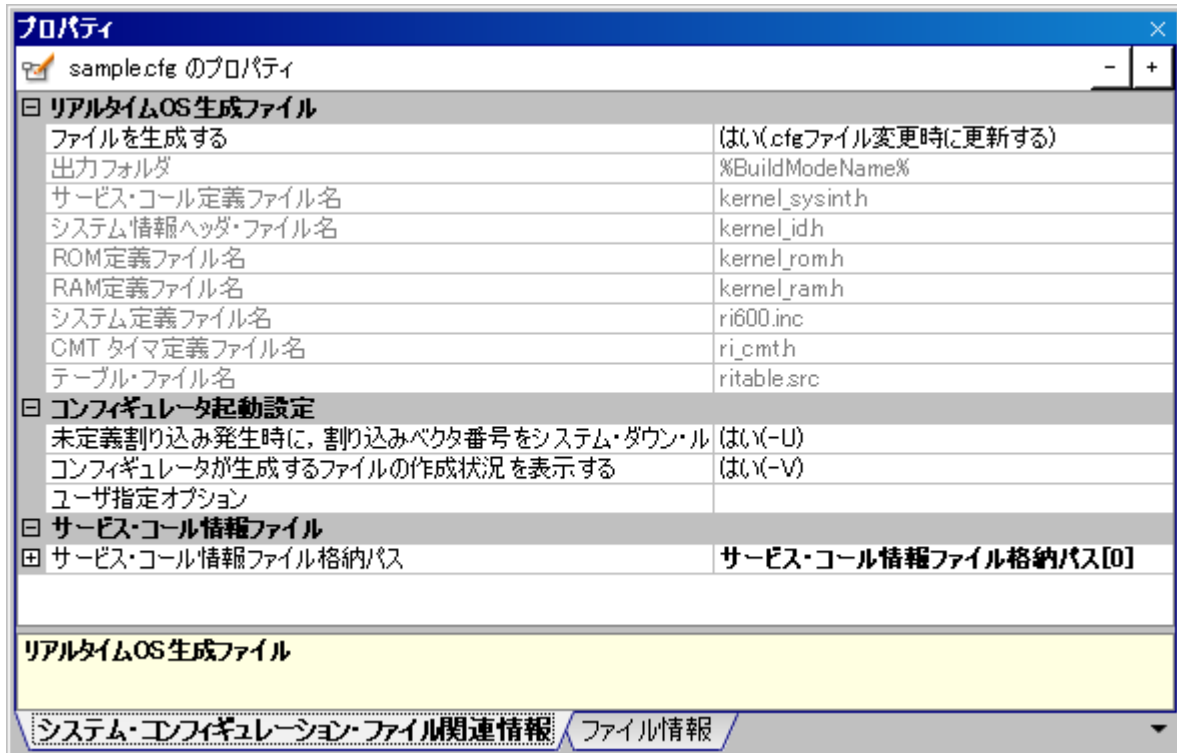
備考 3 システム・コンフィギュレーション・ファイルは、プロジェクトに複数追加することができますが、有効となるのは最初に追加したファイルです。有効なファイルをプロジェクトから外しても、追加済みのファイルは有効にならないため、再度ファイルを追加してください。

5) リアルタイム OS 生成ファイルの出力指定

プロジェクト・ツリーでシステム・コンフィギュレーション・ファイルを選択し、プロパティパネルをオープンします。

[システム・コンフィギュレーション・ファイル関連情報] タブにおいて、リアルタイム OS 生成ファイルの出力設定などを行います。

図 2-4 プロパティパネル：[システム・コンフィギュレーション・ファイル情報] タブ



6) ロード・モジュール・ファイルの出力指定

ビルドの生成物として、ロード・モジュール・ファイルを出力することを設定します。

備考 ロード・モジュールの出力指定についての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

7) ビルド・オプションの設定

コンパイラ、アセンブラ、リンカなどに対するオプションを設定します。

必ず「2.6 ビルド・オプション」を参照してください。

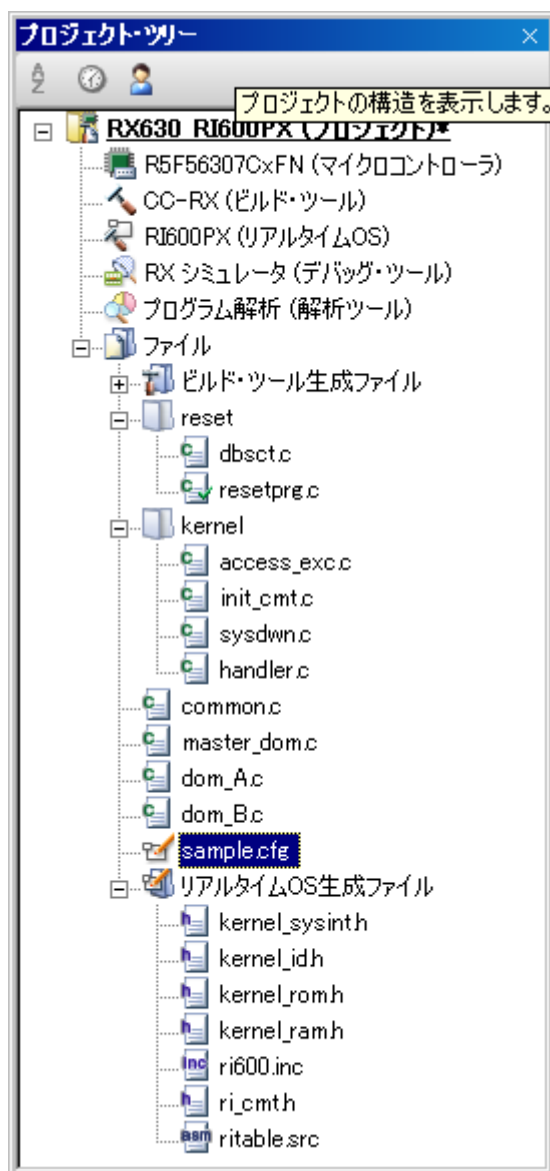
備考 ビルド・オプションの設定についての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

8) ビルドの実行

ビルドを実行し、ロード・モジュール・ファイルを生成します。

備考 ビルドの実行についての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

図2-5 プロジェクト・ツリーパネル (ビルド実行後)



9) プロジェクトの保存

プロジェクトの設定情報をプロジェクト・ファイルに保存します。

備考 プロジェクトの保存についての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

2.6 ビルド・オプション

ここでは、特に留意すべきビルド・オプションについて解説します。

2.6.1 サービス・コール情報ファイルと“-ri600_preinit_mrc”コンパイラ・オプション

サービス・コール情報ファイル（mrc ファイル）は、kernel.h をインクルードするファイルのコンパイルによって、オブジェクト・ファイルと同じフォルダに生成されます。

mrc ファイルには、ソース中で使用しているサービス・コール名が出力されます。テーブル生成ユーティリティ mkritblpx には、mrc ファイルが格納されたパスを漏れなく指定する必要があります。漏れがある場合、アプリケーションで使用しているサービス・コール・モジュールがリンクされない場合があります。この場合、そのサービス・コールを呼び出したときにシステム・ダウンとなります。

逆に、過去に生成され、現在は無効な mrc ファイルを mkritblpx に入力した場合は、アプリケーションで使用していないサービス・コール・モジュールがリンクされる場合があります。この場合、RI600PX の動作に問題は生じませんが、コード・サイズが無駄に大きくなってしまいます。

また、kernel.h をインクルードするファイルのコンパイル時には、“-ri600_preinit_mrc” オプションを指定してください。本オプションを指定しなくても RI600PX の動作に問題は生じませんが、アプリケーションで使用していないサービス・コール・モジュールがリンクされる場合があります。この場合、RI600PX の動作に問題は生じませんが、コード・サイズが無駄に大きくなってしまいます。

アプリケーションをライブラリ化する場合は、コンパイル時に生成された mrc ファイルも mkritblpx に入力してください。これが難しい場合、使用しているサービス・コール名を羅列した mrc ファイル（下記例を参照）を作成し、mkritblpx に入力してください。

なお、組み込まれていないサービス・コールを呼び出した場合は、システム・ダウンとなります。

```
sta_tsk
snd_mbx
rcv_mbx
prcv_mbx
```

2.6.2 ブート処理ファイルのコンパイラ・オプション

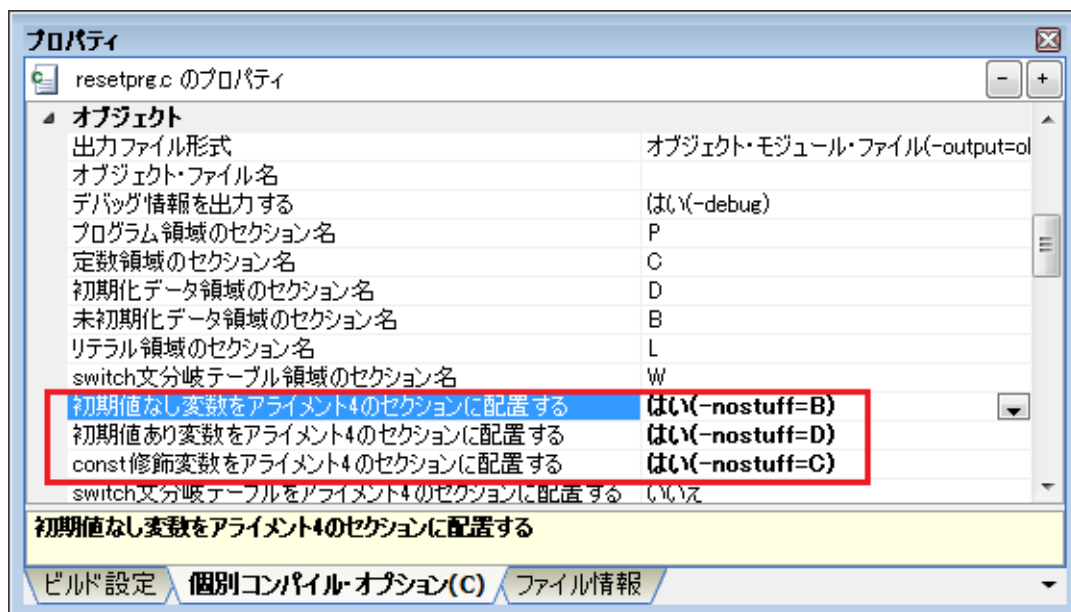
「17.2.3 ブート処理ファイルのコンパイラ・オプション」に記載のように、ブート処理ファイル（サンプル・プロジェクトでは“resetprg.c”）にはオプション“-nostuff”を設定する必要があります。そうでない場合、RI600PXは正常に動作しません。

オプション“-nostuff”をブート処理ファイルのみに設定するにはブート処理ファイルの[プロパティ]パネルの[個別コンパイル・オプション]タブで、オプション“-nostuff”を全ファイルに設定するには[CC-RX（ビルド・ツール）]の[プロパティ]パネルの[コンパイル・オプション]タブで、以下のいずれかを設定してください。

1) [オブジェクト]カテゴリで設定する

図2-6のように、[初期値なし変数をアライメント4のセクションに配置する]、[初期値あり変数をアライメント4のセクションに配置する]、および[const修飾変数をアライメント4のセクションに配置する]を”はい”に設定してください。

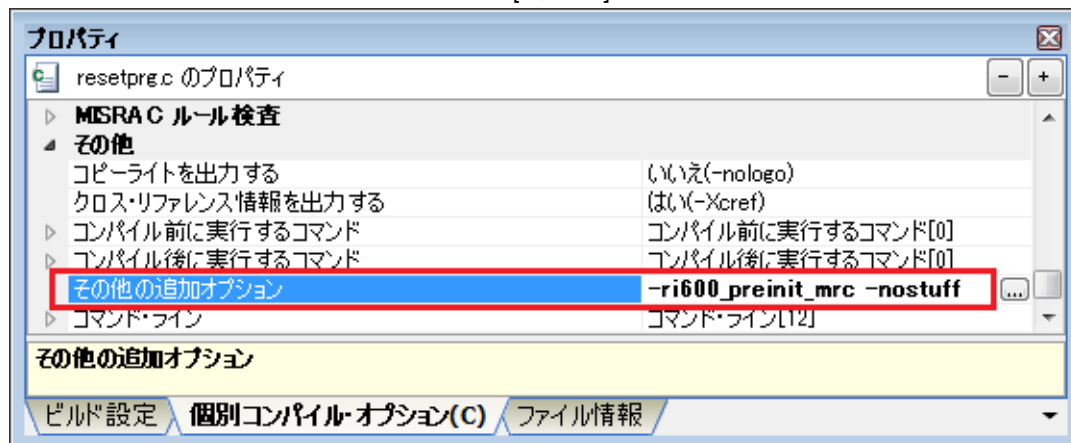
図2-6 [オブジェクト]カテゴリ



2) [その他]カテゴリで設定する

図2-7のように、[その他の追加オプション]に“-nostuff”を追加してください。

図2-7 [その他]カテゴリ



2.6.3 カーネル・ライブラリ

カーネル・ライブラリは、表 2-1 に示すフォルダに格納されています。ただし、CubeSuite+ が自動的に適切なカーネル・ライブラリをリンクするので、ユーザはカーネル・ライブラリを意識する必要はありません。

表 2-1 カーネル・ライブラリ

	フォルダ	対応コンパイラ・バージョン	対応 CPU コア	ファイル名	説明
1	<ri_root>%library%rxv1	V1.02.01 以降	RXv1 アーキテクチャ	ri600lit.lib	リトル・エンディアン用
				ri600big.lib	ビッグ・エンディアン用
2	<ri_root>%library%rxv2	V2.01.00 以降	RXv1 アーキテクチャおよび RXv2 アーキテクチャ	ri600lit.lib	リトル・エンディアン用
				ri600big.lib	ビッグ・エンディアン用

備考 1 <ri_root> は、RI600PX のインストール・フォルダを表しています。

デフォルトは、“C:%Program Files%Renesas Electronics%CubeSuite+%RI600PX”です。

備考 2 コンパイラ CC-RX V2.01 以降を使用時は項番 2、それ以外の場合は項番 1 のライブラリがリンクされます。

2.6.4 セクション配置

1) セクション名の命名規則

各セクションを適切なメモリ・オブジェクトに配置するために、以下の例のようなセクション命名規則を定めることを推奨します。

- 1文字目：セクション種別

- P：プログラム領域
- C：定数領域
- B：未初期化データ領域
- D：初期化データ領域（ROM部）
- R：初期化データ領域（RAM部）（リンカ生成）
- W：switch文分岐テーブル領域（コンパイラ生成）
- L：リテラル領域（コンパイラ生成）

- 2文字目以降

- RI*：RI600PX 予約
この領域は、ユーザ・モード（=タスク・コンテキスト）からアクセスされません。
- U*：メモリ・オブジェクトまたはユーザ・スタック
この領域は、ユーザ・モード（=タスク・コンテキスト）からアクセスされます。
- S*：上記以外
この領域は、ユーザ・モード（=タスク・コンテキスト）からアクセスされません。

2) RI600PX のセクション

表 2-2 に、RI600PX のセクションを示します。

アプリケーションでは、SURI_STACK および BURI_HEAP 以外の RI600PX のセクションを使用しないでください。

表 2-2 RI600PX セクション一覧

セクション名	属性	アライメント数	ROM/RAM	意味
PRI_KERNEL	CODE	1	ROM/RAM	RI600PX プログラム
CRI_ROM	ROMDATA	4	ROM/RAM	RI600PX 定数
DRI_ROM	ROMDATA	4	ROM/RAM	RI600PX 初期化データ (ROM)
FIX_INTERRUPT_VECTOR	ROMDATA	4	ROM	固定ベクタ・テーブル/例外ベクタ・テーブルです。後述の「FIX_INTERRUPT_VECTOR セクション」を参照してください。
INTERRUPT_VECTOR	ROMDATA	4	ROM/RAM	可変ベクタ・テーブル (1kB)
SI	DATA	4	RAM	システム・スタック
SURI_STACK	DATA	4	RAM	ユーザ・スタック領域のデフォルトのセクション
BRI_RAM	DATA	4	RAM	<ul style="list-style-type: none"> - RI600PX 変数 - システム・コンフィギュレーション・ファイルで生成されたデータ・キュー領域 - システム・コンフィギュレーション・ファイルで生成されたメッセージ・バッファ領域 (セクション指定省略時)
RRI_RAM	DATA	4	RAM	RI600PX 初期化データ (RAM)
BURI_HEAP	DATA	4	RAM	<ul style="list-style-type: none"> - システム・コンフィギュレーション・ファイルで生成された固定長メモリ・プール領域 (セクション指定省略時) - システム・コンフィギュレーション・ファイルで生成された可変長メモリ・プール領域 (セクション指定省略時) <p>通常、本セクションはメモリ・オブジェクトにしてください。</p>

3) FIX_INTERRUPT_VECTOR セクション

コンフィギュレータ `cfg600px` は、システム・コンフィギュレーション・ファイルの `"interrupt_fvector[]"` の定義内容に従って、固定ベクタ・テーブル/例外ベクタ・テーブルを `FIX_INTERRUPT_VECTOR` セクションとして生成します。

- RXv1 アーキテクチャ使用時

RXv1 アーキテクチャでは、固定ベクタ・テーブルは `0xFFFFFFFF80` 番地に固定されています。`FIX_INTERRUPT_VECTOR` セクションは `0xFFFFFFFF80` 番地に配置してください。`FIX_INTERRUPT_VECTOR` セクションを `0xFFFFFFFF80` 番地以外に配置した場合、RI600PX は正常に動作しません。

- RXv2 アーキテクチャ使用時

RXv2 アーキテクチャでは、RXv1 アーキテクチャの「固定ベクタ・テーブル」は「例外ベクタ・テーブル」と呼称が変更され、その先頭アドレスを `EXTB` レジスタで設定できるようになりました。`EXTB` レジスタのリセット時の初期値は、RXv1 アーキテクチャの固定ベクタ・テーブルと同じ `0xFFFFFFFF80` です。`FIX_INTERRUPT_VECTOR` セクションは、通常はリンク時に `0xFFFFFFFF80` 番地に配置してください。`FIX_INTERRUPT_VECTOR` セクションを `0xFFFFFFFF80` 番地以外に配置した場合は、システム・コンフィギュレーション・ファイルの `"interrupt_fvector[31]"` (リセット・ベクタ) は無視されます。ユーザ側でリセット・ベクタ (`0xFFFFFFFFFC` 番地) を用意してください。また、ブート処理関数 (`PowerON_Reset_PC()`) で `EXTB` レジスタを `FIX_INTERRUPT_VECTOR` セクションの先頭アドレスに初期化してください。

4) "aligned_section" リンカ・オプション

以下のセクションは、`"aligned_section"` リンカ・オプションを指定する必要があります。

- システム・コンフィギュレーション・ファイルで、`memory_object[].start_address` に指定したセクション
- システム・コンフィギュレーション・ファイルで、`task[].stack_section` に指定したセクション
- `SURI_STACK` セクション

5) L および W セクションに関する注意

L セクションはリテラル領域、W セクションは `switch` 文分岐テーブル領域です。これらはコンパイラによって生成されます。これらのセクションは、`#pragma section` でセクション名を変更することはできません。

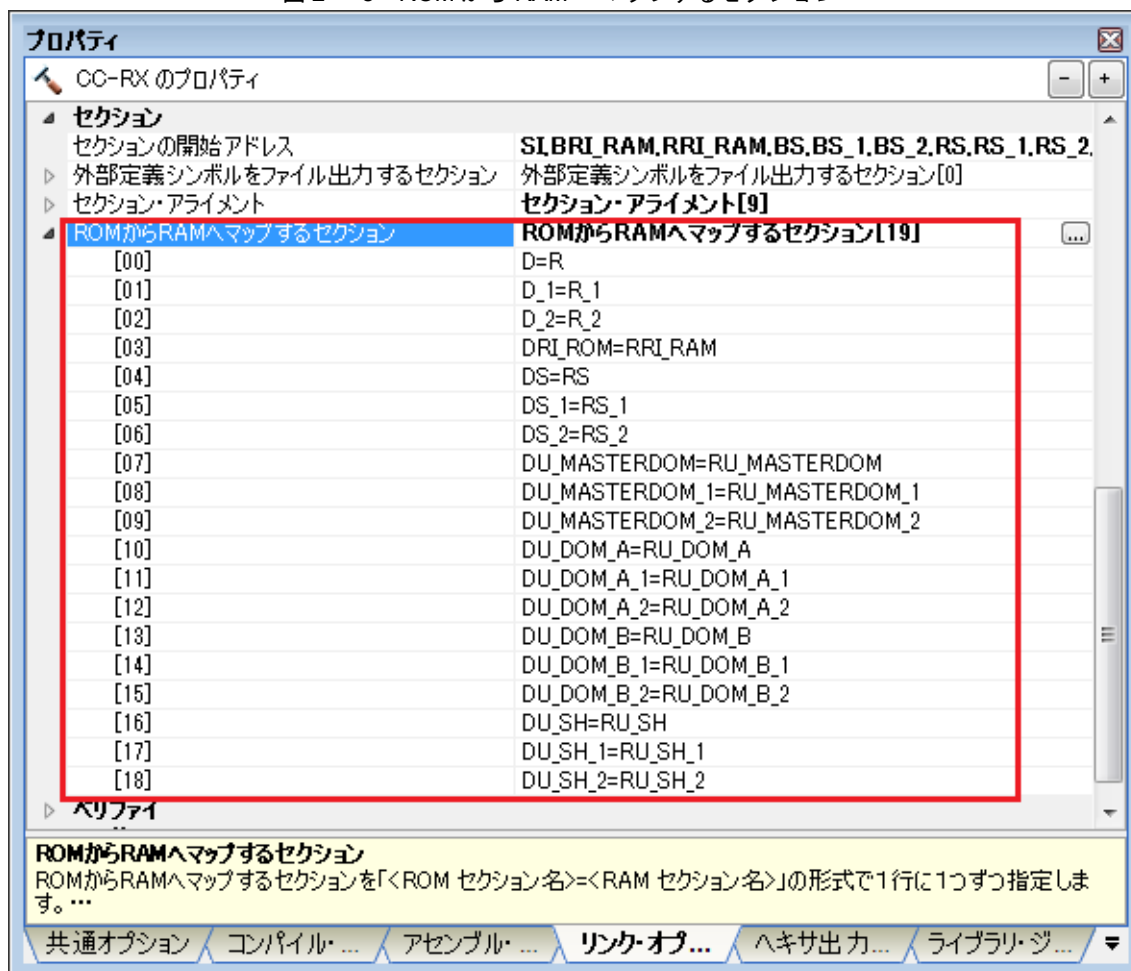
L および W セクションが生成される可能性のあるソース・ファイル中の関数をタスクの一部として実行させる場合は、以下に留意してください。

- A) ソース・ファイル中の全関数が、同一ドメインに所属するタスクとしてのみ実行する場合
特に留意事項はありません。L および W セクションを、そのドメインからリード・アクセス可能なメモリ・オブジェクトとしてください。
- B) ソース・ファイル中の関数が、複数のドメインに所属するタスクとして実行する場合
関数ごとのリテラル領域および分岐テーブル領域に異なるセクション名を付与することはできないため、それらを異なるアクセス許可を持つ個別のメモリ・オブジェクトに分離することはできません。したがって、実行時のドメインごとに関数を別ファイル化して A を適用するか、または L および W セクションを全てのドメインからリード・アクセス可能なメモリ・オブジェクトとしてください。

2.6.5 初期化データ・セクション

セクション情報ファイル（ユーザ・OWN・コーディング部）の DTBL に記述した初期化データセクションは、リンクのオプション“-rom”を用いてROMに配置されたセクションをRAM上のセクションにマップする設定を行う必要があります。図2-8のように、[CC-RX（ビルド・ツール）]の[プロパティ]パネルの[リンク・オプション]タブの[セクション]カテゴリを設定してください。

図2-8 ROMからRAMへマップするセクション



備考 RI600PXが提供するサンプルプロジェクトでは、RI600PXの“DRI_ROM”セクションの“RRI_RAM”セクションへのマップを設定済みです。

第3章 メモリ保護機能

本章では、RI600PX が提供しているメモリ保護機能について解説しています。

3.1 概要

RI600PX は、マイコンに搭載された MPU (Memory Protection Unit) を利用して、タスク・コンテキストからのメモリ・アクセスに関して、以下の保護機能を実現しています。なお、ハンドラは本機能に関係なく、すべてのアドレス空間にアクセスできます。

- 1) タスクおよびタスク例外処理ルーチンによる不正メモリ・アクセスの検出
タスクおよびタスク例外処理ルーチンは、アクセス許可された領域 (メモリ・オブジェクト) のみにアクセスできます。許可されていない領域をアクセスすると、アクセス例外ハンドラが起動されます。
- 2) ユーザ・スタック保護
各タスクのユーザ・スタックは、他のタスクからはアクセスできません。ユーザ・スタックがオーバフローしたり、タスクが他のタスクのユーザ・スタックをアクセスすると、アクセス例外ハンドラが起動されます。
- 3) RI600PX による不正メモリ・アクセスの検出
いくつかのサービス・コールは、引数としてポインタを受け取ります。RI600PX は、サービス・コールを呼び出したタスクがポインタで指定された領域をアクセス可能かを検査します。アクセス許可がない場合には、サービス・コールはエラー E_MACV を返します。
また、いくつかのサービス・コールはユーザ・スタックにタスクのコンテキスト・レジスタを保存します。このときにユーザ・スタックがオーバフローする場合には、サービス・コールはエラー E_MACV を返します。
この機能は、タスク・コンテキストから呼び出されたサービス・コールでのみ行われます。

3.2 ドメイン、メモリオブジェクト、アクセス許可ベクタ

メモリ保護機能は、以下を制御することで実現しています。

- だれが
- どこに対して
- どのアクセスが許可されるか

「だれが」に相当するものが「ドメイン」です。タスクとそのタスク例外処理ルーチンは、必ずいずれかのドメインに所属します。ドメインは 1 ~ 15 のドメイン ID で識別されます。ドメインは、システム・コンフィギュレーション・ファイルで静的に生成されます。

「どこに対して」に相当するものが「メモリ・オブジェクト」で、「どのアクセスが許可されるか」に相当するものが「アクセス許可ベクタ」です。

メモリ・オブジェクトは、通常はシステム・コンフィギュレーション・ファイルで静的に登録しますが、動的に登録 (`ata_mem` サービス・コール)、登録解除 (`det_mem` サービス・コール) することもできます。メモリ・オブジェクトは、先頭アドレスが 16 バイト境界で、かつサイズが 16 バイトの整数倍でなければなりません。

アクセス許可ベクタは、各ドメインに所属するタスクからのオペランド・リード・アクセス、オペランド・ライト・アクセス、および実行アクセスの可否を表します。

タスクから、メモリ・オブジェクトに対して許可されていないアクセスをした場合、およびメモリ・オブジェクトでも自タスクのユーザ・スタックでもない領域をアクセスした場合、アクセス例外ハンドラが起動されます。

一方、ハンドラ (割り込みハンドラ、周期ハンドラ、アラーム・ハンドラ、およびアクセス例外ハンドラ) にはドメイン所属の概念はなく、全アドレス空間にアクセスできます。

図 3-1 メモリ保護の概要

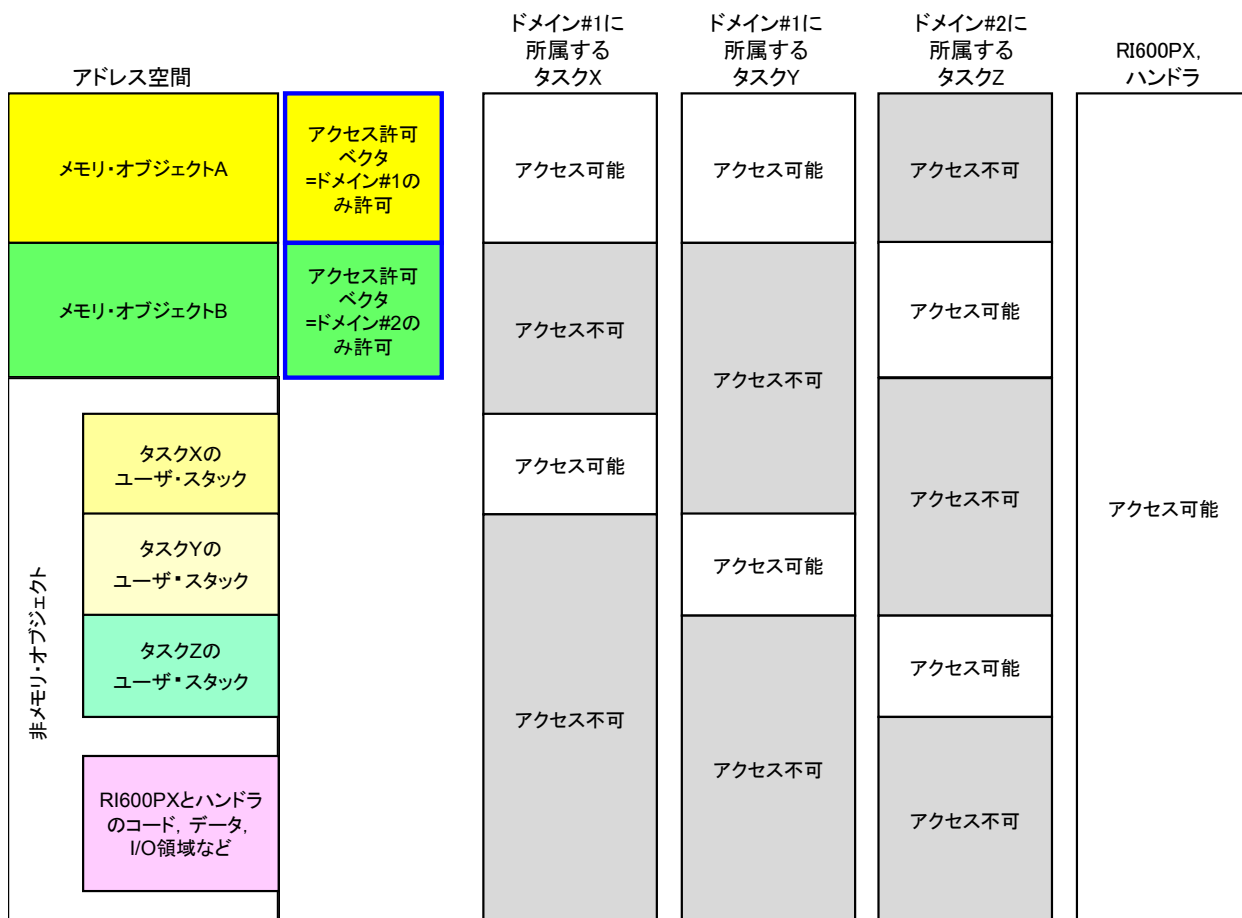


表 3-1 にメモリ・オブジェクトに関する操作をまとめます。

表 3-1 メモリ・オブジェクト操作

操作	静的 (システム・コンフィ ギュレーション・ファイル)	動的 (サービス・コール)
登録	memory_object[]	ata_mem
登録解除	—	det_mem
アクセス許可変更	—	sac_mem
アクセス権チェック (メモリ・オブジェクト以 外の領域もチェック可能)	—	vprb_mem
状態参照	—	ref_mem

3.3 メモリ・オブジェクト数の制約

あるドメインにリード・アクセス、ライト・アクセス、実行アクセスの少なくともひとつのアクセスを許可されているメモリ・オブジェクトの数は、最大 7 つです。このことに留意して、[メモリ・マップ設計](#)を行ってください。[ata_mem](#) または [sac_mem](#) によってこの制約を満たさなくなる場合には、E_OACV エラーが検出されます。

3.4 信頼されたドメイン

メモリ・アクセス以外の保護機能をサポートしていないシステムでは、例えば以下のような不正アクセスの可能性が考えられます。

- 1) 悪意のあるタスク A は、メモリ・オブジェクト M に対するアクセス許可がない。
- 2) タスク A の作成者は、メモリ・オブジェクト M に対するアクセス許可があるドメインに所属するタスク B を生成して起動するように、タスク A を実装する。
- 3) タスク B がメモリ・オブジェクト M にアクセスしても、不正アクセスは検出されない。

RI600PX では、このような悪意による不正メモリ・アクセスを防ぐために、「信頼されたドメイン」と呼ぶ機能をサポートしています。以下に示すソフトウェア構成に変更を与えるサービス・コールは、信頼されたドメインに所属するタスクからのみ呼出し可能となっています。これらのサービス・コールを信頼されていないドメインから呼び出した場合は、E_OACV エラーが検出されます。

- cre_???, acre_???, del_???, def_tex, ata_mem, det_mem, sac_mem

3.5 アクセス許可の変更

メモリ・オブジェクトに対するアクセス許可は、動的に変更 ([sac_mem](#) サービス・コール) することができます。例えば、以下のような使い方が想定されます。

アプリケーションを外部からダウンロードし、別のドメインに所属するタスクとして実行させるケースを考えます。

- 1) ダウンロードする領域をメモリ・オブジェクトとして登録 ([ata_mem](#)) する。その際、そのメモリ・オブジェクトに対し、ダウンロードを実行するタスクが所属するドメイン A からのライトアクセスを許可する。その後、メモリ・オブジェクト領域にダウンロードを行う。
- 2) ダウンロード完了後、そのメモリ・オブジェクトをドメイン B からアクセスできるように設定 ([sac_mem](#)) する。そして、ダウンロードしたコードを、ドメイン B に所属するタスクとして生成し、起動する。

3.6 ユーザ・スタック保護

各タスクのユーザ・スタックは、そのタスクのみがアクセスできます。ユーザ・スタックがオーバフローしたり、他のタスクからアクセスすると、アクセス例外ハンドラが起動されます。

また、タスクから呼び出されたサービス・コールでカーネルがユーザ・スタックを使用する場合、カーネルはスタック・ポインタがそのタスクのユーザ・スタック領域内にあるかどうかを検査します。範囲外の場合にはエラー E_MACV を返します。

3.7 アクセス許可のチェック

共通ライブラリ関数など、複数のドメインから呼び出されるプログラムでは、メモリ・アクセスが可能かどうかを判定したい場合があります。このような場合は、`vprb_mem` サービス・コールを利用してください。`vprb_mem` サービス・コールは、指定したタスクが指定した領域に対して指定したアクセスが可能かを検査します。

3.8 プロセッサのモード

MPU (Memory Protection Unit) によるメモリ保護は、ユーザ・モードのときにのみ機能します。

RI600PX では、タスク・コンテキストはユーザ・モード、非タスク・コンテキストはスーパーバイザ・モードで動作する仕様となっています。

3.9 MPU (Memory Protection Unit) の有効化

RI600PX は、起動時 (`vsta_knl`, `ivsta_knl`) に常に MPU を有効化します。システム動作中は、MPU を無効にしてはなりません。無効にした場合の動作は保証されません。

3.10 アクセス例外ハンドラ (`_RI_sys_access_exception()`)

タスクまたはタスク例外処理ルーチンが、許可されていないメモリにアクセスした時には、アクセス例外ハンドラが起動されます。アクセス例外ハンドラでは、アクセス違反の要因を取り除いて復帰するなどの処理を行うことができます。あるいは、デバッグ目的のみで使用しても構いません。

3.10.1 ユーザ・OWN・コーディング部

アクセス例外ハンドラは、ユーザ・OWN・コーディング部として実装する必要があります。

備考 RI600PX で提供するサンプルのアクセス例外ハンドラのソース・ファイルは、“access_exc.c”です。

- アクセス例外ハンドラの基本型
以下に、アクセス例外ハンドラの基本型を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */

// Access exception handler

void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr);
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr)
{
    .....
    .....
}
```

備考 アクセス例外ハンドラの関数名は“_RI_sys_access_exception”です。

- パラメータ

I/O	パラメータ	レジスタ	説明
I	UW <i>pc</i> ;	R1	アクセス例外を発生させた命令のアドレス
I	UW <i>psw</i> ;	R2	アクセス例外発生時の PSW
I	UW <i>sts</i> ;	R3	アクセス違反要因 MPU (Memory Protection Unit) の MPESTS レジスタ値が設定されます。
I	UW <i>addr</i> ;	R4	オペランド・アクセス・エラーの場合は、そのアクセス・アドレス (= MPU の MPDEA レジスタの値) が設定されます。 実行アクセス・エラーの場合は、不定です。

- スタック
アクセス例外ハンドラは、システム・スタックを使用します。
- サービス・コールの発行
アクセス例外ハンドラでは、“発行有効範囲”が“非タスク”のサービスコールを発行可能です。

- 処理開始時の PSW

表 3 - 2 アクセス例外ハンドラ処理開始時の PSW

ビット	値	備考
I	0	全割り込み禁止
IPL	アクセス例外発生前と同じ	処理開始時より下げてもなりません。
PM	0	スーパーバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

3.11 メモリ・マップ設計

本節では、メモリ・マップ設計に必要な情報を説明します。本節を参考に、セクション分割、メモリ・オブジェクト定義、リンカでのセクション配置を行ってください。

3.11.1 メモリ・オブジェクトのアドレス制約

メモリ・オブジェクトの先頭アドレスは 16 バイト境界、サイズは 16 の倍数でなければなりません。

システム・コンフィギュレーション・ファイルで、静的 API“memory_object[]”によってメモリ・オブジェクトを登録するとき、メモリ・オブジェクトのアドレスを絶対アドレスまたはセクション名で指定することができます。

絶対アドレス指定は、I/O レジスタ領域などの指定するときに使用します。

セクション名指定では、メモリ・オブジェクトを構成する先頭のセクション名と最後のセクション名を指定します。この場合、リンク時に必ず想定通りのセクション配置となるように注意してください。たとえば、メモリ・オブジェクトの先頭は 16 バイト境界でなければならぬので、メモリ・オブジェクトの先頭 (memory_object[].start_address) に指定したセクションに対し、リンカの “aligned_section” オプションを指定してください。

また、メモリ・オブジェクトのサイズは 16 の整数倍、すなわちメモリ・オブジェクトの終端アドレスは 16 の整数倍 +15 でなければなりません。しかし、メモリ・オブジェクトの最後のセクション (memory_object[].end_address) がこの通りになるとは限りません。最後のセクションの終端が 16 の倍数 + 15 でない場合には、終端 +1 から次の 16 の倍数 +15 までの範囲も、そのメモリ・オブジェクトの一部と扱われます。したがって、リンク時には終端セクションの終端 +1 から 16 の倍数 +15 の範囲には他のセクションを配置してはなりません。

- 例

memory_object[].end_address に “CU_DOM1” を指定し、CU_DOM1 セクションの終端アドレスが 0xFFFF1003 の場合、0xFFFF1004 ~ 0xFFFF100F の範囲に他のセクションを配置してはなりません。リンク時に、CU_DOM1 の後続のセクションに “aligned_section” オプションを指定することで、0xFFFF1004 ~ 0xFFFF100F にはどのセクションも配置されなくなります。

3.11.2 メモリ・オブジェクト化すべき領域

- 1) タスクがアクセスする領域
タスクがアクセスできるのは、そのタスク自身のユーザ・スタックを除くと、アクセス許可が適切に設定されたメモリ・オブジェクトのみです。したがって、タスクがアクセスするプログラムセクション、定数セクション、初期化データ・セクション、未初期化データ・セクションは、メモリ・オブジェクト内に配置する必要があります。また、タスクで I/O 領域をアクセスする場合には、その領域もメモリ・オブジェクトとする必要があります。
- 2) メールボックスで扱うメッセージ
メッセージは、送受信双方のタスクからアクセス可能なメモリ・オブジェクト内に作成する必要があります。ただし、メッセージ先頭には、カーネルの管理テーブルがあります。カーネル管理テーブルが破壊されると、システムの正常な動作は保証されません。この理由から、メッセージ通信には、データ・キューまたはメッセージ・バッファの使用を推奨します。
- 3) 固定長・可変長メモリ・プール領域
メモリ・プール領域は、メモリ・ブロックを使用するタスクからアクセス可能なメモリ・オブジェクト内とする必要があります。
ただし、RI600PX はメモリ・プール領域内に管理テーブルを生成します。カーネル管理テーブルが破壊されると、システムの正常な動作は保証されません。
 - システム・コンフィギュレーション・ファイルで固定長メモリ・プールを生成する場合
固定長メモリ・プール領域は `memorypool[].section` で指定したセクションに生成されます。`memorypool[].section` を省略した場合、固定長メモリ・プール領域は `BURI_HEAP` セクションに生成されます。
 - `cre_mpf`, `acre_mpf` で固定長メモリ・プールを生成する場合
固定長メモリ・プール領域はアプリケーション側で確保し、これらのサービス・コールでその先頭アドレスを指定します。
 - システム・コンフィギュレーション・ファイルで可変長メモリ・プールを生成する場合
可変長メモリ・プール領域は `variable_memorypool[].mpl_section` で指定したセクションに生成されます。`variable_memorypool[].mpl_section` を省略した場合、可変長メモリ・プール領域は `BURI_HEAP` セクションに生成されます。
 - `cre_mpl`, `acre_mpl` で可変長メモリ・プールを生成する場合
可変長メモリ・プール領域はアプリケーション側で確保し、これらのサービス・コールで先頭そのアドレスを指定します。

3.11.3 メモリ・オブジェクトにしてはならない領域

- 1) BURI_HEAP 以外の RI600PX のセクション
BURI_HEAP 以外の RI600PX のセクションは RI600PX のみがアクセスするため、メモリ・オブジェクトにしてはなりません。RI600PX のセクションについては、「[2.6.4 セクション配置](#)」を参照してください。
- 2) タスクのユーザ・スタック領域
タスクのユーザ・スタック領域はメモリ・オブジェクト外でなければなりません。他のユーザ・スタックおよびメモリ・オブジェクトと重なっていた場合、システムの正常な動作は保証されません。
 - システム・コンフィギュレーション・ファイルでタスクを生成する場合
ユーザ・スタック領域は `task[].stack_section` で指定したセクションに生成されます。`task[].stack_section` を省略した場合、ユーザ・スタック領域は `SURI_STACK` セクションに生成されます。
 - `cre_tsk`, `acre_tsk` でタスクを生成する場合
ユーザ・スタック領域はアプリケーション側で確保し、これらのサービス・コールでそのアドレスを指定します。
- 3) データ・キュー領域
データ・キュー領域はメモリ・オブジェクト外でなければなりません。ユーザ・スタックおよびメモリ・オブジェクトと重なっていた場合、システムの正常な動作は保証されません。
 - システム・コンフィギュレーション・ファイルでデータ・キューを生成する場合
データ・キュー領域は RI600PX の `BRI_RAM` セクションに生成されます。
 - `cre_dtq`, `acre_dtq` でデータ・キューを生成する場合
データ・キュー領域はアプリケーション側で確保し、これらのサービス・コールでそのアドレスを指定します。
- 4) メッセージ・バッファ領域
メッセージ・バッファ領域はメモリ・オブジェクト外でなければなりません。ユーザ・スタックおよびメモリ・オブジェクトと重なっていた場合、システムの正常な動作は保証されません。
 - システム・コンフィギュレーション・ファイルでメッセージ・バッファを生成する場合
データ・キュー領域は RI600PX の `BRI_RAM` セクションに生成されます。
 - `cre_mbf`, `acre_mbf` でメッセージ・バッファを生成する場合
メッセージ・バッファ領域はアプリケーション側で確保し、これらのサービス・コールでそのアドレスを指定します。
- 5) 固定長メモリ・プール管理領域
固定長メモリ・プール管理領域はメモリ・オブジェクト外でなければなりません。ユーザ・スタックおよびメモリ・オブジェクトと重なっていた場合、システムの正常な動作は保証されません。
 - システム・コンフィギュレーション・ファイルで固定長メモリ・プールを生成する場合
固定長メモリ・プール管理領域は RI600PX の `BRI_RAM` セクションに生成されます。
 - `cre_mpf`, `acre_mpf` で固定長メモリ・プールを生成する場合
固定長メモリ・プール管理領域はアプリケーション側で確保し、これらのサービス・コールでそのアドレスを指定します。

第4章 タスク管理機能

本章では、RI600PX が提供しているタスク管理機能について解説しています。

4.1 概要

RI600PX におけるタスク管理機能では、タスクの生成／起動／終了などといったタスクの状態を操作する機能のほか、優先度の参照、タスク詳細情報の参照などといったタスクの状態を参照する機能も提供しています。

4.2 タスク

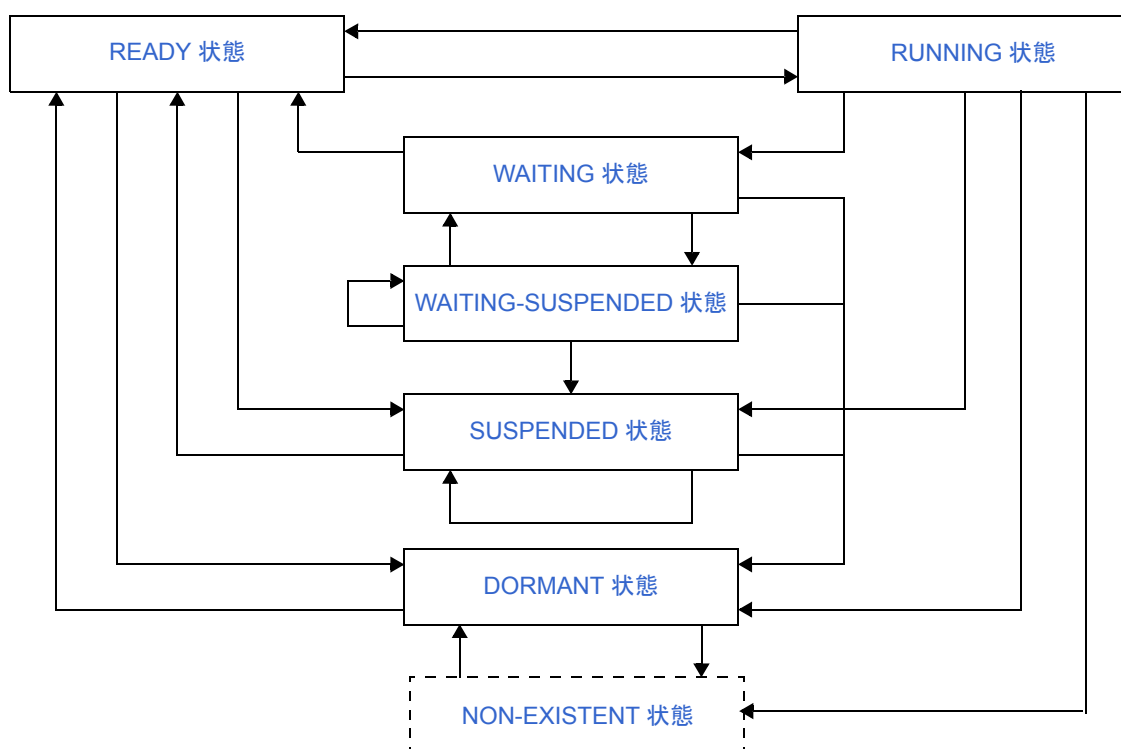
タスクは、他処理プログラム（割り込みハンドラ、周期ハンドラ、およびアラーム・ハンドラ）とは異なり、RI600PX が提供するサービス・コールを使用して明示的に操作しないかぎり実行されることのない処理プログラムです。

備考 タスクが処理を実行するうえで必要となるプログラム・カウンタ、汎用レジスタなどの実行環境情報は、“タスク・コンテキスト”と呼ばれ、タスクの実行が切り替わる際には、現在実行中のタスクのタスク・コンテキストがセーブされ、次に実行されるタスクのタスク・コンテキストがロードされます。

4.2.1 タスクの状態

タスクは、処理を実行するうえで必要となる資源の獲得状況、および、事象発生の有無などにより、さまざまな状態へと遷移していきます。そこで、RI600PXでは、各タスクが現在どのような状態にあるかを認識し、管理する必要があります。なお、RI600PXでは、タスクが取り得る状態を以下に示した7種類に分類し、管理しています。

図4-1 タスクの状態遷移



1) NON-EXISTENT 状態

RI600PX に登録されていない仮想的な状態です。

2) DORMANT 状態

タスクとして起動されていない状態、またはタスクとしての処理を終了した際に遷移する状態です。

なお、DORMANT 状態のタスクは、RI600PX の管理下でありながらも、RI600PX のスケジューリング対象からは除外されています。

3) READY 状態

処理を実行するうえで必要となる準備は整っているが、より高い優先度（同一優先度の場合もある）を持つタスクが処理を実行中のため、CPU の利用権が割り当てられるのを待っている状態です。

4) RUNNING 状態

CPU の利用権が割り当てられ、処理を実行中の状態です。

なお、RUNNING 状態のタスクは、システム全体を通して同時に複数存在することはありません。

5) WAITING 状態

処理を実行するうえで必要となる条件が整わないため、処理の実行が中断した状態です。

なお、WAITING 状態からの処理再開は、処理の実行が中断した箇所からとなります。したがって、処理を再開するうえで必要となる情報（タスク・コンテキスト：プログラム・カウンタ、汎用レジスタなど）は、中断直前の値が復元されます。

また、RI600PX では、要求条件の種類により、WAITING 状態を以下に示す 12 状態に細分化し、管理しています。

表 4 - 1 WAITING 状態の種類

状態種別	状態遷移するサービス・コール
起床待ち状態	slp_tsk または tslp_tsk
時間経過待ち状態	dly_tsk。
資源獲得待ち状態	wai_sem または twai_sem
イベントフラグ待ち状態	wai_flg または twai_flg
データ送信待ち状態	snd_dtq または tsnd_dtq
データ受信待ち状態	rcv_dtq または trcv_dtq
メッセージ受信待ち状態	rcv_mbx または trcv_mbx
ミューテックス待ち状態	loc_mtx または tloc_mtx
メッセージ・バッファ送信待ち状態	snd_mbf または tsnd_mbf
メッセージ・バッファ受信待ち状態	rcv_mbf または trcv_mbf
固定長メモリ・ブロック待ち状態	get_mpf または tget_mpf
可変長メモリ・ブロック待ち状態	get_mpl または tget_mpl

6) SUSPENDED 状態

強制的に処理の実行を中断させられた状態です。

なお、SUSPENDED 状態からの処理再開は、処理の実行が中断した箇所からの再開となります。したがって、処理を再開するうえで必要となる情報（タスク・コンテキスト：プログラム・カウンタ、汎用レジスタなど）は、中断直前の値が復元されます。

7) WAITING-SUSPENDED 状態

WAITING 状態と SUSPENDED 状態が複合した状態です。

なお、WAITING 状態が解除された際には SUSPENDED 状態へ、SUSPENDED 状態が解除された際には WAITING 状態へと遷移します。

4.2.2 タスクの優先度

タスクには、処理を実行するうえでの優先順位を決定する優先度が付けられています。そこで、RI600PX のスケジューラでは、実行可能な状態（RUNNING 状態および READY 状態）にあるタスクの優先度を参照し、その中から最も高い優先度（最高優先度）を持つタスクを選び出し、CPU の利用権を与えています。

なお、RI600PX では、“タスクの優先度”を以下に示した2種類に分類し、管理しています。

- 現在優先度

RI600PX は、以下の処理を現在優先度にしたがって実施します。

- タスクのスケジューリング（「[16.4 タスクのスケジューリング方式](#)」参照）
- 優先度順の待ちキューへのキューイング

備考 タスクが DORMANT 状態から READY 状態へと遷移した直後の現在優先度（初期優先度）は、タスク生成時に指定します。

- ベース優先度

ミューテックスを使用しない限りは、ベース優先度と現在優先度は同じです。ミューテックスを使用する場合は、「[8.2.2 現在優先度とベース優先度](#)」を参照してください。

備考 1 RI600PX におけるタスクの優先度は、その値が小さいほど、高い優先度であることを意味します。

備考 2 システム内で利用可能な優先度の範囲については、システム・コンフィギュレーション・ファイル作成時に [システム情報（system）](#) の [タスク優先度の最大値（priority）](#) を定義することにより実現されます。

4.2.3 タスクの基本型

以下に、タスクを記述する場合の基本型を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 4 参照 */
void Task1 ( VP_INT exinf );      /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    .....
    .....

    ext_tsk ( );                  /* タスクの終了 */
}

```

備考 1 *exinf* には、以下の情報が渡されます。

起動方法	<i>exinf</i>
タスク生成時に TA_ACT 属性を指定	タスクの生成時に指定した拡張情報
<i>act_tsk</i> または <i>iact_tsk</i>	
<i>sta_tsk</i> または <i>ista_tsk</i>	<i>sta_tsk</i> または <i>ista_tsk</i> で指定した起動コード <i>stacd</i>

備考 2 タスク内で return 命令が発行された場合、*ext_tsk* と同等の処理が実行されます。

備考 3 タスクの拡張情報についての詳細は、「4.5 タスクの起動」を参照してください。

備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

4.2.4 タスク内での処理

RI600PX では、タスクを切り替える際、独自のスケジューリング処理を行っています。このため、タスクを記述する際には、以下に示す注意点ががあります。

- スタック
タスクは、**タスク情報 (task[])**、**cre_tsk** または **acre_tsk** において指定されたユーザ・スタックを使用します。
- サービス・コールの発行
タスクでは、“発行有効範囲”が“タスク”のサービスコールを発行可能です。
- 処理開始時の PSW

表 4-2 タスク処理開始時の PSW

ビット	値	備考
I	1	全割り込み受け付け可能
IPL	0	
PM	1	ユーザ・モード
U	1	ユーザ・スタック
C, Z, S, O	不定	
その他	0	

- 処理開始時の FPSW
システム情報 (system) の**タスク・コンテキスト・レジスタ (context)** に FPSW を含む設定をした場合の処理開始時の FPSW を表 4-3 に示します。それ以外の場合は不定です。

表 4-3 タスク処理開始時の FPSW

コンパイラ・オプション		値
-round	-denormalize	
nearest (デフォルト)	off (デフォルト)	0x00000100 (DN ビットのみ 1)
	on	0
zero	off (デフォルト)	0x00000101 (DN ビットと RM ビットのみ 1)
	on	1 (RM ビットのみ 1)

4.2.5 タスク実行時のプロセッサ・モード

タスクは、常にユーザ・モードで実行されます。タスクををスーパーバイザ・モードで実行させることはできません。スーパーバイザ・モードで実行させたい処理は、INT 命令の割り込みハンドラとして実装してください。

例えば、CPU を低消費電力モードに遷移させる WAIT 命令は特権命令なので、スーパーバイザ・モードで実行する必要があります。

なお、INT #1 ~ 8 は、RI600PX で予約されているので、使用しないでください。

4.3 タスクの生成

タスクは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API“task[]” を使用してタスクを生成します。
静的 API“task[]” の詳細は、「[20.10 タスク情報 \(task\[\]\)](#)」を参照してください。
- 2) `cre_tsk` または `acre_tsk` サービスコールによる生成
`cre_tsk` は、パラメータ `pk_ctsk` が指す領域に設定されたタスク生成情報にしたがって、パラメータ `tskid` で指定されたタスク ID のタスクを生成します。
`acre_tsk` は、パラメータ `pk_ctsk` が指す領域に設定されたタスク生成情報にしたがってタスクを生成し、生成されたタスク ID を返します。
指定するタスク生成情報は、以下の通りです。
 - タスク属性 (`tskatr`)
以下を指定します。
 - タスクが所属するドメイン
ドメインについては、「[3.2 ドメイン, メモリオブジェクト, アクセス許可ベクタ](#)」を参照してください。
 - 生成後に起動する指定 (`TA_ACT` 属性)
生成されたタスクの状態は、`TA_ACT` 属性を指定した場合は `READY` 状態、`TA_ACT` 属性を指定しなかった場合は `DORMANT` 状態となります。
 - 拡張情報 (`exinf`)
 - タスクの実行開始アドレス (`task`)
 - タスク起動時優先度 (`itskpri`)
 - ユーザ・スタック・サイズ (`stksz`)、ユーザ・スタック領域の先頭アドレス (`stk`)
ユーザ・スタック領域は、以下の条件を満たさなければなりません。
 - 先頭アドレスが 16 バイト境界
 - サイズが 16 の倍数
 - メモリ・オブジェクトおよび他のタスクのユーザ・スタック領域と重ならないこと

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_tsk` の記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */

extern void Task2 ( VP_INT exinf );
#pragma section B SU_STACK2      /* ユーザ・スタック領域のセクション */
                                   /* リンク時に 16 バイト境界アドレスに配置必要 */
#define STKSZ2 256               /* ユーザ・スタック・サイズは 16 の倍数 */
static UW Stack2[STKSZ2/sizeof(UW)]; /* ユーザ・スタック領域 */
#pragma section

#pragma task Task1                /* 備考参照 */
void Task1 ( VP_INT exinf );     /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER_ID    tskid;              /* 変数の宣言 */
    T_CTSK   pk_ctsk = {        /* 変数の宣言, 初期化 */
        TA_DOM(1)|TA_ACT,      /* タスク属性 (tskatr) */
        0,                     /* 拡張情報 (exinf) */
        (FP)Task2,             /* タスクの実行開始アドレス (task) */
        1,                     /* 起動時優先度 (itskpri) */
        STKSZ2,                /* ユーザ・スタック・サイズ (stksz) */
        (VP)Stack2             /* ユーザ・スタック領域の先頭アドレス (stk) */
    };
    .....
    .....
    tskid = acre_tsk ( &pk_ctsk ); /* タスクの生成 */
    .....
    .....
}

```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.4 タスクの削除

タスクは、以下のいずれかの方法で削除します。

- `exd_tsk` サービス・コールによる自タスクの削除
`exd_tsk` は、自タスクを正常終了し、さらに削除します。
 以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    .....
    .....

    exd_tsk( );                  /* 自タスクの正常終了と削除 */
}
```

備考 1 本サービス・コールでは、自タスクがミューテックスをロックしていた場合には、ロック状態の解除(`unl_mtx` と同等の処理) もあわせて行われます。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `del_tsk` サービス・コールによる他タスクの削除
`del_tsk` は、パラメータ `tskid` で指定された DORMANT 状態の他のタスクを削除します。
 本サービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。
 以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考参照 */
void Task1 ( VP_INT exinf );     /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;           /* 変数の宣言、初期化 */

    .....
    .....

    del_tsk( tskid );           /* 他タスクの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.5 タスクの起動

RI600PX では、タスクの起動において、“起動要求をキューイングする”、“起動要求をキューイングしない”の2種類のインタフェースを用意しています。

4.5.1 起動要求をキューイングする起動

タスクの起動（起動要求をキューイングする）は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `act_tsk`, `iact_tsk`

パラメータ `tskid` で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち、初期優先度のレディ・キューの最後尾にキューイングします。これにより、対象タスクは、RI600PX のスケジューリング対象となります。ただし、本サービス・コールを発行した際、対象タスクが DORMANT 状態以外の場合には、対象タスクのキューイング処理、および、状態操作処理は行わず、対象タスクに起動要求をキューイング（起動要求カウンタに1を加算）しています。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考3 参照 */
void Task1 ( VP_INT exinf );           /* 備考3 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                  /* 変数の宣言, 初期化 */

    .....

    act_tsk ( tskid );                 /* タスクの起動 (起動要求をキューイングする) */

    .....
}
```

備考1 RI600PX が管理する起動要求カウンタは、8ビット幅で構成されています。このため、本サービス・コールでは、起動要求数が255回を超える場合には、起動要求の発行（起動要求カウンタの加算処理）は行わず、戻り値として E_QOVR を返します。

備考2 本サービス・コールの発行により起動されたタスクには、[タスクの生成時](#)に指定した拡張情報が渡されます。

備考3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.5.2 起動要求をキューイングしない起動

タスクの起動（起動要求をキューイングしない）は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `sta_tsk`, `ista_tsk`

パラメータ `tskid` で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち、初期優先度のレディ・キューの最後尾にキューイングします。これにより、対象タスクは、RI600PX のスケジューリング対象となります。ただし、本サービス・コールでは、起動要求のキューイングが行われません。このため、対象タスクが DORMANT 状態以外の場合には、対象タスクの状態操作処理は行わず、戻り値として `E_OBJ` を返します。

なお、パラメータ `stacd` には、対象タスクに引き渡す拡張情報を指定します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                  /* 変数の宣言, 初期化 */
    VP_INT  stacd = 123;                /* 変数の宣言, 初期化 */

    .....
    .....

    sta_tsk ( tskid, stacd );           /* タスクの起動 (起動要求をキューイングしない) */

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.6 起動要求のキューイング解除

起動要求のキューイング解除は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `can_act`, `ican_act`

パラメータ `tskid` で指定されたタスクにキューイングされている起動要求をすべて解除（起動要求カウンタに 0 を設定）します。

なお、正常終了時は戻り値として本サービス・コールの発行により解除した起動要求数を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );           /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER_UINT ercd;                       /* 変数の宣言 */
    ID      tskid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = can_act ( tskid );           /* 起動要求のキューイング解除 */

    if ( ercd >= 0 ) {
        .....                          /* 正常終了処理 */
        .....
    }

    .....
    .....
}
```

備考 1 本サービス・コールでは、状態操作処理は行わず、起動要求カウンタの設定処理のみを行います。したがって、READY 状態などから DORMANT 状態に遷移することはありません。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.7 タスクの終了

4.7.1 自タスクの終了

自タスクの終了は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ext_tsk`

自タスクを RUNNING 状態から DORMANT 状態へと遷移させ、レディ・キューから外します。これにより、自タスクは、RI600PX のスケジューリング対象から除外されます。

ただし、本サービス・コールを発行した際、自タスクの起動要求がキューイングされていた（起動要求カウンタ > 0）場合には、自タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち、自タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 3 参照 */
void Task1 ( VP_INT exinf );           /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    .....
    .....

    ext_tsk ( );                        /* タスクの終了 */
}
```

備考 1 本サービス・コールでは、自タスクがミューテックスをロックしていた場合には、ロック状態の解除(`unl_mtx` と同等の処理)もあわせて行われます。

備考 2 タスク内で `return` 命令が発行された場合、本サービス・コールと同等の処理が実行されます。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.7.2 他タスクの強制終了

タスクの強制終了は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ter_tsk`

パラメータ `tskid` で指定されたタスクを強制的に DORMANT 状態へと遷移させます。これにより、対象タスクは、RI600PX のスケジューリング対象から除外されます。

ただし、本サービス・コールを発行した際、対象タスクの起動要求がキューイングされていた（起動要求カウンタ > 0）場合には、対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち、対象タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                /* 変数の宣言, 初期化 */

    .....

    ter_tsk ( tskid );                /* タスクの強制終了 */

    .....
}
```

備考 1 本サービス・コールでは、対象タスクがミューテックスをロックしていた場合には、ロック状態の解除（`unl_mtx` と同等の処理）もあわせて行われます。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.8 タスク優先度の変更

タスク優先度の変更は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `chg_pri`, `ichg_pri`

`tskid` で指定されたタスクのベース優先度を `tskpri` で指定された値に変更します。

変更されたタスクのベース優先度は、タスクが終了、または本サービス・コールを呼び出すまで有効です。次回のタスク起動時、タスクのベース優先度はタスク生成時に指定した初期タスク優先度になります。

本サービス・コールは、対象タスクの現在優先度も、`tskpri` で示された値に変更します。ただし、対象タスクがミューテックスをロックしている場合は、現在優先度は変更しません。

対象タスクがミューテックスをロックしているかロックを待っている場合で、`tskpri` がそれらのミューテックスのいずれかの上限優先度よりも高い場合には、戻り値として `E_ILUSE` を返します。

現在優先度が変わった場合、以下の状態変化が生じます。

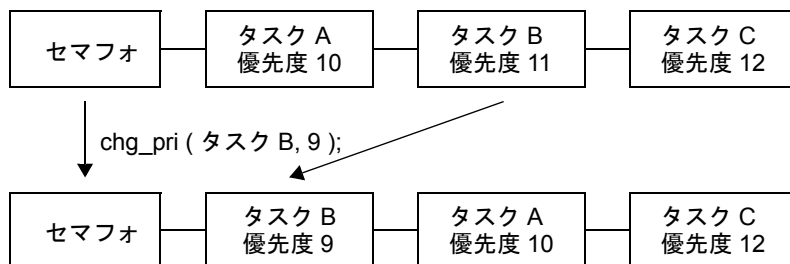
1) 対象タスクが RUNNING 状態または READY 状態の場合

本サービス・コールは、対象タスクを `tskpri` で指定された優先度に応じたレディ・キューの最後尾につなぎかえます。

2) 対象タスクが TA_TPRI 属性または TA_CEILING 属性のオブジェクトの待ちキューにキューイングされている場合

本サービス・コールは、`tskpri` で指定された優先度にしたがって対象タスクを待ちキューにつなぎかえます。待ちキューに `tskpri` で指定された現在優先度のタスクが複数ある場合は、対象タスクをそれらの中の最後尾につなぎかえます。

例 セマフォの待ちキューに3つのタスク（タスク A：優先度 10、タスク B：優先度 11、タスク C：優先度 12）が優先度順でキューイングされているとき、タスク B の優先度を 11 から 9 に変更した場合、待ちキューの順序は、以下のように変更されます。



以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;           /* 変数の宣言, 初期化 */
    PRI      tskpri = 9;        /* 変数の宣言, 初期化 */
    .....
    .....
    chg_pri ( tskid, tskpri );   /* タスク優先度の変更 */
    .....
    .....
}
  
```

備考 1 現在優先度とベース優先度については、「8.2.2 現在優先度とベース優先度」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.9 タスク優先度の参照

タスク優先度の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `get_pri`, `iget_pri`

パラメータ `tskid` で指定されたタスクの現在優先度をパラメータ `p_tskpri` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                 /* 変数の宣言, 初期化 */
    PRI     p_tskpri;                 /* 変数の宣言 */

    .....
    .....

    get_pri ( tskid, &p_tskpri );     /* タスク優先度の参照 */

    .....
    .....
}
```

備考 1 現在優先度とベース優先度については、「[8.2.2 現在優先度とベース優先度](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.10 タスク状態の参照

4.10.1 タスク詳細情報の参照

タスク詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_tsk`, `iref_tsk`

パラメータ `tskid` で指定されたタスクのタスク詳細情報（現在状態、現在優先度など）をパラメータ `pk_rtsk` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );           /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                  /* 変数の宣言, 初期化 */
    T_RTSK  pk_rtsk;                    /* データ構造体の宣言 */
    STAT    tskstat;                    /* 変数の宣言 */
    PRI     tskpri;                      /* 変数の宣言 */
    PRI     tskbpri;                     /* 変数の宣言 */
    STAT    tskwait;                     /* 変数の宣言 */
    ID      wobjid;                       /* 変数の宣言 */
    TMO     lefttmo;                      /* 変数の宣言 */
    UINT    actcnt;                       /* 変数の宣言 */
    UINT    wupcnt;                       /* 変数の宣言 */
    UINT    suscnt;                       /* 変数の宣言 */

    .....
    .....

    ref_tsk ( tskid, &pk_rtsk );        /* タスク詳細情報の参照 */

    tskstat = pk_rtsk.tskstat;           /* 現在状態の獲得 */
    tskpri = pk_rtsk.tskpri;              /* 現在優先度の獲得 */
    tskbpri = pk_rtsk.tskbpri;           /* ベース優先度の獲得 */
    tskwait = pk_rtsk.tskwait;           /* 待ち要因の獲得 */
    wobjid = pk_rtsk.wobjid;              /* 待ちオブジェクトの ID の獲得 */
    lefttmo = pk_rtsk.lefttmo;           /* タイムアウトするまでの時間の獲得 */
    actcnt = pk_rtsk.actcnt;              /* 起動要求数の獲得 */
    wupcnt = pk_rtsk.wupcnt;              /* 起床要求数の獲得 */
    suscnt = pk_rtsk.suscnt;              /* サスペンド要求数の獲得 */

    .....
    .....
}

```

備考 1 タスク詳細情報 T_RTSK についての詳細は、「[【タスク詳細情報 T_RTSK の構造】](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

4.10.2 タスク基本情報の参照

タスク基本情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- [ref_tst](#), [iref_tst](#)

パラメータ *tskid* で指定されたタスクのタスク基本情報（現在状態、待ち要因）をパラメータ *pk_rtst* で指定された領域に格納します。

タスク情報のうち、現在状態、待ち要因のみを参照したい場合に使用します。

取得する情報が少ないので [ref_tsk](#), [iref_tsk](#) より高速に応答します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                 /* 変数の宣言, 初期化 */
    T_RTST  pk_rtst;                  /* データ構造体の宣言 */
    STAT    tskstat;                  /* 変数の宣言 */
    STAT    tskwait;                  /* 変数の宣言 */

    .....
    .....

    ref_tst ( tskid, &pk_rtst );      /* タスク基本情報の参照 */

    tskstat = pk_rtst.tskstat;        /* 現在状態の獲得 */
    tskwait = pk_rtst.tskwait;       /* 待ち要因の獲得 */

    .....
    .....
}
```

備考 1 タスク基本情報 T_RTST についての詳細は、「[【タスク基本情報 T_RTST の構造】](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

第5章 タスク付属同期機能

本章では、RI600PX が提供しているタスク付属同期機能について解説しています。

5.1 概要

RI600PX におけるタスク付属同期機能では、タスクに従属した同期機能を提供しています。

5.2 起床待ち状態への移行

5.2.1 永久待ち

起床待ち状態への移行（永久待ち）は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `slp_tsk`

自タスクを RUNNING 状態から WAITING 状態（起床待ち状態）へと遷移させます。

ただし、本サービス・コールを発行した際、自タスクの起床要求がキューイングされていた（起床要求カウンタ > 0）場合には、状態操作処理は行わず、起床要求カウンタから 1 を減算します。

なお、起床待ち状態の解除は、以下の場合に行われます。

起床待ち状態の解除操作	戻り値
<code>wup_tsk</code> の発行により、起床要求が発行された。	E_OK
<code>iwup_tsk</code> の発行により、起床要求が発行された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                        /* 変数の宣言 */

    .....

    ercd = slp_tsk ( );                  /* 起床待ち状態への移行（永久待ち） */
    if ( ercd == E_OK ) {
        .....                          /* 正常終了処理 */
    } else if ( ercd == E_RLWAI ) {
        .....                          /* 強制終了処理 */
    }

    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

5.2.2 タイムアウト付き

起床待ち状態への移行（タイムアウト付き）は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `tslp_tsk`

自タスクを `RUNNING` 状態からタイムアウト付きの `WAITING` 状態（起床待ち状態）へと遷移させます。ただし、本サービス・コールを発行した際、自タスクの起床要求がキューイングされていた（起床要求カウンタ > 0）場合には、状態操作処理は行わず、起床要求カウンタから 1 を減算します。なお、起床待ち状態の解除は、以下の場合に行われます。

起床待ち状態の解除操作	戻り値
<code>wup_tsk</code> の発行により、起床要求が発行された。	<code>E_OK</code>
<code>iwup_tsk</code> の発行により、起床要求が発行された。	<code>E_OK</code>
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	<code>E_RLWAI</code>
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	<code>E_RLWAI</code>
パラメータ <code>tmout</code> で指定された待ち時間が経過した	<code>E_TMOUT</code>

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                      /* 変数の宣言 */
    TMO      tmout = 3600;             /* 変数の宣言, 初期化 */

    .....

    ercd = tslp_tsk ( tmout );        /* 起床待ち状態への移行 (タイムアウト付き) */

    if ( ercd == E_OK ) {
        .....                        /* 正常終了処理 */
    } else if ( ercd == E_RLWAI ) {
        .....                        /* 強制終了処理 */
    } else if ( ercd == E_TMOUT ) {
        .....                        /* タイムアウト処理 */
    }

    .....
}
```

備考 1 待ち時間 `tmout` に `TMO_FEVR` が指定された際には“`slp_tsk` と同等の処理”を実行します。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

5.3 タスクの起床

タスクの起床は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- wup_tsk, iwup_tsk

パラメータ *tskid* で指定されたタスクを WAITING 状態（起床待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。

ただし、本サービス・コールを発行した際、対象タスクが起床待ち状態以外の場合には、状態操作処理は行わず、起床要求カウンタに1を加算します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                 /* 変数の宣言, 初期化 */

    .....
    .....

    wup_tsk ( tskid );                /* タスクの起床 */

    .....
    .....
}
```

備考 1 RI600PX が管理する起床要求カウンタは、8 ビット幅で構成されています。このため、本サービス・コールでは、起床要求数が 255 回を超える場合には、起床要求のキューイング（起床要求カウンタの加算処理）は行わず、戻り値として E_QOVR を返します。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

5.4 起床要求の解除

起床要求の解除は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `can_wup`, `ican_wup`

パラメータ `tskid` で指定されたタスクにキューイングされている起床要求をすべて解除（起床要求カウンタに 0 を設定）します。

なお、本サービス・コールは戻り値として解除した起床要求数を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考参照 */
void Task1 ( VP_INT exinf );      /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER_UINT ercd;                 /* 変数の宣言 */
    ID      tskid = 8;            /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = can_wup ( tskid );      /* 起床要求の解除 */

    if ( ercd >= 0 ) {
        .....                    /* 正常終了処理 */
        .....
    }

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

5.5 WAITING 状態の強制解除

WAITING 状態の強制解除は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rel_wai`, `irel_wai`

パラメータ `tskid` で指定されたタスクの WAITING 状態を強制的に解除します。これにより、対象タスクは待ちキューから外れ、WAITING 状態から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

なお、本サービス・コールの発行により WAITING 状態を解除されたタスクには、WAITING 状態へと遷移するきっかけとなったサービス・コール (`slp_tsk`, `wai_sem` など) の戻り値として `E_RLWAI` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 3 参照 */
void Task1 ( VP_INT exinf );            /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                  /* 変数の宣言, 初期化 */

    .....

    rel_wai ( tskid );                  /* WAITING 状態の強制解除 */

    .....
}
```

備考 1 本サービス・コールでは、解除要求のキューイングが行われません。このため、対象タスクが WAITING 状態または WAITING-SUSPENDED 状態以外の場合には、戻り値として `E_OBJ` を返します。

備考 2 本サービス・コールでは、SUSPENDED 状態の解除は行われません。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

5.6 SUSPENDED 状態への移行

SUSPENDED 状態への移行は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- [sus_tsk](#), [isus_tsk](#)

パラメータ *tskid* で指定されたタスクを RUNNING 状態から SUSPENDED 状態へ、READY 状態から SUSPENDED 状態へ、または WAITING 状態から WAITING-SUSPENDED 状態へと遷移させます。

ただし、本サービス・コールを発行した際、対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態へと遷移していた場合には、戻り値として E_QOVR を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    sus_tsk ( tskid );                 /* SUSPENDED 状態への移行 */

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

5.7 SUSPENDED 状態の解除

5.7.1 SUSPENDED 状態の解除

SUSPENDED 状態の解除は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rsm_tsk`, `irmsm_tsk`

パラメータ `tskid` で指定されたタスクを SUSPENDED 状態から READY 状態へ、または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 3 参照 */
void Task1 ( VP_INT exinf );      /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;            /* 変数の宣言, 初期化 */

    .....

    rsm_tsk ( tskid );           /* SUSPENDED 状態の解除 */

    .....
}
```

備考 1 本サービス・コールでは、解除要求のキューイングが行われません。このため、対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態以外の場合には、戻り値として E_OBJ を返します。

備考 2 RI600PX では、サスペンド要求のキューイング機能はサポートしていません。サスペンド要求のキューイングも含めて SUSPEND 要求を解除する `frsm_tsk`, `ifrsn_tsk` もサポートしていますが、これらの振る舞いは `rsm_tsk`, `irmsm_tsk` と同じです。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

5.7.2 SUSPENDED 状態の強制解除

SUSPENDED 状態の強制解除は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- frsm_tsk, ifrsm_tsk

パラメータ *tskid* で指定されたタスクを SUSPENDED 状態から READY 状態へ、または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 3 参照 */
void Task1 ( VP_INT exinf );     /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;           /* 変数の宣言, 初期化 */

    .....

    frsm_tsk ( tskid );        /* SUSPENDED 状態の強制解除 */

    .....
}
```

備考 1 本サービス・コールでは、解除要求のキューイングが行われません。このため、対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態以外の場合には、戻り値として E_OBJ を返します。

備考 2 RI600PX では、サスペンド要求のネストはできません。このため、本サービス・コールは *rsm_tsk*, *irms_tsk* とまったく同じ処理を行います。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

5.8 時間経過待ち状態への移行

時間経過待ち状態への移行は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `dly_tsk`

自タスクをパラメータ `dlytim` で指定された遅延時間が経過するまでの間、RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。

なお、時間経過待ち状態の解除は、以下の場合に行われます。

時間経過待ち状態の解除操作	戻り値
パラメータ <code>dlytim</code> で指定された遅延時間が経過した。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    RELTIM  dlytim = 3600;       /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = dly_tsk ( dlytim );   /* 時間経過待ち状態への移行 */

    if ( ercd == E_OK ) {
        .....                /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                /* 強制終了処理 */
        .....
    }

    .....
    .....
}
```

備考 1 `dlytim` に 0 を指定すると、次回の基本クロック割り込み発生までが遅延時間となります。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

5.9 タイムアウト付き起床待ちと時間経過待ちの違い

タイムアウト付き起床待ちと時間経過待ちには、以下にのような違いがあります。

表 5-1 タイムアウト付き起床待ちと時間経過待ちの違い

	タイムアウト付き起床待ち	時間経過待ち
状態遷移するサービス・コール	<code>tslp_tsk</code>	<code>dly_tsk</code>
時間経過時の戻り値	E_TMOUT	E_OK
<code>wup_tsk</code> , <code>iwup_tsk</code> が発行された際の動作	起床	起床要求をキューイング(時間経過待ちの解除は行われない)

第6章 タスク例外処理機能

本章では、RI600PX が提供しているタスク例外処理機能について解説しています。

6.1 概要

タスクに対してタスク例外を要求すると、タスク毎に定義されたタスク例外処理ルーチンが起動されます。タスク例外処理ルーチンには、要求された例外要因が渡されます。

本機能により、タスクにおける例外処理を容易に実装することができます。

タスク例外処理機能に関するサービス・コールを以下に示します。

表 6-1 タスク例外処理機能

サービス・コール名	機能概要	発行有効範囲
def_tex	タスク例外処理ルーチンの定義	タスク
ras_tex	タスク例外処理の要求	タスク
iras_tex	タスク例外処理の要求	非タスク
dis_tex	タスク例外処理の禁止	タスク
ena_tex	タスク例外処理の許可	タスク
sns_tex	タスク例外禁止状態の参照	タスク／非タスク
ref_tex	タスク例外処理の詳細情報の参照	タスク
iref_tex	タスク例外処理の詳細情報の参照	非タスク

6.2 タスク例外処理ルーチン

タスク例外処理ルーチンでは、要求された例外要因に応じた例外処理を実装します。

6.2.1 タスク例外処理ルーチンの基本型

以下に、タスク例外処理ルーチンを記述する場合の基本型を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma taskexception Texrtn1    /* 備考 3 参照 */
void Texrtn1 ( TEXPTN texptn, VP_INT exinf ); /* 備考 3 参照 */
void Texrtn1 ( TEXPTN texptn, VP_INT exinf )
{
    .....
    .....
}
```

備考 1 *texptn* には、受理した例外要因が渡されます。

備考 2 *exinf* には、[タスクの生成時](#)に指定した拡張情報が渡されます。

備考 3 システム・コンフィギュレーション・ファイルで定義したタスク例外処理ルーチンについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

6.2.2 タスク例外処理ルーチン内での処理

- スタック
タスク例外処理ルーチンは、タスクのユーザ・スタックを使用します。
- サービス・コールの発行
タスク例外処理ルーチンでは、“発行有効範囲”が“タスク”のサービスコールを発行可能です。
- 処理開始時の PSW

表 6 - 2 タスク例外処理ルーチン処理開始時の PSW

ビット	値	備考
I	1	
IPL	タスク例外処理ルーチン起動直前のタスク本体と同じ	
PM	1	ユーザ・モード
U	1	ユーザ・スタック
C, Z, S, O	不定	
その他	0	

- 処理開始時の FPSW
システム情報 (system) のタスク・コンテキスト・レジスタ (context) に FPSW を含む設定をした場合の処理開始時の FPSW を表 6 - 3 に示します。それ以外の場合は不定です。

表 6 - 3 タスク例外処理開始時の FPSW

コンパイラ・オプション		値
-round	-denormalize	
nearest (デフォルト)	off (デフォルト)	0x00000100 (DN ビットのみ 1)
	on	0
zero	off (デフォルト)	0x00000101 (DN ビットと RM ビットのみ 1)
	on	1 (RM ビットのみ 1)

6.2.3 タスク例外処理ルーチンの起動条件

「16.4 タスクのスケジューリング方式」にしたがってスケジューリングされたタスクについて、以下のすべての条件が満たされたとき、RI600PXはそのタスク本体を実行させる代わりに、タスク例外処理ルーチンを起動します。タスク例外処理ルーチンが終了すると、元のタスク本体の処理が再開されます。

表 6-4 タスク例外処理ルーチンの起動条件

項番	起動条件
1	タスクはタスク例外許可状態である。
2	タスクの保留例外要因が0でない。

例外要因は、`TEXPTN` 型のビット・パターンで表現されます。

`ras_tex` または `iras_tex` によってタスク例外処理を要求すると、対象タスクの保留例外パターンは、`ras_tex` または `iras_tex` で指定された例外要因パターンとの論理和に更新されます。

また、タスクはタスク例外禁止状態かタスク例外許可状態のいずれかの状態を取ります。タスク起動直後はタスク例外禁止状態です。また、タスク例外処理ルーチンが定義されていない間も、タスク例外禁止状態です。自タスクが `ena_tex` を呼び出すことでタスク例外許可状態に遷移し、`dis_tex` を呼び出すことでタスク例外禁止状態に遷移します。

タスク例外処理ルーチンが起動するとタスク例外禁止状態に遷移し、タスク例外処理ルーチンが終了すると、タスク例外許可状態に遷移します。

表 6-5 タスク例外禁止状態にする操作

項番	操作	タスク例外禁止状態になるタスク
1	<code>タスクの起動</code>	起動されたタスク
2	<code>dis_tex</code>	自タスク
3	タスク例外処理ルーチンの起動	当該タスク
4	<code>タスク例外処理ルーチンの定義解除</code>	当該タスク

表 6-6 タスク例外許可状態にする操作

項番	操作	タスク例外禁止状態になるタスク
1	<code>ena_tex</code>	自タスク
2	タスク例外処理ルーチンの終了	当該タスク

6.3 タスク例外処理ルーチンの定義

タスク例外処理ルーチンは、以下のいずれかの方法で定義します。

- 1) システム・コンフィギュレーション・ファイルでの定義
システム・コンフィギュレーション・ファイルで静的API“task[]”を使用してタスクを生成する際に、タスク例外処理ルーチンを定義できます。
静的API“task[]”の詳細は、「20.10 タスク情報 (task[])」を参照してください。
- 2) `def_tex` サービスコールによる定義
`def_tex` は、パラメータ `pk_dtex` 指す領域に設定されたタスク例外処理ルーチン定義情報にしたがって、パラメータ `tskid` で指定されたタスクのタスク例外処理ルーチンを定義します。
指定するタスク例外処理ルーチン定義情報は、以下の通りです。
 - タスク例外処理ルーチン属性 (`texatr`)
TA_HLNG のみを指定できます。
 - タスク例外処理ルーチンの実行開始アドレス (`texrtn`)

`def_tex` は、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、`def_tex` によるタスク例外処理ルーチンの定義の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */

extern void Texrtn1 ( TEXPTN texptn, VP_INT exinf );

#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      tskid = TSK_SELF;           /* 変数の宣言, 初期化 */
    T_DTEX  pk_dtex = {                 /* 変数の宣言, 初期化 */
        TA_HLNG,                       /* タスク例外処理ルーチン属性 (texatr) */
        (FP) Texrtn1,                  /* 開始アドレス (texrtn) */
    };
    .....
    .....
    ercd = def_tex ( tskid, pk_dtex );  /* タスク例外処理ルーチンの定義 */
    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

6.4 タスク例外処理ルーチンの定義解除

`def_tex` で、パラメータ `pk_dtex` に NULL を指定すると、パラメータ `tskid` で指定されたタスクのタスク例外処理ルーチンの定義を解除します。

`def_tex` は、**信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に、`def_tex` によるタスク例外処理ルーチンの定義解除の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */

#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                        /* 変数の宣言 */
    ID      tskid = TSK_SELF;           /* 変数の宣言, 初期化 */
    .....
    .....
    ercd = def_tex ( tskid, (T_DTEX *)NULL ); /* タスク例外処理ルーチンの定義解除 */
    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

6.5 タスク例外処理の要求

- `ras_tex`, `iras_tex`

パラメータ `tskid` で指定されたタスクに対して、タスク例外処理を要求します。対象タスクの保留例外要因を、`rasptn` で示された値との論理和に更新します。

本サービス・コールにより、「6.2.3 タスク例外処理ルーチンの起動条件」に示す条件がすべて揃った場合には、RI600PX はタスク例外処理ルーチンを起動します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                  /* 変数の宣言, 初期化 */
    TEXPTN  rasptn = 0x00000001UL;     /* 変数の宣言, 初期化 */

    .....
    .....

    ras_tex ( tskid, rasptn );         /* タスク例外処理の要求 */

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

6.6 タスク例外処理の禁止と許可

自タスクのタスク例外処理を禁止するに `dis_tex`, 許可するには `ena_tex` を使用します。
以下に, これらのサービス・コールの記述例を示します。

```
#include      "kernel.h"           /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"        /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                 /* 備考参照 */
void Task1 ( VP_INT exinf );       /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    .....
    .....

    ena_tex ( );                   /* タスク例外許可状態へ移行 */

    .....                         /* タスク例外許可状態 */
    .....

    dis_tex ( );                  /* タスク例外禁止状態へ移行 */

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては, これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため, 記述不要です。

6.7 タスク例外処理禁止状態の参照

複数のタスクやハンドラから呼ばれる関数では、呼び出し元のタスクのタスク例外処理禁止状態を参照したい場合があります。このような場合には、`sns_tex` を使用します。

- `sns_tex`

本サービス・コールは、RUNNING 状態のタスクがタスク例外禁止状態の場合は TRUE、そうでない場合は FALSE を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ctx;                    /* 変数の宣言 */
    BOOL tex;                    /* 変数の宣言 */
    .....
    .....

    ctx = sns_ctx ( );          /* コンテキスト種別の参照 */

    if ( ctx == TRUE ) {
        .....                  /* 非タスク・コンテキスト処理 */
        .....
    } else if ( ctx == FALSE ) {
        .....                  /* タスク・コンテキスト処理 */
        .....

        tex = sns_tex ( );      /* タスク例外禁止状態の参照 */

        if ( tex == TRUE ) {
            .....              /* 自タスクはタスク例外禁止状態 */
            .....
        } else if ( tex == FALSE ) {
            .....              /* 自タスクはタスク例外許可状態 */
            .....
        }
    }
    .....
    .....
}
```


6.8 タスク例外処理の詳細情報の参照

- [ref_tex](#), [iref_tex](#)

パラメータ *tskid* で指定されたタスクのタスク例外処理の詳細情報をパラメータ *pk_rtex* で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      tskid = 8;                /* 変数の宣言, 初期化 */
    T_RTEX  pk_rtex;                 /* データ構造体の宣言 */
    STAT    texstat;                /* 変数の宣言 */
    TEXPTN  pndptn;                /* 変数の宣言 */

    .....
    .....

    ref_tex ( tskid, &pk_rtex );     /* タスク例外処理の詳細情報の参照 */

    texstat = pk_rtex.texstat;       /* タスク例外処理状態の獲得 */
    pndptn = pk_rtex.pndptn;        /* 保留例外要因の獲得 */

    .....
    .....
}
```

備考 1 タスク詳細情報 T_RTEX についての詳細は、「[【タスク基本情報 T_RTEX の構造】](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

第7章 同期通信機能

本章では、RI600PX が提供している同期通信機能について解説しています。

7.1 概要

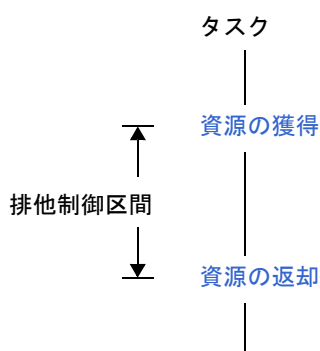
RI600PX における同期通信機能では、タスク間の排他制御、同期、通信を実現する手段としてセマフォ、イベントフラグ、データ・キュー、メールボックスを提供しています。

7.2 セマフォ

マルチタスク処理では、並行に動作するタスクが限られた数の資源（A/D コンバータ、コプロセッサ、ファイルなど）を同時に使用するといった資源使用の競合を防ぐ機能（排他制御機能）が必要となります。そこで、RI600PX では、このような資源使用の競合を防ぐ機能として“非負数の計数型セマフォ”を提供しています。

以下に、セマフォを利用した場合の処理の流れを示します。

図 7-1 処理の流れ（セマフォ）



7.2.1 セマフォの生成

セマフォは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API "semaphore[]" を使用してセマフォを生成します。
静的 API "semaphore[]" の詳細は、「[20.11 セマフォ情報 \(semaphore\[\]\)](#)」を参照してください。
- 2) `cre_sem` または `acre_sem` サービスコールによる生成
`cre_sem` は、パラメータ `pk_csem` が指す領域に設定されたセマフォ生成情報にしたがって、パラメータ `semid` で指定されたセマフォ ID のタスクを生成します。
`acre_sem` は、パラメータ `pk_csem` が指す領域に設定されたセマフォ生成情報にしたがってセマフォ生成し、生成されたセマフォ ID を返します。
指定するセマフォ生成情報は、以下の通りです。
 - セマフォ属性 (`sematr`)
以下を指定します。
 - タスク待ちキューの順序 (FIFO 順またはタスクの現在優先度順)
 - セマフォ資源数の初期値 (`isemcnt`)
 - セマフォ資源数の最大値 (`maxsem`)

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。
以下に、代表として `acre_sem` の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      semid;                      /* 変数の宣言 */
    T_CSEM  pk_csem = {                 /* 変数の宣言, 初期化 */
        TA_TFIFO,                      /* セマフォ属性 (sematr) */
        1,                              /* セマフォ資源数の初期値 (isemcnt) */
        1                                /* セマフォ資源数の最大値 (maxsem) */
    };
    .....
    .....
    semid = acre_sem ( &pk_csem );     /* セマフォの生成 */
    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.2.2 セマフォの削除

- del_sem

del_sem は、パラメータ *semid* で指定されたセマフォを削除します。

対象セマフォで *wai_sem* または *twai_sem* によって待っているタスクがある場合は、そのタスク待ち状態を解除し、*wai_sem* または *twai_sem* の戻り値として E_DLT を返します。

本サービス・コールは、**信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      semid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_sem( semid );                  /* セマフォの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

7.2.3 資源の獲得

資源の獲得は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `wai_sem` (待つ)
- `pol_sem`, `ipol_sem` (ポーリング)
- `twai_sem` (タイムアウト付きで待つ)
- `wai_sem` (待つ)

パラメータ `semid` で指定されたセマフォから資源を獲得 (セマフォ・カウンタから 1 を減算) します。

ただし、本サービス・コールを発行した際、対象セマフォから資源を獲得することができなかった (空き資源が存在しなかった) 場合には、資源の獲得は行わず、自タスクを対象セマフォの待ちキューにキューイングしたのち、`RUNNING` 状態から `WAITING` 状態 (資源獲得待ち状態) へと遷移させます。

なお、資源獲得待ち状態の解除は、以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
<code>sig_sem</code> の発行により、対象セマフォに資源が返却された。	<code>E_OK</code>
<code>isig_sem</code> の発行により、対象セマフォに資源が返却された。	<code>E_OK</code>
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	<code>E_RLWAI</code>
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	<code>E_RLWAI</code>
<code>del_sem</code> の発行により、待ち状態を強制的に解除された。	<code>E_DLT</code>

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      semid = 1;           /* 変数の宣言, 初期化 */

    .....

    ercd = wai_sem ( semid );    /* 資源の獲得 */

    if ( ercd == E_OK ) {
        .....                  /* 正常終了処理 */

        sig_sem ( semid );      /* 資源の返却 */
    } else if ( ercd == E_RLWAI ) {
        .....                  /* 強制終了処理 */
    }

    .....
}
```

備考 1 自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は、セマフォの生成時に指定した順 (FIFO 順または現在優先度順) に行われます。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `pol_sem`, `ipol_sem` (ポーリング)

パラメータ `semid` で指定されたセマフォから資源を獲得 (セマフォ・カウンタから 1 を減算) します。

ただし、本サービス・コールを発行した際、対象セマフォから資源を獲得することができなかった (空き資源が存在しなかった) 場合には、資源の獲得は行わず、戻り値として `E_TMOU` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考参照 */
void Task1 ( VP_INT exinf );     /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      semid = 1;           /* 変数の宣言, 初期化 */

    .....

    ercd = pol_sem ( semid );    /* 資源の獲得 */

    if ( ercd == E_OK ) {
        .....                  /* ポーリング成功処理 */

        sig_sem ( semid );      /* 資源の返却 */
    } else if ( ercd == E_TMOU ) {
        .....                  /* ポーリング失敗処理 */
    }

    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- twai_sem (タイムアウト付きで待つ)

パラメータ *semid* で指定されたセマフォから資源を獲得 (セマフォ・カウンタから 1 を減算) します。
 ただし、本サービス・コールを発行した際、対象セマフォから資源を獲得することができなかった (空き資源が存在しなかった) 場合には、資源の獲得は行わず、自タスクを対象セマフォの待ちキューにキューイングしたのち、**RUNNING** 状態からタイムアウト付きの **WAITING** 状態 (資源獲得待ち状態) へと遷移させます。
 なお、資源獲得待ち状態の解除は、以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
sig_sem の発行により、対象セマフォに資源が返却された。	E_OK
isig_sem の発行により、対象セマフォに資源が返却された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <i>tmout</i> で指定された待ち時間が経過した。	E_TMOUT
del_sem の発行により、待ち状態を強制的に解除された	E_DLT

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 3 参照 */
void Task1 ( VP_INT exinf );     /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      semid = 1;           /* 変数の宣言, 初期化 */
    TMO     tmout = 3600;        /* 変数の宣言, 初期化 */

    .....

                                /* 資源の獲得 */
    ercd = twai_sem ( semid, tmout );

    if ( ercd == E_OK ) {
        .....                /* 正常終了処理 */

        sig_sem ( semid );     /* 資源の返却 */
    } else if ( ercd == E_RLWAI ) {
        .....                /* 強制終了処理 */
    } else if ( ercd == E_TMOUT ) {
        .....                /* タイムアウト処理 */
    }

    .....
}
```

- 備考 1 自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は、セマフォの生成時に指定した順 (FIFO 順または現在優先度順) に行われます。
- 備考 2 待ち時間 *tmout* に **TMO_FEVR** が指定された際には “wai_sem と同等の処理” を、**TMO_POL** が指定された際には “pol_sem と同等の処理” を実行します。
- 備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは **cfg600px** が **kernel_id.h** に出力するため、記述不要です。

7.2.4 資源の返却

資源の返却は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sig_sem, isig_sem

パラメータ *semid* で指定されたセマフォに資源を返却（セマフォ・カウンタに1を加算）します。

ただし、本サービス・コールを発行した際、対象セマフォの待ちキューにタスクがキューイングされていた場合には、資源の返却（セマフォ・カウンタの加算処理）は行わず、該当タスク（待ちキューの先頭タスク）に資源を渡します。これにより、該当タスクは、待ちキューから外れ、WAITING 状態（資源獲得待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );           /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      semid = 1;                  /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = wai_sem ( semid );          /* 資源の獲得 */

    if ( ercd == E_OK ) {
        .....                          /* 正常終了処理 */
        .....

        sig_sem ( semid );             /* 資源の返却 */
    } else if ( ercd == E_RLWAI ) {
        .....                          /* 強制終了処理 */
        .....
    }

    .....
    .....
}
```

備考 1 RI600PX では、セマフォの資源数として取り得る最大値（最大資源数）をコンフィギュレーション時に定義させています。このため、本サービス・コールでは、資源数が最大資源数を超える場合には、資源の返却（セマフォ・カウンタの加算処理）は行わず、戻り値として E_QOVR を返します。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

7.2.5 セマフォ詳細情報の参照

セマフォ詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_sem`, `iref_sem`

パラメータ `semid` で指定されたセマフォのセマフォ詳細情報(待ちタスクの有無, 現在資源数)をパラメータ `pk_rsem` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考2 参照 */
void Task1 ( VP_INT exinf );           /* 備考2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      semid = 1;                  /* 変数の宣言, 初期化 */
    T_RSEM  pk_rsem;                   /* データ構造体の宣言 */
    ID      wtskid;                     /* 変数の宣言 */
    UINT    semcnt;                     /* 変数の宣言 */

    .....
    .....

    ref_sem ( semid, &pk_rsem );       /* セマフォ詳細情報の参照 */

    wtskid = pk_rsem.wtskid;           /* 待ちタスクの有無の獲得 */
    semcnt = pk_rsem.semcnt;           /* 現在資源数の獲得 */

    .....
    .....
}
```

備考1 セマフォ詳細情報 T_RSEM についての詳細は、「[【セマフォ詳細情報 T_RSEM の構造】](#)」を参照してください。

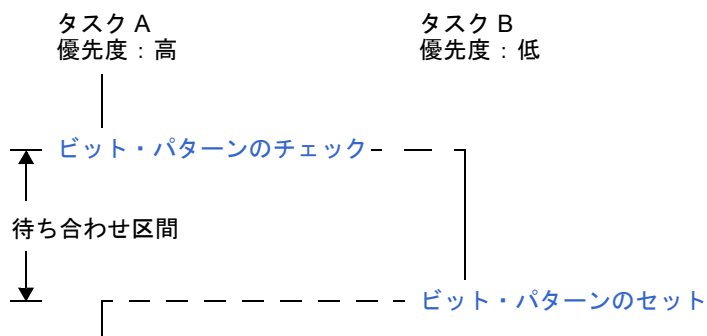
備考2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.3 イベントフラグ

マルチタスク処理では、あるタスクの処理結果が出るまでの間、他タスクが処理の実行を待つといったタスク間の待ち合わせ機能（事象の発生有無を判断できる機能）が必要となります。そこで、RI600PX では、このようなタスク間の待ち合わせ機能として“32 ビット幅のイベントフラグ”を提供しています。

以下に、イベントフラグを利用した場合の処理の流れを示します。

図7-2 処理の流れ（イベントフラグ）



7.3.1 イベントフラグの生成

イベントフラグは、以下のいずれかの方法で生成します。

1) システム・コンフィギュレーション・ファイルによる生成

システム・コンフィギュレーション・ファイルで静的 API“flag[]” を使用してイベントフラグを生成します。

静的 API“flag[]” の詳細は、「[20.12 イベントフラグ情報 \(flag\[\]\)](#)」を参照してください。

2) `cre_flg` または `acre_flg` サービスコールによる生成

`cre_flg` は、パラメータ `pk_cflg` が指す領域に設定されたイベントフラグ生成情報にしたがって、パラメータ `flgid` で指定されたイベントフラグ ID のイベントフラグを生成します。

`acre_flg` は、パラメータ `pk_cflg` が指す領域に設定されたイベントフラグ生成情報にしたがってイベントフラグを生成し、生成されたイベントフラグ ID を返します。

指定するイベントフラグ生成情報は、以下の通りです。

- イベントフラグ属性 (`flgatr`)
以下を指定します。
 - タスク待ちキューの順序 (FIFO 順またはタスクの現在優先度順)
 - イベントフラグに対して複数のタスクが待つことを禁止するかどうか
 - 待ち解除時にイベントフラグのビット・パターンを 0 クリアするかどうか
- 初期ビット・パターン (`iflgptn`)

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_flg` の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      flgid;                       /* 変数の宣言 */
    T_CFLG  pk_cflg = {                 /* 変数の宣言, 初期化 */
        TA_TFIFO|TA_WSGL|TA_CLR, /* イベントフラグ属性 (flgatr) */
        OUL                               /* 初期ビット・パターン (iflgptn) */
    };
    .....
    .....
    flgid = acre_flg ( &pk_cflg );     /* イベントフラグ生成 */
    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.3.2 イベントフラグの削除

- del_flg

パラメータ *flgid* で指定されたイベントフラグを削除します。

対象イベントフラグで *wai_flg* または *twai_flg* によって待っているタスクがある場合は、そのタスク待ち状態を解除し、*wai_flg* または *twai_flg* の戻り値として E_DLT を返します。

本サービス・コールは、**信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      flgid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_flg( flgid );                  /* イベントフラグの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

7.3.3 ビット・パターンのセット

ビット・パターンのセットは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `set_flg`, `iset_flg`

パラメータ `flgid` で指定されたイベントフラグのビット・パターンとパラメータ `setptn` で指定されたビット・パターンの論理和 OR をとり、その結果を対象イベントフラグにセットします。

そして、待ちキューの順に待ちキューにつながれているタスクの待ち解除条件を満たすかどうかを調べます。待ち解除条件を満たせば、該当タスクを待ちキューから外し、WAITING 状態（イベントフラグ待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。このとき、対象のイベントフラグ属性に TA_CLR 属性が指定されている場合には、イベントフラグのビット・パターンを 0 クリアし、処理を終了します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      flgid = 1;                  /* 変数の宣言, 初期化 */
    FLGPNTN setptn = 0x00000001UL;     /* 変数の宣言, 初期化 */

    .....
    .....

    set_flg ( flgid, setptn );         /* ビット・パターンのセット */

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.3.4 ビット・パターンのクリア

ビット・パターンのクリアは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `clr_flg`, `iclr_flg`

パラメータ `flgid` で指定されたイベントフラグのビット・パターンとパラメータ `clrptn` で指定されたビット・パターンの論理積 AND をとり、その結果を対象イベントフラグに設定します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考参照 */
void Task1 ( VP_INT exinf );     /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      flgid = 1;           /* 変数の宣言, 初期化 */
    FLGPTN  clrptn = 0xFFFFFFFFEUL; /* 変数の宣言, 初期化 */

    .....
    .....

    clr_flg ( flgid, clrptn );   /* ビット・パターンのクリア */

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.3.5 ビット・パターンのチェック

ビット・パターンのチェックは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `wai_flg` (待つ)
 - `pol_flg`, `ipol_flg` (ポーリング)
 - `twai_flg` (タイムアウト付きで待つ)
- `wai_flg` (待つ)

パラメータ `waiptn` で指定された要求ビット・パターンとパラメータ `wfmode` で指定された要求条件を満足するビット・パターンがパラメータ `flgid` で指定されたイベントフラグに設定されているか否かをチェックします。

なお、要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には、対象イベントフラグのビット・パターンをパラメータ `p_flgptn` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には、自タスクを対象イベントフラグの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (イベントフラグ待ち状態) へと遷移させます。

なお、イベントフラグ待ち状態の解除は、以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
<code>set_flg</code> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<code>iset_flg</code> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>del_flg</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、要求条件 `wfmode` の指定形式を示します。

- `wfmode = TWF_ANDW`
`waiptn` で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
- `wfmode = TWF_ORW`
`waiptn` で 1 を設定しているビットのうち、いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考4 参照 */
void Task1 ( VP_INT exinf );      /* 備考4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      flgid = 1;            /* 変数の宣言, 初期化 */
    FLGPTN  waiptn = 14;          /* 変数の宣言, 初期化 */
    MODE    wfmode = TWF_ANDW;    /* 変数の宣言, 初期化 */
    FLGPTN  p_flgptn;             /* 変数の宣言 */

    .....
    .....

                                /* ビット・パターンのチェック */
    ercd = wai_flg ( flgid, waiptn, wfmode, &p_flgptn );

    if ( ercd == E_OK ) {
        .....                  /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                  /* 強制終了処理 */
        .....
    }

    .....
    .....
}

```

- 備考1 TA_WSGL 属性（複数タスクの待ちを許さない）を指定して生成されたイベントフラグに、すでに待ちタスクがキューイングされている場合、本サービス・コールは E_ILUSE エラーを返します。
- 備考2 自タスクを対象イベントフラグ（TA_WMUL 属性）の待ちキューにキューイングする際のキューイング方式は、イベントフラグ生成時に指定した順（FIFO 順または現在優先度順）に行われます。ただし、TA_CLR 属性が指定されていない場合は、優先度順の指定の場合でも FIFO 順に行われます。この振る舞いは、μITRON4.0 仕様の範囲外です。
- 備考3 対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際、RI600PX はビット・パターンのクリア（0 の設定）を行います。
- 備考4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

- `pol_flg`, `ipol_flg` (ポーリング)

パラメータ `waitptn` で指定された要求ビット・パターンとパラメータ `wfmode` で指定された要求条件を満足するビット・パターンがパラメータ `flgid` で指定されたイベントフラグに設定されているか否かをチェックします。

なお、要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には、対象イベントフラグのビット・パターンをパラメータ `p_flgptn` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には、戻り値として `E_TMOUT` を返します。

以下に、要求条件 `wfmode` の指定形式を示します。

- `wfmode = TWF_ANDW`

`waitptn` で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- `wfmode = TWF_ORW`

`waitptn` で 1 を設定しているビットのうち、いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考 3 参照 */
void Task1 ( VP_INT exinf );     /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      flgid = 1;           /* 変数の宣言, 初期化 */
    FLGPTN  waitptn = 14;       /* 変数の宣言, 初期化 */
    MODE    wfmode = TWF_ANDW;  /* 変数の宣言, 初期化 */
    FLGPTN  p_flgptn;          /* 変数の宣言 */

    .....
    .....

                                /* ビット・パターンのチェック */
    ercd = pol_flg ( flgid, waitptn, wfmode, &p_flgptn );

    if ( ercd == E_OK ) {
        .....                /* ポーリング成功処理 */
        .....
    } else if ( ercd == E_TMOUT ) {
        .....                /* ポーリング失敗処理 */
        .....
    }

    .....
    .....
}
```

備考 1 `TA_WSGL` 属性 (複数タスクの待ちを許さない) を指定して生成されたイベントフラグに、すでに待ちタスクがキューイングされている場合、本サービス・コールは `E_ILUSE` エラーを返します。

備考 2 対象イベントフラグ (`TA_CLR` 属性) の要求条件が満足した際、RI600PX はビット・パターンのクリア (0 の設定) を行います。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `twai_flg` (タイムアウト付きで待つ)

パラメータ `waiptn` で指定された要求ビット・パターンとパラメータ `wfmode` で指定された要求条件を満足するビット・パターンがパラメータ `flgid` で指定されたイベントフラグに設定されているか否かをチェックします。

なお、要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には、対象イベントフラグのビット・パターンをパラメータ `p_flgptn` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には、自タスクを対象イベントフラグの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態 (イベントフラグ待ち状態) へと遷移させます。

なお、イベントフラグ待ち状態の解除は、以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
<code>set_flg</code> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<code>iset_flg</code> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_flg</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、要求条件 `wfmode` の指定形式を示します。

- `wfmode = TWF_ANDW`

`waiptn` で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- `wfmode = TWF_ORW`

`waiptn` で 1 を設定しているビットのうち、いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* *cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                     /* 備考 5 参照 */
void Task1 ( VP_INT exinf );           /* 備考 5 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      flgid = 1;                   /* 変数の宣言, 初期化 */
    FLGPTN  waiptn = 14;                 /* 変数の宣言, 初期化 */
    MODE    wfmode = TWF_ANDW;          /* 変数の宣言, 初期化 */
    FLGPTN  p_flgptn;                   /* 変数の宣言 */
    TMO     tmout = 3600;                /* 変数の宣言, 初期化 */

    .....

    .....

    /* ビット・パターンのチェック */
    ercd = twai_flg ( flgid, waiptn, wfmode, &p_flgptn, tmout );

    if ( ercd == E_OK ) {
        .....                          /* 正常終了処理 */
    }
}
```

```

.....
} else if ( ercd == E_RLWAI ) {
..... /* 強制終了処理 */
.....
} else if ( ercd == E_TMOUT ) {
..... /* タイムアウト処理 */
.....
}

.....
.....
}

```

- 備考 1 **TA_WSGL** 属性（複数タスクの待ちを許さない）を指定して生成されたイベントフラグに、すでに待ちタスクがキューイングされている場合、本サービス・コールは **E_ILUSE** エラーを返します。
- 備考 2 自タスクを対象イベントフラグ（**TA_WMUL** 属性）の待ちキューにキューイングする際のキューイング方式は、イベントフラグ生成時に指定した順（FIFO 順または現在優先度順）に行われます。ただし、**TA_CLR** 属性が指定されていない場合は、優先度順の指定の場合でも FIFO 順に行われます。この振る舞いは、 μ ITRON4.0 仕様の範囲外です。
- 備考 3 対象イベントフラグ（**TA_CLR** 属性）の要求条件が満足した際、RI600PX はビット・パターンのクリア（0 の設定）を行います。
- 備考 4 待ち時間 *tmout* に **TMO_FEVR** が指定された際には“wai_flg と同等の処理”を、**TMO_POL** が指定された際には“pol_flg と同等の処理”を実行します。
- 備考 5 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは **cfg600px** が **kernel_id.h** に出力するため、記述不要です。

7.3.6 イベントフラグ詳細情報の参照

イベントフラグ詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- [ref_flg](#), [iref_flg](#)

パラメータ *flgid* で指定されたイベントフラグのイベントフラグ詳細情報（待ちタスクの有無、現在ビット・パターン）をパラメータ *pk_rflg* で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      flgid = 1;                 /* 変数の宣言, 初期化 */
    T_RFLG  pk_rflg;                   /* データ構造体の宣言 */
    ID      wtskid;                    /* 変数の宣言 */
    FLGPNTN flgpntn;                  /* 変数の宣言 */

    .....
    .....

    ref_flg ( flgid, &pk_rflg );      /* イベントフラグ詳細情報の参照 */

    wtskid = pk_rflg.wtskid;          /* 待ちタスクの有無の獲得 */
    flgpntn = pk_rflg.flgpntn;       /* 現在ビット・パターンの獲得 */

    .....
    .....
}
```

備考 1 イベントフラグ詳細情報 T_RFLG についての詳細は、「[【 イベントフラグ詳細情報 T_RFLG の構造 】](#)」を参照してください。

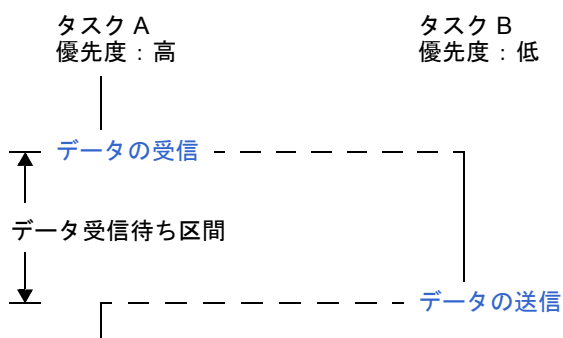
備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.4 データ・キュー

マルチタスク処理では、あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（データの受け渡し機能）が必要となります。そこで、RI600PXでは、規定されたデータ・サイズの通信機能として“データ・キュー”を提供しています。

以下に、データ・キューを利用した場合の処理の流れを示します。

図7-3 処理の流れ（データ・キュー）



備考 1回のデータ送信／受信処理で送信／受信するデータのサイズは、4バイトです。

7.4.1 データ・キューの生成

データ・キューは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API「dataqueue[]」を使用してデータ・キューを生成します。
静的 API「dataqueue[]」の詳細は、「[20.13 データ・キュー情報 \(dataqueue\[\]\)](#)」を参照してください。
- 2) `cre_dtq` または `acre_dtq` サービスコールによる生成
`cre_dtq` は、パラメータ `pk_cdtq` が指す領域に設定されたデータ・キュー生成情報にしたがって、パラメータ `dtqid` で指定されたデータ・キュー ID のデータ・キューを生成します。
`acre_dtq` は、パラメータ `pk_cdtq` が指す領域に設定されたデータ・キュー生成情報にしたがってデータ・キューを生成し、生成されたデータ・キュー ID を返します。
指定するデータ・キュー生成情報は、以下の通りです。
 - データ・キュー属性 (`dtqatr`)
以下を指定します。
 - タスク送信待ちキューの順序 (FIFO 順またはタスクの現在優先度順)
 - データ・キューの容量 (データの個数) (`dtqcnt`)、データ・キュー領域の先頭アドレス (`dtq`)
`dtq` から `TSZ_DTQ(dtqcnt)` バイトをデータ・キュー領域とします。TSZ_DTQ マクロについては「[18.3.2 データ・キュー関連](#)」を参照してください。
データ・キュー領域は、メモリ・オブジェクト以外でかつユーザ・スタック以外の領域に作成してください。

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_dtq` の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */

#define DTQCNT 10                      /* データ・キューの容量 (データの個数) */

#pragma section B BRI_RAM              /* データ・キュー領域にのセクション */
static UW dtq_area[ TSZ_DTQ(NUM_DATA)/sizeof(UW) ]; /* データ・キュー領域 */
#pragma section

#pragma task Task1                     /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      dtqid;                      /* 変数の宣言 */
    T_CDTQ  pk_cdtq = {                 /* 変数の宣言, 初期化 */
        TA_TFIFO,                      /* データ・キュー属性 (dtqatr) */
        DTQCNT,                        /* データ・キューの容量 (データの個数) (dtqcnt) */
        (VP) dtq_area                  /* データ・キュー領域の先頭アドレス (dtq) */
    };
    .....
    .....
    dtqid = acre_dtq ( &pk_cdtq );     /* データ・キュー生成 */
    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.4.2 データ・キューの削除

- del_dtq

パラメータ *dtqid* で指定されたデータ・キューを削除します。

対象データ・キューで *snd_dtq*, *tsnd_dtq*, *rcv_dtq*, または *trcv_dtq* によって待っているタスクがある場合は、そのタスク待ち状態を解除し、*snd_dtq*, *tsnd_dtq*, *rcv_dtq*, または *trcv_dtq* の戻り値として E_DLT を返します。

また、データ・キューに蓄えられていたデータは破棄されます。

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      dtqid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_dtq( dtqid );                  /* データ・キューの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

7.4.3 データの送信

データの送信は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `snd_dtq` (待つ)
- `psnd_dtq`, `ipsnd_dtq` (ポーリング)
- `tsnd_dtq` (タイムアウト付きで待つ)
- `snd_dtq` (待つ)
 - パラメータ `dtqid` で指定されたデータ・キューの状況に応じて、以下の処理を行います。
 - 受信待ちキューにタスクがキューイングされている場合
パラメータ `data` で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態 (データ受信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
 - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がある場合
パラメータ `data` で指定されたデータをデータ・キューに格納します。
 - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
自タスクを対象データ・キューの送信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (データ送信待ち状態) へと遷移させます。
なお、データ送信待ち状態の解除は、以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
<code>rcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>prcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>iprcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>trcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_dtq</code> の発行により、対象データ・キューがリセットされた。	EV_RST
<code>del_dtq</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。


```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考3 参照 */
void Task1 ( VP_INT exinf );      /* 備考3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      dtqid = 1;            /* 変数の宣言, 初期化 */
    VP_INT  data = 123;          /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = snd_dtq ( dtqid, data ); /* データの送信 */

    if ( ercd == E_OK ) {
        .....                  /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                  /* 強制終了処理 */
        .....
    }

    .....
    .....
}

```

- 備考1 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は、データの送信要求を行った順に行われます。
- 備考2 自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は、**データ・キューの生成時に指定された順**（FIFO 順または現在優先度順）に行われます。
- 備考3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `psnd_dtq`, `ipsnd_dtq` (ポーリング)

パラメータ `dtqid` で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
パラメータ `data` で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態(データ受信待ち状態)から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がある場合
パラメータ `data` で指定されたデータをデータ・キューに格納します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
戻り値として `E_TMOU` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 2 参照 */
void Task1 ( VP_INT exinf );      /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      dtqid = 1;            /* 変数の宣言, 初期化 */
    VP_INT  data = 123;          /* 変数の宣言, 初期化 */

    .....

    ercd = psnd_dtq ( dtqid, data ); /* データの送信 */

    if ( ercd == E_OK ) {
        .....                    /* ポーリング成功処理 */
    } else if ( ercd == E_TMOU ) {
        .....                    /* ポーリング失敗処理 */
    }

    .....
    .....
}
```

備考 1 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は、データの送信要求を行った順に行われます。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `tsnd_dtq` (タイムアウト付きで待つ)

パラメータ `dtqid` で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
パラメータ `data` で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態 (データ受信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がある場合
パラメータ `data` で指定されたデータをデータ・キューに格納します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
自タスクを対象データ・キューの送信待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態 (データ送信待ち状態) へと遷移させます。
なお、データ送信待ち状態の解除は、以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
<code>rcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>prcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>iprcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>trcv_dtq</code> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_dtq</code> の発行により、対象データ・キューがリセットされた。	EV_RST
<code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_dtq</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 4 参照 */
void Task1 ( VP_INT exinf );      /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      dtqid = 1;           /* 変数の宣言, 初期化 */
    VP_INT  data = 123;         /* 変数の宣言, 初期化 */
    TMO      tmout = 3600;      /* 変数の宣言, 初期化 */

    .....
    .....

                                /* データの送信 */
    ercd = tsnd_dtq ( dtqid, data, tmout );

    if ( ercd == E_OK ) {
        .....                 /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                 /* 強制終了処理 */
        .....
    } else if ( ercd == E_TMOUT ) {
        .....                 /* タイムアウト処理 */
        .....
    }
    .....
    .....
}

```

- 備考 1 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は、データの送信要求を行った順に行われます。
- 備考 2 自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は、データ・キュー生成時に指定された順（FIFO 順または現在優先度順）に行われます。
- 備考 3 待ち時間 *tmout* に **TMO_FEVR** が指定された際には“**snd_dtq** と同等の処理”を、**TMO_POL** が指定された際には“**psnd_dtq** と同等の処理”を実行します。
- 備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは **cfg600px** が **kernel_id.h** に出力するため、記述不要です。

7.4.4 データの強制送信

データの強制送信は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `fsnd_dtq`, `ifsnd_dtq`

パラメータ `dtqid` で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
パラメータ `data` で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態 (データ受信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされていない場合
パラメータ `data` で指定されたデータをデータ・キューに格納します。
データ・キューに空きがない場合は、その前に最古のデータを削除します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 2 参照 */
void Task1 ( VP_INT exinf );      /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      dtqid = 1;           /* 変数の宣言, 初期化 */
    VP_INT  data = 123;         /* 変数の宣言, 初期化 */

    .....
    .....

    fsnd_dtq ( dtqid, data );    /* データの強制送信 */

    .....
    .....
}
```

備考 1 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は、データの送信要求を行った順に行われます。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.4.5 データの受信

データの受信は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rcv_dtq` (待つ)
- `prcv_dtq`, `iprcv_dtq` (ポーリング)
- `trcv_dtq` (タイムアウト付きで待つ)
- `rcv_dtq` (待つ)
 - パラメータ `dtqid` で指定されたデータ・キューの状況に応じて、以下の処理を行います。
 - データ・キューにデータが格納されている場合
データ・キューから最古のデータを取り出して `p_data` で指定された領域に格納します。
送信待ちキューにタスクがキューイングされている場合は、送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち、WAITING 状態 (データ送信待ち状態) から READY 状態へと遷移させます。
 - データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したデータを `p_data` で指定された領域に格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (データ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、データ・キューの容量が 0 の場合のみ生じます。
 - データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
自タスクを対象データ・キューの受信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (データ受信待ち状態) へと遷移させます。
なお、データ受信待ち状態の解除は、以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
<code>snd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>psnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>ipsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>tsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>fsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>ifsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>del_dtq</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考2 参照 */
void Task1 ( VP_INT exinf );      /* 備考2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      dtqid = 1;            /* 変数の宣言, 初期化 */
    VP_INT  p_data;               /* 変数の宣言 */

    .....
    .....

                                /* データの受信 */
    ercd = rcv_dtq ( dtqid, &p_data );

    if ( ercd == E_OK ) {
        .....                    /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                    /* 強制終了処理 */
        .....
    }

    .....
    .....
}

```

備考1 自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は、データの受信要求を行った順に行われます。

備考2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

- `prcv_dtq`, `iprcv_dtq` (ポーリング)

パラメータ `dtqid` で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- データ・キューにデータが格納されている場合
データ・キューから最古のデータを取り出して `p_data` で指定された領域に格納します。
送信待ちキューにタスクがキューイングされている場合は、送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち、WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したデータを `p_data` で指定された領域に格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（データ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、データ・キューの容量が 0 の場合のみ生じます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
戻り値として `E_TMOUT` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                        /* 変数の宣言 */
    ID      dtqid = 1;                   /* 変数の宣言, 初期化 */
    VP_INT  p_data;                      /* 変数の宣言 */

    .....
    .....

                                /* データの受信 */
    ercd = prcv_dtq ( dtqid, &p_data );

    if ( ercd == E_OK ) {
        .....                    /* ポーリング成功処理 */
        .....
    } else if ( ercd == E_TMOUT ) {
        .....                    /* ポーリング失敗処理 */
        .....
    }

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `trcv_dtq` (タイムアウト付きで待つ)

パラメータ `dtqid` で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- データ・キューにデータが格納されている場合
データ・キューから最古のデータを取り出して `p_data` で指定された領域に格納します。
送信待ちキューにタスクがキューイングされている場合は、送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち、WAITING 状態 (データ送信待ち状態) から READY 状態へと遷移させます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したデータを `p_data` で指定された領域に格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (データ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、データ・キューの容量が 0 の場合のみ生じます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
自タスクを対象データ・キューの受信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (データ受信待ち状態) へと遷移させます。
なお、データ受信待ち状態の解除は、以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
<code>snd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>psnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>ipsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>tsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>fsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>ifsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_dtq</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考3 参照 */
void Task1 ( VP_INT exinf );      /* 備考3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      dtqid = 1;           /* 変数の宣言, 初期化 */
    VP_INT  p_data;              /* 変数の宣言 */
    TMO      tmout = 3600;       /* 変数の宣言, 初期化 */

    .....
    .....

                                /* データの受信 */
    ercd = trcv_dtq ( dtqid, &p_data, tmout );

    if ( ercd == E_OK ) {
        .....                 /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                 /* 強制終了処理 */
        .....
    } else if ( ercd == E_TMOUT ) {
        .....                 /* タイムアウト処理 */
        .....
    }

    .....
    .....
}

```

- 備考1 自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は、データの受信要求を行った順に行われます。
- 備考2 待ち時間 *tmout* に **TMO_FEVR** が指定された際には“**rcv_dtq** と同等の処理”を、**TMO_POL** が指定された際には“**prcv_dtq** と同等の処理”を実行します。
- 備考3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは **cfg600px** が **kernel_id.h** に出力するため、記述不要です。

7.4.6 データ・キュー詳細情報の参照

データ・キュー詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_dtq`, `iref_dtq`

パラメータ `dtqid` で指定されたデータ・キューのデータ・キュー詳細情報（待ちタスクの有無，未受信データの総数）をパラメータ `pk_rdtq` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考2参照 */
void Task1 ( VP_INT exinf );          /* 備考2参照 */
void Task1 ( VP_INT exinf )
{
    ID      dtqid = 1;                /* 変数の宣言，初期化 */
    T_RDTQ  pk_rdtq;                /* データ構造体の宣言 */
    ID      stskid;                  /* 変数の宣言 */
    ID      rtskid;                  /* 変数の宣言 */
    UINT    sdtqcnt;                /* 変数の宣言 */

    .....
    .....

    ref_dtq ( dtqid, &pk_rdtq );    /* データ・キュー詳細情報の参照 */

    stskid = pk_rdtq.stskid;        /* データ送信待ちタスクの有無の獲得 */
    rtskid = pk_rdtq.rtskid;        /* データ受信待ちタスクの有無の獲得 */
    sdtqcnt = pk_rdtq.sdtqcnt;      /* 未受信データの総数の獲得 */

    .....
    .....
}
```

備考1 データ・キュー詳細情報 T_RDTQ についての詳細は、「【[データ・キュー詳細情報 T_RDTQ の構造](#)】」を参照してください。

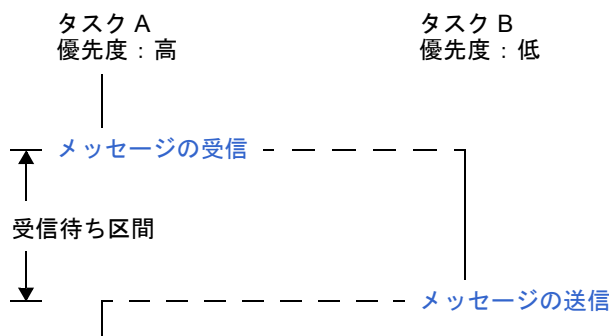
備考2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.5 メールボックス

マルチタスク処理では、あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（メッセージの受け渡し機能）が必要となります。そこで、RI600PX では、共有されているメモリ領域に書き込まれたメッセージの先頭アドレス受け渡し機能として“メールボックス”を提供しています。

以下に、メールボックスを利用した場合の処理の流れを示します。

図7-4 処理の流れ（メールボックス）



7.5.1 メッセージ

RI600PX では、処理プログラム間でメールボックスを介してやり取りされる情報を“メッセージ”と呼んでいます。

なお、メッセージは、メールボックスを介することにより任意の処理プログラムに対して送信することができますが、RI600PX における同期通信機能（メールボックス）では、メッセージの先頭アドレスを受信側処理プログラムに渡すだけであり、メッセージの内容が他領域にコピーされるわけではないので注意が必要です。

- メッセージの領域

メッセージは、送受信双方のタスクからアクセス可能なメモリ・オブジェクト内に作成する必要があります。

ただし、メッセージ先頭には、カーネルの管理テーブルがあります。カーネル管理テーブルが破壊されると、システムの正常な動作は保証されません。この理由から、メッセージ通信には、データ・キューまたはメッセージ・バッファの使用を推奨します。

- メッセージの基本型

RI600PX では、メッセージの内容、長さについては、利用するメールボックスの属性により、以下の規定を設けています。

- TA_MFIFO 属性のメールボックスを利用する場合

メッセージの先頭には T_MSG 構造体が必要です。この領域は、RI600PX が使用します。ユーザのメッセージは、T_MSG 構造体以降に設定します。

ユーザのメッセージの長さについては、メールボックスを利用して情報のやり取りを行う処理プログラム間で規定することになります。

以下に、TA_MFIFO 属性用メッセージを記述する場合の基本型を示します。

【TA_MFIFO 属性用メッセージの構造】

```
/* kernel.h で定義されている T_MSG 構造体 */
typedef struct {
    VP    *msghead;        /*RI600PX 管理領域 */
} T_MSG;

/* ユーザが定義するメッセージ構造体 */
typedef struct user_msg {
    T_MSG    t_msg;        /* T_MSG 構造体 */
    B        data[8];      /* ユーザのメッセージ */
} USER_MSG;
```

- TA_MPRI 属性のメールボックスを利用する場合

メッセージの先頭には T_MSG_PRI 構造体が必要です。T_MSG_PRI.msgque の領域は、RI600PX が使用します。T_MSG_PRI.msgpri には、メッセージの優先度を設定してください。ユーザのメッセージは、T_MSG_PRI 構造体以降に設定します。

ユーザのメッセージの長さについては、メールボックスを利用して情報のやり取りを行う処理プログラム間で規定することになります。

以下に、TA_MPRI 属性用メッセージを記述する場合の基本型を示します。

【TA_MPRI 属性用メッセージの構造】

```
/* kernel.h で定義されている T_MSG 構造体 */
typedef struct {
    VP    *msghead;        /*RI600PX 管理領域 */
} T_MSG;

/* kernel.h で定義されている T_MSG_PRI 構造体 */
typedef struct {
    T_MSG    msgque;        /* メッセージヘッダ */
    PRI    msgpri;        /* メッセージ優先度 */
} T_MSG_PRI;

/* ユーザが定義するメッセージ構造体 */
typedef struct user_msg {
    T_MSG_PRI    t_msg;    /* T_MSG_PRI 構造体 */
    B        data[8];      /* ユーザのメッセージ */
} USER_MSG;
```

備考1 RI600PX におけるメッセージの優先度は、その値が小さいほど、高い優先度であることを意味します。

備考2 メッセージの優先度として指定可能な値は、システム・コンフィギュレーション・ファイル作成時にメールボックス情報 (mailbox[]) の最大メッセージ優先度 (max_pri) で定義された値域に限定されます。

7.5.2 メールボックスの生成

メールボックスは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API“mailbox[]” を使用してメールボックスを生成します。
静的 API“mailbox[]” の詳細は、「[20.14 メールボックス情報 \(mailbox\[\]\)](#)」を参照してください。
- 2) `cre_mbx`, または `acre_mbx` サービスコールによる生成
`cre_mbx` は、パラメータ `pk_cmbx` が指す領域に設定されたメールボックス生成情報にしたがって、パラメータ `mbxid` で指定されたメールボックス ID のタスクを生成します。
`acre_mbx` は、パラメータ `pk_cmbx` が指す領域に設定されたメールボックス生成情報にしたがってメールボックスを生成し、生成されたメールボックス ID を返します。
指定するメールボックス生成情報は、以下の通りです。
 - メールボックス属性 (`mbxatr`)
以下を指定します。
 - タスク待ちキューの順序 (FIFO 順またはタスクの現在優先度順)
 - メッセージ・キューの順序 (FIFO 順またはメッセージ優先度順)
 - メッセージ優先度の最大値 (`maxmpri`)

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_mbx` の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mbxid;                       /* 変数の宣言 */
    T_CMBX  pk_cmbx = {                  /* 変数の宣言, 初期化 */
        TA_TFIFO|TA_MFIFO, /* メールボックス属性 (mbxatr) */
        1, /* メッセージ優先度の最大値 (maxmpri) */
        (VP)0 /* 予約 (mprihd) */
    };
    .....
    .....
    mbxid = acre_mbx ( &pk_cmbx ); /* メールボックスの生成 */
    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.5.3 メールボックスの削除

- del_mbx

パラメータ *mbxid* で指定されたメールボックスを削除します。

対象メールボックスで *rcv_mbx*, または *trcv_mbx* によって待っているタスクがある場合は, そのタスク待ち状態を解除し, *rcv_mbx*, または *trcv_mbx* の戻り値として E_DLT を返します。

本サービス・コールは, **信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に, 本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                     /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mbxid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_mbx( mbxid );                  /* メールボックスの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては, これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため, 記述不要です。

7.5.4 メッセージの送信

メッセージの送信は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `snd_mbx`, `isnd_mbx`

パラメータ `mbxid` で指定されたメールボックスにパラメータ `pk_msg` で指定されたメッセージを送信します。ただし、本サービス・コールを発行した際、対象メールボックスの待ちキューにタスクがキューイングされていた場合には、メッセージの送信（メッセージのキューイング処理）は行わず、該当タスクにメッセージを渡します。これにより、該当タスクは、待ちキューから外れ、WAITING 状態（メッセージ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 3 参照 */
void Task1 ( VP_INT exinf );          /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ID      mbxid = 1;                /* 変数の宣言, 初期化 */
    T_MSG_PRI      *pk_msg;          /* データ構造体の宣言 */

    .....

    .....

    .....                          /* メモリ領域（メッセージ用）を確保し, /
    pk_msg = ...                      /* そのポインタを pk_msg に設定 */
    .....

    .....                          /* 本体（内容）の作成 */
    .....

    pk_msg->msgpri = 8;                /* メッセージ優先度を設定 */

    .....                          /* メッセージの送信 */
    snd_mbx ( mbxid, ( T_MSG * ) pk_msg );

    .....

    .....
}
```

備考 1 メッセージを対象メールボックスのメッセージ・キューにキューイングする際のキューイング方式は、[メールボックスの生成時に指定した順](#)（FIFO 順またはメッセージ優先度順）に行われます。

備考 2 メッセージ `T_MSG`, `T_MSG_PRI` についての詳細は、「[7.5.1 メッセージ](#)」を参照してください。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.5.5 メッセージの受信

メッセージの受信は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rcv_mbx` (待つ)
- `prcv_mbx`, `iprcv_mbx` (ポーリング)
- `trcv_mbx` タイムアウト付きで待つ)
- `rcv_mbx` (待つ)

パラメータ `mbxid` で指定されたメールボックスからメッセージを受信し、その先頭アドレスをパラメータ `ppk_msg` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には、メッセージの受信は行わず、自タスクを対象メールボックスの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
<code>snd_mbx</code> の発行により、対象メールボックスにメッセージが送信された。	E_OK
<code>isnd_mbx</code> の発行により、対象メールボックスにメッセージが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>del_mbx</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 3 参照 */
void Task1 ( VP_INT exinf );            /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                        /* 変数の宣言 */
    ID      mbxid = 1;                   /* 変数の宣言, 初期化 */
    T_MSG   *pk_msg;                     /* データ構造体の宣言 */
    .....

    ercd = rcv_mbx ( mbxid, &pk_msg );   /* メッセージの受信 */

    if ( ercd == E_OK ) {
        .....                          /* 正常終了処理 */
    } else if ( ercd == E_RLWAI ) {
        .....                          /* 強制終了処理 */
    }
    .....
}
```

備考 1 自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は、**メールボックスの生成時に**指定した順（FIFO 順または現在優先度順）に行われます。

備考 2 メッセージ T_MSG, T_MSG_PRI についての詳細は、「7.5.1 メッセージ」を参照してください。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `prcv_mbx`, `iprcv_mbx` (ポーリング)

パラメータ `mbxid` で指定されたメールボックスからメッセージを受信し、その先頭アドレスをパラメータ `pk_msg` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には、メッセージの受信は行わず、戻り値として `E_TMOUT` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );            /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      mbxid = 1;                   /* 変数の宣言, 初期化 */
    T_MSG   *pk_msg;                     /* データ構造体の宣言 */
    .....

    ercd = prcv_mbx ( mbxid, &pk_msg ); /* メッセージの受信 */

    if ( ercd == E_OK ) {
        .....                          /* ポーリング成功処理 */
    } else if ( ercd == E_TMOUT ) {
        .....                          /* ポーリング失敗処理 */
    }
    .....
}
```

備考 1 メッセージ `T_MSG`, `T_MSG_PRI` についての詳細は、「[7.5.1 メッセージ](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `trcv_mbx` タイムアウト付きで待つ)

パラメータ `mbxid` で指定されたメールボックスからメッセージを受信し、その先頭アドレスをパラメータ `ppk_msg` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には、メッセージの受信は行わず、自タスクを対象メールボックスの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
<code>snd_mbx</code> の発行により、対象メールボックスにメッセージが送信された。	E_OK
<code>isnd_mbx</code> の発行により、対象メールボックスにメッセージが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_mbx</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 4 参照 */
void Task1 ( VP_INT exinf );            /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                        /* 変数の宣言 */
    ID      mbxid = 1;                   /* 変数の宣言, 初期化 */
    T_MSG   *pk_msg;                     /* データ構造体の宣言 */
    TMO     tmout = 3600;                 /* 変数の宣言, 初期化 */
    .....

    ercd = trcv_mbx ( mbxid, &pk_msg, tmout ); /* メッセージの受信 */

    if ( ercd == E_OK ) {
        .....                          /* 正常終了処理 */
    } else if ( ercd == E_RLWAI ) {
        .....                          /* 強制終了処理 */
    } else if ( ercd == E_TMOUT ) {
        .....                          /* タイムアウト処理 */
    }
    .....
}
```

備考 1 自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式はメールボックスの生成時に指定した順（FIFO 順または現在優先度順）に行われます。

備考 2 待ち時間 `tmout` に `TMO_FEVR` が指定された際には“`rcv_mbx` と同等の処理”を、`TMO_POL` が指定された際には“`prcv_mbx` と同等の処理”を実行します。

備考 3 メッセージ `T_MSG`, `T_MSG_PRI` についての詳細は、「7.5.1 メッセージ」を参照してください。

備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

7.5.6 メールボックス詳細情報の参照

メールボックス詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_mbx`, `iref_mbx`

パラメータ `mbxid` で指定されたメールボックスのメールボックス詳細情報（待ちタスクの有無、待ちメッセージの有無）をパラメータ `pk_rmbx` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      mbxid = 1;                /* 変数の宣言, 初期化 */
    T_RMBX  pk_rmbx;                 /* データ構造体の宣言 */
    ID      wtskid;                  /* 変数の宣言 */
    T_MSG   *pk_msg;                /* データ構造体の宣言 */

    .....

    ref_mbx ( mbxid, &pk_rmbx );     /* メールボックス詳細情報の参照 */

    wtskid = pk_rmbx.wtskid;         /* 待ちタスクの有無の獲得 */
    pk_msg = pk_rmbx.pk_msg;        /* 待ちメッセージの有無の獲得 */

    .....
}
```

備考 1 メールボックス詳細情報 T_RMBX についての詳細は、「[【メールボックス詳細情報 T_RMBX の構造】](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

第8章 拡張同期通信機能

本章では、RI600PX が提供している拡張同期通信機能について解説しています。

8.1 概要

RI600PX における拡張同期通信機能では、タスク間の排他制御を実現する手段として**ミューテックス**、任意のサイズのメッセージをコピーして受け渡す手段として**メッセージ・バッファ**を提供しています。

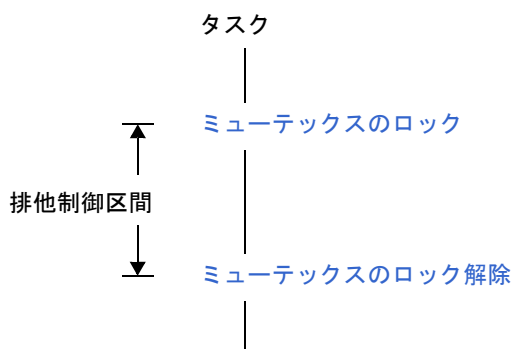
8.2 ミューテックス

マルチタスク処理では、並行に動作するタスクが限られた数の資源（A/D コンバータ、コプロセッサ、ファイルなど）を同時に使用するといった資源使用の競合を防ぐ機能（排他制御機能）が必要となります。そこで、RI600PX では、このような資源使用の競合を防ぐ機能として“ミューテックス”を提供しています。

以下に、ミューテックスを利用した場合の処理の流れを示します。

RI600PX のミューテックスは優先度上限プロトコルをサポートしています。

図 8-1 処理の流れ（ミューテックス）



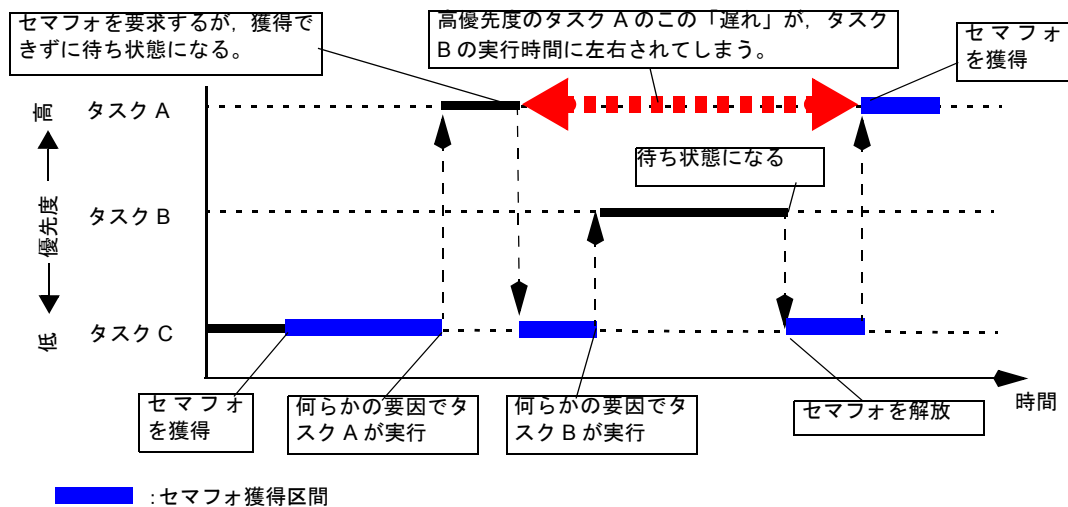
8.2.1 優先度逆転問題

セマフォを用いた排他制御では、優先度逆転という問題が発生するケースがあります。優先度逆転とは、資源を要求するタスクの実行が、資源を使用しない別のタスクによって遅延されてしまうという現象です。

この様子を図8-2に示します。この図では、タスクAとタスクCは同じ資源を使用し、タスクBはその資源を使用しない例になっています。タスクAは資源を使用するためにセマフォを獲得しようとしませんが、すでにタスクCがセマフォを獲得しているため待ち状態になります。ところが、タスクCがセマフォを解放する前に、優先度がタスクCよりも高くタスクAより低く、かつ資源とは関係のないタスクBが実行すると、タスクCによるセマフォの解放はタスクBの実行によって遅れ、その結果タスクAがセマフォを獲得するのも遅れます。タスクAの立場では、資源競合しておらず、かつ自分より優先度の低いタスクBが優先的に実行されてしまうこととなります。

ミューテックスは、この問題を回避するために導入された機能です。

図8-2 優先度逆転問題



8.2.2 現在優先度とベース優先度

タスクの優先度には、ベース優先度と現在優先度があります。タスクのスケジューリングは、現在優先度に従って行われます。

ミューテックスをロックしていない時は、両者は常に同じです。

ミューテックスをロックすると、現在優先度のみがそのミューテックスの上限優先度に引き上げられます。

タスクの優先度を変更する `chg_pri`, `ichg_pri` では、ミューテックスをロックしていないタスクの場合は、ベース優先度・現在優先度とも変更されますが、ミューテックスをロックしているタスクの場合はベース優先度のみが変更されます。また、ミューテックスロック中、またはミューテックスのロックを待っているタスクの場合は、ロック中、またはロックを待っているミューテックスのいずれかの上限優先度よりも高い優先度を指定すると、戻り値として `E_ILUSE` を返します。

なお、`get_pri`, `iget_pri` を用いると、現在優先度を参照することができます。また、`ref_tsk`, `iref_tsk` を用いると現在優先度とベース優先度を参照することができます。

8.2.3 簡略化した優先度上限プロトコル

優先度上限プロトコルの本来の振る舞いは、タスクの現在優先度を、そのタスクがロックしているミューテックスの中で最高の上限優先度に制御することです。これは、ミューテックスのロック・アンロック時に、タスクの現在優先度を以下のように制御することで実現されます。

- ミューテックスのロック時に、タスクの現在優先度をそのタスクがロックしているミューテックスの中で最高の上限優先度に変更する。
- ミューテックスのアンロック時に、タスクの現在優先度をそのタスクが以降もロックを継続するミューテックスの中で最高の上限優先度に変更する。ロックしているミューテックスがなくなる場合は、現在優先度をベース優先度に戻す。

しかし、RI600PX ではオーバーヘッドの低減を目的に「簡略化した優先度上限プロトコル」を採用しているため、上記の下線部の制御は行われません。

8.2.4 セマフォとの相違点

ミューテックスは最大資源数が 1 つのセマフォ（バイナリ・セマフォ）と似た動作をしますが、以下のような違いがあります。

- ミューテックスをロックしたタスクの現在優先度は、ミューテックスのロックを解除するまで、ミューテックスの上限優先度に引き上げられます。これによって、優先度逆転問題が回避されます。
 - セマフォでは優先度は変更されません。
- ミューテックスのロック解除（資源返却に相当）できるのはミューテックスをロックしたタスクのみです。
 - セマフォはどのタスク／ハンドラからでも資源の返却が可能です。
- ミューテックスをロックしているタスクを終了する (`ext_tsk`, `ter_tsk`) 際に、自動的にロック解除処理が行われます。
 - セマフォは自動的に資源の返却を行わないので、資源を獲得したまま終了します。
- セマフォは複数の資源を管理できる（最大資源数を指定できる）がミューテックスの資源最大数に相当する値は 1 固定です。

8.2.5 ミューテックスの生成

ミューテックスは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API "mutex[]" を使用してミューテックスを生成します。
静的 API "mutex[]" の詳細は、「20.15 ミューテックス情報 (mutex[])」を参照してください。
- 2) `cre_mtx` または `acre_mtx` サービスコールによる生成
`cre_mtx` は、パラメータ `pk_cmtx` が指す領域に設定されたミューテックス生成情報にしたがって、パラメータ `mtxid` で指定されたミューテックス ID のタスクを生成します。
`acre_mtx` は、パラメータ `pk_cmtx` が指す領域に設定されたミューテックス生成情報にしたがってミューテックスを生成し、生成されたミューテックス ID を返します。
指定するミューテックス生成情報は、以下の通りです。
 - ミューテックス属性 (`mtxatr`)
「優先度上限プロトコル (TA_CEILING)」のみを指定できます。
なお、タスク待ちキューは、タスクの現在優先度順で管理されます。
 - 上限優先度 (`ceilpri`)

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_mtx` の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考参照 */
void Task1 ( VP_INT exinf );          /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mtxid;                    /* 変数の宣言 */
    T_CMTX  pk_cmtx = {               /* 変数の宣言, 初期化 */
        TA_CEILING,                  /* ミューテックス属性 (mtxatr) */
        1,                           /* 上限優先度 (ceilpri) */
    };
    .....
    .....
    mtxid = acre_mtx ( &pk_cmtx );    /* ミューテックスの生成 */
    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

8.2.6 ミューテックスの削除

- del_mtx

パラメータ *mtxid* で指定されたミューテックスを削除します。

対象ミューテックスをロックしているタスクがある場合、そのロックを解除します。その結果、そのタスクがロックしているミューテックスがなくなった場合には、そのタスクの現在優先度をベース優先度に戻します。対象ミューテックスをロックしていたタスクには、ミューテックスが削除されたことは通知されません。後で *unl_mtx* によってミューテックスをロック解除しようとした時点で、エラー E_NOEXS が返されます。

対象ミューテックスで *loc_mtx* または *tloc_mtx* によって待っているタスクがある場合は、そのタスク待ち状態を解除し、*loc_mtx* または *tloc_mtx* の戻り値として E_DLT を返します。

本サービス・コールは、**信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mtxid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_mtx( mtxid );                  /* ミューテックスの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

8.2.7 ミューテックスのロック

ミューテックスのロックは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `loc_mtx` (待つ)
- `ploc_mtx` (ポーリング)
- `tlloc_mtx` (タイムアウト付きで待つ)
- `loc_mtx` (待つ)

パラメータ `mtxid` で指定されたミューテックスをロックします。

ただし、本サービス・コールを発行した際、対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には、自タスクを対象ミューテックスの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（ミューテックス待ち状態）へと遷移させます。

なお、ミューテックス待ち状態の解除は、以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
<code>unl_mtx</code> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<code>ext_tsk</code> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<code>exd_tsk</code> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<code>ter_tsk</code> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>del_mtx</code> の発行により、待ち状態を強制的に解除された。	E_DLT

ミューテックスのロック時には、自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、自タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合には、自タスクの現在優先度は変更しません。

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考3 参照 */
void Task1 ( VP_INT exinf );      /* 備考3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      mtxid = 8;            /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = loc_mtx ( mtxid );     /* ミューテックスのロック */

    if ( ercd == E_OK ) {
        .....                   /* ロック状態 */
        .....

        unl_mtx ( mtxid );       /* ミューテックスのロック解除 */
    } else if ( ercd == E_RLWAI ) {
        .....                   /* 強制終了処理 */
        .....
    }

    .....
    .....
}

```

- 備考1 自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は、優先度順に行われます。ただし、同じ優先度のタスクの中ではFIFO順に行われます。
- 備考2 自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には、戻り値としてE_ILUSEを返します。
- 備考3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントはcfg600pxがkernel_id.hに出力するため、記述不要です。

- `ploc_mtx` (ポーリング)

パラメータ `mtxid` で指定されたミューテックスをロックします。

ただし、本サービス・コールを発行した際、対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には、戻り値として `E_TMOUT` を返します。

ミューテックスのロック時には、自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、自タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合には、自タスクの現在優先度は変更しません。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      mtxid = 8;           /* 変数の宣言, 初期化 */

    .....

    ercd = ploc_mtx ( mtxid );   /* ミューテックスのロック */

    if ( ercd == E_OK ) {
        .....                  /* ポーリング成功処理 */
        .....

        unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
    } else if ( ercd == E_TMOUT ) {
        .....                  /* ポーリング失敗処理 */
        .....

    }

    .....
    .....
}
```

備考 1 自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には、戻り値として `E_ILUSE` を返します。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `tloc_mtx` (タイムアウト付きで待つ)

パラメータ `mtxid` で指定されたミューテックスをロックします。

ただし、本サービス・コールを発行した際、対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には、自タスクを対象ミューテックスの待ちキューにキューイングしたのち、`RUNNING` 状態からタイムアウト付きの `WAITING` 状態（ミューテックス待ち状態）へと遷移させます。

なお、ミューテックス待ち状態の解除は、以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
<code>unl_mtx</code> の発行により、対象ミューテックスのロック状態が解除された。	<code>E_OK</code>
<code>ext_tsk</code> の発行により、対象ミューテックスのロック状態が解除された。	<code>E_OK</code>
<code>exd_tsk</code> の発行により、対象ミューテックスのロック状態が解除された。	<code>E_OK</code>
<code>ter_tsk</code> の発行により、対象ミューテックスのロック状態が解除された。	<code>E_OK</code>
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	<code>E_RLWAI</code>
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	<code>E_RLWAI</code>
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	<code>E_TMOUT</code>
<code>del_mtx</code> の発行により、待ち状態を強制的に解除された。	<code>E_DLT</code>

ミューテックスのロック時には、自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、自タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合には、自タスクの現在優先度は変更しません。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 4 参照 */
void Task1 ( VP_INT exinf );          /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                      /* 変数の宣言 */
    ID      mtxid = 8;                 /* 変数の宣言, 初期化 */
    TMO      tmout = 3600;             /* 変数の宣言, 初期化 */
    .....
    ercd = tloc_mtx ( mtxid, tmout ); /* ミューテックスのロック */
    if ( ercd == E_OK ) {
        .....                        /* ロック状態 */
        unl_mtx ( mtxid );            /* ミューテックスのロック解除 */
    } else if ( ercd == E_RLWAI ) {
        .....                        /* 強制終了処理 */
    } else if ( ercd == E_TMOUT ) {
        .....                        /* タイムアウト処理 */
    }
}
```

備考 1 自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は、優先度順に行われます。ただし、同じ優先度のタスクの中では FIFO 順に行われます。

備考 2 自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には、戻り値として `E_ILUSE` を返します。

備考 3 待ち時間 `tmout` に `TMO_FEVR` が指定された際には“`loc_mtx` と同等の処理”を、`TMO_POL` が指定された際には“`ploc_mtx` と同等の処理”を実行します。

備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

8.2.8 ミューテックスのロック解除

ミューテックスのロック解除は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `unl_mtx`

パラメータ `mtxid` で指定されたミューテックスのロック状態を解除します。その結果、自タスクがロックしているミューテックスがなくなった場合には、自タスクの現在優先度をベース優先度に変更します。

本サービス・コールを発行した際、対象ミューテックスの待ちキューにタスクがキューイングされていた場合には、ミューテックスのロック解除処理後、ただちに該当タスク（待ちキューの先頭タスク）によるミューテックスのロック処理が行われます。このとき、該当タスクは、待ちキューから外れ、WAITING 状態（ミューテックス待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、該当タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、該当タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合には、該当タスクの現在優先度は変更しません。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 3 参照 */
void Task1 ( VP_INT exinf );           /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      mtxid = 8;                   /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = loc_mtx ( mtxid );           /* ミューテックスのロック */

    if ( ercd == E_OK ) {
        .....                           /* ロック状態 */
        .....

        unl_mtx ( mtxid );              /* ミューテックスのロック解除 */
    } else if ( ercd == E_RLWAI ) {
        .....                           /* 強制終了処理 */
        .....

    }

    .....
    .....
}
```

備考 1 ミューテックスのロック解除が可能なタスクは“対象ミューテックスをロックしたタスク”に限られます。このため、自タスクがロックしていないミューテックスに対して本サービス・コールを発行した場合には、何も処理は行わず、戻り値として E_ILUSE を返します。

備考 2 タスクの終了時、そのタスクがロックしていたミューテックスは解除されます。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

8.2.9 ミューテックス詳細情報の参照

ミューテックス詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mtx

パラメータ *mtxid* で指定されたミューテックスのミューテックス詳細情報（ロックの有無、待ちタスクの有無）をパラメータ *pk_rmtx* で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      mtxid = 1;                 /* 変数の宣言, 初期化 */
    T_RMTX  pk_rmtx;                  /* データ構造体の宣言 */
    ID      htskid;                   /* 変数の宣言 */
    ID      wtskid;                   /* 変数の宣言 */

    .....
    .....

    ref_mtx ( mtxid, &pk_rmtx );      /* ミューテックス詳細情報の参照 */

    htskid = pk_rmtx.htskid;          /* ロックの有無の獲得 */
    wtskid = pk_rmtx.wtskid;          /* 待ちタスクの有無の獲得 */

    .....
    .....
}
```

備考 1 ミューテックス詳細情報 T_RMTX についての詳細は、「[【ミューテックス詳細情報 T_RMTX の構造】](#)」を参照してください。

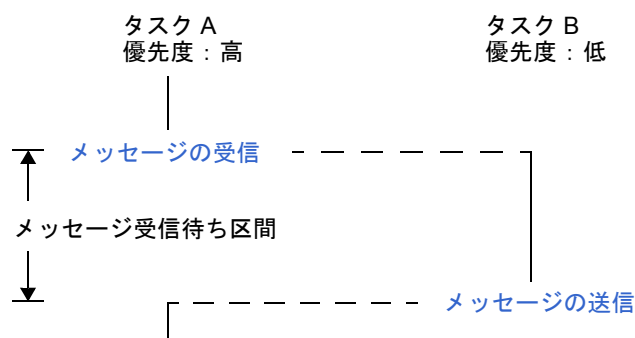
備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

8.3 メッセージ・バッファ

マルチタスク処理では、あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（メッセージの受け渡し機能）が必要となります。そこで、RI600PX では、任意のサイズのメッセージをコピーして受け渡す機能として“メッセージ・バッファ”を提供しています。

以下に、メッセージ・バッファを利用した場合の処理の流れを示します。

図 8 - 3 処理の流れ（メッセージ・バッファ）



8.3.1 メッセージ・バッファの生成

メッセージ・バッファは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API "message_buffer[]" を使用してメッセージ・バッファを生成します。
静的 API "message_buffer[]" の詳細は、「20.16 メッセージ・バッファ情報 (message_buffer[])」を参照してください。
- 2) `cre_mbf` または `acre_mbf` サービスコールによる生成
`cre_mbf` は、パラメータ `pk_cmbf` が指す領域に設定されたメッセージ・バッファ生成情報にしたがって、パラメータ `mbfid` で指定されたメッセージ・バッファ ID のメッセージ・バッファを生成します。
`acre_mbf` は、パラメータ `pk_cmbf` が指す領域に設定されたメッセージ・バッファ生成情報にしたがってメッセージ・バッファを生成し、生成されたメッセージ・バッファ ID を返します。
指定するメッセージ・バッファ生成情報は、以下の通りです。
 - メッセージ・バッファ属性 (`mbfattr`)
「送信タスク待ちキューは FIFO 順 (TA_TFIFO)」のみを指定できます。
なお、受信タスク待ちキューはも FIFO 順で管理されます。
 - メッセージの最大サイズ (バイト数) (`maxmsz`)
 - メッセージ・バッファ領域のサイズ (バイト数) (`mbfsz`)、メッセージ・バッファ領域の先頭アドレス (`mbf`)
メッセージ・バッファ領域は、メモリ・オブジェクト以外でかつユーザ・スタック以外の領域に作成してください。

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_mbf` の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */

#define MAX_MSGSZ 64                  /* メッセージの最大サイズ (バイト数) */
#define MBFSZ 256                     /* メッセージ・バッファ領域のサイズ (バイト数) */

#pragma section B BRI_RAM             /* メッセージ・バッファ領域のセクション */
static UW mbf_area[ MBFSZ/sizeof(UW) ]; /* メッセージ・バッファ領域 */
#pragma section

#pragma task Task1                    /* 備考参照 */
void Task1 ( VP_INT exinf );          /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mbfid;                    /* 変数の宣言 */
    T_CMBF  pk_cmbf = {               /* 変数の宣言, 初期化 */
        TA_TFIFO,                    /* メッセージ・バッファ属性 (mbfattr) */
        MAX_MSGSZ,                   /* メッセージの最大サイズ (バイト数) (maxmsz) */
        MBFSZ,                       /* メッセージ・バッファ領域のサイズ (バイト数) (mbfsz) */
        (VP)mbf_area                 /* メッセージ・バッファ領域の先頭アドレス (mbf) */
    };
    .....
    mbfid = acre_mbf ( &pk_cmbf );    /* メッセージ・バッファ生成 */
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

8.3.2 メッセージ・バッファの削除

- del_mbf

パラメータ *mbfid* で指定されたメッセージ・バッファを削除します。

対象メッセージ・バッファで *snd_mbf*, *tsnd_mbf*, *rcv_mbf*, または *trcv_mbf* によって待っているタスクがある場合は、そのタスク待ち状態を解除し、*snd_mbf*, *tsnd_mbf*, *rcv_mbf*, または *trcv_mbf* の戻り値として E_DLT を返します。

また、メッセージ・バッファに蓄えられていたメッセージは破棄されます。

本サービス・コールは、**信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mbfid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_mbf( mbfid );                  /* メッセージ・バッファの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

8.3.3 メッセージの送信

メッセージの送信は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `snd_mbf` (待つ)
 - `psnd_mbf`, `ipsnd_mbf` (ポーリング)
 - `tsnd_mbf` (タイムアウト付きで待つ)
- `snd_mbf` (待つ)
- パラメータ `mbfid` で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。
- 受信待ちキューにタスクがキューイングされている場合
`msg` で指定されたメッセージを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは受信待ちキューから外れ、WAITING 状態 (メッセージ受信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
 - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がある場合
`msg` で指定されたメッセージをメッセージ・バッファに格納します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

$$\text{減少サイズ} = \text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL}$$
 - 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (メッセージ送信待ち状態) へと遷移させます。
 なお、メッセージ送信待ち状態の解除は、以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
<code>rcv_mbf</code> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<code>prcv_mbf</code> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<code>trcv_mbf</code> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <code>rel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>ter_tsk</code> の発行により、待ち状態を強制的に解除された。 - <code>tsnd_mbf</code> のパラメータ <code>tmout</code> で指定された待ち時間が経過した。 	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_mbf</code> の発行により、対象メッセージ・バッファがリセットされた。	EV_RST
<code>del_mbf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考3参照 */
void Task1 ( VP_INT exinf );      /* 備考3参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      mbfid = 1;            /* 変数の宣言, 初期化 */
    B       msg[] = {1,2,3};      /* 変数の宣言, 初期化 */
    UINT    msgsz = sizeof( msg ); /* 変数の宣言, 初期化 */

    .....
    .....

                                /* メッセージの送信 */
    ercd = snd_mbf ( mbfid, (VP)msg, msgsz );

    if ( ercd == E_OK ) {
        .....                    /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                    /* 強制終了処理 */
        .....
    }

    .....
    .....
}

```

- 備考1 メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は、メッセージの送信要求を行った順に行われます。
- 備考2 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は、FIFO順に行われます。
- 備考3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントはcfg600px が kernel_id.h に出力するため、記述不要です。

- `psnd_mbf`, `ipsnd_mbf` (ポーリング)

パラメータ `mbfid` で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
`msg` で指定されたメッセージを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは受信待ちキューから外れ、WAITING 状態 (メッセージ受信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がある場合
`msg` で指定されたメッセージをメッセージ・バッファに格納します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

$$\text{減少サイズ} = \text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL}$$
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
 戻り値として `E_TMOUT` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );           /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      mbfid = 1;                  /* 変数の宣言, 初期化 */
    B       msg[] = {1,2,3};           /* 変数の宣言, 初期化 */
    UINT    msgsz = sizeof( msg );     /* 変数の宣言, 初期化 */

    .....
    .....

                                /* メッセージの送信 */
    ercd = psnd_mbf ( mbfid, (VP)msg, msgsz );

    if ( ercd == E_OK ) {
        .....                    /* ポーリング成功処理 */
        .....
    } else if ( ercd == E_TMOUT ) {
        .....                    /* ポーリング失敗処理 */
        .....
    }

    .....
    .....
}
```

備考 1 メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は、メッセージの送信要求を行った順に行われます。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- **tsnd_mbf** (タイムアウト付きで待つ)

パラメータ *mbfid* で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは受信待ちキューから外れ、WAITING 状態 (メッセージ受信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がある場合
msg で指定されたメッセージをメッセージ・バッファに格納します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
$$\text{減少サイズ} = \text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL}$$
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態 (メッセージ送信待ち状態) へと遷移させます。
なお、メッセージ送信待ち状態の解除は、以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
<i>rcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<i>prcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<i>trcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <i>rel_wai</i> の発行により、待ち状態を強制的に解除された。 - <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。 - <i>ter_tsk</i> の発行により、待ち状態を強制的に解除された。 - <i>tsnd_mbf</i> のパラメータ <i>tmout</i> で指定された待ち時間が経過した。 	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>vrst_mbf</i> の発行により、対象メッセージ・バッファがリセットされた。	EV_RST
パラメータ <i>tmout</i> で指定された待ち時間が経過した。	E_TMOUT
<i>del_mbf</i> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 4 参照 */
void Task1 ( VP_INT exinf );      /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      mbfid = 1;            /* 変数の宣言, 初期化 */
    B       msg[] = {1,2,3};     /* 変数の宣言, 初期化 */
    UINT    msgsz = sizeof( msg ); /* 変数の宣言, 初期化 */
    TMO     tmout = 3600;        /* 変数の宣言, 初期化 */

    .....
    .....

                                /* メッセージの送信 */
    ercd = tsnd_mbf ( mbfid, (VP)msg, msgsz, tmout );

    if ( ercd == E_OK ) {
        .....                 /* 正常終了処理 */
        .....
    } else if ( ercd == E_RLWAI ) {
        .....                 /* 強制終了処理 */
        .....
    } else if ( ercd == E_TMOUT ) {
        .....                 /* タイムアウト処理 */
        .....
    }

    .....
    .....
}

```

- 備考 1 メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は、メッセージの送信要求を行った順に行われます。
- 備考 2 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考 3 待ち時間 *tmout* に **TMO_FEVR** が指定された際には “**snd_mbf** と同等の処理 ” を、**TMO_POL** が指定された際には “**psnd_mbf** と同等の処理 ” を実行します。
- 備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは **cfg600px** が **kernel_id.h** に出力するため、記述不要です。

8.3.4 メッセージの受信

メッセージの受信は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rcv_mbf` (待つ)
- `prcv_mbf` (ポーリング)
- `trcv_mbf` (タイムアウト付きで待つ)

- `rcv_mbf` (待つ)

パラメータ `mbfid` で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
 メッセージ・バッファから最古のメッセージを取り出して `msg` で指定された領域に格納し、そのメッセージサイズを戻り値として返します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。

$$\text{増加サイズ} = \text{up4(戻り値)} + \text{VTSZ_MBFTBL}$$

さらに以下を、送信待ちキューにタスクがなくなるか、メッセージを格納できなくなるまで繰り返します。

- 送信待ちキューにタスクがキューイングされており、キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は、そのメッセージをメッセージ・バッファに格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (メッセージ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

$$\text{減少サイズ} = \text{up4(該当タスクが送信したメッセージ・サイズ)} + \text{VTSZ_MBFTBL}$$

- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされている場合
 送信待ちキュー先頭のタスクが指定したメッセージを `msg` で指定された領域に格納し、そのメッセージサイズを戻り値として返します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (メッセージ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

なお、この状況は、メッセージ・バッファのサイズが0の場合のみ生じます。

- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
 自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (メッセージ受信待ち状態) へと遷移させます。
 なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
<code>snd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>psnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>ipsnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>tsnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>del_mbf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。


```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考3参照 */
void Task1 ( VP_INT exinf );      /* 備考3参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      mbfid = 1;            /* 変数の宣言, 初期化 */
    B      msg[16];              /* 変数の宣言 (最大メッセージサイズ) */

    .....

    ercd = rcv_mbf ( mbfid, (VP)msg ); /* メッセージの受信 */

    if ( ercd == E_OK ) {
        ..... /* 正常終了処理 */
    } else if ( ercd == E_RLWAI ) {
        ..... /* 強制終了処理 */
    }
    .....
}

```

- 備考1 最大メッセージ・サイズは、メッセージ・バッファの生成時に指定します。msg の指す領域は、必ず最大メッセージ・サイズ以上としてください。
- 備考2 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

- `prcv_mbf` (ポーリング)

パラメータ `mbfid` で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
メッセージ・バッファから最古のメッセージを取り出して `msg` で指定された領域に格納し、そのメッセージサイズを戻り値として返します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。

$$\text{増加サイズ} = \text{up4(戻り値)} + \text{VTSZ_MBFTBL}$$

さらに以下を、送信待ちキューにタスクがなくなるか、メッセージを格納できなくなるまで繰り返します。

- 送信待ちキューにタスクがキューイングされており、キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は、そのメッセージをメッセージ・バッファに格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (メッセージ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

$$\text{減少サイズ} = \text{up4(該当タスクが送信したメッセージ・サイズ)} + \text{VTSZ_MBFTBL}$$

- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したメッセージを `msg` で指定された領域に格納し、そのメッセージサイズを戻り値として返します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (メッセージ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、メッセージ・バッファのサイズが 0 の場合のみ生じます。
- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
戻り値として `E_TMOUT` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );            /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                        /* 変数の宣言 */
    ID      mbfid = 1;                   /* 変数の宣言, 初期化 */
    B      msg[16];                      /* 変数の宣言 ( 最大メッセージサイズ ) */

    .....

    ercd = prcv_mbf ( mbfid, (VP)msg ); /* メッセージの受信 */

    if ( ercd == E_OK ) {
        .....                          /* ポーリング成功処理 */
    } else if ( ercd == E_TMOUT ) {
        .....                          /* ポーリング失敗処理 */
    }

    .....
}
```

備考 1 最大メッセージ・サイズは、メッセージ・バッファの生成時に指定します。`msg` の指す領域は、必ず最大メッセージ・サイズ以上としてください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `trcv_mbf` (タイムアウト付きで待つ)

パラメータ `mbfid` で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
メッセージ・バッファから最古のメッセージを取り出して `msg` で指定された領域に格納し、そのメッセージサイズを戻り値として返します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。

$$\text{増加サイズ} = \text{up4(戻り値)} + \text{VTSZ_MBFTBL}$$

さらに以下を、送信待ちキューにタスクがなくなるか、メッセージを格納できなくなるまで繰り返します。

- 送信待ちキューにタスクがキューイングされており、キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は、そのメッセージをメッセージ・バッファに格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (メッセージ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。

$$\text{減少サイズ} = \text{up4(該当タスクが送信したメッセージ・サイズ)} + \text{VTSZ_MBFTBL}$$

- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したメッセージを `msg` で指定された領域に格納し、そのメッセージサイズを戻り値として返します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態 (メッセージ送信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

なお、この状況は、メッセージ・バッファのサイズが0の場合のみ生じます。

- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされていない場合

自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態 (メッセージ受信待ち状態) へと遷移させます。

なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
<code>snd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>psnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>ipsnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>tsnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_mbf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 4 参照 */
void Task1 ( VP_INT exinf );      /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      mbfid = 1;            /* 変数の宣言, 初期化 */
    B       msg[16];              /* 変数の宣言 (最大メッセージサイズ) */
    TMO     tmout = 3600;         /* 変数の宣言, 初期化 */

    .....

                                /* メッセージの受信 (タイムアウト付き) */
    ercd = trcv_mbf ( mbfid, (VP)msg, tmout );

    if ( ercd == E_OK ) {
        .....                  /* 正常終了処理 */
    } else if ( ercd == E_RLWAI ) {
        .....                  /* 強制終了処理 */
    } else if ( ercd == E_TMOUT ) {
        .....                  /* タイムアウト処理 */
    }
    .....
}

```

- 備考 1 最大メッセージ・サイズは、メッセージ・バッファの生成時に指定します。msg の指す領域は、必ず最大メッセージ・サイズ以上としてください。
- 備考 2 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考 3 待ち時間 tmout に TMO_FEVR が指定された際には“rcv_mbf と同等の処理”を、TMO_POL が指定された際には“prcv_mbf と同等の処理”を実行します。
- 備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

8.3.5 メッセージ・バッファ詳細情報の参照

メッセージ・バッファ詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_mbf`, `iref_mbf`

パラメータ `mbfid` で指定されたメッセージ・バッファの詳細情報（待ちタスクの有無、空き領域のサイズなど）をパラメータ `pk_rmbf` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      mbfid = 1;                /* 変数の宣言, 初期化 */
    T_RMBF  pk_rmbf;                 /* データ構造体の宣言 */
    ID      stskid;                  /* 変数の宣言 */
    ID      rtskid;                  /* 変数の宣言 */
    UINT    msgcnt;                  /* 変数の宣言 */
    SIZE    fmbfsz;                  /* 変数の宣言 */

    .....
    .....

    ref_mbf ( mbfid, &pk_rmbf );     /* メッセージ・バッファ詳細情報の参照 */

    stskid = pk_rmbf.stskid;         /* メッセージ送信待ちタスクの有無の獲得 */
    rtskid = pk_rmbf.rtskid;         /* メッセージ受信待ちタスクの有無の獲得 */
    msgcnt = pk_rmbf.msgcnt;         /* 未受信メッセージ総数の獲得 */
    fmbfsz = pk_rmbf.fmbfsz;        /* 空き領域のサイズ */

    .....
    .....
}
```

備考 1 メッセージ・バッファ詳細情報 `T_RMBF` についての詳細は、「[【メッセージ・バッファ詳細情報 T_RMBF の構造】](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

第9章 メモリ・プール管理機能

本章では、RI600PX が提供しているメモリ・プール管理機能について解説しています。

9.1 概 要

RI600PXでは、動的なメモリ獲得機能として“固定長メモリ・プール”および“可変長メモリ・プール”を提供しています。固定長メモリ・プールは、扱えるメモリ・サイズは固定ですが、獲得／解放のオーバーヘッドが小さいという特長があります。

一方、可変長メモリ・プールは任意のメモリサイズを扱うことができますが、獲得／解放のオーバーヘッドは固定長メモリ・プールより劣ります。また、メモリ・プール領域の断片化の問題が発生します。

9.2 固定長メモリ・プール

RI600PX では、処理プログラムから動的なメモリ操作要求が行われた際に利用するメモリ領域として“固定長メモリ・プール”を提供しています。

なお、固定長メモリ・プールに対する動的なメモリ操作は、固定サイズの固定長メモリ・ブロックを単位として行われます。

9.2.1 固定長メモリ・プールの生成

固定長メモリ・プールは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API“`memorypool[]`”を使用して固定長メモリ・プールを生成します。
静的 API“`memorypool[]`”の詳細は、「[20.17 固定長メモリ・プール情報 \(memorypool\[\]\)](#)」を参照してください。
- 2) `cre_mpf` または `acre_mpf` サービスコールによる生成
`cre_mpf` は、パラメータ `pk_cmpf` が指す領域に設定された固定長メモリ・プール生成情報にしたがって、パラメータ `mpfid` で指定された固定長メモリ・プール ID の固定長メモリ・プールを生成します。
`acre_mpf` は、パラメータ `pk_cmpf` が指す領域に設定された固定長メモリ・プール生成情報にしたがって固定長メモリ・プールを生成し、生成された固定長メモリ・プール ID を返します。
指定する固定長メモリ・プール生成情報は、以下の通りです。
 - 固定長メモリ・プール属性 (`mpfatr`)
以下を指定します。
 - タスク待ちキューの順序 (FIFO 順またはタスクの現在優先度順)
 - 獲得可能なメモリ・ブロック数 (`blkcnt`)、メモリ・ブロックのサイズ (バイト数) (`blksz`)、固定長メモリ・プール領域の先頭アドレス (`mpf`)
`mpf` から `TSZ_MPF(blkcnt, blksz)` バイトを固定長メモリ・プール領域とします。TSZ_MPF マクロについては「[18.3.3 固定長メモリ・プール関連](#)」を参照してください。
固定長メモリ・プール領域は、メモリ・ブロックにアクセスするタスクがアクセス可能なメモリオブジェクト内としてください。
 - 固定長メモリ・プール管理領域の先頭アドレス (`mpfmb`)
`mpfmb` から `TSZ_MPFMB(blkcnt, blksz)` バイトを固定長メモリ・プール管理領域とします。TSZ_MPFMB マクロについては「[18.3.3 固定長メモリ・プール関連](#)」を参照してください。
固定長メモリ・プール管理領域は、メモリ・オブジェクト以外でかつユーザ・スタック以外の領域に作成してください。

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_mpf` の記述例を示します。

```

#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */

#define BLKCNT 32                      /* 獲得可能なメモリ・ブロック数 */
#define BLKSZ  16                      /* メモリ・ブロックのサイズ (バイト数) */

#pragma section B BU_SH                /* 固定長メモリ・プール領域のセクション */
static UW mpf_area[ TSZ_MPF( BLKCNT, BLKSZ)/sizeof(UW)]; /* 固定長メモリ・プール領域 */

#pragma section B BRI_RAM              /* 固定長メモリ・プール管理領域のセクション */
static UW mpfmb_area[ TSZ_MPFMB( BLKCNT, BLKSZ)/sizeof(UW)]; /* 固定長メモリ・プール管理領域 */

#pragma section
#pragma task Task1                    /* 備考参照 */
void Task1 ( VP_INT exinf );          /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mpfid;                    /* 変数の宣言 */
    T_CMPF  pk_cmpf = {               /* 変数の宣言, 初期化 */
        TA_TFIFO,                    /* 固定長メモリ・プール属性 (mpfatr) */
        BLKCNT,                      /* 獲得可能なメモリ・ブロック数 (blkcnt) */
        BLKSZ,                       /* メモリ・ブロックのサイズ (バイト数) (blksz) */
        (VP)mpf_area,                /* 固定長メモリ・プール領域の先頭アドレス (mpf) */
        (VP)mpfmb_area               /* 固定長メモリ・プール管理領域の先頭アドレス (mpfmb) */
    };
    .....
    .....
    mpfid = acre_mpf ( &pk_cmpf );    /* 固定長メモリ・プールの生成 */
    .....
    .....
}

```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

9.2.2 固定長メモリ・プールの削除

- del_mpf

パラメータ *mpfid* で指定された固定長メモリ・プールを削除します。

対象固定長メモリ・プールで *get_mpf* または *tget_mpf* によって待っているタスクがある場合は、そのタスク待ち状態を解除し、*get_mpf* または *tget_mpf* の戻り値として E_DLT を返します。

本サービス・コールは、**信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mpfid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_mpf( mpfid );                  /* 固定長メモリ・プールの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

9.2.3 固定長メモリ・ブロックの獲得

固定長メモリ・ブロックの獲得は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

RI600PX では、固定長メモリ・ブロックを獲得する際、メモリ・クリア処理を行っていません。したがって、獲得された固定長メモリ・ブロックの内容は不定となります。

- `get_mpf` (待つ)
 - `pget_mpf`, `ipget_mpf` (ポーリング)
 - `tget_mpf` (タイムアウト付きで待つ)
- `get_mpf` (待つ)

パラメータ `mpfid` で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し、その先頭アドレスをパラメータ `p_blk` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には、固定長メモリ・ブロックの獲得は行わず、自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお、固定長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<code>rel_mpf</code> の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
<code>irel_mpf</code> の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_mpf</code> の発行により、対象固定長メモリ・プールがリセットされた。	EV_RST
<code>del_mpf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 4 参照 */
void Task1 ( VP_INT exinf );      /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      mpfid = 1;            /* 変数の宣言, 初期化 */
    VP      p_blk;               /* 変数の宣言 */

    .....
    .....

                                /* 固定長メモリ・ブロックの獲得 */
    ercd = get_mpf ( mpfid, &p_blk );

    if ( ercd == E_OK ) {
        .....                  /* 正常終了処理 */
        .....

                                /* 固定長メモリ・ブロックの返却 */
        rel_mpf ( mpfid, p_blk );
    } else if ( ercd == E_RLWAI ) {
        .....                  /* 強制終了処理 */
        .....

    }

    .....
    .....
}

```

- 備考 1 自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、**固定長メモリ・プールの生成**に指定した順（FIFO 順または現在優先度順）に行われます。
- 備考 2 獲得したメモリ・ブロックの内容は不定です。
- 備考 3 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は、以下を守ってください。
- 固定長メモリ・プール生成時に指定するメモリ・ブロック・サイズを、目的のアライメント数の倍数とする。
 - 固定長メモリ・プール領域の先頭アドレスを、目的のアライメント数のアドレスとする。
- 備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

- `pget_mpf`, `ipget_mpf` (ポーリング)

パラメータ `mpfid` で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し、その先頭アドレスをパラメータ `p_blk` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には、固定長メモリ・ブロックの獲得は行わず、戻り値として `E_TMOUT` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                     /* 備考3 参照 */
void Task1 ( VP_INT exinf );           /* 備考3 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      mpfid = 1;                  /* 変数の宣言, 初期化 */
    VP      p_blk;                      /* 変数の宣言 */

    .....
    .....

                                /* 固定長メモリ・ブロックの獲得 */
    ercd = pget_mpf ( mpfid, &p_blk );

    if ( ercd == E_OK ) {
        .....                       /* ポーリング成功処理 */
        .....

                                /* 固定長メモリ・ブロックの返却 */
        rel_mpf ( mpfid, p_blk );
    } else if ( ercd == E_TMOUT ) {
        .....                       /* ポーリング失敗処理 */
        .....

    }

    .....
    .....
}
```

備考1 獲得したメモリ・ブロックの内容は不定です。

備考2 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は、以下を守ってください。

- 固定長メモリ・プール生成時に指定するメモリ・ブロック・サイズを、目的のアライメント数の倍数とする。
- 固定長メモリ・プール領域の先頭アドレスを、目的のアライメント数のアドレスとする。

備考3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `tget_mpf` (タイムアウト付きで待つ)

パラメータ `mpfid` で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し、その先頭アドレスをパラメータ `p_blk` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった(空き固定長メモリ・ブロックが存在しなかった)場合には、固定長メモリ・ブロックの獲得は行わず、自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態(固定長メモリ・ブロック獲得待ち状態)へと遷移させます。

なお、固定長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<code>rel_mpf</code> の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
<code>irel_mpf</code> の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_mpf</code> の発行により、対象固定長メモリ・プールがリセットされた。	EV_RST
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_mpf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 5 参照 */
void Task1 ( VP_INT exinf );           /* 備考 5 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      mpfid = 1;                  /* 変数の宣言, 初期化 */
    VP      p_blk;                      /* 変数の宣言 */
    TMO      tmout = 3600;              /* 変数の宣言, 初期化 */

    .....
    .....

    /* 固定長メモリ・ブロックの獲得 */
    ercd = tget_mpf ( mpfid, &p_blk, tmout );

    if ( ercd == E_OK ) {
        .....                          /* 正常終了処理 */
        .....

        /* 固定長メモリ・ブロックの返却 */
        rel_mpf ( mpfid, p_blk );
    } else if ( ercd == E_RLWAI ) {
        .....                          /* 強制終了処理 */
        .....
    } else if ( ercd == E_TMOUT ) {
        .....                          /* タイムアウト処理 */
        .....
    }

    .....
    .....
}
```

- 備考 1 自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、固定長メモリ・プール生成時に指定した順（FIFO 順または現在優先度順）に行われます。
- 備考 2 獲得したメモリ・ブロックの内容は不定です。
- 備考 3 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は、以下を守ってください。
- 固定長メモリ・プール生成時に指定するメモリ・ブロック・サイズを、目的のアライメント数の倍数とする。
 - 固定長メモリ・プール領域の先頭アドレスを、目的のアライメント数のアドレスとする。
- 備考 4 待ち時間 *tmout* に `TMO_FEVR` が指定された際には“`get_mpf` と同等の処理”を、`TMO_POL` が指定された際には“`pget_mpf` と同等の処理”を実行します。
- 備考 5 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

9.2.4 固定長メモリ・ブロックの返却

固定長メモリ・ブロックの返却は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rel_mpf`, `irel_mpf`

パラメータ `mpfid` で指定された固定長メモリ・プールにパラメータ `blk` で指定された固定長メモリ・ブロックを返却します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には、固定長メモリ・ブロックの返却は行わず、該当タスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。これにより、該当タスクは、待ちキューから外れ、WAITING 状態（固定長メモリ・ブロック獲得待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );            /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                        /* 変数の宣言 */
    ID      mpfid = 1;                   /* 変数の宣言, 初期化 */
    VP      blk;                          /* 変数の宣言 */

    .....
    .....

    ercd = get_mpf ( mpfid, &blk ); /* 固定長メモリ・ブロックの獲得 */

    if ( ercd == E_OK ) {
        .....                          /* 正常終了処理 */
        .....

        rel_mpf ( mpfid, blk ); /* 固定長メモリ・ブロックの返却 */
    } else if ( ercd == E_RLWAI ) {
        .....                          /* 強制終了処理 */
        .....
    }

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

9.2.5 固定長メモリ・プール詳細情報の参照

固定長メモリ・プール詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_mpf`, `iref_mpf`

パラメータ `mpfid` で指定された固定長メモリ・プールの固定長メモリ・プール詳細情報（待ちタスクの有無、空き固定長メモリ・ブロックの総数など）をパラメータ `pk_rmpf` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );           /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      mpfid = 1;                  /* 変数の宣言, 初期化 */
    T_RMPF  pk_rmpf;                   /* データ構造体の宣言 */
    ID      wtskid;                     /* 変数の宣言 */
    UINT    fblkcnt;                   /* 変数の宣言 */

    .....
    .....

    ref_mpf ( mpfid, &pk_rmpf );      /* 固定長メモリ・プール詳細情報の参照 */

    wtskid = pk_rmpf.wtskid;           /* 待ちタスクの有無の獲得 */
    fblkcnt = pk_rmpf.fblkcnt;        /* 空き固定長メモリ・ブロックの総数の獲得 */

    .....
    .....
}
```

備考 1 固定長メモリ・プール詳細情報 T_RMPF についての詳細は、「【[固定長メモリ・プール詳細情報 T_RMPF の構造](#)】」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

9.3 可変長メモリ・プール

RI600PX では、処理プログラムから動的なメモリ操作要求が行われた際に利用するメモリ領域として“可変長メモリ・プール”を提供しています。

なお、可変長メモリ・プールに対する動的なメモリ操作は、任意サイズの可変長メモリ・ブロックを単位として行われます。

9.3.1 可変長メモリ・ブロックのサイズ

現在の RI600PX の実装では、実際に獲得されるメモリ・ブロックのサイズは、最大で 12 種類のバリエーションから選択されます。このバリエーションは、可変長メモリ・プール生成時に指定した最大メモリ・ブロック・サイズを元に、あらかじめ規定された 24 種類のサイズの中から選択されます。表 9-1 に、メモリ・ブロックサイズのバリエーションを示します。ただし、この振る舞いは将来変更される可能性があります。

表 9-1 メモリ・ブロック・サイズのバリエーション

項番	メモリ・ブロックのサイズ (16 進数)	例 1: 最大メモリ・ブロック・サイズ =0x100 の場合	例 2: 最大メモリ・ブロック・サイズ =0x20000 の場合
1	12 (0xC)	○	×
2	36 (0x24)	○	×
3	84 (0x54)	○	○
4	180 (0xB4)	○	○
5	372 (0x174)	×	○
6	756 (0x2F4)	×	○
7	1524 (0x5F4)	×	○
8	3060 (0xBF4)	×	○
9	6132 (0x17F4)	×	○
10	12276 (0x2FF4)	×	○
11	24564 (0x5FF4)	×	○
12	49140 (0xBFF4)	×	○
13	98292 (0x17FF4)	×	○
14	196596 (0x2FFF4)	×	○
15	393204 (0x5FFF4)	×	×
16	786420 (0xBFFF4)	×	×
17	1572852 (0x17FFF4)	×	×
18	3145716 (0x2FFFF4)	×	×
19	6291444 (0x5FFFF4)	×	×
20	12582900 (0xBFFFF4)	×	×
21	25165812 (0x17FFFF4)	×	×
22	50331636 (0x2FFFFFF4)	×	×
23	100663284 (0x5FFFFFF4)	×	×
24	201326580 (0xBFFFFFF4)	×	×

9.3.2 可変長メモリ・プールの生成

可変長メモリ・プールは、以下のいずれかの方法で生成します。

1) システム・コンフィギュレーション・ファイルによる生成

システム・コンフィギュレーション・ファイルで静的 API“variable_memorypool[]” を使用して可変長メモリ・プールを生成します。

静的 API“variable_memorypool[]” の詳細は、「[20.18 可変長メモリ・プール情報 \(variable_memorypool\[\]\)](#)」を参照してください。

2) `cre_mpl` または `acre_mpl` サービスコールによる生成

`cre_mpl` は、パラメータ `pk_cmpl` が指す領域に設定された可変長メモリ・プール生成情報にしたがって、パラメータ `mplid` で指定された可変長メモリ・プール ID の可変長メモリ・プールを生成します。

`acre_mpl` は、パラメータ `pk_cmpl` が指す領域に設定された可変長メモリ・プール生成情報にしたがって可変長メモリ・プールを生成し、生成された可変長メモリ・プール ID を返します。

指定する可変長メモリ・プール生成情報は、以下の通りです。

- 可変長メモリ・プール属性 (`mplatr`)

「タスク待ちキューは FIFO 順 (TA_TFIFO)」のみを指定できます。

- 可変長メモリ・プール領域のサイズ (バイト数) (`mplsz`)、可変長メモリ・プール領域の先頭アドレス (`mpl`)

可変長メモリ・プールの先頭アドレスは、4 バイト境界でなければなりません。

可変長メモリ・プール領域は、メモリ・ブロックにアクセスするタスクがアクセス可能なメモリオブジェクト内としてください。

- 最大メモリ・ブロック・サイズ (バイト数) (`maxblksz`)

可変長メモリ・プールから獲得するメモリ・ブロックの最大サイズを指定します。「[9.3.1 可変長メモリ・ブロックのサイズ](#)」も参照してください。

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_mpl` の記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */

#define MPLSZ      1024          /* 可変長メモリ・プール領域のサイズ (バイト数) */
#define MAXBLKSZ  128          /* 最大メモリ・ブロック・サイズ (バイト数) */

#pragma section B BU_SH         /* 可変長メモリ・プール領域のセクション */
static UW mpl_area[ MPLSZ/sizeof(UW) ]; /* 可変長メモリ・プール領域 */
#pragma section

#pragma task Task1              /* 備考参照 */
void Task1 ( VP_INT exinf );    /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mplid;              /* 変数の宣言 */
    T_CMPL  pk_cmpl = {        /* 変数の宣言, 初期化 */
        TA_TFIFO,              /* 可変長メモリ・プール属性 (mplatr) */
        MPLSZ,                  /* 可変長メモリ・プール領域のサイズ (バイト数) (mplsz) */
        (VP)mpl_area,          /* 可変長メモリ・プール領域の先頭アドレス (mpl) */
        MAXBLKSZ                /* 最大メモリ・ブロック・サイズ (バイト数) (maxblksz) */
    };
    .....
    mplid = acre_mpl ( &pk_cmpl ); /* 可変長メモリ・プールの生成 */
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

9.3.3 可変長メモリ・プールの削除

- del_mpl

パラメータ *mplid* で指定された可変長メモリ・プールを削除します。

対象可変長メモリ・プールで *get_mpl* または *tget_mpl* によって待っているタスクがある場合は、そのタスク待ち状態を解除し、*get_mpl* または *tget_mpl* の戻り値として E_DLT を返します。

本サービス・コールは、**信頼されたドメイン**に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );           /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      mplid = 8;                  /* 変数の宣言, 初期化 */

    .....
    .....

    del_mpl( mplid );                  /* 可変長メモリ・プールの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

9.3.4 可変長メモリ・ブロックの獲得

可変長メモリ・ブロックの獲得は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

RI600PX では、可変長メモリ・ブロックを獲得する際、メモリ・クリア処理を行っていません。したがって、獲得された可変長メモリ・ブロックの内容は不定となります。

- `get_mpl` (待つ)
- `pget_mpl`, `ipget_mpl` (ポーリング)
- `tget_mpl` (タイムアウト付きで待つ)
- `get_mpl` (待つ)

パラメータ `mplid` で指定された可変長メモリ・プールからパラメータ `blksz` で指定されたサイズの可変長メモリ・ブロックを獲得し、その先頭アドレスをパラメータ `p_blk` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には、可変長メモリ・ブロックの獲得は行わず、自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお、可変長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<code>rel_mpl</code> の発行により、対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <code>rel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>ter_tsk</code> の発行により、待ち状態を強制的に解除された。 - <code>tget_mpl</code> のパラメータ <code>tmout</code> で指定された待ち時間が経過した。 	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_mpl</code> の発行により、対象可変長メモリ・プールがリセットされた。	EV_RST
<code>del_mpl</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 5 参照 */
void Task1 ( VP_INT exinf );      /* 備考 5 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                 /* 変数の宣言 */
    ID      mplid = 1;            /* 変数の宣言, 初期化 */
    UINT    blkksz = 256;        /* 変数の宣言, 初期化 */
    VP      p_blk;                /* 変数の宣言 */

    .....
    .....

                                /* 可変長メモリ・ブロックの獲得 */
    ercd = get_mpl ( mplid, blkksz, &p_blk );

    if ( ercd == E_OK ) {
        .....                    /* 正常終了処理 */
        .....

                                /* 可変長メモリ・ブロックの返却 */
        rel_mpl ( mplid, p_blk );
    } else if ( ercd == E_RLWAI ) {
        .....                    /* 強制終了処理 */
        .....

    }

    .....
    .....
}

```

- 備考 1 実際に獲得されるメモリ・ブロックのサイズについては、「9.3.1 可変長メモリ・ブロックのサイズ」を参照してください。
- 備考 2 自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考 3 獲得したメモリ・ブロックの内容は不定です。
- 備考 4 獲得するメモリ・ブロックのアライメント数は、可変長メモリ・プールの生成方法によって異なります。
- システム・コンフィギュレーション・ファイルで静的に生成した場合
獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには、可変長メモリ・プール情報 (variable_memorypool[]) のメモリ・プール領域に付与するセクション名 (mpl_section) に個別のセクション名を指定し、リンク時にそのセクションを 4 バイト境界アドレスに配置してください。
 - cre_mpl または acre_mpl サービスコールで動的に生成した場合
獲得するメモリ・ブロックのアライメント数は 4 です。
- 備考 5 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

- `pget_mpl`, `ipget_mpl` (ポーリング)

パラメータ `mplid` で指定された可変長メモリ・プールからパラメータ `blksz` で指定されたサイズの可変長メモリ・ブロックを獲得し、その先頭アドレスをパラメータ `p_blk` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には、可変長メモリ・ブロックの獲得は行わず、戻り値として `E_TMOU` を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 4 参照 */
void Task1 ( VP_INT exinf );           /* 備考 4 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      mplid = 1;                  /* 変数の宣言, 初期化 */
    UINT    blksz = 256;               /* 変数の宣言, 初期化 */
    VP      p_blk;                     /* 変数の宣言 */

    .....
    .....

                                /* 可変長メモリ・ブロックの獲得 */
    ercd = pget_mpl ( mplid, blksz, &p_blk );

    if ( ercd == E_OK ) {
        .....                       /* ポーリング成功処理 */
        .....

                                /* 可変長メモリ・ブロックの返却 */
        rel_mpl ( mplid, p_blk );
    } else if ( ercd == E_TMOU ) {
        .....                       /* ポーリング失敗処理 */
        .....

    }

    .....
    .....
}
```

備考 1 実際に獲得されるメモリ・ブロックのサイズについては、「9.3.1 可変長メモリ・ブロックのサイズ」を参照してください。

備考 2 獲得したメモリ・ブロックの内容は不定です。

備考 3 獲得するメモリ・ブロックのアライメント数は、可変長メモリ・プールの生成方法によって異なります。

- システム・コンフィギュレーション・ファイルで静的に生成した場合
獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには、[可変長メモリ・プール情報 \(variable_memorypool\[\]\)](#) のメモリ・プール領域に付与するセクション名 (`mpl_section`) に個別のセクション名を指定し、リンク時にそのセクションを 4 バイト境界アドレスに配置してください。
- `cre_mpl` または `acre_mpl` サービスコールで動的に生成した場合
獲得するメモリ・ブロックのアライメント数は 4 です。

備考 4 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

- `tget_mpl` (タイムアウト付きで待つ)

パラメータ `mplid` で指定された可変長メモリ・プールからパラメータ `blksz` で指定されたサイズの可変長メモリ・ブロックを獲得し、その先頭アドレスをパラメータ `p_blk` で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった(要求サイズ分の連続する空き領域が存在しなかった)場合には、可変長メモリ・ブロックの獲得は行わず、自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態(可変長メモリ・ブロック獲得待ち状態)へと遷移させます。

なお、可変長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<code>rel_mpl</code> の発行により、対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <code>rel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>ter_tsk</code> の発行により、待ち状態を強制的に解除された。 - <code>tget_mpl</code> のパラメータ <code>tmout</code> で指定された待ち時間が経過した。 	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_mpl</code> の発行により、対象可変長メモリ・プールがリセットされた。	EV_RST
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_mpl</code> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、本サービス・コールの記述例を示します。


```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考 6 参照 */
void Task1 ( VP_INT exinf );     /* 備考 6 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      mplid = 1;           /* 変数の宣言, 初期化 */
    UINT    blkksz = 256;       /* 変数の宣言, 初期化 */
    VP      p_blk;              /* 変数の宣言 */
    TMO     tmout = 3600;       /* 変数の宣言, 初期化 */

    .....
    .....

                                /* 可変長メモリ・ブロックの獲得 */
    ercd = tget_mpl ( mplid, blkksz, &p_blk, tmout );

    if ( ercd == E_OK ) {
        .....                  /* 正常終了処理 */
        .....

                                /* 可変長メモリ・ブロックの返却 */
        rel_mpl ( mplid, p_blk );
    } else if ( ercd == E_RLWAI ) {
        .....                  /* 強制終了処理 */
        .....

    } else if ( ercd == E_TMOUT ) {
        .....                  /* タイムアウト処理 */
        .....

    }

    .....
    .....
}

```

- 備考 1 実際に獲得されるメモリ・ブロックのサイズについては、「9.3.1 可変長メモリ・ブロックのサイズ」を参照してください。
- 備考 2 自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考 3 獲得したメモリ・ブロックの内容は不定です。
- 備考 4 獲得するメモリ・ブロックのアライメント数は、可変長メモリ・プールの生成方法によって異なります。
- システム・コンフィギュレーション・ファイルで静的に生成した場合
獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには、可変長メモリ・プール情報 (variable_memorypool[]) のメモリ・プール領域に付与するセクション名 (mpl_section) に個別のセクション名を指定し、リンク時にそのセクションを 4 バイト境界アドレスに配置してください。
 - cre_mpl または acre_mpl サービスコールで動的に生成した場合
獲得するメモリ・ブロックのアライメント数は 4 です。
- 備考 5 待ち時間 tmout に TMO_FEVR が指定された際には “get_mpl と同等の処理” を、TMO_POL が指定された際には “pget_mpl と同等の処理” を実行します。
- 備考 6 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

9.3.5 可変長メモリ・ブロックの返却

可変長メモリ・ブロックの返却は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rel_mpl`

パラメータ `mplid` で指定された可変長メモリ・プールにパラメータ `blk` で指定された可変長メモリ・ブロックを返却します。

可変長メモリ・ブロックを返却したあと、対象可変長メモリ・プールの待ちキューにキューイングされているタスクをキューの先頭から調べていき、待ちタスクが要求するサイズのメモリを割り当てられる場合はメモリを割り当てます。この動作を待ちキューにタスクがなくなるか、メモリが割り当てられなくなるまで繰り返します。これにより、メモリを獲得できたタスクは、待ちキューから外れ、WAITING 状態（可変長メモリ・ブロック獲得待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );           /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                       /* 変数の宣言 */
    ID      mplid = 1;                  /* 変数の宣言, 初期化 */
    UINT    blkksz = 256;              /* 変数の宣言, 初期化 */
    VP      blk;                       /* 変数の宣言 */

    .....
    .....

                                /* 可変長メモリ・ブロックの獲得 */
    ercd = get_mpl ( mplid, blkksz, &blk );

    if ( ercd == E_OK ) {
        .....                      /* 正常終了処理 */
        .....

        rel_mpl ( mplid, blk ); /* 可変長メモリ・ブロックの返却 */
    } else if ( ercd == E_RLWAI ) {
        .....                      /* 強制終了処理 */
        .....
    }

    .....
    .....
}
```

備考 1 RI600PX では、`blk` に関して簡易的なエラー検出しか行っていません。`blk` には、必ず正しい値を指定してください。`blk` がエラー検出されない不正な値の場合、以後の動作は保証されません。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

9.3.6 可変長メモリ・プール詳細情報の参照

可変長メモリ・プール詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_mpl`, `iref_mpl`

パラメータ `mplid` で指定された可変長メモリ・プールの可変長メモリ・プール詳細情報（待ちタスクの有無、空き可変長メモリ・ブロックの合計サイズなど）をパラメータ `pk_rmpl` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      mplid = 1;                 /* 変数の宣言, 初期化 */
    T_RMPL  pk_rmpl;                  /* データ構造体の宣言 */
    ID      wtskid;                   /* 変数の宣言 */
    SIZE    fmplsz;                   /* 変数の宣言 */
    UINT    fblksz;                   /* 変数の宣言 */

    .....
    .....

    ref_mpl ( mplid, &pk_rmpl );      /* 可変定長メモリ・プール詳細情報の参照 */

    wtskid = pk_rmpl.wtskid;          /* 待ちタスクの有無の獲得 */
    fmplsz = pk_rmpl.fmplsz;         /* 空き領域の合計サイズの獲得 */
    fblksz = pk_rmpl.fblksz;        /* 獲得可能なメモリ・ブロックの最大サイズの獲得 */

    .....
    .....
}
```

備考 1 可変長メモリ・プール詳細情報 T_RMPL についての詳細は、「【[可変長メモリ・プール詳細情報 T_RMPL の構造](#)】」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

第10章 時間管理機能

本章では、RI600PX が提供している時間管理機能について解説しています。

10.1 概要

RI600PX における時間管理機能では、一定周期で発生する基本クロック用タイマ割り込みを利用してシステム時刻を操作／参照する機能のほかに、時間に依存した処理を実現する手段（タイマ・オペレーション機能：[タスクの遅延](#)、[タイムアウト](#)、[周期ハンドラ](#)、[アラーム・ハンドラ](#)、[システム時刻](#)）を提供しています。

10.2 システム時刻

システム時刻とは、RI600PX が時間管理を行う際に使用する“時間（単位：ミリ秒）”です。

システム時刻は、[カーネル初期化部 \(vsta_knl, ivsta_knl\)](#) において 0 に初期化されたのち、[システム情報 \(system\)](#) の[基本クロック割り込み周期の分母 \(tic_deno\)](#)、および[基本クロック割り込み周期の分子 \(tic_num\)](#) で定義された基本クロック周期ごとに更新されます。

10.2.1 基本クロック用タイマ割り込み

RI600PX では、時間管理機能を実現するために、一定周期で発生する割り込み（基本クロック用タイマ割り込み）を使用します。

基本クロック用タイマ割り込みが発生した際は、RI600PX の時間に関連した処理（システム時刻の更新、タスクのタイムアウト／遅延、周期ハンドラの起動、アラーム・ハンドラの起動など）が実行されます。

基本クロック用タイマは、RX MCU 内蔵のコンペア・マッチ・タイマ（CMT）のチャンネル 0～3 のいずれかを使用することを基本としています。どのチャンネルを使用するかは、システム・コンフィギュレーション・ファイルの[基本クロック割り込み情報 \(clock\)](#) の[基本クロック用タイマ・チャンネルの選択 \(timer\)](#) で指定します。

基本クロック用タイマ割り込みを発生するためのハードウェアの初期化は、ユーザ・OWN・コーディング部として切り出されています。詳細は、「[10.9 基本クロック用タイマ初期化ルーチン \(_RI_init_cmt_knl\(\)\)](#)」を参照してください。

10.2.2 基本クロック周期

RI600PX のサービス・コールでは時間指定パラメータの単位は「ミリ秒」になっています。

基本クロック用タイマ割り込みの発生周期は 1 ミリ秒とするのが望ましいですが、ターゲット・システムの性質上（処理能力、必要とする時間分解能など）1 ミリ秒とすることが困難な場合があります。

この場合、システム・コンフィギュレーション・ファイルの[システム情報 \(system\)](#) の[基本クロック割り込み周期の分母 \(tic_deno\)](#)、および[基本クロック割り込み周期の分子 \(tic_num\)](#) で基本クロック用タイマ割り込みの発生周期を指定することができます。

基本クロック周期を指定すると 1 回の基本クロック用タイマ割り込みで基本クロック周期分の時間が経過したとして処理されます。

10.3 タイマ・オペレーション機能

RI600PXにおけるタイマ・オペレーション機能では、時間に依存した処理を実現する手段として“[タスクの遅延](#)、[タイムアウト](#)、[周期ハンドラ](#)、[アラーム・ハンドラ](#)、[システム時刻](#)”を提供しています。

10.4 タスクの遅延

タスクの遅延は、一定の時間が経過するまでの間、自タスクを RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させ、一定の時間が経過した際には、該当タスクを WAITING 状態から READY 状態へと遷移させるものです。

なお、遅延起床は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

[dly_tsk](#)

10.5 タイムアウト

タイムアウトとは、タスクから発行された要求条件が即時成立しなかった場合、一定の時間が経過するまでの間、該当タスクを RUNNING 状態から WAITING 状態（起床待ち状態、資源獲得待ち状態、イベントフラグ待ち状態など）へと遷移させ、一定の時間が経過した際には、要求条件の成立／不成立を問わず、該当タスクを WAITING 状態から READY 状態へと遷移させるものです。

なお、タイムアウトは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

[tslp_tsk](#)

[twai_sem](#)

[twai_flg](#)

[tsnd_dtq](#)

[trcv_dtq](#)

[trcv_mbx](#)

[tlloc_mtx](#)

[tsnd_mbf](#)

[trcv_mbf](#)

[tget_mpf](#)

[tget_mpl](#)

10.6 周期ハンドラ

周期ハンドラは、一定の時間ごとに周期的に起動される周期処理専用ルーチンです。

なお、RI600PXでは、周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため、一定の時間が経過した際には、システム内で最高優先度を持つタスクが処理を実行中であっても、その処理は中断され、周期ハンドラに制御が移ります。

10.6.1 周期ハンドラの基本形

以下に、周期ハンドラを記述する場合の基本型を示します。exinfには、周期ハンドラ生成時に指定した拡張情報が渡されます。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma cychandler Cychdr1      /* 備考参照 */
void Cychdr1 ( VP_INT exinf );   /* 備考参照 */
void Cychdr1 ( VP_INT exinf )
{
    .....
    .....

    return;                      /* 周期ハンドラの終了 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成した周期ハンドラについては、これらのステートメントはcfg600pxがkernel_id.hに出力するため、記述不要です。

10.6.2 周期ハンドラ内での処理

- スタック
周期ハンドラは、システム・スタックを使用します。
- サービス・コールの発行
RI600PX では、周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。
周期ハンドラでは、“発行有効範囲”が“非タスク”のサービスコールを発行可能です。

備考 RI600PX では、周期ハンドラ内の処理を高速に終了させる目的から、周期ハンドラ内の処理が完了するまでの間、スケジューラの起動を遅延しています。したがって、周期ハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（`isig_sem`, `iset_flg` など）が発行された際には、キュー操作などといった処理が行われるだけであり、実際のディスパッチ処理の実行は“周期ハンドラからの復帰命令（`return` 命令の発行）”が発行されるまで遅延され、一括して行うようにしています。

- 処理開始時の PSW

表 10 - 1 周期ハンドラ処理開始時の PSW

ビット	値	備考
I	1	
IPL	基本クロック割り込み優先レベル (IPL)	処理開始時より下げたはなりません。
PM	0	スーパーバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

10.6.3 周期ハンドラの生成

周期ハンドラは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API "cyclic_hand[]" を使用して周期ハンドラを生成します。
静的 API "cyclic_hand[]" の詳細は、「20.19 周期ハンドラ情報 (cyclic_hand[])」を参照してください。
- 2) `cre_cyc` または `acre_cyc` サービスコールによる生成
`cre_cyc` は、パラメータ `pk_ccyc` が指す領域に設定された周期ハンドラ生成情報にしたがって、パラメータ `cycid` で指定された周期ハンドラ ID のタスクを生成します。
`acre_cyc` は、パラメータ `pk_ccyc` が指す領域に設定された周期ハンドラ生成情報にしたがって周期ハンドラ生成し、生成された周期ハンドラ ID を返します。
指定する周期ハンドラ生成情報は、以下の通りです。
 - 周期ハンドラ属性 (`cycatr`)
以下を指定します。
 - 生成後に動作状態にする指定 (TA_STA 属性)
 - 起動位相を保存する指定 (TA_PHS 属性)
 - 拡張情報 (`exinf`)
 - 周期ハンドラの実行開始アドレス (`cychdr`)
 - 起動周期 (`cyctim`)
 - 起動位相 (`cycphs`)

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_cyc` の記述例を示します。

```
#include      "kernel.h"           /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"        /* cfg600px が出力するヘッダ・ファイルの定義 */

extern void Cychdr1 ( VP_INT exinf );

#pragma task Task1                /* 備考参照 */
void Task1 ( VP_INT exinf );      /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      cycid;                /* 変数の宣言 */
    T_CCYC  pk_ccyc = {           /* 変数の宣言, 初期化 */
        TA_STA,                    /* 周期ハンドラ属性 (cycatr) */
        0,                          /* 拡張情報 (exinf) */
        (FP) Cychdr1,              /* 周期ハンドラの開始アドレス (cyhdr) */
        10,                         /* 起動周期 (cyctim) */
        2,                          /* 起動位相 (cycphs) */
    };
    .....
    cycid = acre_cyc ( &pk_ccyc ); /* 周期ハンドラの生成 */
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.6.4 周期ハンドラの削除

- del_cyc

パラメータ *cycid* で指定された周期ハンドラを削除します。

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"           /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"        /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                /* 備考参照 */
void Task1 ( VP_INT exinf );      /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      cycid = 8;             /* 変数の宣言, 初期化 */

    .....
    .....

    del_cyc( cycid );             /* 周期ハンドラの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

10.6.5 周期ハンドラの動作開始

周期ハンドラの動作開始は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `sta_cyc`, `ista_cyc`

パラメータ `cycid` で指定された周期ハンドラの動作状態を停止状態 (STP 状態) から動作状態 (STA 状態) へと遷移させます。これにより、対象周期ハンドラは、RI600PX の起動対象となります。

なお、本サービス・コールの発行から 1 回目の起動要求が発行されるまでの相対時間間隔は、[周期ハンドラの生成](#)時に `TA_PHS` 属性を指定しているか否かにより異なります。

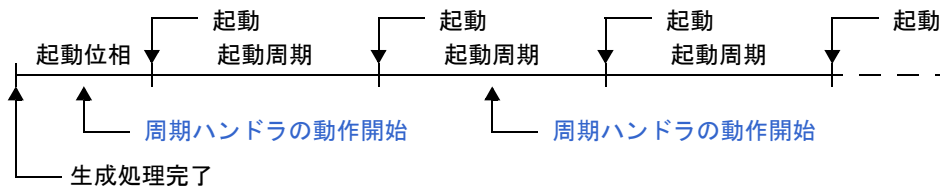
- `TA_PHS` 属性を指定した場合

起動位相および起動周期にしたがって、起動タイミングが設定されます。

ただし、対象周期ハンドラの動作状態が開始状態の場合には、本サービス・コールを発行しても何も処理は行わず、エラーとしても扱いません

[図 10 - 1](#) に、周期ハンドラの起動タイミング・イメージを示します。

図 10 - 1 `TA_PHS` 属性：指定あり



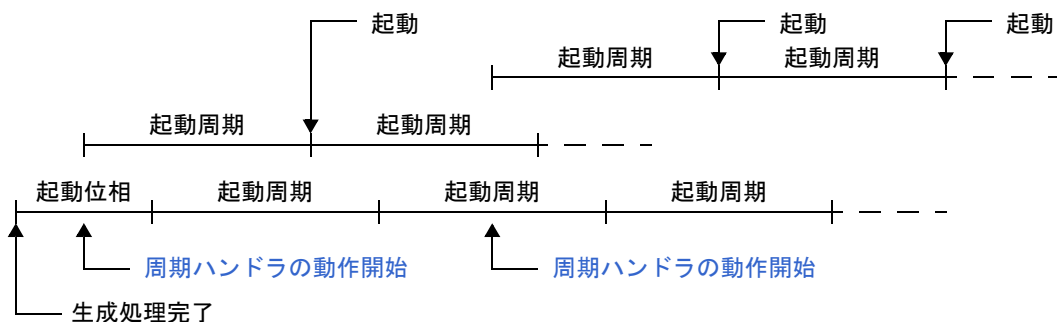
- `TA_PHS` 属性を指定しない場合

本サービス・コールの発行時点を中心に、起動周期にしたがって起動タイミングが設定されます。

なお、起動タイミング設定処理については、対象周期ハンドラの動作状態に関係なく実行されます。

[図 10 - 2](#) に、周期ハンドラの起動タイミング・イメージを示します。

図 10 - 2 `TA_PHS` 属性：指定なし



以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考参照 */
void Task1 ( VP_INT exinf );     /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      cycid = 1;           /* 変数の宣言, 初期化 */

    .....
    .....

    sta_cyc ( cycid );         /* 周期ハンドラの動作開始 */

    .....
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは cfg600px が kernel_id.h に出力するため、記述不要です。

10.6.6 周期ハンドラの動作停止

周期ハンドラの動作停止は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `stp_cyc`, `istp_cyc`

パラメータ `cycid` で指定された周期ハンドラの動作状態を動作状態 (STA 状態) から停止状態 (STP 状態) へと遷移させます。これにより、本サービス・コールの発行から `sta_cyc` または `ista_cyc` が発行されるまでの間、対象周期ハンドラは、RI600PX の起動対象から除外されます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      cycid = 1;           /* 変数の宣言, 初期化 */

    .....

    stp_cyc ( cycid );          /* 周期ハンドラの動作停止 */

    .....
}
```

備考 1 本サービス・コールでは、停止要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、対象周期ハンドラの動作状態が停止状態 (STP 状態) へと遷移していた場合には、何も処理は行わず、エラーとしても扱いません。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.6.7 周期ハンドラ詳細情報の参照

周期ハンドラ詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_cyc`, `iref_cyc`

パラメータ `cycid` で指定された周期ハンドラの周期ハンドラ詳細情報（現在状態、残り時間）をパラメータ `pk_rcyc` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );            /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      cycid = 1;                  /* 変数の宣言, 初期化 */
    T_RCYC  pk_rcyc;                   /* データ構造体の宣言 */
    STAT    cycstat;                   /* 変数の宣言 */
    RELTIM  lefttim;                  /* 変数の宣言 */

    .....
    .....

    ref_cyc ( cycid, &pk_rcyc );       /* 周期ハンドラ詳細情報の参照 */

    cycstat = pk_rcyc.cycstat;         /* 現在状態の獲得 */
    lefttim = pk_rcyc.lefttim;        /* 残り時間の獲得 */

    .....
    .....
}
```

備考 1 周期ハンドラ詳細情報 T_RCYC についての詳細は、[【周期ハンドラ詳細情報 T_RCYC の構造】](#)を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.7 アラーム・ハンドラ

アラーム・ハンドラは、指定した時間が経過したときに起動されるルーチンです。

なお、RI600PXでは、アラーム・ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため、指定した時間が経過した際には、システム内で最高優先度を持つタスクが処理を実行中であっても、その処理は中断され、アラーム・ハンドラに制御が移ります。

10.7.1 アラーム・ハンドラの基本形

以下に、アラーム・ハンドラを記述する場合の基本型を示します。*exinf*には、アラーム・ハンドラ生成時に指定した拡張情報が渡されます。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma almhandler Almhdr1             /* 備考参照 */
void Almhdr1 ( VP_INT exinf );         /* 備考参照 */
void Almhdr1 ( VP_INT exinf )
{
    .....
    .....

    return;                             /* アラーム・ハンドラの終了 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したアラーム・ハンドラについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

10.7.2 アラーム・ハンドラ内での処理

- スタック
アラーム・ハンドラは、システム・スタックを使用します。
- サービス・コールの発行
RI600PX では、アラーム・ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。
アラーム・ハンドラでは、“発行有効範囲”が“非タスク”のサービスコールを発行可能です。

備考 RI600PX では、アラーム・ハンドラ内の処理を高速に終了させる目的から、アラーム・ハンドラ内の処理が完了するまでの間、スケジューラの起動を遅延しています。したがって、アラーム・ハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（`isig_sem`、`iset_flg` など）が発行された際には、キュー操作などといった処理が行われるだけであり、実際のディスパッチ処理の実行は“アラーム・ハンドラからの復帰命令（return 命令の発行）”が発行されるまで遅延され、一括して行うようにしています。

- 処理開始時の PSW

表 10 - 2 アラーム・ハンドラ処理開始時の PSW

ビット	値	備考
I	1	
IPL	基本クロック割り込み優先レベル (IPL)	処理開始時より下げたはなりません。
PM	0	スーパーバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

10.7.3 アラーム・ハンドラの生成

アラーム・ハンドラは、以下のいずれかの方法で生成します。

- 1) システム・コンフィギュレーション・ファイルによる生成
システム・コンフィギュレーション・ファイルで静的 API "alarm_hand[]" を使用してアラーム・ハンドラを生成します。
静的 API "alarm_hand[]" の詳細は、「[20.20 アラーム・ハンドラ情報 \(alarm_hand\[\]\)](#)」を参照してください。
- 2) `cre_alm` または `acre_alm` サービスコールによる生成
`cre_alm` は、パラメータ `pk_calm` が指す領域に設定されたアラーム・ハンドラ生成情報にしたがって、パラメータ `almid` で指定されたアラーム・ハンドラ ID のタスクを生成します。
`acre_alm` は、パラメータ `pk_calm` が指す領域に設定されたアラーム・ハンドラ生成情報にしたがってアラーム・ハンドラ生成し、生成されたアラーム・ハンドラ ID を返します。
生成されたアラーム・ハンドラは、停止状態 (STA 状態) となります。
指定するアラーム・ハンドラ生成情報は、以下の通りです。
 - アラーム・ハンドラ属性 (`almatr`)
TA_HLNG のみを指定できます。
 - 拡張情報 (`exinf`)
 - アラーム・ハンドラの開始アドレス (`almhdr`)

これらのサービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、代表として `acre_alm` の記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */

extern void Almhdr1 (VP_INT exinf);

#pragma task Task1                      /* 備考参照 */
void Task1 ( VP_INT exinf );            /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      almid;                       /* 変数の宣言 */
    T_CALM  pk_calm = {                 /* 変数の宣言, 初期化 */
        TA_HLNG,                       /* アラーム・ハンドラ属性 (almatr) */
        0,                               /* 拡張情報 (exinf) */
        (FP) Almhdr1                    /* アラーム・ハンドラの開始アドレス (almhdr) */
    };
    .....
    almid = acre_alm ( &pk_calm );      /* アラーム・ハンドラの生成 */
    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.7.4 アラーム・ハンドラの削除

- del_alm

パラメータ *almid* で指定されたアラーム・ハンドラを削除します。

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクだけが呼び出せます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考参照 */
void Task1 ( VP_INT exinf );     /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ID      almid = 8;           /* 変数の宣言, 初期化 */

    .....
    .....

    del_alm( almid );           /* アラーム・ハンドラの削除 */
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.7.5 アラーム・ハンドラの動作開始

アラーム・ハンドラの動作開始は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `sta_alm`, `ista_alm`

`almid` で指定されたアラーム・ハンドラの起動時刻を `almtim` ミリ秒後に設定し、動作状態（STA 状態）にします。これにより、対象アラーム・ハンドラは、RI600PX の起動対象となります。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 3 参照 */
void Task1 ( VP_INT exinf );          /* 備考 3 参照 */
void Task1 ( VP_INT exinf )
{
    ID      almid = 1;                /* 変数の宣言, 初期化 */

    .....

    sta_alm ( almid );                /* アラーム・ハンドラの動作開始 */

    .....
}
```

備考 1 `almtim` に 0 を指定すると、次の基本クロック割り込み時にアラーム・ハンドラが起動されます。

備考 2 対象アラーム・ハンドラがすでに動作状態の場合でも、本サービス・コールは起動時刻を再設定します。以前の起動時刻の設定は無効となります。

備考 3 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.7.6 アラーム・ハンドラの動作停止

アラーム・ハンドラの動作停止は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `stp_alm`, `istp_alm`

パラメータ `almid` で指定されたアラーム・ハンドラの動作状態を動作状態 (STA 状態) から停止状態 (STP 状態) へと遷移させます。これにより、本サービス・コールの発行から `sta_alm` または `ista_alm` が発行されるまでの間、対象アラーム・ハンドラは、RI600PX の起動対象から除外されます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"             /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                      /* 備考 2 参照 */
void Task1 ( VP_INT exinf );           /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      almid = 1;                  /* 変数の宣言, 初期化 */

    .....
    .....

    stp_alm ( almid );                  /* 周期ハンドラの動作停止 */

    .....
    .....
}
```

備考 1 本サービス・コールでは、停止要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、対象アラーム・ハンドラの動作状態が停止状態 (STP 状態) へと遷移していた場合には、何も処理は行わず、エラーとしても扱いません。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.7.7 アラーム・ハンドラ詳細情報の参照

アラーム・ハンドラ詳細情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_alm`, `iref_alm`

パラメータ `almid` で指定されたアラーム・ハンドラの周期ハンドラ詳細情報（現在状態、残り時間）をパラメータ `pk_ralm` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考2 参照 */
void Task1 ( VP_INT exinf );          /* 備考2 参照 */
void Task1 ( VP_INT exinf )
{
    ID      almid = 1;                 /* 変数の宣言、初期化 */
    T_RALM  pk_ralm;                  /* データ構造体の宣言 */
    STAT    almstat;                  /* 変数の宣言 */
    RELTIM  lefttim;                  /* 変数の宣言 */

    .....
    .....

    ref_alm ( almid, &pk_ralm );      /* 周期ハンドラ詳細情報の参照 */

    almstat = pk_ralm.almstat;        /* 現在状態の獲得 */
    lefttim = pk_ralm.lefttim;        /* 残り時間の獲得 */

    .....
    .....
}
```

備考1 アラーム・ハンドラ詳細情報 T_RALM についての詳細は、「[【アラーム・ハンドラ詳細情報 T_RALM の構造】](#)」を参照してください。

備考2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.8 システム時刻

10.8.1 システム時刻の設定

システム時刻の設定は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

なお、システム時刻を変更しても、それ以前に行われた時間管理要求（タスクのタイムアウト、`dly_tsk`によるタスクの遅延、周期ハンドラ、およびアラーム・ハンドラ）が発生する実時刻は変化しません。

- `set_tim`, `iset_tim`

RI600PX のシステム時刻（単位：ミリ秒）をパラメータ `p_sysstim` で指定された時間に変更します。

以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    SYSTEMIM p_sysstim;                /* データ構造体の宣言 */

    p_sysstim.ltime = 3600;            /* データ構造体の初期化 */
    p_sysstim.uptime = 0;              /* データ構造体の初期化 */

    .....
    .....

    set_tim ( &p_sysstim );            /* システム時刻の設定 */

    .....
    .....
}

```

備考 1 システム時刻情報 `SYSTEMIM` についての詳細は、「【システム時刻情報 `SYSTEMIM` の構造】」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.8.2 システム時刻の参照

システム時刻の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `get_tim`, `iget_tim`

RI600PX のシステム時刻（単位：ミリ秒）をパラメータ `p_systim` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    SYSTIM  p_systim;            /* データ構造体の宣言 */
    UW      ltime;               /* 変数の宣言 */
    UH      utime;               /* 変数の宣言 */

    .....
    .....

    get_tim ( &p_systim );       /* システム時刻の参照 */

    ltime = p_systim.ltime;      /* システム時刻（下位 32 ビット）の獲得 */
    utime = p_systim.utime;      /* システム時刻（上位 16 ビット）の獲得 */

    .....
    .....
}
```

備考 1 システム時刻情報 SYSTIM についての詳細は、「[【システム時刻情報 SYSTIM の構造】](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

10.9 基本クロック用タイマ初期化ルーチン（_RI_init_cmt_knl()）

10.9.1 ユーザ・OWN・コーディング部

基本クロック用タイマ初期化ルーチンは、ユーザ・OWN・コーディング部として実装する必要があります。

備考 RI600PXで提供するサンプルの基本クロック用タイマ初期化ルーチンのソース・ファイルは“init_cmt.c”です。

- 基本クロック用タイマ初期化ルーチンの基本型
以下に、基本クロック用タイマ初期化ルーチンの基本型を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */

#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
#include      "ri_cmt.h"
#endif

// Timer initialize call-back
// Timer initialize call-back
void _RI_init_cmt_knl(void);
void _RI_init_cmt_knl(void)
{
#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
    _RI_init_cmt();
#endif
}
```

備考 基本クロック用タイマ初期化ルーチンの関数名は“_RI_init_cmt_knl”です。

- 解説
基本クロック用タイマ初期化ルーチンは、[vsta_knl](#)、[ivsta_knl](#) から呼び出されます。
_RI_CLOCK_TIMER は、[cfg600px](#) が [kernel_id.h](#) に出力するマクロです。_RI_CLOCK_TIMER の定義値は、システム・コンフィギュレーション・ファイルの[基本クロック用タイマ・チャンネルの選択 \(timer\)](#) にしたがって、以下のようになります。

clock.timer 設定	_RI_CLOCK_TIMER 定義値
“CMT0”	0
“CMT1”	1
“CMT2”	2
“CMT3”	3
“OTHER”	0x7FFFFFFF
“NOTIMER”	-1

clock.timer に“CMT0”、“CMT1”、“CMT2”、または“CMT3”を指定した場合、[cfg600px](#) は [ri_cmt.h](#) に基本タイマ用初期化用のインライン関数“void _RI_init_cmt(void)”を出力するので、_RI_init_cmt_knl() はこれ呼び出すように実装してください。

clock.timer に“OTHER”を指定した場合は、アプリケーションで _RI_init_cmt_knl() を実装してください。

clock.timer に “NOTIMER” を指定した場合は、_RI_init_cmt_knl() は何もせずに終了するように実装してください。

- スタック
基本クロック用タイマ初期化ルーチンは、システム・スタックを使用します。
- サービス・コールの発行
基本クロック用タイマ初期化ルーチンでは、“発行有効範囲”が“非タスク”のサービスコールを発行可能です。
- 処理開始時の PSW

表 10 - 3 基本クロック用タイマ初期化ルーチン処理開始時の PSW

ビット	値	備考
I	1	
IPL	カーネル割り込みマスクレベル (system_IPL)	処理開始時より下げてはなりません。
PM	0	スーパーバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

第 11 章 システム状態管理機能

本章では、RI600PX が提供しているシステム状態管理機能について解説しています。

11.1 概 要

RI600PX におけるシステム状態管理機能では、タスクの優先順位の回転、ディスパッチ禁止状態への遷移などといったシステムの状態を操作する機能のほかに、コンテキスト種別の参照、CPU ロック状態の参照などといったシステムの状態を参照する機能も提供しています。

なお、システム・ダウン (`vsys_dwn`, `ivsys_dwn`) については「[第 15 章 システム・ダウン](#)」、RI600PX の起動 (`vsta_knl`, `ivsta_knl`) については「[第 17 章 システム初期化処理](#)」を参照してください。

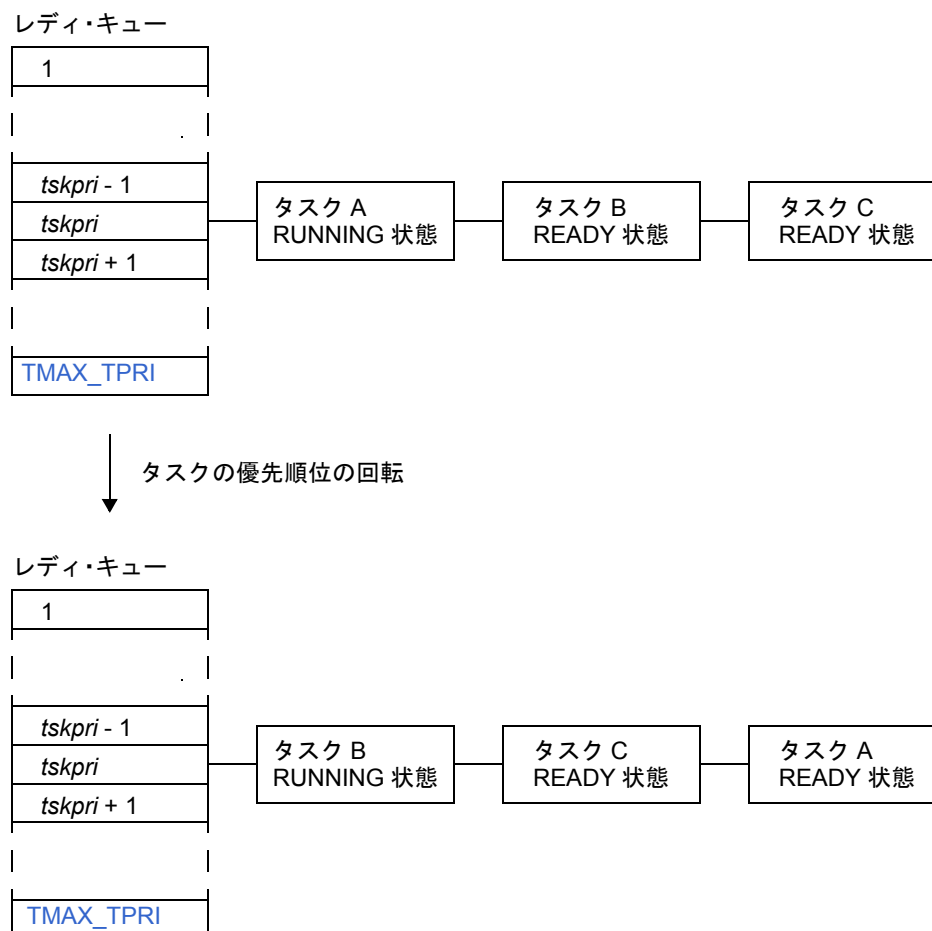
11.2 タスクの優先順位の回転

タスクの優先順位の回転は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `rot_rdq`, `irot_rdq`

パラメータ `tskpri` で指定された優先度に対応したレディ・キューの先頭タスクを最後尾につなぎかえ、タスクの実行順序を明示的に変更します。

図 11 - 1 タスク優先順位の回転



以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma cychandler Cychdr1       /* 備考 5 参照 */
void Cychdr1 ( VP_INT exinf );    /* 備考 5 参照 */
void Cychdr1 ( VP_INT exinf );    /* 周期ハンドラ */
{
    PRI      tskpri = 8;          /* 変数の宣言, 初期化 */

    .....

    irot_rdq ( tskpri );         /* タスクの優先順位の回転 */

    .....

    return;                      /* 周期ハンドラの終了 */
}

```

- 備考 1 本サービス・コールでは、レディ・キューの対象優先度にタスクが 1 つもキューイングされていなかった場合には、何も処理は行わず、エラーとしても扱いません。
- 備考 2 本サービス・コールを周期ハンドラなどから一定周期で発行することにより、ラウンドロビン・スケジューリングを実現することができます。
- 備考 3 RI600PX におけるレディ・キューは、優先度をキーとしたハッシュ・テーブルであり、実行可能な状態 (RUNNING 状態または READY 状態) へと遷移したタスクが FIFO 順でキューイングされます。このため、スケジューラは、起動された際にレディ・キューの優先度高位から検出処理を実行し、キューイングされているタスクを検出した場合には、該当優先度の先頭タスクに CPU の利用権を与えることにより、RI600PX のスケジューリング方式 (優先度方式, FCFS 方式) を実現しています。
- 備考 4 `tskpri` に `TPRI_SELF` を指定すると、自タスクのベース優先度を対象とします。ミューテックスをロック中のタスクの現在優先度は、ベース優先度と異なる場合があります。この場合、そのタスクが本サービス・コールで `tskpri` に `TPRI_SELF` を指定しても、自タスクが属する現在優先度のレディ・キューを回転することはできません。
- 備考 5 システム・コンフィギュレーション・ファイルで生成した周期ハンドラについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

11.3 RUNNING 状態のタスクの参照

RUNNING 状態のタスクの参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- [get_tid](#), [iget_tid](#)

RUNNING 状態のタスクの ID をパラメータ *p_tskid* で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */

void Inthdr ( VP_INT exinf );     /* 割り込みハンドラ */
{
    ID      p_tskid;              /* 変数の宣言 */

    .....
    .....

    iget_tid ( &p_tskid );        /* RUNNING 状態のタスクの参照 */

    .....
    .....

    return;                       /* 割り込みハンドラの終了 */
}
```

備考 1 本サービス・コールでは、RUNNING 状態へと遷移しているタスクが存在しなかった場合には、パラメータ *p_tskid* で指定された領域に [TSK_NONE](#) を格納します。

11.4 CPU ロック状態への移行と解除

CPU ロック状態では、タスクのスケジューリングは禁止され、カーネル管理割り込みもマスクされます。つまり、カーネル管理外割り込みハンドラを除くすべての処理プログラムに対して、排他的に処理を行うことができます。

以下のサービス・コールによって、CPU ロック状態へ移行します。

- `loc_cpu`, `iloc_cpu`

システムを CPU ロック状態へ移行させます。

CPU ロック状態では、発行可能なサービス・コールは以下に制限されます。

発行可能なサービス・コール	機能概要
<code>ext_tsk</code>	自タスクの終了 (CPU ロック状態は解除されます)
<code>exd_tsk</code>	自タスクの終了と削除 (CPU ロック状態は解除されます)
<code>sns_tex</code>	タスク例外禁止状態の参照
<code>loc_cpu</code> , <code>iloc_cpu</code>	CPU ロック状態への移行
<code>unl_cpu</code> , <code>iunl_cpu</code>	CPU ロックの解除
<code>sns_loc</code>	CPU ロック状態の参照
<code>sns_dsp</code>	ディスパッチ禁止状態の参照
<code>sns_ctx</code>	コンテキスト種別の参照
<code>sns_dpn</code>	ディスパッチ保留状態の参照
<code>vsys_dwn</code> , <code>ivsys_dwn</code>	システム・ダウン

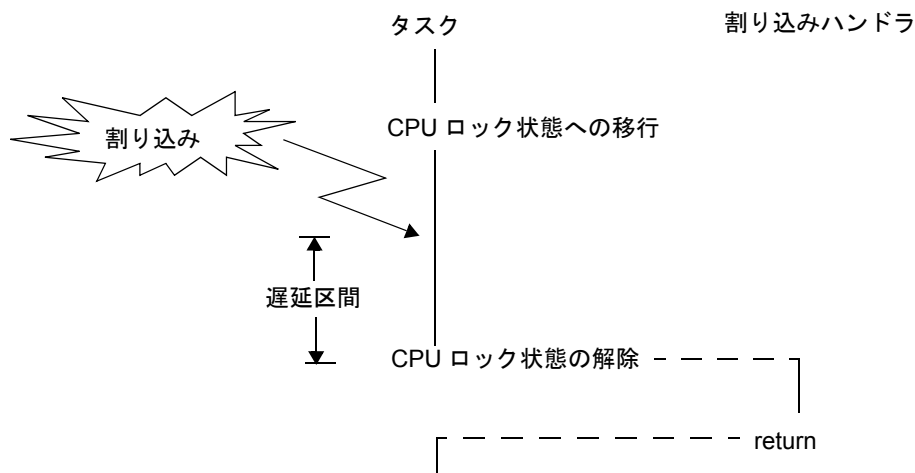
CPU ロック状態は、以下のサービス・コールと、`ext_tsk`、および `exd_tsk` によって解除されます。

- `unl_cpu`, `iunl_cpu`

システムを CPU ロック解除状態へ移行させます。

以下に、“CPU ロック状態” を利用した際の処理の流れを示します。

図 11 - 2 CPU ロック状態



以下に、CPU ロック状態への移行とその解除の記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考7参照 */
void Task1 ( VP_INT exinf );     /* 備考7参照 */
void Task1 ( VP_INT exinf )
{
    .....
    .....

    loc_cpu ( );                 /*CPU ロック状態への移行 */

    .....                       /*CPU ロック状態 */
    .....

    unl_cpu ( );                 /*CPU ロック状態の解除 */

    .....
    .....
}

```

- 備考1 CPU ロック状態の解除は、`loc_cpu` または `iloc_cpu` を発行した処理プログラムが終了する以前に行う必要があります。
- 備考2 `loc_cpu`, `iloc_cpu` では、ロック要求のキューイングが行われません。このため、すでに CPU ロック状態の場合には、何も処理は行わず、エラーとしても扱いません。
- 備考3 `unl_cpu`, `iunl_cpu` では、解除要求のキューイングが行われません。このため、すでに非 CPU ロック状態の場合には、何も処理は行わず、エラーとしても扱いません。
- 備考4 `unl_cpu`, `iunl_cpu` では、`dis_dsp` によるディスパッチ禁止状態は解除されません。
- 備考5 CPU ロック状態の期間は基本クロック用タイマ割り込みはマスクされます。このため、CPU ロック状態の期間が長くなると、[時間管理機能](#)で扱う時間が遅れる場合があります。
- 備考6 “カーネル管理割り込み”については、「[12.1 割り込みの種類](#)」を参照してください。
- 備考7 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

11.5 CPU ロック状態の参照

複数のタスクやハンドラから呼ばれる関数では、現在のCPUロック状態を参照したい場合があります。このような場合には、`sns_loc`を使用します。

- `sns_loc`

本サービス・コールは、CPUロック状態か否かを調べます。本サービス・コールは、戻り値として、CPUロック状態の場合はTRUE、CPUロック解除状態の場合はFALSEを返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /*cfg600px が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ercd;                    /* 変数の宣言 */

    .....
    .....

    ercd = sns_loc ( );          /*CPU ロック状態の参照 */

    if ( ercd == TRUE ) {
        .....                  /*CPU ロック状態 */
    } else if ( ercd == FALSE ) {
        .....                  /* 非 CPU ロック状態 */
        .....
    }

    .....
    .....
}
```

11.6 ディスパッチ禁止状態への移行と解除

ディスパッチ禁止状態では、タスクのスケジューリングが禁止されるため、他のタスクに対して排他的に処理を行うことができます。

以下のサービス・コールによって、ディスパッチ禁止状態へ移行します。また、`chg_ims` によって割り込みマスク (PSW.IPL) を 0 以外に変更したときにも、ディスパッチ禁止状態へ移行します。

- `dis_dsp`

システムをディスパッチ禁止状態へ移行させます。

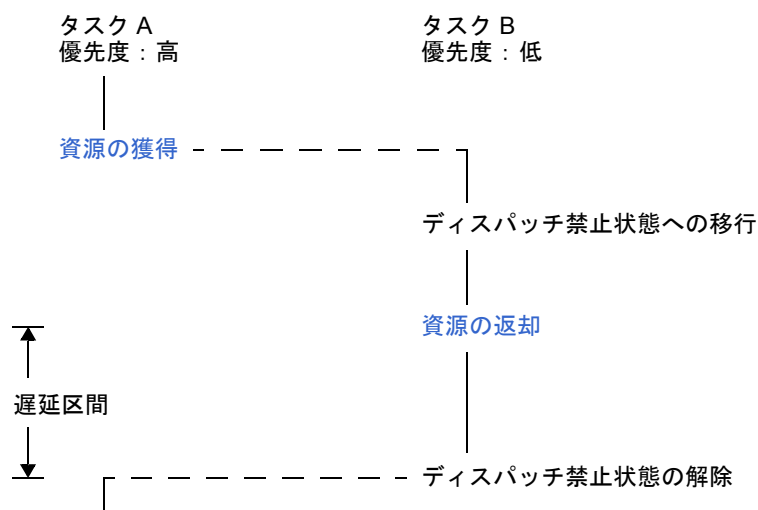
ディスパッチ禁止状態は、以下のサービス・コール、`ext_tsk`、`exd_tsk`、および `chg_ims` によって割り込みマスク (PSW.IPL) を 0 に変更したときに解除されます。

- `ena_dsp`

システムをディスパッチ許可状態へ移行させます。

以下に、“ディスパッチ禁止状態”を利用した際の処理の流れを示します。

図 11 - 3 ディスパッチ禁止状態



以下に、本サービス・コールの記述例を示します。

```

#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 5 参照 */
void Task1 ( VP_INT exinf );      /* 備考 5 参照 */
void Task1 ( VP_INT exinf )
{
    .....
    .....

    dis_dsp ( );                  /* ディスパッチ禁止状態への移行 */

    .....                        /* ディスパッチ禁止状態 */
    .....

    ena_dsp ( );                  /* ディスパッチ禁止状態の解除 */

    .....
    .....
}

```

- 備考 1 ディスパッチ禁止状態の解除は、`dis_dsp` を発行したタスクが DORMANT 状態へ遷移する以前に行う必要があります。
- 備考 2 `dis_dsp` では、禁止要求のキューイングが行われません。このため、すでにディスパッチ禁止状態の場合には、何も処理は行わず、エラーとしても扱いません。
- 備考 3 `ena_dsp` では、許可要求のキューイングが行われません。このため、すでにディスパッチ許可状態の場合には、何も処理は行わず、エラーとしても扱いません。
- 備考 4 ディスパッチ禁止状態の間に“自タスクを状態遷移させる可能性のあるサービス・コール(`wai_sem`, `wai_flg` など)”を発行した場合には、要求条件の即時成立／不成立を問わず、戻り値として E_CTX を返します。
- 備考 5 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

11.7 ディスパッチ禁止状態の参照

複数のタスクから呼ばれる関数では、現在のディスパッチ禁止状態を参照したい場合があります。このような場合には、`sns_dsp` を使用します。

- `sns_dsp`

本サービス・コールは、ディスパッチ禁止状態か否かを調べます。本サービス・コールは、戻り値として、ディスパッチ禁止状態の場合は TRUE、ディスパッチ許可状態の場合は FALSE を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ercd;                    /* 変数の宣言 */

    .....

    ercd = sns_dsp ( );           /* ディスパッチ禁止状態の参照 */

    if ( ercd == TRUE ) {
        .....                    /* ディスパッチ禁止状態 */
    } else if ( ercd == FALSE ) {
        .....                    /* ディスパッチ許可状態 */
    }

    .....
}
```

11.8 コンテキスト種別の参照

複数のタスクやハンドラから呼ばれる関数では、現在のコンテキスト種別を参照したい場合があります。このような場合には、`sns_ctx` を使用します。

- `sns_ctx`

本サービス・コールは、本サービス・コールを発行した処理プログラムのコンテキスト種別を調べます。本サービス・コールは、戻り値として、非タスク・コンテキストの場合は TRUE、タスク・コンテキストの場合は FALSE を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* *cfg600px が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ercd;                    /* 変数の宣言 */

    .....

    ercd = sns_ctx ( );           /* コンテキスト種別の参照 */

    if ( ercd == TRUE ) {
        .....                    /* 非タスク・コンテキスト処理 */
    } else if ( ercd == FALSE ) {
        .....                    /* タスク・コンテキスト処理 */
    }

    .....
}
```

11.9 ディスパッチ保留状態の参照

以下のいずれかの条件を満たすときをディスパッチ保留状態と呼びます。

- ディスパッチ禁止状態
- CPU ロック状態
- ハンドラなど、PSW.IPL>0 の状態

複数のタスクやハンドラから呼ばれる関数では、現在のディスパッチ保留状態を参照したい場合があります。このような場合には、[sns_dpn](#) を使用します。

- [sns_dpn](#)

本サービス・コールは、ディスパッチ保留状態か否かを調べます。本サービス・コールは、戻り値として、ディスパッチ保留状態の場合は TRUE、ディスパッチ保留状態でない場合は FALSE を返します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */

void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ercd;                    /* 変数の宣言 */

    .....
    .....

    ercd = sns_dpn ( );           /* ディスパッチ保留状態の参照 */

    if ( ercd == TRUE ) {
        .....                    /* ディスパッチ保留状態 */
        .....
    } else if ( ercd == FALSE ) {
        .....                    /* 非ディスパッチ保留状態 */
        .....
    }

    .....
    .....
}
```

第12章 割り込み管理機能

本章では、RI600PX が提供している割り込み管理機能について解説しています。

12.1 割り込みの種類

割り込みは、カーネル管理割り込みとカーネル管理外割り込みに分類されます。

- カーネル管理割り込み

カーネル割込マスクレベルより割り込み優先レベルが低い割り込みをカーネル管理割り込みといいます。

カーネル管理割り込みハンドラでは、サービス・コールを呼び出すことができます。

サービス・コール処理中にカーネル管理割り込みが発生した場合、カーネル管理割り込みを受け付け可能となるまで割り込み受理が遅延されます。

- カーネル管理外割り込み

カーネル割込マスクレベルより割り込み優先レベルが高い割り込みをカーネル管理外割り込みといいます。ノンマスク割込割り込みは、カーネル管理外割り込みの扱いとなります。

カーネル管理外割り込みハンドラでは、サービス・コールを呼び出してはなりません。

サービス・コール処理中にカーネル管理外割り込みが発生した場合でも、直ちに割り込み受理が受理されるため、カーネル処理に依存しない高速な割り込み応答が可能です。

備考 カーネル割り込みマスクレベルは、システム情報 (system) のカーネル割り込みマスクレベル (system_IPL) で定義します。

12.2 RX MCU の高速割り込み

RX MCU は「高速割り込み」機能をサポートしています。ひとつの割り込み要因だけを高速割り込みとすることができます。高速割り込みは、割り込み優先レベル 15 として扱われます。高速割り込みを使用する場合は、割り込み優先レベル 15 の割り込み要因をひとつに限定する必要があります。

RI600PX で高速割り込みを使用する場合は、その割り込みはカーネル管理外割り込みとして扱う必要があります。つまり、カーネル割り込みマスクレベル (system.system_IPL) は、14 以下に設定する必要があります。

システム・コンフィギュレーション・ファイルで高速割り込みを定義する際には、os_int に NO を指定し、さらに pragma_switch に F を指定する必要があります。

また、ブート処理関数 (PowerON_Reset_PC()) で RX-MCU の FINTV レジスタをそのハンドラの開始アドレスに初期化する必要があります。

12.3 CPU 例外の扱い

以下の CPU 例外はカーネル管理外割り込みの扱いとなります。

- 無条件トラップ (INT, BRK 命令)

なお、INT #1 ~ #8 は RI600PX で予約されています。

- 未定義命令例外

- 特権命令例外

- 浮動小数点例外

一方、アクセス例外ハンドラはカーネル管理割り込みの扱いとなります。

12.4 基本クロック用タイマ割り込み

[時間管理機能](#)は、一定周期で発生する基本クロック用タイマ割り込みを利用して実現されています。基本クロック用タイマ割り込みが発生した際には、RI600PX が提供している時間管理用割り込みハンドラが起動し、時間に関連した処理（システム時刻の更新、タスクの遅延起床／タイムアウト、周期ハンドラの起動など）が実行されます。

12.5 多重割り込み

RI600PX では、割り込みハンドラ内で再び割り込みが発生することを“多重割り込み”と呼んでいます。

多重割り込みを許可するかどうかは、可変ベクタの割り込みハンドラごとに設定できます。詳細は、「[20.21 可変ベクタ情報 \(interrupt_vector\[\]\)](#)」を参照してください。

12.6 割り込みハンドラ

割り込みハンドラは、割り込みが発生した際に起動される割り込み処理専用ルーチンです。

なお、RI600PXでは、割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため、割り込みが発生した際には、システム内で最高優先度を持つタスクが処理を実行中であっても、その処理は中断され、割り込みハンドラに制御が移ります。

12.6.1 割り込みハンドラの基本型

以下に、割り込みハンドラを記述する場合の基本型を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */

void Inthdr1 ( void )
{
    .....
    .....

    return;                            /* 割り込みハンドラの終了 */
}
```

備考 ハンドラ関数のプロトタイプ宣言および `#pragma interrupt` ディレクティブは、`cfg600px` が `kernel_id.h` に出力します。

- スタック

割り込みハンドラは、システム・スタックを使用します。

- サービス・コールの発行

RI600PXでは、割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。

カーネル管理割り込みハンドラでは、“発行有効範囲”が“非タスク”のサービスコールを発行可能です。なお、カーネル管理外割り込みハンドラでは、サービス・コールを呼び出してはなりません。

備考 RI600PXでは、カーネル管理割り込みハンドラ内の処理を高速に終了させる目的から、カーネル管理割り込みハンドラ内の処理が完了するまでの間、スケジューラの起動を遅延しています。したがって、カーネル管理割り込みハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（`isig_sem`、`iset_flg` など）が発行された際には、キュー操作などといった処理が行われるだけであり、実際のディスパッチ処理の実行はカーネル管理割り込みハンドラが終了するまで遅延され、一括して行うようにしています。

- 処理開始時の PSW

表 12-1 割り込みハンドラ処理開始時の PSW

ビット	値	備考
I	登録時に pragma_switch に “E” を指定した場合は 1, そうでない場合は 0	
IPL	割り込み：当該割り込み優先レベル CPU 例外：例外発生前と同じ	処理開始時より下げてはなりません。
PM	0	スーパーバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

12.6.2 割り込みハンドラの登録

RI600PX では、割り込みハンドラの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

割り込みハンドラの静的登録とは、システム・コンフィギュレーション・ファイルで静的 API “interrupt_vector[]” (可変ベクタ割り込みハンドラの登録)、および “interrupt_fvector[]” (固定ベクタ/例外ベクタ割り込みハンドラの登録) を使用して割り込みハンドラを定義することをいいます。

静的 API “interrupt_vector[]” の詳細は、「[20.21 可変ベクタ情報 \(interrupt_vector\[\]\)](#)」を、静的 API “interrupt_fvector[]” の詳細は、「[20.22 固定ベクタ/例外ベクタ情報 \(interrupt_fvector\[\]\)](#)」を参照してください。

12.7 処理プログラム内におけるマスカブル割り込み受け付け状態

RX MCU のマスカブル割り込みの受け付け状態は、PSW.I, PSW.IPL の値で変わります。詳細はハードウェアのマニュアルを参照してください。

初期状態は処理プログラムごとに決まっています。その詳細は、[表 12-2](#) を参照してください。

表 12-2 処理プログラム起動時のマスカブル割り込み受け付け状態

処理プログラム名	PSW.I	PSW.IPL
タスク	1	0
タスク例外処理ルーチン	1	タスク例外処理ルーチン起動直前のタスク本体と同じ
周期ハンドラ, アラーム・ハンドラ	1	基本クロック割り込み優先レベル (IPL)
割り込みハンドラ	登録時に pragma_switch に “E” を指定した場合は 1, そうでない場合は 0	割り込み：当該割り込み優先レベル CPU 例外：例外発生前と同じ

12.8 マスカブル割り込みの受け付け禁止

マスカブル割り込みの受け付け禁止には、以下の方法があります。

- `loc_cpu`, `iloc_cpu` を使用して CPU ロック状態にする
- `chg_ims`, `ichg_ims` を使用して PSW.IPL を変更する
- PSW.I, PSW.IPL を直接変更する (ハンドラ限定)

12.8.1 `loc_cpu`, `iloc_cpu` を使用して CPU ロック状態にする

CPU ロック状態では、PSW.IPL はカーネル割り込みマスクレベル (`system_IPL`) となります。つまり、CPU ロック状態で禁止される割り込みは、カーネル管理割り込みのみです。

また、CPU ロック状態では使用可能なサービス・コールに制限があります。詳細は、「11.4 CPU ロック状態への移行と解除」を参照してください。

12.8.2 `chg_ims`, `ichg_ims` を使用して PSW.IPL を変更する

`chg_ims`, `ichg_ims` では、PSW.IPL を任意の値に変更することができます。

タスクで `chg_ims` を用いて PSW.IPL を 0 以外に変更すると、同時にディスパッチ禁止状態に移行し、`chg_ims` を用いて PSW.IPL を 0 に戻すと、同時にディスパッチ許可状態に戻ります。

タスクが `chg_ims` によって PSW.IPL を 0 以外に変更している間は、`ena_dsp` を呼び出さないようにしてください。`ena_dsp` を呼び出すと、その時点でディスパッチ許可状態に遷移します。タスク・ディスパッチが発生すると、PSW はディスパッチ先のタスクの状態に更新されるので、意図せずに IPL 値が下がってしまうことがあります。

ハンドラでは、PSW.IPL を処理開始時より下げてはなりません。

12.8.3 PSW.I, PSW.IPL を直接変更する (ハンドラ限定)

ハンドラでは、PSW.I, PSW.IPL を直接変更することができます。この方法は、`ichg_ims` よりも高速です。

ハンドラでは、PSW.IPL を処理開始時より下げてはなりません。

なお、コンパイラでは PSW を操作する以下の組み込み関数を用意しています。組み込み関数の詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX コーディング編」を参照してください。

- `set_ipl()` : PSW レジスタの IPL ビットの変更
- `get_ipl()` : PSW レジスタの IPL ビットの参照
- `set_psw()` : PSW レジスタの設定
- `get_psw()` : PSW レジスタの参照

第13章 システム構成管理機能

本章では、RI600PX が提供しているシステム構成管理機能について解説しています。

13.1 概要

RI600PX におけるシステム構成管理機能では、バージョン情報を参照する機能を提供しています。

13.2 バージョン情報の参照

バージョン情報の参照は、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `ref_ver`, `iref_ver`

バージョン情報をパラメータ `pk_rver` で指定された領域に格納します。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    T_RVER  pk_rver;                  /* データ構造体の宣言 */
    UH      maker;                   /* 変数の宣言 */
    UH      prid;                    /* 変数の宣言 */

    .....
    .....

    ref_ver ( &pk_rver );            /* バージョン情報の参照 */

    maker = pk_rver.maker;           /* メーカーコードの獲得 */
    prid = pk_rver.prid;             /* カーネル識別番号の獲得 */

    .....
    .....
}
```

備考 1 バージョン情報 T_RVER についての詳細は、「[【バージョン情報 T_RVER の構造】](#)」を参照してください。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

第14章 オブジェクト・リセット機能

本章では、RI600PX が提供しているオブジェクト・リセット機能について解説しています。

14.1 概 要

オブジェクト・リセット機能は、データ・キュー、メールボックス、メッセージ・バッファ、固定長メモリ・プール、および可変長メモリ・プールを初期状態に戻す機能で、 μ ITRON4.0 仕様外の機能です。

14.2 データ・キューのリセット

データ・キューのリセットは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- vrst_dtq

パラメータ *dtqid* で指定されたデータ・キューをリセットします。

データ・キューに蓄えられていたデータは破棄されます。また、データ送信を待っていたタスクの待ち状態は解除され、そのタスクには戻り値として EV_RST が返されます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考2 参照 */
void Task1 ( VP_INT exinf );          /* 備考2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                      /* 変数の宣言 */
    ID      dtqid = 1;                /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = vrst_dtq ( dtqid );        /* データ・キューのリセット */

    .....
    .....
}
```

備考1 本サービス・コールでは、データ受信を待っていたタスクの待ち状態は解除されません。

備考2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは *cfg600px* が *kernel_id.h* に出力するため、記述不要です。

14.3 メールボックスのリセット

メールボックスのリセットは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `vrst_mbx`

パラメータ `mbxid` で指定されたメールボックスをリセットします。

メールボックスに蓄えられたメッセージは、RI600PX の管理から外れます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考 2 参照 */
void Task1 ( VP_INT exinf );     /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      mbxid = 1;           /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = vrst_mbx( mbxid );    /* メールボックスのリセット */

    .....
    .....
}
```

備考 1 本サービス・コールでは、メッセージ受信を待っていたタスクの待ち状態は解除されません。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

14.4 メッセージ・バッファのリセット

メッセージ・バッファのリセットは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `vrst_mbf`

パラメータ `mbfid` で指定されたメッセージ・バッファをリセットします。

メッセージ・バッファに蓄えられていたメッセージは破棄されます。また、メッセージ送信を待っていたタスクの待ち状態は解除され、そのタスクには戻り値として `EV_RST` が返されます。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"                /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"            /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1                    /* 備考 2 参照 */
void Task1 ( VP_INT exinf );          /* 備考 2 参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                      /* 変数の宣言 */
    ID      mbfid = 1;                 /* 変数の宣言, 初期化 */

    .....
    .....

    ercd = vrst_mbf( mbfid );         /* メッセージ・バッファのリセット */

    .....
    .....
}
```

備考 1 本サービス・コールでは、メッセージ受信を待っていたタスクの待ち状態は解除されません。

備考 2 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

14.5 固定長メモリ・プールのリセット

固定長メモリ・プールのリセットは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `vrst_mpf`

パラメータ `mpfid` で指定された固定長メモリ・プールをリセットします。

固定長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され、戻り値として `EV_RST` が返されます。

また、すでに獲得されていた固定長メモリ・ブロックはすべて、固定長メモリ・プールに返却されます。このため、本サービス・コール以降はそれらの固定長メモリ・ブロックにアクセスしてはなりません。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1              /* 備考参照 */
void Task1 ( VP_INT exinf );     /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      mpfid = 1;           /* 変数の宣言, 初期化 */

    .....

    ercd = vrst_mpf( mpfid );    /* 固定長メモリ・プールのリセット */

    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

14.6 可変長メモリ・プールのリセット

可変長メモリ・プールのリセットは、以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- `vrst_mpl`

パラメータ `mplid` で指定された可変長メモリ・プールをリセットします。

可変長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され、戻り値として `EV_RST` が返されます。

また、すでに獲得されていた可変長メモリ・ブロックはすべて、可変長メモリ・プールに返却されます。このため、本サービス・コール以降はそれらの可変長メモリ・ブロックにアクセスしてはなりません。

以下に、本サービス・コールの記述例を示します。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */
#pragma task Task1               /* 備考参照 */
void Task1 ( VP_INT exinf );      /* 備考参照 */
void Task1 ( VP_INT exinf )
{
    ER      ercd;                /* 変数の宣言 */
    ID      mplid = 1;           /* 変数の宣言, 初期化 */

    .....

    ercd = vrst_mpl( mplid );    /* 可変長メモリ・プールのリセット */

    .....
}
```

備考 システム・コンフィギュレーション・ファイルで生成したタスクについては、これらのステートメントは `cfg600px` が `kernel_id.h` に出力するため、記述不要です。

第15章 システム・ダウン

本章では、RI600PX が提供しているシステム・ダウン機能について解説しています。

15.1 概要

RI600PX の稼働中に回復不可能な事象が発生するとシステム・ダウンとなり、システム・ダウン・ルーチンが呼び出されます。

15.2 ユーザ・OWN・コーディング部

システム・ダウン・ルーチンは、ユーザ・OWN・コーディング部として実装する必要があります。

備考 RI600PX で提供するサンプルのシステム・ダウンのソース・ファイルは“sysdwn.c”です。

15.2.1 システム・ダウン・ルーチン (`_RI_sys_dwn__()`)

以下に、システム・ダウン・ルーチンを記述する場合の基本型を示します。システム・ダウン・ルーチンからリターンしてはなりません。

```
#include      "kernel.h"          /* 標準ヘッダ・ファイルの定義 */
#include      "kernel_id.h"       /* cfg600px が出力するヘッダ・ファイルの定義 */

/* プロトタイプ宣言 */
void _RI_sys_dwn__ ( W type, VW inf1, VW inf2, VW inf3 );

void _RI_sys_dwn__ ( W type, VW inf1, VW inf2, VW inf3 )
{
    .....
    .....

    while(1);
}
```

備考 システム・ダウン・ルーチンの関数名は“_RI_sys_dwn__”です。

- スタック
システム・ダウン・ルーチンは、システム・スタックを使用します。
- サービス・コールの発行
システム・ダウン・ルーチンからサービス・コールを呼び出してはなりません。

- 処理開始時の PSW

表 15 - 1 システム・ダウン・ルーチン処理開始時の PSW

ビット	値	備考
I	0	
IPL	- <i>type</i> < 0 の場合 : 不定 - <i>type</i> ≥ 0 の場合 : システム・ダウン発生前と同じ	処理開始時より下げてはなりません。
PM	0	スーパーバイザ・モード
U	0	システム・スタック
C, Z, S, O	不定	
その他	0	

15.2.2 システム・ダウン・ルーチンのパラメータ

- *type* == -1 (カーネル管理割り込みハンドラ終了時のエラー)

表 15 - 2 システム・ダウン・ルーチンのパラメータ (*type* == -1)

inf1	inf2	inf3	解説
E_CTX (-25)	2	不定	カーネル管理割り込みハンドラ終了時点で, PSW.PMが1(ユーザ・モード)である。
	3	不定	カーネル管理割り込みハンドラ終了時点で, PSW.IPL > カーネル割り込みマスクレベルである。
	5	不定	カーネル管理割り込みハンドラ終了時点で, CPU ロック状態である。
E_MACV (-26)	12	不定	割り込まれたタスクのスタック・ポインタが, ユーザ・スタック領域の範囲外を指している。

- *type* == -2 ([ext_tsk](#) のエラー)

表 15 - 3 システム・ダウン・ルーチンのパラメータ (*type* == -2)

inf1	inf2	inf3	解説
E_CTX (-25)	1	不定	非タスク・コンテキストから ext_tsk を呼び出した。
	4	不定	ext_tsk 呼び出し時点で, PSW.IPL > カーネル割り込みマスクレベルである。

- `type == -3` (組み込まれていないサービス・コールの呼び出し)

表 15 - 4 システム・ダウン・ルーチンのパラメータ (`type == -3`)

inf1	inf2	inf3	解説
E_NOSPT (-9)	不定	不定	組み込まれていないサービス・コールを呼び出した。

備考 「2.6.1 サービス・コール情報ファイルと“-ri600_preinit_mrc”コンパイラ・オプション」参照してください。

- `type == -4` (タスク例外処理ルーチン終了時のエラー)

表 15 - 5 システム・ダウン・ルーチンのパラメータ (`type == -4`)

inf1	inf2	inf3	解説
E_CTX (-25)	7	不定	タスク例外処理ルーチン終了時点で、PSW.IPL > カーネル割り込みマスクレベルである。
	8	不定	タスク例外処理ルーチン終了時点で、CPU ロック状態である。
	9	不定	タスク例外処理ルーチン終了時点で、非タスクコンテキストである。

- `type == -5` (`exd_tsk` のエラー)

表 15 - 6 システム・ダウン・ルーチンのパラメータ (`type == -5`)

inf1	inf2	inf3	解説
E_CTX (-25)	10	不定	<code>exd_tsk</code> 呼び出し時点で、PSW.IPL > カーネル割り込みマスクレベルである。
	11	不定	非タスク・コンテキストから <code>exd_tsk</code> を呼び出した。

- type == -6 (vsta_knl, ivsta_knl のエラー)

表 15 - 7 システム・ダウン・ルーチンのパラメータ (type == -6)

inf1	inf2	inf3	解説	
E_PAR (-17)	15	不定	メモリ・オブジェクト登録 (memory_object[]) に関するエラー	1) 先頭アドレスが 16 バイト境界でない。 2) acptn1,acptn2,acptn3 いずれかの bit15 に 1 がセットされている。 3) acptn1 == acptn2 == acptn3 == 0 4) acptn1,acptn2,acptn3 に最大ドメイン ID (VTMAX_DOMAIN) より大きな値のドメインに対応するビットがセットされている。 5) 先頭アドレス > 終端アドレス
E_OBJ (-41)		不定		先頭アドレスが同じメモリ・オブジェクトが複数定義されている。
E_OACV (-27)		不定		あるドメインにアクセス許可されているメモリ・オブジェクト数が 7 を超える。
E_PAR (-17)	16	不定	タスク生成 (task[]) に関するエラー	ユーザ・スタックの終端アドレス +1 が 16 バイト境界でない。

- type == -16 (未定義の可変ベクタ割り込み)

表 15 - 8 システム・ダウン・ルーチンのパラメータ (type == -16)

inf1	inf2	inf3
- cfg600px で -U オプションを指定しない場合 不定 - cfg600px で -U オプションを指定した場合 ベクタ番号	CPU の割り込み処理によってスタックに退避された PC	CPU の割り込み処理によってスタックに退避された PSW

- type == -17 (未定義の固定ベクタ/例外ベクタ割り込み)

表 15 - 9 システム・ダウン・ルーチンのパラメータ (type == -17)

inf1	inf2	inf3
- cfg600px で -U オプションを指定しない場合 不定 - cfg600px で -U オプションを指定した場合 ベクタ番号	CPU の割り込み処理によってスタックに退避された PC	CPU の割り込み処理によってスタックに退避された PSW

- *type* > 0 (アプリケーションからの *vsys_dwn*, *ivsys_dwn* の呼び出し)
0, および負の *type* 値は RI600PX 用に予約されています。アプリケーションから *vsys_dwn*, *ivsys_dwn* を呼び出す場合は、正の *type* 値を使用してください。

表 15 - 10 システム・ダウン・ルーチンのパラメータ (*type* > 0)

inf1	inf2	inf3
vsys_dwn, ivsys_dwn に指定した値		

第16章 スケジューリング機能

本章では、RI600PX が提供しているスケジューリング機能について解説しています。

16.1 概要

RI600PX におけるスケジューリング機能では、動的に変化していくタスクの状態を直接参照することにより、タスクの実行順序を管理/決定し、最適なタスクにCPUの利用権を与える機能を提供しています。

16.2 処理の単位と優先順位

アプリケーションは、以下の処理単位によって実行制御されます。

- タスク（タスク例外処理ルーチンを含む）
- 割り込みハンドラ
- 周期ハンドラ
- アラーム・ハンドラ
- アクセス例外ハンドラ

各処理単位は、以下の優先順位で処理されます。

- 1) 割り込みハンドラ、周期ハンドラ、アラーム・ハンドラ
- 2) アクセス例外ハンドラ
- 3) スケジューラ
- 4) タスク（タスク例外処理ルーチンを含む）

なお、スケジューラとは、実行するタスクを特定し、切り換える RI600PX の処理のことです。

割り込みハンドラ、周期ハンドラ、アラーム・ハンドラ、およびアクセス例外ハンドラはスケジューラよりも優先順位が高いため、これらが実行している間は、タスク（タスク例外処理ルーチンを含む）は実行されません（「[16.7 非タスク内におけるタスク・スケジューリング処理](#)」参照）。

割り込みハンドラは、割り込み優先レベルが高いほど優先順位が高くなります。

周期ハンドラおよびアラーム・ハンドラの優先順位は、基本クロック割り込みの優先レベルの割り込みハンドラと同じです。

タスク間の優先順位は、タスクに付与された現在優先度に従います。

16.3 タスクの駆動方式

RI600PX では、スケジューラの駆動方式として何らかの事象(きっかけ)が発生した際に起動する“[事象駆動方式](#)”を採用しています。

- 事象駆動方式

RI600PX における事象駆動方式では、以下に示した事象が発生した場合にスケジューラを起動し、タスクのスケジューリング処理を実行します。

- タスクの状態遷移を引き起こす可能性があるサービス・コールの発行
- 非タスク（周期ハンドラ、割り込みハンドラなど）からの復帰命令の発行
- [時間管理機能](#)を行う際に使用している基本クロック用タイマ割り込みの発生

16.4 タスクのスケジューリング方式

RI600PX では、タスクのスケジューリング方式として各タスクに定義されている優先度を利用した“優先度方式”，および、RI600PX のスケジューリング対象となつてからの経過時間を利用した“FCFS 方式”を採用しています。

- 優先度方式

RI600PX における優先度方式では、実行可能な状態（RUNNING 状態または READY 状態）にある全タスクの中から“最も高い現在優先度を持つタスク”を選び出し、CPU の利用権を与えます。

- FCFS 方式

“最も高い現在優先度を持つタスク”が複数存在する場合は、優先度方式だけではスケジューリング対象のタスクを決定することができません。この場合、RI600PX は FCFS 方式（First Come First Served 方式）でスケジューリング対象のタスクを決定します。具体的は、それらの中で最も早く実行可能な状態（READY 状態）へ遷移したタスクを選び出し、CPU の利用権を与えます。

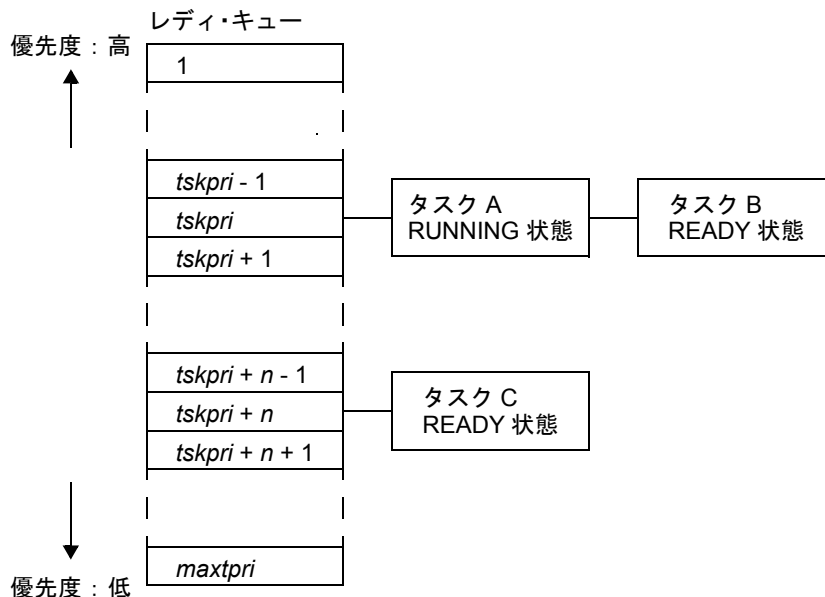
16.4.1 レディ・キュー

RI600PX では、タスクのスケジューリング方式を実現する手段として“レディ・キュー”を提供しています。

なお、RI600PX におけるレディ・キューは、優先度をキーとしたハッシュ・テーブルであり、実行可能な状態（RUNNING 状態または READY 状態）へと遷移したタスクが FIFO 順でキューイングされます。このため、スケジューラは、起動された際にレディ・キューの優先度高位から検出処理を実行し、キューイングされているタスクを検出した場合には、該当優先度の先頭タスクに CPU の利用権を与えることにより、RI600PX のスケジューリング方式（優先度方式、FCFS 方式）を実現しています。

以下に、複数のタスクがレディ・キューにキューイングされている場合を示します。

図 16 - 1 スケジューリング方式（優先度方式、FCFS 方式）の実現



- レディ・キューの生成

RI600PX では、レディ・キューの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

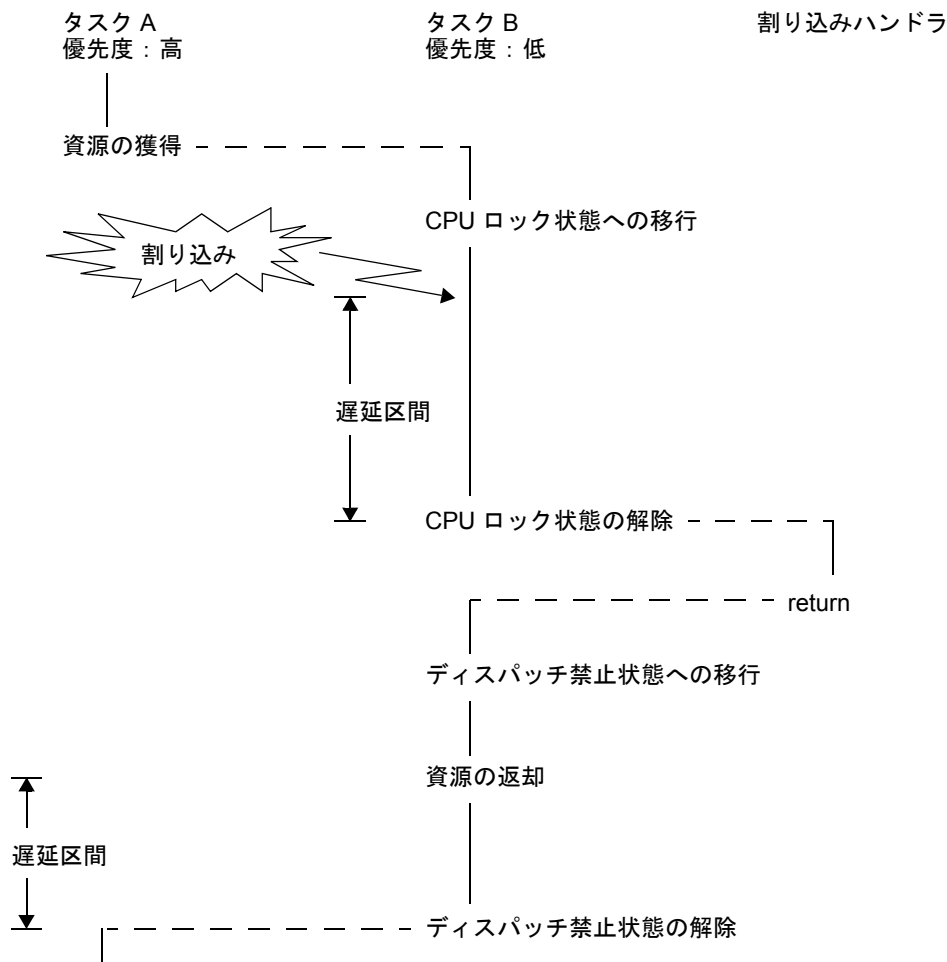
レディ・キューの静的生成とは、システム・コンフィギュレーション・ファイルでシステム情報（system）のタスク優先度の最大値（priority）を定義することをいいます。

16.5 タスク・スケジューリングのロック機能

RI600PX では、処理プログラムからスケジューラの状態を明示的に操作し、ディスパッチ処理を禁止／許可する機能（スケジューリングのロック機能）を提供しています。

以下に、スケジューリングのロック機能を利用した際の処理の流れを示します。

図 16 - 2 スケジューリングのロック機能



詳細は、「11.4 CPU ロック状態への移行と解除」,「11.6 ディスパッチ禁止状態への移行と解除」を参照してください。

16.6 アイドリング

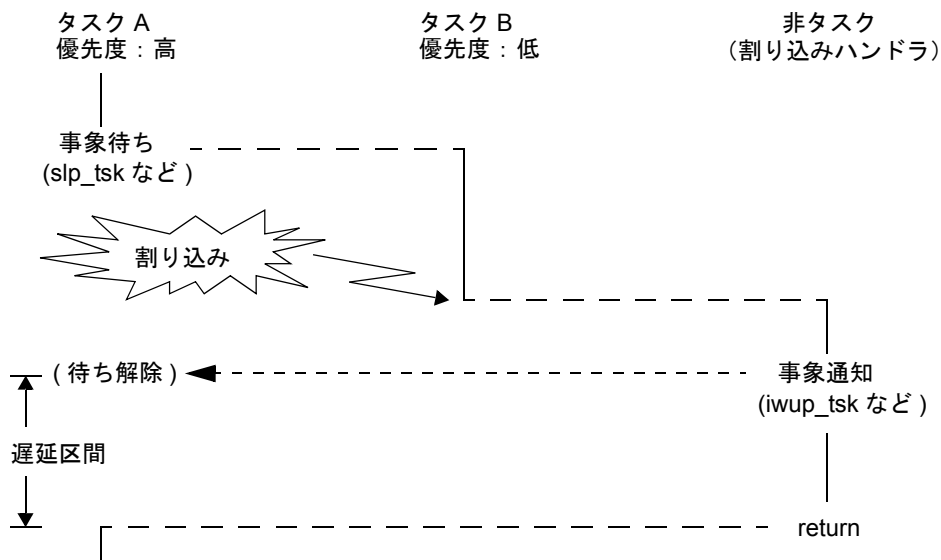
RUNNING 状態または READY 状態のタスクが存在しなくなると、RI600PX 内部で無限ループとなり、割り込みが発生するのを待ちます。

16.7 非タスク内におけるタスク・スケジューリング処理

「16.2 処理の単位と優先順位」に示したように、非タスク（割り込みハンドラ、周期ハンドラ、アラーム・ハンドラ、およびアクセス例外ハンドラ）はタスクよりも優先順位が高いため、非タスク処理が始まるとそれが完了するまで、タスクは実行されません。

以下に、非タスク内でディスパッチ処理を伴うサービス・コールを発行した際の例を示します。この例では、割り込みハンドラが事象通知を行った時点でタスク B よりも優先度の高いタスク A の待ち状態が解除されますが、この時点ではタスク A はまだ実行されず、非タスク（割り込みハンドラ）の処理が継続されます。非タスク処理が完了した時点でスケジューラが起動され、その結果タスク A が実行されます。

図 16-3 非タスク内におけるスケジューリング処理



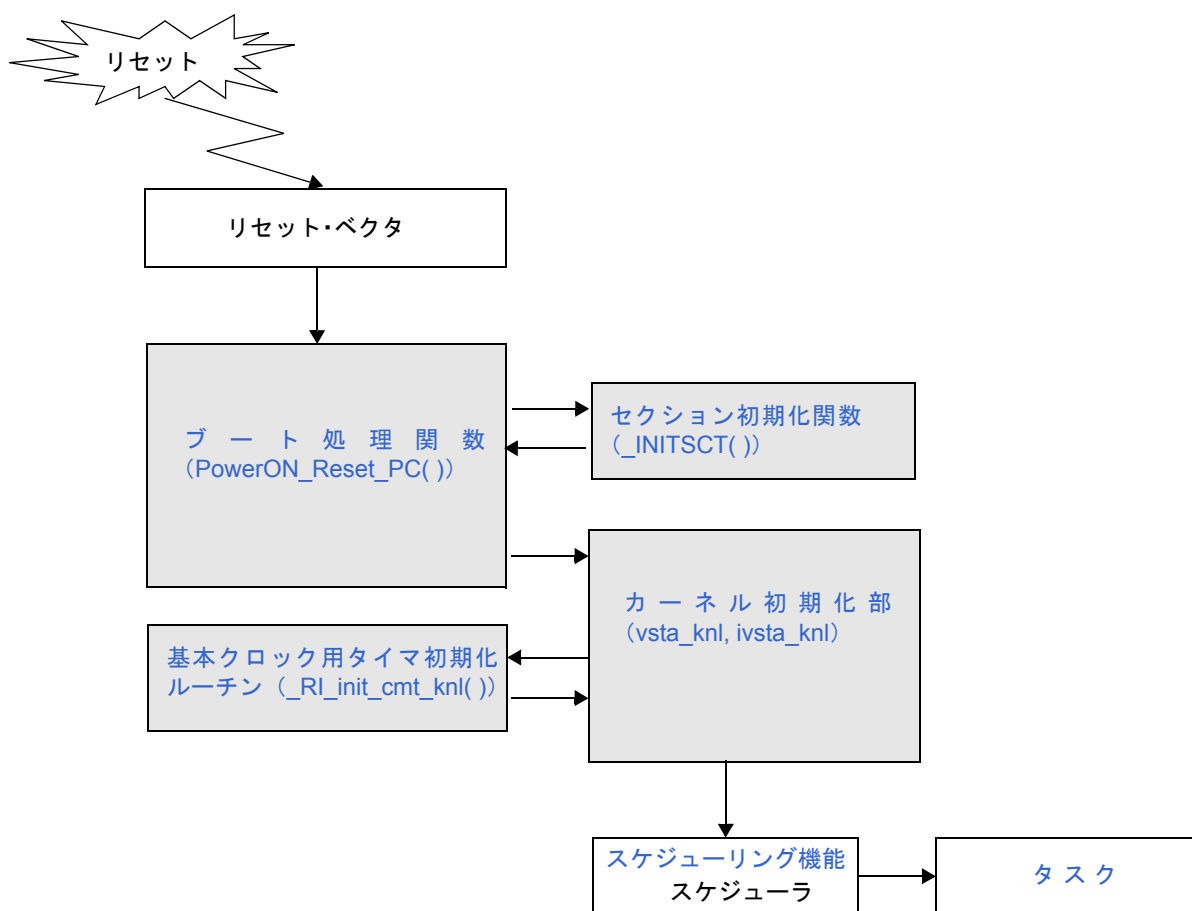
第17章 システム初期化処理

本章では、システム初期化処理について解説しています。

17.1 概 要

以下に、リセットの発生から処理プログラム（タスク）に制御が移るまでに実行される処理の流れを示します。

図 17 - 1 処理の流れ（システム初期化処理）



17.2 ブート処理ファイル（ユーザ・OWN・コーディング部）

ブート処理ファイルには、以下を記述します。

- 1) ブート処理関数 (PowerON_Reset_PC())
- 2) kernel_ram.h および kernel_rom.h の取り込み

備考 RI600PX で提供するサンプルのブート処理ファイルは“resetprg.c”です。

17.2.1 ブート処理関数 (PowerON_Reset_PC())

ブート処理関数はリセットベクタに登録されるプログラムで、スーパーバイザ・モードで実行します。通常は、以下のよ
うな処理を行います。

- プロセッサ、ハードウェアの初期化
RX MCU の高速割り込みを使用する場合は、FINTV レジスタを高速割り込みハンドラの開始アドレスに初期化し
てください。
- C / C++ 言語ソフトウェアの実行環境の初期化（セクションの初期化など）
- RI600PX を起動 (vsta_knl, ivsta_knl の呼び出し)
- ブート処理関数の基本型
ブート処理関数は“void PowerON_Reset_PC (void)”としてください。これ以外の関数名とする場合は、システム・コ
ンフィギュレーション・ファイルで interrupt_fvector[31] にその関数名を定義する必要があります。
ブート処理関数の例については、「17.2.4 ブート処理ファイルの例」を参照してください。

備考 静的 API“interrupt_fvector[]”については、「20.22 固定ベクタ／例外ベクタ情報 (interrupt_fvector[])」を
参照してください。

- ブート処理関数に関する留意事項

- スタック
以下のように #pragma entry ディレクティブを記述してください。これにより、ブート処理関数の先頭でスタック
・ポインタ (ISP) をシステム・スタックに設定するコードが生成されます。

```
#pragma entry PowerON_Reset_PC
```

- PSW レジスタ
カーネル初期化部 (vsta_knl, ivsta_knl) を呼び出すまでは、全割り込みを禁止した状態とスーパーバイザ・モー
ドを維持してください。CPU リセット直後はこの状態 (PSW.I=0, PSW.PM=0) となっているので、通常は
PSW レジスタを変更しないでください。
- EXTB レジスタ (RXv2 アーキテクチャ)
必要に応じ、EXTB レジスタを FIX_INTERRUPT_VECTOR セクションの先頭アドレスに初期化してください。
2.6.4 節の「FIX_INTERRUPT_VECTOR セクション」も参照してください。
- サービス・コールの発行
ブート処理関数はカーネル初期化部 (vsta_knl, ivsta_knl) の実行前に実行されるため、vsta_knl, ivsta_knl を除
くサービス・コールを呼び出してはなりません。

17.2.2 kernel_ram.h および kernel_rom.h の取り込み

cfg600px が出力した RAM 定義ファイル (kernel_ram.h) と ROM 定義ファイル (kernel_rom.h) を、ブート処理ファイルの最後で、この順にインクルードしてください。

17.2.3 ブート処理ファイルのコンパイラ・オプション

以下のコンパイラ・オプションの指定が必須です。

- "-lang=c" または "-lang=c99"
- "-nostuff"
- 適切な "-isa" または "-cpu"

備考 コンパイラ・オプション "-isa" は、コンパイラ CC-RX V2.01 以降でサポートされています。

17.2.4 ブート処理ファイルの例

```

#include <machine.h>
#include <_h_c_lib.h>
// #include <stddef.h> // Remove the comment when you use errno
// #include <stdlib.h> // Remove the comment when you use rand()
#include "typedefine.h" // Define Types
#include "kernel.h" // Provided by RI600PX
#include "kernel_id.h" // Generated by cfg600px

#ifdef __cplusplus
extern "C" {
#endif
void PowerON_Reset_PC(void);
#ifdef __cplusplus
}
#endif

// #ifdef __cplusplus // Use SIM I/O
// extern "C" {
// #endif
// extern void _INIT_IOLIB(void);
// extern void _CLOSEALL(void);
// #ifdef __cplusplus
// }
// #endif

#define FPSW_init 0x00000000 // FPSW bit base pattern

// extern void srand(_UINT); // Remove the comment when you use rand()
// extern _SBYTE *_slptr; // Remove the comment when you use strtok()

// #ifdef __cplusplus // Use Hardware Setup
// extern "C" {
// #endif
// extern void HardwareSetup(void);
// #ifdef __cplusplus
// }

// #endif

// #ifdef __cplusplus // Remove the comment when you use global class object
// extern "C" { // Sections C$INIT and C$END will be generated
// #endif
// extern void _CALL_INIT(void);
// extern void _CALL_END(void);
// #ifdef __cplusplus
// }
// #endif

```

```

////////////////////////////////////
// Section definition
////////////////////////////////////
#pragma section P PS
#pragma section B BS
#pragma section C CS
#pragma section D DS

#pragma entry PowerON_Reset_PC

////////////////////////////////////
// Boot processing
////////////////////////////////////
void PowerON_Reset_PC(void)
{
#ifdef __ROZ                // Initialize FPSW
#define _ROUND 0x00000001    // Let FPSW RMbits=01 (round to zero)
#else
#define _ROUND 0x00000000    // Let FPSW RMbits=00 (round to nearest)
#endif
#ifdef __DOFF
#define _DENOM 0x00000100    // Let FPSW DNbit=1 (denormal as zero)
#else
#define _DENOM 0x00000000    // Let FPSW DNbit=0 (denormal as is)
#endif

//   set_extb(__sectop("FIX_INTERRUPT_VECTOR")); // Initialize EXTB register
//                                               //   (only for RXv2 arch.)
   set_fpsw(FPSW_init | _ROUND | _DENOM);

   _INIT_SCT();                // Initialize Sections

//   _INIT_IOLIB();            // Use SIM I/O

//   errno=0;                  // Remove the comment when you use errno
//   srand((_UINT)1);          // Remove the comment when you use rand()
//   _slptr=NULL;              // Remove the comment when you use strtok()

//   HardwareSetup();          // Use Hardware Setup
   nop();

//   set_fintv(<handler address>; // Initialize FINTV register

//   _CALL_INIT();             // Remove the comment when you use global class object
   vsta_knl();                  // Start RI600PX
//                               // Never return from vsta_knl

//   _CLOSEALL();              // Use SIM I/O

//   _CALL_END();              // Remove the comment when you use global class object
   brk();

}
////////////////////////////////////
// RI600PX system data
////////////////////////////////////
#include "kernel_ram.h"        // generated by cfg600px
#include "kernel_rom.h"        // generated by cfg600px

```

17.3 カーネル初期化部 (vsta_knl, ivsta_knl)

カーネル初期化部は、`vsta_knl`、`ivsta_knl` の呼び出しによって実行されます。`vsta_knl`、`ivsta_knl` は、通常はブート処理関数 (`PowerON_Reset_PC()`) から呼び出します。

カーネル初期化部では、以下に示した処理が実行されます。

- 1) ISP レジスタを、SI セクションの最終アドレス +1 に初期化
- 2) INTB レジスタを、`cfg600px` によって生成された可変ベクタ・テーブル (INTERRUPT_VECTOR セクション) 先頭アドレスに初期化
- 3) システム時刻を 0 に初期化
- 4) システム・コンフィギュレーション・ファイルで定義された各種オブジェクトの生成
ここで異常を検出した場合は、システム・ダウンとなります。
- 5) MPU (Memory Protection Unit) を初期化
ここで異常を検出した場合は、システム・ダウンとなります。
- 6) 基本クロック用タイマの初期化 (基本クロック用タイマ初期化ルーチン (`_RI_init_cmt_knl()`) の呼び出し)
- 7) スケジューラに制御を移す

17.4 セクション初期化関数 (_INITSCT())

ブート処理関数(PowerON_Reset_PC())から呼び出されるセクション初期化関数“_INITSCT()”は、コンパイラによって提供されます。_INITSCT()は、セクション情報ファイル(ユーザ・OWN・コーディング部)に記述されたテーブルにしたがって、未初期化データ・セクションの0クリア、および初期化データ・セクションの初期化を行います。

初期化対象のセクションは、セクション情報ファイルのセクション初期化用テーブル(DTBL,BTBL)へ記述する必要があります。_INITSCT()が使用するセクションの先頭アドレスおよび最終アドレスを、セクションアドレス演算子を用いて設定します。セクション初期化用テーブルのセクション名は、未初期化データ領域を C\$BSEC、初期化データ領域を C\$DSEC で宣言します。

また、DTBLに記述した初期化データ・セクションは、リンカ・オプション“-rom”を用いてROMからRAMにマップする必要があります。詳細は「2.6.5 初期化データ・セクション」を参照してください。

備考 _INITSCT()の詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX コーディング編」を参照してください。

17.4.1 セクション情報ファイル（ユーザ・オウン・コーディング部）

セクション情報ファイルは、ユーザ・オウン・コーディング部としてユーザが作成する必要があります。以下に、セクション情報ファイルの例を示します。

備考 RI600PX が提供するサンプルのセクション情報ファイルは“dbsct.c”です。

```
#include "typedefine.h"

#pragma unpack

#pragma section C C$DSEC
extern const struct {
    _UBYTE *rom_s;          /* Start address of the initialized data section in ROM */
    _UBYTE *rom_e;          /* End address of the initialized data section in ROM */
    _UBYTE *ram_s;          /* Start address of the initialized data section in RAM */
}
_DTBL[] = {
    { __sectop("D"), __secend("D"), __sectop("R") },
    { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
    { __sectop("D_1"), __secend("D_1"), __sectop("R_1") },
    { __sectop("DS"), __secend("DS"), __sectop("RS") },
    { __sectop("DS_2"), __secend("DS_2"), __sectop("RS_2") },
    { __sectop("DS_1"), __secend("DS_1"), __sectop("RS_1") },
    { __sectop("DU_SH"), __secend("DU_SH"), __sectop("RU_SH") },
    { __sectop("DU_SH_2"), __secend("DU_SH_2"), __sectop("RU_SH_2") },
    { __sectop("DU_SH_1"), __secend("DU_SH_1"), __sectop("RU_SH_1") },
    { __sectop("DU_MASTERDOM"), __secend("DU_MASTERDOM"),
      __sectop("RU_MASTERDOM") },
    { __sectop("DU_MASTERDOM_2"), __secend("DU_MASTERDOM_2"),
      __sectop("RU_MASTERDOM_2") },
    { __sectop("DU_MASTERDOM_1"), __secend("DU_MASTERDOM_1"),
      __sectop("RU_MASTERDOM_1") },
    { __sectop("DU_DOM_A"), __secend("DU_DOM_A"), __sectop("RU_DOM_A") },
    { __sectop("DU_DOM_A_2"), __secend("DU_DOM_A_2"), __sectop("RU_DOM_A_2") },
    { __sectop("DU_DOM_A_1"), __secend("DU_DOM_A_1"), __sectop("RU_DOM_A_1") },
    { __sectop("DU_DOM_B"), __secend("DU_DOM_B"), __sectop("RU_DOM_B") },
    { __sectop("DU_DOM_B_2"), __secend("DU_DOM_B_2"), __sectop("RU_DOM_B_2") },
    { __sectop("DU_DOM_B_1"), __secend("DU_DOM_B_1"), __sectop("RU_DOM_B_1") }
};

#pragma section C C$BSEC
extern const struct {
    _UBYTE *b_s;           /* Start address of non-initialized data section */
    _UBYTE *b_e;           /* End address of non-initialized data section */
}
_BTBL[] = {
    { __sectop("B"), __secend("B") },
    { __sectop("B_2"), __secend("B_2") },
    { __sectop("B_1"), __secend("B_1") },
    { __sectop("BS"), __secend("BS") },
    { __sectop("BS_2"), __secend("BS_2") },
    { __sectop("BS_1"), __secend("BS_1") },
};
```

```

{ __sectop("BU_SH"), __secend("BU_SH") },
{ __sectop("BU_SH_2"), __secend("BU_SH_2") },
{ __sectop("BU_SH_1"), __secend("BU_SH_1") },
{ __sectop("DU_MASTERDOM"), __secend("DU_MASTERDOM"),
  __sectop("RU_MASTERDOM") },
{ __sectop("DU_MASTERDOM_2"), __secend("DU_MASTERDOM_2"),
  __sectop("RU_MASTERDOM_2") },
{ __sectop("DU_MASTERDOM_1"), __secend("DU_MASTERDOM_1"),
  __sectop("RU_MASTERDOM_1") },
{ __sectop("DU_DOM_A"), __secend("DU_DOM_A"), __sectop("RU_DOM_A") },
{ __sectop("DU_DOM_A_2"), __secend("DU_DOM_A_2"), __sectop("RU_DOM_A_2") },
{ __sectop("DU_DOM_A_1"), __secend("DU_DOM_A_1"), __sectop("RU_DOM_A_1") },
{ __sectop("DU_DOM_B"), __secend("DU_DOM_B"), __sectop("RU_DOM_B") },
{ __sectop("DU_DOM_B_2"), __secend("DU_DOM_B_2"), __sectop("RU_DOM_B_2") },
{ __sectop("DU_DOM_B_1"), __secend("DU_DOM_B_1"), __sectop("RU_DOM_B_1") }
};

#pragma section C C$BSEC
extern const struct {
  _UBYTE *b_s;          /* Start address of non-initialized data section */
  _UBYTE *b_e;          /* End address of non-initialized data section */
}
_BTBL[] = {
{ __sectop("B"), __secend("B") },
{ __sectop("B_2"), __secend("B_2") },
{ __sectop("B_1"), __secend("B_1") },
{ __sectop("BS"), __secend("BS") },
{ __sectop("BS_2"), __secend("BS_2") },
{ __sectop("BS_1"), __secend("BS_1") },
{ __sectop("BU_SH"), __secend("BU_SH") },
{ __sectop("BU_SH_2"), __secend("BU_SH_2") },
{ __sectop("BU_SH_1"), __secend("BU_SH_1") },
{ __sectop("BU_MASTERDOM"), __secend("BU_MASTERDOM") },
{ __sectop("BU_MASTERDOM_2"), __secend("BU_MASTERDOM_2") },
{ __sectop("BU_MASTERDOM_1"), __secend("BU_MASTERDOM_1") },
{ __sectop("BU_DOM_A"), __secend("BU_DOM_A") },
{ __sectop("BU_DOM_A_2"), __secend("BU_DOM_A_2") },
{ __sectop("BU_DOM_A_1"), __secend("BU_DOM_A_1") },
{ __sectop("BU_DOM_B"), __secend("BU_DOM_B") },
{ __sectop("BU_DOM_B_2"), __secend("BU_DOM_B_2") },
{ __sectop("BU_DOM_B_1"), __secend("BU_DOM_B_1") }
};

#pragma section

/*
** CTBL prevents excessive output of L1100 messages when linking.
** Even if CTBL is deleted, the operation of the program does not change.
*/
_UBYTE * const _CTBL[] = {
  __sectop("C_1"), __sectop("C_2"), __sectop("C"),
  __sectop("W_1"), __sectop("W_2"), __sectop("W"),
  __sectop("CU_MASTERDOM_1"), __sectop("CU_MASTERDOM_2"),
  __sectop("CU_MASTERDOM"),
  __sectop("CU_DOM_A_1"), __sectop("CU_DOM_A_2"), __sectop("CU_DOM_A"),
  __sectop("CU_DOM_B_1"), __sectop("CU_DOM_B_2"), __sectop("CU_DOM_B"),
  __sectop("WU_SH_1"), __sectop("WU_SH_2"), __sectop("WU_SH"),
  __sectop("LU_SH"),
  __sectop("CU_SH_1"), __sectop("CU_SH_2"), __sectop("CU_SH"),
  __sectop("CS_1"), __sectop("CS_2"), __sectop("CS"), __sectop("P")
};

#pragma packoption

```


17.5 固定ベクタ・テーブル／例外ベクタ・テーブルのレジスタなど

MCUによっては、固定ベクタ・テーブル（RXv1 アーキテクチャ）／例外ベクタ・テーブル（RXv2 アーキテクチャ）内の 0xFFFFF80 ~ 0xFFFFFBF 番地に、エンディアン選択レジスタやオンチップ・デバッガ ID コード・プロテクトなどが割り当てられています。これらを設定するには、システム・コンフィギュレーション・ファイルに“interrupt_fvector[]”を記述してください。詳細は、「[20.22 固定ベクタ／例外ベクタ情報 \(interrupt_fvector\[\]\)](#)」を参照してください。

第18章 データ・タイプとマクロ

本章では、RI600PX が提供するサービス・コールを発行する際に使用するデータ・タイプ、マクロについて解説しています。

備考 <ri_root> は、RI600PX のインストール・フォルダを表しています。
デフォルトは、“C:\Program Files\Renesas Electronics\CubeSuite+¥RI600PX” です。

18.1 データ・タイプ

以下に、サービス・コールを発行する際に指定する各種パラメータのデータ・タイプ一覧を示します。

データ・タイプのマクロ定義は、<ri_root>\inc600\kernel.h、または kernel.h から呼び出される <ri_root>\inc600\itron.h で行われています。

表 18 - 1 データ・タイプ

マクロ	型	意味
B	signed char	符号付き 8 ビット整数
H	signed short	符号付き 16 ビット整数
W	signed long	符号付き 32 ビット整数
D	signed long long	符号付き 64 ビット整数
UB	unsigned char	符号なし 8 ビット整数
UH	unsigned short	符号なし 16 ビット整数
UW	unsigned long	符号なし 32 ビット整数
UD	unsigned long long	符号なし 64 ビット整数
VB	signed char	データ・タイプが一定しない値 (8 ビット)
VH	signed short	データ・タイプが一定しない値 (16 ビット)
VW	signed long	データ・タイプが一定しない値 (32 ビット)
VD	signed long long	データ・タイプが一定しない値 (64 ビット)
VP	void *	データ・タイプが一定しない値 (ポインタ)
FP	void (*)	処理プログラムの起動アドレス (ポインタ)
INT	signed long	符号付き 32 ビット整数
UINT	unsigned long	符号なし 32 ビット整数
BOOL	signed long	真偽値 (TRUE または FALSE)
ER	signed long	エラー・コード
ID	signed short	オブジェクト ID
ATR	unsigned short	オブジェクト属性
STAT	unsigned short	オブジェクトの状態
MODE	unsigned short	サービス・コールの動作モード
PRI	signed short	タスクまたはメッセージの優先度
SIZE	unsigned long	領域のサイズ (単位: バイト)
TMO	signed long	タイムアウト (単位: ミリ秒)

マクロ	型	意味
RELTIM	unsigned long	相対時間（単位：ミリ秒）
VP_INT	signed long	データ・タイプが一定しない値(ポインタ), または符号付き 32 ビット整数
ER_ID	signed long	エラー・コード, またはオブジェクト ID
ER_UINT	signed long	エラー・コード, または符号なし 32 ビット整数
ER_BOOL	signed long	エラー・コード, または真偽値 (TRUE または FALSE)
ACPTN	unsigned short	アクセス許可パターン
FLGPTN	unsigned long	イベントフラグのビット・パターン
IMASK	unsigned short	割り込みマスクレベル
TEXPTN	unsigned long	タスク例外要因

18.2 定数マクロ

以下に、定数マクロ一覧を示します。

なお、定数マクロの定義は、以下のいずれかで行われています。

- <ri_root>%inc600%kernel.h
- kernel.h から呼び出される <ri_root>%inc600%itron.h
- cfg600px が出力するシステム情報ヘッダファイル kernel_id.
本ファイルの内容は、システム・コンフィギュレーション・ファイルの定義内容によって変化します。

表 18 - 2 定数マクロ

分類	マクロ	定義内容	定義場所	説明
一般	NULL	0	itron.h	無効ポインタ
	TRUE	1	itron.h	真
	FALSE	0	itron.h	偽
	E_OK	0	itron.h	正常終了

分類	マクロ	定義内容	定義場所	説明
属性	TA_NULL	0	itron.h	オブジェクト属性を指定しない
	TA_HLNG	0x0000	kernel.h	高級言語用インタフェース
	TA_TFIFO	0x0000	kernel.h	タスクの待ち行列は FIFO 順
	TA_TPRI	0x0001	kernel.h	タスクの待ち行列はタスクの現在優先度順。ただし、同じ現在優先度のタスクの中では FIFO 順。
	TA_MFIFO	0x0000	kernel.h	メッセージ・キューは FIFO 順
	TA_MPRI	0x0002	kernel.h	メッセージ・キューはメッセージ優先度順。ただし、同じメッセージ優先度のメッセージの中では FIFO 順。
	TA_ACT	0x0002	kernel.h	タスクを生成と同時に起動
	TA_WSGL	0x0000	kernel.h	イベント・フラグに複数タスクの待ちを許さない
	TA_WMUL	0x0002	kernel.h	イベント・フラグに複数タスクの待ちを許す
	TA_CLR	0x0004	kernel.h	待ち解除時にイベント・フラグをクリア
	TA_CEILING	0x0003	kernel.h	優先度上限プロトコル
	TA_STA	0x0002	kernel.h	周期ハンドラを動作状態で生成
	TA_PHS	0x0004	kernel.h	周期ハンドラ位相を保存
タイムアウト	TMO_POL	0	itron.h	ポーリング
	TMO_FEVR	-1	itron.h	永久待ち
動作モード	TWF_ANDW	0x0000	kernel.h	イベント・フラグの AND 待ち
	TWF_ORW	0x0001	kernel.h	イベント・フラグの OR 待ち
タスク例外	TTEX_ENA	0x0000	kernel.h	タスク例外許可状態
	TTEX_DIS	0x0001	kernel.h	タスク例外禁止状態

分類	マクロ	定義内容	定義場所	説明
状態	TTS_RUN	0x0001	kernel.h	RUNNING 状態
	TTS_RDY	0x0002	kernel.h	READY 状態
	TTS_WAI	0x0004	kernel.h	WAITING 状態
	TTS_SUS	0x0008	kernel.h	SUSPENDED 状態
	TTS_WAS	0x000C	kernel.h	WAITING-SUSPENDED 状態
	TTS_DMT	0x0010	kernel.h	DORMANT 状態
	TTW_SLP	0x0001	kernel.h	起床待ち状態
	TTW_DLY	0x0002	kernel.h	時間経過待ち状態
	TTW_SEM	0x0004	kernel.h	セマフォ資源獲得待ち状態
	TTW_FLG	0x0008	kernel.h	イベント・フラグ待ち状態
	TTW_SDTQ	0x0010	kernel.h	データ・キューへの送信待ち状態
	TTW_RDTQ	0x0020	kernel.h	データ・キューからの受信待ち状態
	TTW_MBX	0x0040	kernel.h	メールボックスからの受信待ち状態
	TTW_MTX	0x0080	kernel.h	ミューテックス待ち状態
	TTW_SMBF	0x0100	kernel.h	メッセージ・バッファへの送信待ち状態
	TTW_RMBF	0x0200	kernel.h	メッセージ・バッファからの受信待ち状態
	TTW_MPF	0x2000	kernel.h	固定長メモリ・ブロック獲得待ち待ち状態
	TTW_MPL	0x4000	kernel.h	可変長メモリ・ブロック獲得待ち待ち状態
	TCYC_STP	0x0000	kernel.h	周期ハンドラ非動作状態
	TCYC_STA	0x0001	kernel.h	周期ハンドラ動作状態
TALM_STP	0x0000	kernel.h	アラーム・ハンドラ非動作状態	
TALM_STA	0x0001	kernel.h	アラーム・ハンドラ動作状態	
その他	TSK_SELF	0	kernel.h	自タスク指定
	TSK_NONE	0	kernel.h	該当するタスクがない
	TPRI_SELF	0	kernel.h	自タスクのベース優先度の指定
	TPRI_INI	0	kernel.h	タスクの起動時優先度の指定

分類	マクロ	定義内容	定義場所	説明
カーネル構成	TMIN_TPRI	1	kernel.h	タスク優先度の最小値
	TMAX_TPRI	system.priority	kernel_id.h	タスク優先度の最大値
	TMIN_MPRI	1	kernel.h	メッセージ優先度の最小値
	TMAX_MPRI	system.message_pri	kernel_id.h	メッセージ優先度の最大値
	TKERNEL_MAKER	0x011B	kernel.h	カーネルのメーカー・コード
	TKERNEL_PRID	0x0004	kernel.h	カーネルの識別番号
	TKERNEL_SPVER	0x5403	kernel.h	ITRON 仕様のバージョン番号
	TKERNEL_PRVER	0x0120	kernel.h	カーネルのバージョン番号
	TMAX_ACTCNT	255	kernel.h	タスク起動要求キューイング数の最大値
	TMAX_WUPCNT	255	kernel.h	タスク起床要求キューイング数の最大値
	TMAX_SUSCNT	1	kernel.h	タスク強制待ち要求ネスト数の最大値
	TBIT_FLGPTN	32	kernel.h	イベントフラグのビット数
	TBIT_TEXPTN	32	kernel.h	タスク例外要因のビット数
	TIC_NUME	system.tic_nume	kernel_id.h	基本クロック割り込み周期の分子
	TIC_DENO	system.tic_deno	kernel_id.h	基本クロック割り込み周期の分母
	TMAX_MAXSEM	65535	kernel.h	セマフォの最大資源数の最大値
	VTMAX_DOMAIN	備考 1 参照	kernel_id.h	最大ドメイン ID
	VTMAX_TSK	備考 1 参照	kernel_id.h	最大タスク ID
	VTMAX_SEM	備考 1 参照	kernel_id.h	最大セマフォ ID
	VTMAX_FLG	備考 1 参照	kernel_id.h	最大イベントフラグ ID
	VTMAX_DTQ	備考 1 参照	kernel_id.h	最大データ・キュー ID
	VTMAX_MBX	備考 1 参照	kernel_id.h	最大メールボックス ID
	VTMAX_MTX	備考 1 参照	kernel_id.h	最大ミューテックス ID
	VTMAX_MBF	備考 1 参照	kernel_id.h	最大メッセージ・バッファ ID
	VTMAX_MPF	備考 1 参照	kernel_id.h	最大固定長メモリ・プール ID
	VTMAX_MPL	備考 1 参照	kernel_id.h	最大可変長メモリ・プール ID
	VTMAX_CYH	備考 1 参照	kernel_id.h	最大周期ハンドラ ID
	VTMAX_ALH	備考 1 参照	kernel_id.h	最大アラーム・ハンドラ ID
	VTSZ_MBFTBL	4	kernel.h	メッセージ・バッファのメッセージ管理テーブルのサイズ (単位: バイト)
	VTMAX_AREASIZE	0x10000000	kernel.h	各種領域サイズの最大値 (単位: バイト)
VTKNL_LVL	system.system_IPL	kernel_id.h	カーネル割り込みマスクレベル	
VTIM_LVL	clock.IPL	kernel_id.h	基本クロック用タイマ割り込み優先レベル	

分類	マクロ	定義内容	定義場所	説明
エラー・コード	E_SYS	-5	itron.h	システム・エラー
	E_NOSPT	-9	itron.h	未サポート機能
	E_RSFN	-10	itron.h	予約機能コード
	E_RSATR	-11	itron.h	予約属性
	E_PAR	-17	itron.h	パラメータ・エラー
	E_ID	-18	itron.h	不正 ID 番号
	E_CTX	-25	itron.h	コンテキスト・エラー
	E_MACV	-26	itron.h	メモリ・アクセス違反
	E_OACV	-27	itron.h	オブジェクト; アクセス違反
	E_ILUSE	-28	itron.h	サービス・コール不正使用
	E_NOMEM	-33	itron.h	メモリ不足
	E_NOID	-34	itron.h	ID 番号不足
	E_OBJ	-41	itron.h	オブジェクト状態エラー
	E_NOEXS	-42	itron.h	オブジェクト未生成
	E_QOVR	-43	itron.h	キューイング・オーバフロー
	E_RLWAI	-49	itron.h	待ち状態の強制解除
	E_TMOUT	-50	itron.h	ポーリング失敗またはタイムアウト
	E_DLT	-51	itron.h	待ちオブジェクトの削除
E_CLS	-52	itron.h	待ちオブジェクトの状態変化	
EV_RST	-127	itron.h	オブジェクト・リセットによる待ち解除	
保護機能拡張	TDOM_SELF	0	kernel.h	自タスクが所属するドメイン
	TACP_SHARED	((1u << (VTMAX_DOMAIN)) - 1)	kernel.h	すべてのドメインがアクセスできることを示すアクセス許可パターン
	TACT_SRW	{TACP_SHARED, TACP_SHARED, TACP_SHARED}	kernel.h	すべてのアクセスがすべてのドメインに許可されることを示すアクセス許可ベクタ 備考2参照
	TACT_SRO	{TACP_SHARED, 0, TACP_SHARED}	kernel.h	オペランド・ライト・アクセスはすべてのドメインに禁止され、オペランド・リードアクセスおよび実行アクセスはすべてのドメインに許可されていることを示すアクセス許可ベクタ 備考2参照
	TPM_READ	1	kernel.h	オペランド・リード・アクセス
	TPM_WRITE	2	kernel.h	オペランド・ライト・アクセス
	TPM_EXEC	4	kernel.h	実行アクセス

備考1 「20.7 最大 ID (maxdefine)」を参照してください。

備考2 初期代入文の右辺にのみ記述できます。

18.3 関数マクロ

以下に、関数マクロ一覧を示します。

なお、関数マクロの定義は、以下のいずれかで行われています。

- <ri_root>%inc600%kernel.h
- kernel.h から呼び出される <ri_root>%inc600%itron.h

18.3.1 エラー・コード関連

- 1) ER_MERCD (ER *erccd*)
erccd のメイン・エラー・コードを返します。
- 2) ER_SERCD (ER *erccd*)
erccd のサブ・エラー・コードを返します。
- 3) ER_ERCD (ER *merccd*, ER *serccd*)
merccd のメイン・エラー・コードと *serccd* のサブ・エラー・コードからなるエラー・コードを返します。

備考 RI600PX のサービス・コールが返すエラー・コードのサブ・エラー・コードはすべて -1、メイン・エラー・コードは表 18-2 に記載の値です。

18.3.2 データ・キュー関連

- 1) SIZE_TSZ_DTQ (UINT *dtqcnt*)
データ数が *dtqcnt* のデータ・キュー領域のサイズ (単位: バイト) を返します。

18.3.3 固定長メモリ・プール関連

- 1) SIZE_TSZ_MPF (UINT *blkcnt*, UINT *blkksz*)
blkksz バイトのメモリ・ブロックを *blkcnt* 個獲得可能な固定長メモリ・プール領域のサイズ (単位: バイト) を返します。
- 2) SIZE_TSZ_MPFMB (UINT *blkcnt*, UINT *blkksz*)
blkksz バイトのメモリ・ブロックを *blkcnt* 個獲得可能な固定長メモリ・プールの管理領域のサイズ (単位: バイト) を返します。

18.3.4 ドメイン関連

- 1) ATR_TA_DOM (ID *domid*)
domid で指定されたドメインに所属することを示すタスク属性値を返します。本マクロは、タスク生成時に指定する *tskatr* (タスク属性) の bit7-4 を設定するために使用します。

18.3.5 アクセス許可関連

1) ACPTN TACP(ID *domid*)

domid で指定されたドメインからのみアクセス可能であることを示すアクセス許可パターンを返します。

2) ACVCT TACT_PRW(ID *domid*)

すべてのアクセス（オペランド・リード、オペランド・ライト、実行）が、*domid* で指定されたドメインにのみ許可されていることを示すアクセス許可ベクタを返します。
本マクロは、初期代入文の右辺にのみ記述できます。

3) ACVCT TACT_PRO(ID *domid*)

オペランド・リード・アクセスおよび実行アクセスが *domid* で指定されたドメインにのみ許可され、オペランド・ライト・アクセスがすべてのドメインに禁止されていることを示すアクセス許可ベクタを返します。
本マクロは、初期代入文の右辺にのみ記述できます。

4) ACVCT TACT_SRPW(ID *domid*)

オペランド・リード・アクセスおよび実行アクセスがすべてのドメインにのみ許可され、オペランド・ライト・アクセスが *domid* で指定されたドメインにのみ許可されていることを示すアクセス許可ベクタを返します。
本マクロは、初期代入文の右辺にのみ記述できます。

第19章 サービス・コール

本章では、RI600PX が提供しているサービス・コールについて解説しています。

19.1 概 要

RI600PX が提供しているサービス・コールは、ユーザが記述した処理プログラムから RI600PX が管理している資源（タスク、セマフォなど）を操作するために用意されたサービス・ルーチンです。

以下に、RI600PX が提供しているサービス・コールを管理モジュール別に示します。

- タスク管理機能

cre_tsk	acre_tsk	del_tsk	act_tsk
iact_tsk	can_act	ican_act	sta_tsk
ista_tsk	ext_tsk	exd_tsk	ter_tsk
chg_pri	ichg_pri	get_pri	iget_pri
ref_tsk	iref_tsk	ref_tst	iref_tst

- タスク付属同期機能

slp_tsk	tslp_tsk	wup_tsk	iwup_tsk
can_wup	ican_wup	rel_wai	irel_wai
sus_tsk	isus_tsk	rsm_tsk	irms_tsk
frsm_tsk	ifrm_tsk	dly_tsk	

- タスク例外処理機能

def_tex	ras_tex	iras_tex	dis_tex
ena_tex	sns_tex	ref_tex	iref_tex

- 同期通信機能（セマフォ）

cre_sem	acre_sem	del_sem	wai_sem
pol_sem	ipol_sem	twai_sem	sig_sem
isig_sem	ref_sem	iref_sem	

- 同期通信機能（イベントフラグ）

cre_flg	acre_flg	del_flg	set_flg
iset_flg	clr_flg	iclr_flg	wai_flg
pol_flg	ipol_flg	twai_flg	ref_flg
iref_flg			

- 同期通信機能（データ・キュー）

cre_dtq	acre_dtq	del_dtq	snd_dtq
psnd_dtq	ipsnd_dtq	tsnd_dtq	fsnd_dtq
ifsnd_dtq	rcv_dtq	prcv_dtq	iprcv_dtq
trcv_dtq	ref_dtq	iref_dtq	

- 同期通信機能（メールボックス）

cre_mbx	acre_mbx	del_mbx	snd_mbx
isnd_mbx	rcv_mbx	prcv_mbx	iprcv_mbx
trcv_mbx	ref_mbx	iref_mbx	

- 拡張同期通信機能（ミューテックス）

cre_mtx	acre_mtx	del_mtx	loc_mtx
ploc_mtx	tlloc_mtx	unl_mtx	ref_mtx

- 拡張同期通信機能 (メッセージ・バッファ)

cre_mbf	acre_mbf	del_mbf	snd_mbf
psnd_mbf	ipsnd_mbf	tsnd_mbf	rcv_mbf
prcv_mbf	trcv_mbf	ref_mbf	iref_mbf

- メモリ・プール管理機能 (固定長メモリ・プール)

cre_mpf	acre_mpf	del_mpf	get_mpf
pget_mpf	ipget_mpf	tget_mpf	rel_mpf
irel_mpf	ref_mpf	iref_mpf	

- メモリ・プール管理機能 (可変長メモリ・プール)

cre_mpl	acre_mpl	del_mpl	get_mpl
pget_mpl	ipget_mpl	tget_mpl	rel_mpl
ref_mpl	iref_mpl		

- 時間管理機能

set_tim	iset_tim	get_tim	iget_tim
cre_cyc	acre_cyc	del_cyc	sta_cyc
ista_cyc	stp_cyc	istp_cyc	ref_cyc
iref_cyc	cre_alm	acre_alm	del_alm
sta_alm	ista_alm	stp_alm	istp_alm
ref_alm	iref_alm		

- システム状態管理機能

rot_rdq	irotd_rdq	get_tid	iget_tid
loc_cpu	iloc_cpu	unl_cpu	iunl_cpu
dis_dsp	ena_dsp	sns_ctx	sns_loc
sns_dsp	sns_dpn	vsys_dwn	ivsys_dwn
vsta_knl	ivsta_knl		

- 割り込み管理機能

chg_ims	ichg_ims	get_ims	iget_ims
---------	----------	---------	----------

- システム構成管理機能

ref_ver	iref_ver		
---------	----------	--	--

- オブジェクト・リセット機能

vrst_dtq	vrst_mbx	vrst_mbf	vrst_mpf
vrst_mpl			

- メモリ・オブジェクト保護機能

ata_mem	det_mem	sac_mem	vprb_mem
ref_mem			

19.1.1 サービス・コールの呼び出し方法

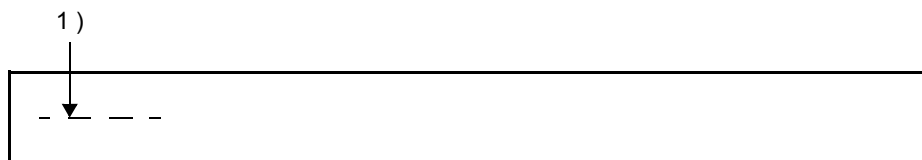
サービス・コールは、通常の C 言語関数と同様の方法で呼び出します。

備考 RI600PX が提供するサービス・コールを処理プログラムから発行する場合、以下に示したヘッダ・ファイルの定義 (インクルード処理) を行う必要があります。

- kernel.h : 標準ヘッダ・ファイル
- kernel_id.h : cfg600px が出力するシステム情報ヘッダ・ファイル

19.2 サービス・コール解説

次項から RI600PX が提供しているサービス・コールについて、以下の記述フォーマットにしたがって解説します。



2) → **概要**

3) → **C 言語形式**

4) → **パラメータ**

I/O	パラメータ	説明

5) → **機能**

6) → **戻り値**

マクロ	数値	意味

- 1) 名称
サービス・コールの名称を示しています。
- 2) 概要
サービス・コールの機能概要を示しています。
- 3) C 言語形式
サービス・コールを C 言語で記述された処理プログラムから発行する際の記述形式を示しています。
- 4) パラメータ
サービス・コールのパラメータを以下の形式で示しています。

I/O	パラメータ	説明
A	B	C

- A) パラメータの種類
 - I: RI600PX への入力パラメータ
 - O: RI600PX からの出力パラメータ
 - B) パラメータのデータ・タイプ
 - C) パラメータの説明
- 5) 機能
サービス・コールの機能詳細を示しています。
 - 6) 戻り値
サービス・コールからの戻り値を以下の形式で示しています。

マクロ	数値	意味
A	B	C

- A) 戻り値のマクロ
- B) 戻り値の数値
- C) 戻り値の意味

19.2.1 タスク管理機能

以下に、RI600PX がタスク管理機能として提供しているサービス・コールの一覧を示します。

表 19 - 1 タスク管理機能

サービス・コール名	機能概要	発行有効範囲
cre_tsk	タスクの生成	タスク
acre_tsk	タスクの生成 (ID 番号自動割付け)	タスク
del_tsk	タスクの削除	タスク
act_tsk	タスクの起動 (起動要求をキューイングする)	タスク
iact_tsk	タスクの起動 (起動要求をキューイングする)	非タスク
can_act	起動要求のキューイング解除	タスク
ican_act	起動要求のキューイング解除	非タスク
sta_tsk	タスクの起動 (起動要求をキューイングしない)	タスク
ista_tsk	タスクの起動 (起動要求をキューイングしない)	非タスク
ext_tsk	自タスクの終了	タスク
exd_tsk	自タスクの終了と削除	タスク
ter_tsk	タスクの強制終了	タスク
chg_pri	タスク優先度の変更	タスク
ichg_pri	タスク優先度の変更	非タスク
get_pri	タスク現在優先度の参照	タスク
iget_pri	タスク現在優先度の参照	非タスク
ref_tsk	タスク詳細情報の参照	タスク
iref_tsk	タスク詳細情報の参照	非タスク
ref_tst	タスク基本情報の参照	タスク
iref_tst	タスク基本情報の参照	非タスク

cre_tsk acre_tsk

概要

タスクの生成

C 言語形式

```
ER      cre_tsk ( ID tskid, T_CTSK *pk_ctsk );
ER_ID  acre_tsk ( T_CTSK *pk_ctsk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID
I	T_CTSK <i>*pk_ctsk</i> ;	タスク生成情報を格納した領域へのポインタ

【タスク生成情報 T_CTSK の構造】

```
typedef struct t_ctsk {
    ATR    tskatr;          /* タスク属性 */
    VP_INT exinf;          /* 拡張情報 */
    FP     task;            /* タスクの実行開始アドレス */
    PRI    itskpri;         /* タスク起動時優先度 */
    SIZE   stksz;           /* ユーザ・スタック・サイズ (バイト数) */
    VP     stk;             /* ユーザ・スタック領域の先頭アドレス */
} T_CTSK;
```

機能

本サービス・コールは、**信頼されたドメイン**に所属するタスクからのみ呼び出せます。

`cre_tsk` は、`tskid` で指定されたタスクを `pk_ctsk` で指定された内容で生成します。`acre_tsk` は `pk_ctsk` で指定された内容でタスクを生成し、生成したタスク ID を返します。

タスク生成時に行われる処理を、表 19-2 に示します。

表 19-2 タスク生成時に行われる処理

No.	処理内容
1	タスク起動要求キューイング数をクリアする。
2	タスク例外処理ルーチンを定義されていない状態にする。

1) タスク属性 (*tskatr*)

*tskatr*には以下を指定できます。

```
tskatr := ( TA_HLNG | [TA_ACT] | [TA_DOM (domid) ] )
```

以下に、*tskatr*のビット位置を示します。

bit15 ~ bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	ドメイン ID (TA_DOM(<i>domid</i>))				0	0	TA_ACT :1	TA_HLNG : 0

- TA_HLNG (= 0x0000)

タスクの記述言語は、C言語のみをサポートしています。

- TA_ACT (= 0x0002)

TA_ACTを指定すると、生成されたタスクは起動され、READY状態になります。タスク起動時に行われる処理は、表 19-3の通りです。TA_ACTを指定しない場合は、生成されたタスクはDORMANT状態になります。

- TA_DOM(*domid*)

*domid*には、所属するドメインIDを指定します。*domid*にTDOM_SELFを指定した場合、およびTA_DOM(*domid*)の指定を省略した場合は、本サービス・コールを呼び出したタスクと同じドメインとします。

備考 TA_DOMマクロについては、「18.3.4 ドメイン関連」を参照してください。

2) 拡張情報 (*exinf*)

TA_ACT属性、または *act_tsk*, *iact_tsk* によってタスクが起動された場合、*exinf* がタスクに引数として渡されます。また、*exinf* はタスク例外処理ルーチンにも引数として渡されます。*exinf* は、ユーザが生成するタスクに関する情報を設定するなどの目的で自由に使用できます。

3) タスクの実行開始アドレス (*task*)

*task*には、タスクの実行開始アドレスを指定します。

4) タスク起動時優先度 (*itskpri*)

*itskpri*には、タスク起動時の優先度を指定します。指定可能な範囲は、1 ~ TMAX_TPRIです。

5) ユーザ・スタック・サイズ (*stksz*)、ユーザ・スタック領域の先頭アドレス (*stk*)

アプリケーション側でユーザ・スタック領域を確保し、その先頭アドレスを *stk* に、サイズを *stksz* に指定します。ユーザ・スタック領域は、以下を満たす必要があります。

A) *stk* は 16 バイト境界であること。そうでない場合は E_PAR エラーを返します。

B) *stksz* は 16 の整数倍であること。そうでない場合は E_PAR エラーを返します。

C) ユーザ・スタック領域は、他のユーザ・スタックおよびメモリ・オブジェクトと重なっていないこと。そうでない場合、エラーは検出されず、システムの正常な動作は保証されません。

備考 μ ITRON4.0 仕様には、*stk* に NULL を指定することで RI600PX がユーザ・スタック領域を割り当てる機能がありますが、本サービス・コールはこの機能をサポートしていません。

戻り値

マクロ	数値	意味
—	正の値	acre_tsk の正常終了 (生成されたタスク ID)
E_OK	0	cre_tsk の正常終了
E_RSATR	-11	予約属性 <ul style="list-style-type: none"> - <i>tskatr</i> の bit0,bit2,bit3,bit8 ~ bit15 のいずれかが 1 - <code>VTMAX_DOMAIN</code> < (<i>tskatr</i> の bit4 ~ bit7 の値)
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>pk_ctsk</i> == NULL - <i>task</i> == NULL - <i>itskpri</i> < 0 - <code>TMAX_TPRI</code> < <i>itskpri</i> - <i>stk</i> が 16 バイト境界でない。 - <i>stksz</i> が 16 の倍数でない。 - <i>stksz</i> < 表 20 - 8 に示す下限値 - <code>VTMAX_AREASIZE</code> < <i>stksz</i> - <i>stk</i> + <i>stksz</i> > 0x100000000
E_ID	-18	不正 ID 番号 (cre_tsk のみ) <ul style="list-style-type: none"> - <i>tskid</i> ≤ 0 - <i>tskid</i> > <code>VTMAX_TSK</code>
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_ctsk</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 <ul style="list-style-type: none"> - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOMEM	-33	メモリ不足 <ul style="list-style-type: none"> - <i>stk</i> == NULL
E_NOID	-34	空き ID なし (acre_tsk のみ)
E_OBJ	-41	オブジェクト状態不正 (cre_tsk のみ) <ul style="list-style-type: none"> - <i>tskid</i> のタスクが存在する。

del_tsk**概要**

タスクの削除

C 言語形式

```
ER      del_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
tskid で指定されたタスクを削除します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>tskid</i> ≤ 0 - <i>tskid</i> > VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_OBJ	-41	オブジェクト状態不正 - 対象タスクが DORMANT 状態でない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

act_tsk
iact_tsk

概要

タスクの起動（起動要求をキューイングする）

C 言語形式

```
ER    act_tsk ( ID tskid );
ER    iact_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID

機能

tskid で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち、初期優先度のレディ・キューの最後尾にキューイングします。これにより、対象タスクは、RI600PX のスケジューリング対象となります。また、このとき以下に示す処理が行われます。

表 19 - 3 タスク起動時に行われる処理

No.	処理内容
1	ベース優先度と現在優先度を、初期優先度にする。
2	起床要求キューイング数をクリアする。
3	強制待ち要求ネスト数をクリアする。
4	保留例外要因をクリアする。
5	例外処理禁止状態にする。

ただし、本サービス・コールを発行した際、対象タスクが DORMANT 状態以外の場合には、対象タスクのキューイング処理、および、状態操作処理は行わず、対象タスクに起動要求をキューイング（起動要求カウンタに 1 を加算）しています。

備考 1 RI600PX が管理する起動要求カウンタは、8 ビット幅で構成されています。このため、本サービス・コールでは、起動要求数が 255 を超える場合には、起動要求の発行（起動要求カウンタの加算処理）は行わず、戻り値として E_QOVR を返します。

備考 2 本サービス・コールの発行により起動されたタスクには、タスク生成時に指定した拡張情報が渡されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <i>tskid</i> < 0 - <i>tskid</i> > <code>VTMAX_TSK</code> - <i>iact_tsk</i> を発行した際、<i>tskid</i> に <code>TSK_SELF</code> を指定した。
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - タスクから <i>iact_tsk</i> を発行した。 - 非タスクから <i>act_tsk</i> を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (<i>act_tsk</i> のみ) <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <i>tskid</i> のタスクが存在しない。
E_QOVR	-43	キューイング・オーバーフロー <ul style="list-style-type: none"> - 起動要求数が 255 回を超えた。

can_act
ican_act

概要

起動要求のキューイング解除

C 言語形式

```
ER_UINT can_act ( ID tskid );
ER_UINT ican_act ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID

機能

tskid で指定されたタスクにキューイングされている起動要求をすべて解除（起動要求カウンタに 0 を設定）します。なお、正常終了時は戻り値として本サービス・コールの発行により解除した起動要求数を返します。

備考 本サービス・コールでは、状態操作処理は行わず、起動要求カウンタの設定処理のみを行います。したがって、READY 状態などから DORMANT 状態に遷移することはありません。

戻り値

マクロ	数値	意味
—	正の値	正常終了（解除した起動要求数）
—	0	正常終了 - 起動要求数が 0 である。 - 対象タスクが DORMANT 状態である。
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - 非タスクから本サービス・コールを発行した際、 <i>tskid</i> に TSK_SELF を指定した。

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ican_act を発行した場合、および非タスクから can_act を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_NOEXS	-42	オブジェクト未生成 - tskid のタスクが存在しない。

sta_tsk
ista_tsk

概要

タスクの起動（起動要求をキューイングしない）

C 言語形式

```
ER    sta_tsk ( ID tskid, VP_INT stacd );
ER    ista_tsk ( ID tskid, VP_INT stacd );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID
I	VP_INT <i>stacd</i> ;	タスクの起動コード

機能

tskid で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち、初期優先度のレディ・キューの最後尾にキューイングします。これにより、対象タスクは、RI600PX のスケジューリング対象となります。また、このとき表 19-3 に示す処理が行われます。

ただし、本サービス・コールでは、起動要求のキューイングが行われません。このため、対象タスクが DORMANT 状態以外の場合には、対象タスクの状態操作処理は行わず、戻り値として E_OBJ を返します。

対象タスクには、*stacd* が引き渡されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>tskid</i> ≤ 0 - <i>tskid</i> > VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ista_tsk を発行した。 - 非タスクから sta_tsk を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 (sta_tsk のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態不正 - 対象タスクが DORMANT 状態でない。
E_NOEXS	-42	オブジェクト未生成 - tskid のタスクが存在しない。

ext_tsk**概要**

自タスクの終了

C 言語形式

```
void ext_tsk ( void );
```

パラメータ

なし

機能

自タスクを RUNNING 状態から DORMANT 状態へと遷移させ、レディ・キューから外します。これにより、自タスクは、RI600PX のスケジューリング対象から除外されます。また、このとき以下に示す処理が行われます。

表 19 - 4 タスク終了時に行われる処理

No	処理内容
1	終了するタスクがロックしていたミューテックスをロック解除する。(unl_mtx と同等の処理)

また、CPU ロック状態とディスパッチ禁止状態は解除されます。

本サービス・コールを発行した際、自タスクの起動要求がキューイングされていた（起動要求カウンタ > 0）場合には、自タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。このとき表 19 - 3 に示す処理が行われます。

本サービス・コールは、呼び出し元には戻りません。以下の場合、システム・ダウンとなります。

- 非タスクから本サービス・コールを発行した。
- 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

備考 1 タスク開始関数で return 命令が発行された場合、本サービス・コールと同等の処理が行われます。

備考 2 本サービス・コールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリ・ブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

戻り値

なし

exd_tsk

概要

自タスクの終了と削除

C 言語形式

```
void    exd_tsk ( void );
```

パラメータ

なし

機能

自タスクを正常終了後、削除します。

タスク終了時に行われる処理は、表 19-4 の通りです。

また、CPU ロック状態とディスパッチ禁止状態は解除されます。

本サービス・コールは、呼び出し元には戻りません。以下の場合、システム・ダウンとなります。

- 非タスクから本サービス・コールを発行した。
- 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

備考 本サービス・コールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリ・ブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

戻り値

なし

ter_tsk

概要

タスクの強制終了

C 言語形式

```
ER      ter_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID

機能

tskid で指定されたタスクを強制的に DORMANT 状態へと遷移させます。これにより、対象タスクは、RI600PX のスケジューリング対象から除外されます。また、このとき表 19-4 に示す処理が行われます。

本サービス・コールを発行した際、対象タスクの起動要求がキューイングされていた（起動要求カウンタ > 0）場合には、対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち、対象タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。このとき表 19-3 に示す処理が行われます。

備考 本サービス・コールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリ・ブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $tskid \leq 0$ - $tskid > VTMAX_TSK$
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。

マクロ	数値	意味
E_ILUSE	-28	サービス・コール不正使用 - 対象タスクが自タスクである。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

chg_pri ichg_pri

概要

タスク優先度の変更

C 言語形式

```
ER      chg_pri ( ID tskid, PRI tskpri );
ER      ichg_pri ( ID tskid, PRI tskpri );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
I	PRI <i>tskpri</i> ;	タスクのベース優先度 TPRI_INI : タスクの初期優先度 数値 : タスクのベース優先度

機能

tskid で指定されたタスクのベース優先度を *tskpri* で指定された値に変更します。

変更されたタスクのベース優先度は、タスクが終了、または本サービス・コールを呼び出すまで有効です。次回のタスク起動時、タスクのベース優先度はタスク生成時に指定した初期タスク優先度になります。

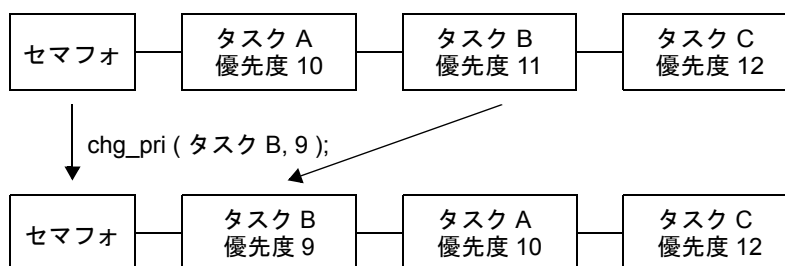
本サービス・コールは、対象タスクの現在優先度も、*tskpri* で示された値に変更します。ただし、対象タスクがミューテックスをロックしている場合は、現在優先度は変更しません。

対象タスクがミューテックスをロックしているかロックを待っている場合で、*tskpri* がそれらのミューテックスのいずれかの上限優先度よりも高い場合には、戻り値として E_ILUSE を返します。

現在優先度に変更された場合、以下の状態変化が生じます。

- 1) 対象タスクが RUNNING 状態または READY 状態の場合
本サービス・コールは、対象タスクを *tskpri* で指定された優先度に応じたレディ・キューの最後尾につながかえます。
- 2) 対象タスクが TA_TPRI 属性または TA_CEILING 属性のオブジェクトの待ちキューにキューイングされている場合
本サービス・コールは、*tskpri* で指定された優先度にしたがって対象タスクを待ちキューにつながかえます。待ちキューに *tskpri* で指定された現在優先度のタスクが複数ある場合は、対象タスクをそれらの中の最後尾につながかえます。

例 セマフォの待ちキューに3つのタスク（タスク A：優先度 10、タスク B：優先度 11、タスク C：優先度 12）が優先度順でキューイングされているとき、タスク B の優先度を 11 から 9 に変更した場合、待ちキューの順序は、以下のように変更されます。



備考 現在優先度とベース優先度については、「[8.2.2 現在優先度とベース優先度](#)」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - $tskpri < 0$ - $tskpri > TMAX_TPRI$
E_ID	-18	不正 ID 番号 - $tskid < 0$ - $tskid > VTMAX_TSK$ - <code>ichg_pri</code> を発行した際、 $tskid$ に <code>TSK_SELF</code> を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから <code>ichg_pri</code> を発行した。 - 非タスクから <code>chg_pri</code> を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (<code>chg_pri</code> のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_ILUSE	-28	サービス・コール不正使用 - $tskpri < \text{対象タスクがロックしているミューテックスの上限優先度}$ - $tskpri < \text{対象タスクがロックを待っているミューテックスの上限優先度}$
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。
E_NOEXS	-42	オブジェクト未生成 - $tskid$ のタスクが存在しない。

get_pri
iget_pri

概要

タスク現在優先度の参照

C 言語形式

```
ER    get_pri ( ID tskid, PRI *p_tskpri );
ER    iget_pri ( ID tskid, PRI *p_tskpri );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
O	PRI * <i>p_tskpri</i> ;	現在優先度を格納する領域へのポインタ

機能

tskid で指定されたタスクの現在優先度を *p_tskpri* で指定された領域に格納します。

備考 現在優先度とベース優先度については、「[8.2.2 現在優先度とベース優先度](#)」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_tskpri</i> == NULL
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - 非タスクから本サービス・コールを発行した際、 <i>tskid</i> に TSK_SELF を指定した。

マクロ	数値	意味
E_CTX	-25	<p>コンテキスト・エラー</p> <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 <p>備考 タスクから <code>iget_pri</code> を発行した場合、および非タスクから <code>get_pri</code> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。</p>
E_MACV	-26	<p>メモリ・アクセス違反 (<code>get_pri</code> のみ)</p> <ul style="list-style-type: none"> - 呼び出しタスクからの <code>p_tskpri</code> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_OBJ	-41	<p>オブジェクト状態エラー</p> <ul style="list-style-type: none"> - 対象タスクが DORMANT 状態である。
E_NOEXS	-42	<p>オブジェクト未生成</p> <ul style="list-style-type: none"> - <code>tskid</code> のタスクが存在しない。

ref_tsk
iref_tsk

概要

タスク詳細情報の参照

C 言語形式

```
ER    ref_tsk ( ID tskid, T_RTsk *pk_rtsk );
ER    iref_tsk ( ID tskid, T_RTsk *pk_rtsk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
O	T_RTsk * <i>pk_rtsk</i> ;	タスク詳細情報を格納する領域へのポインタ

【タスク詳細情報 T_RTsk の構造】

```
typedef struct t_rtsk {
    STAT    tskstat;        /* 現在状態 */
    PRI     tskpri;        /* 現在優先度 */
    PRI     tskbpri;       /* ベース優先度 */
    STAT    tsawait;       /* 待ち要因 */
    ID      wobjid;        /* 待ちオブジェクト ID */
    TMO     lefttmo;       /* タイムアウトするまでの時間 */
    UINT    actcnt;        /* 起動要求数 */
    UINT    wupcnt;        /* 起床要求数 */
    UINT    suscnt;        /* サスペンド要求数 */
} T_RTsk;
```

機能

tskid で指定されたタスクのタスク詳細情報（現在状態、現在優先度など）を *pk_rtsk* で指定された領域に格納します。

- *tskstat*

タスクの現在状態が格納されます。

TTS_RUN : RUNNING 状態
TTS_RDY : READY 状態
TTS_WAI : WAITING 状態
TTS_SUS : SUSPENDED 状態
TTS_WAS : WAITING-SUSPENDED 状態
TTS_DMT : DORMANT 状態

- *tskpri*
タスクの現在優先度が格納されます。
tskpri は、*tskstat* が TTS_DMT 以外の場合のみ有効です。

- *tskbpri*
タスクのベース優先度が格納されます。
tskbpri は、*tskstat* が TTS_DMT 以外の場合のみ有効です。

- *tskwait*
タスクの待ち要因 (WAITING 状態の種類) が格納されます。
tskwait は、*tskstat* が TTS_WAI または TTS_WAS の場合のみ有効です。
 - TTW_SLP : *slp_tsk* または *tslp_tsk* による起床待ち状態
 - TTW_DLY : *dly_tsk* による時間経過待ち状態
 - TTW_SEM : *wai_sem* または *twai_sem* による資源獲得待ち状態
 - TTW_FLG : *wai_flg* または *twai_flg* によるイベントフラグ待ち状態
 - TTW_SDTQ : *snd_dtq* または *tsnd_dtq* によるデータ送信待ち状態
 - TTW_RDTQ : *rcv_dtq* または *trcv_dtq* によるデータ受信待ち状態
 - TTW_MBX : *rcv_mbx* または *trcv_mbx* によるメッセージ受信待ち状態
 - TTW_MTX : *loc_mtx* または *tloc_mtx* によるミューテックス待ち状態
 - TTW_SMBF : *snd_mbf* または *tsnd_mbf* によるメッセージ送信待ち状態
 - TTW_RMBF : *rcv_mbf* または *trcv_mbf* によるメッセージ受信待ち状態
 - TTW_MPF : *get_mpf* または *tget_mpf* による固定長メモリ・ブロック獲得待ち状態
 - TTW_MPL : *get_mpl* または *tget_mpl* による可変長メモリ・ブロック獲得待ち状態

- *wobjid*
待ち対象のオブジェクト (セマフォ, イベントフラグなど) の ID が格納されます。
wobjid は、*tskwait* が TTW_SEM, TTW_FLG, TTW_SDTQ, TTW_RDTQ, TTW_MBX, TTW_MTX, TTW_SMBF, TTW_RMBF, TTW_MPF, または TTW_MPL の場合のみ有効です。

- *lefttmo*
タイムアウトまでの残り時間 (単位: ミリ秒) が格納されます。永久待ちの場合は、*TMO_FEVR* が格納されます。
lefttmo は、*tskstat* が TTS_WAI または TTS_WAS の場合で、かつ *tskwait* が TTW_DLY 以外の場合のみ有効です。

備考 TTW_DLY の場合は、*lefttmo* は不定となります。

- *actcnt*
タスクの起動要求数が格納されます。

- *wupcnt*
タスクの起床要求数が格納されます。
wupcnt は、*tskstat* が TTS_DMT 以外の場合のみ有効です。

- *suscnt*
タスクのサスペンド要求数が格納されます。
suscnt は、*tskstat* が TTS_DMT 以外の場合のみ有効です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>pk_rtsk</i> == NULL
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - 非タスクから本サービス・コールを発行した際、 <i>tskid</i> にTSK_SELFを指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_tsk</i> を発行した場合、および非タスクから <i>ref_tsk</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_tsk</i> のみ) - 呼び出しタスクからの <i>pk_rtsk</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

ref_tst
iref_tst

概要

タスク基本情報の参照

C 言語形式

```
ER    ref_tst ( ID tskid, T_RTST *pk_rtst );
ER    iref_tst ( ID tskid, T_RTST *pk_rtst );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
O	T_RTST * <i>pk_rtst</i> ;	タスク基本情報を格納する領域へのポインタ

【タスク基本情報 T_RTST の構造】

```
typedef struct t_rtst {
    STAT    tskstat;        /* 現在状態 */
    STAT    tskwait;       /* 待ち要因 */
} T_RTST;
```

機能

tskid で指定されたタスクのタスク基本情報（現在状態、待ち要因）を *pk_rtst* で指定された領域に格納します。タスク情報のうち、現在状態、待ち要因のみを参照したい場合に使用します。取得する情報が少ないので [ref_tsk](#)、[iref_tsk](#) より高速に応答します。

- *tskstat*

タスクの現在状態が格納されます。

TTS_RUN : RUNNING 状態
TTS_RDY : READY 状態
TTS_WAI : WAITING 状態
TTS_SUS : SUSPENDED 状態
TTS_WAS : WAITING-SUSPENDED 状態
TTS_DMT : DORMANT 状態

- *tskwait*

タスクの待ち要因（WAITING 状態の種類）が格納されます。

tskwait は、*tskstat* が **TTS_WAI** または **TTS_WAS** の場合のみ有効です。

TTW_SLP : [slp_tsk](#) または [tslp_tsk](#) による起床待ち状態

TTW_DLY :	dly_tsk による時間経過待ち状態
TTW_SEM :	wai_sem または twai_sem による資源獲得待ち状態
TTW_FLG :	wai_flg または twai_flg によるイベントフラグ待ち状態
TTW_SDTQ :	snd_dtq または tsnd_dtq によるデータ送信待ち状態
TTW_RDTQ :	rcv_dtq または trcv_dtq によるデータ受信待ち状態
TTW_MBX :	rcv_mbx または trcv_mbx によるメッセージ受信待ち状態
TTW_MTX :	loc_mtx または tloc_mtx によるミューテックス待ち状態
TTW_SMBF :	snd_mbf または tsnd_mbf によるメッセージ送信待ち状態
TTW_RMBF :	rcv_mbf または trcv_mbf によるメッセージ受信待ち状態
TTW_MPF :	get_mpf または tget_mpf による固定長メモリ・ブロック獲得待ち状態
TTW_MPL :	get_mpl または tget_mpl による可変長メモリ・ブロック獲得待ち状態

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>pk_rtst</i> == NULL
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - 非タスクから本サービス・コールを発行した際、 <i>tskid</i> に TSK_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_tst</i> を発行した場合、および非タスクから <i>ref_tst</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_tst</i> のみ) - 呼び出しタスクからの <i>pk_rtst</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

19.2.2 タスク付属同期機能

以下に、RI600PX がタスク付属同期機能として提供しているサービス・コールの一覧を示します。

表 19 - 5 タスク付属同期機能

サービス・コール名	機能概要	発行有効範囲
slp_tsk	起床待ち状態への移行（永久待ち）	タスク
tslp_tsk	起床待ち状態への移行（タイムアウト付き）	タスク
wup_tsk	タスクの起床	タスク
iwup_tsk	タスクの起床	非タスク
can_wup	起床要求の解除	タスク
ican_wup	起床要求の解除	非タスク
rel_wai	WAITING 状態の強制解除	タスク
irel_wai	WAITING 状態の強制解除	非タスク
sus_tsk	SUSPENDED 状態への移行	タスク
isus_tsk	SUSPENDED 状態への移行	非タスク
rsm_tsk	SUSPENDED 状態の解除	タスク
irms_tsk	SUSPENDED 状態の解除	非タスク
frsm_tsk	SUSPENDED 状態の強制解除	タスク
ifrs_tsk	SUSPENDED 状態の強制解除	非タスク
dly_tsk	時間経過待ち状態への移行	タスク

slp_tsk

概要

起床待ち状態への移行（永久待ち）

C 言語形式

```
ER      slp_tsk ( void );
```

パラメータ

なし

機能

自タスクを RUNNING 状態から WAITING 状態（起床待ち状態）へと遷移させます。

ただし、本サービス・コールを発行した際、自タスクの起床要求がキューイングされていた（起床要求カウンタ > 0）場合には、状態操作処理は行わず、起床要求カウンタから 1 を減算します。

なお、起床待ち状態の解除は、以下の場合に行われます。

起床待ち状態の解除操作	戻り値
wup_tsk の発行により、起床要求が発行された。	E_OK
iwup_tsk の発行により、起床要求が発行された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_RLWAI	-49	待ち状態の強制解除 <ul style="list-style-type: none"> - rel_wai または irel_wai の発行により、待ち状態を強制的に解除された。

tslp_tsk

概要

起床待ち状態への移行（タイムアウト付き）

C 言語形式

```
ER      tslp_tsk ( TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR：永久待ち TMO_POL：ポーリング 数値：待ち時間

機能

自タスクを RUNNING 状態からタイムアウト付きの WAITING 状態（起床待ち状態）へと遷移させます。

ただし、本サービス・コールを発行した際、自タスクの起床要求がキューイングされていた（起床要求カウンタ > 0）場合には、状態操作処理は行わず、起床要求カウンタから 1 を減算します。

なお、起床待ち状態の解除は、以下の場合に行われます。

起床待ち状態の解除操作	戻り値
wup_tsk の発行により、起床要求が発行された。	E_OK
iwup_tsk の発行により、起床要求が発行された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
tmout で指定された待ち時間が経過した。	E_TMOUT

備考 待ち時間 *tmout* に TMO_FEVR が指定された際には“slp_tsk と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>tmout</i> < -1 - <i>tmout</i> > (0x7FFFFFFF - TIC_NUME)/TIC_DENO

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗

wup_tsk
iwup_tsk

概要

タスクの起床

C 言語形式

```
ER      wup_tsk ( ID tskid );
ER      iwup_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID

機能

tskid で指定されたタスクを WAITING 状態（起床待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。

ただし、本サービス・コールを発行した際、対象タスクが起床待ち状態以外の場合には、状態操作処理は行わず、起床要求カウンタに 1 を加算します。

備考 RI600PX が管理する起床要求カウンタは、8 ビット幅で構成されています。このため、本サービス・コールでは、起床要求数が 255 回を超える場合には、起床要求のキューイング（起床要求カウンタの加算処理）は行わず、戻り値として E_QOVR を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - iwup_tsk を発行した際、 <i>tskid</i> に TSK_SELF を指定した。

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから iwup_tsk を発行した。 - 非タスクから wup_tsk を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (wup_tsk のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。
E_NOEXS	-42	オブジェクト未生成 - tskid のタスクが存在しない。
E_QOVR	-43	キューイング・オーバフロー - 起床要求数が 255 回を超えた。

can_wup
ican_wup

概要

起床要求の解除

C 言語形式

```
ER_UINT can_wup ( ID tskid );
ER_UINT ican_wup ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID

機能

tskid で指定されたタスクにキューイングされている起床要求をすべて解除（起床要求カウンタに 0 を設定）し、戻り値として解除した起床要求数を返します。

戻り値

マクロ	数値	意味
—	0 以上	正常終了（解除した起床要求数）
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - 非タスクから本サービス・コールを発行した際、 <i>tskid</i> に TSK_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ican_wup を発行した場合、および非タスクから can_wup を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。

マクロ	数値	意味
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

rel_wai
irel_wai

概要

WAITING 状態の強制解除

C 言語形式

```
ER    rel_wai ( ID tskid );
ER    irel_wai ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID

機能

tskid で指定されたタスクの WAITING 状態を強制的に解除します。これにより、対象タスクは待ちキューから外れ、WAITING 状態から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

なお、本サービス・コールの発行により WAITING 状態を解除されたタスクには、WAITING 状態へと遷移するきっかけとなったサービス・コール ([slp_tsk](#)、[wai_sem](#) など) の戻り値として E_RLWAI を返します。

備考 1 本サービス・コールでは、解除要求のキューイングが行われません。このため、対象タスクが WAITING 状態でも WAITING-SUSPENDED 状態でもない場合には、戻り値として E_OBJ を返します。

備考 2 本サービス・コールでは、SUSPENDED 状態の解除は行われません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $tskid \leq 0$ - $tskid > VTMAX_TSK$
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irel_wai を発行した。 - 非タスクから rel_wai を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 (rel_wai のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが, WAITING 状態でも WAITING-SUSPENDED 状態でもない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

sus_tsk
isus_tsk

概要

SUSPENDED 状態への移行

C 言語形式

```
ER    sus_tsk ( ID tskid );
ER    isus_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID

機能

tskid で指定されたタスクを RUNNING 状態から SUSPENDED 状態へ、READY 状態から SUSPENDED 状態へ、または WAITING 状態から WAITING-SUSPENDED 状態へと遷移させます。

ただし、本サービス・コールを発行した際、対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態へと遷移していた場合には、戻り値として E_QOVR を返します。

備考 RI600PX では、サスペンド要求のネストはできません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - isus_tsk を発行した際、 <i>tskid</i> に TSK_SELF を指定した。

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから <code>isus_tsk</code> を発行した。 - 非タスクから <code>sus_tsk</code> を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 - ディスパッチ禁止状態で、自タスクを指定した。
E_MACV	-26	メモリ・アクセス違反 (<code>sus_tsk</code> のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが DORMANT 状態である。 - ディスパッチ禁止状態のときに、 <code>isus_tsk</code> で <code>tskid</code> に実行状態のタスクを指定
E_NOEXS	-42	オブジェクト未生成 - <code>tskid</code> のタスクが存在しない。
E_QOVR	-43	キューイング・オーバーフロー - 対象タスクが SUSPENDED 状態または WAITING-SUSPENDED 状態である。

rsm_tsk
irmsm_tsk

概要

SUSPENDED 状態の解除

C 言語形式

```
ER    rsm_tsk ( ID tskid );
ER    irsm_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID

機能

tskid で指定されたタスクを SUSPENDED 状態から READY 状態へ、または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。

備考 1 本サービス・コールでは、解除要求のキューイングが行われません。このため、対象タスクが SUSPENDED 状態でない場合、および WAITING-SUSPENDED 状態でない場合には、戻り値として E_OBJ を返します。

備考 2 RI600PX では、サスペンド要求のキューイング機能はサポートしていません。サスペンド要求のキューイングも含めて SUSPEND 要求を解除する [frsm_tsk](#)、[ifrsn_tsk](#) もサポートしていますが、これらの振る舞いは [rsm_tsk](#)、[irmsm_tsk](#) と同じです。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>tskid</i> ≤ 0 - <i>tskid</i> > VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irmsm_tsk を発行した。 - 非タスクから rsm_tsk を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 (rsm_tsk のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが SUSPENDED 状態でない。 - 対象タスクが WAITING-SUSPENDED 状態でない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

frsm_tsk
ifrsn_tsk

概要

SUSPENDED 状態の強制解除

C 言語形式

```
ER    frsm_tsk ( ID tskid );
ER    ifrsn_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID

機能

tskid で指定されたタスクを SUSPENDED 状態から READY 状態へ、または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。

- 備考 1 本サービス・コールでは、解除要求のキューイングが行われません。このため、対象タスクが SUSPENDED 状態でも WAITING-SUSPENDED 状態でもない場合には、戻り値として E_OBJ を返します。
- 備考 2 RI600PX では、サスペンド要求のネストはできません。このため、本サービス・コールは `rsm_tsk`、`irsm_tsk` とまったく同じ処理を行います。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $tskid \leq 0$ - $tskid > VTMAX_TSK$
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ifrsn_tsk を発行した。 - 非タスクから frsm_tsk を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 (frsm_tsk のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態エラー - 対象タスクが SUSPENDED 状態でも WAITING-SUSPENDED 状態でもない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

dly_tsk

概要

時間経過待ち状態への移行

C 言語形式

```
ER      dly_tsk ( RELTIM dlytim );
```

パラメータ

I/O	パラメータ	説明
I	RELTIM dlytim;	遅延時間（単位：ミリ秒）

機能

自タスクを *dlytim* で指定された遅延時間が経過するまでの間、RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。

なお、時間経過待ち状態の解除は、以下の場合に行われます。

時間経過待ち状態の解除操作	戻り値
<i>dlytim</i> で指定された遅延時間が経過した。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI

備考 *dlytim* に 0 を指定すると、次の基本クロック割り込み発生までが遅延時間となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>dlytim</i> > (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。

19.2.3 タスク例外処理機能

以下に、RI600PX がタスク例外処理機能として提供しているサービス・コールの一覧を示します。

表 19 - 6 タスク例外処理機能

サービス・コール名	機能概要	発行有効範囲
def_tex	タスク例外処理ルーチンの定義	タスク
ras_tex	タスク例外処理の要求	タスク
iras_tex	タスク例外処理の要求	非タスク
dis_tex	タスク例外処理の禁止	タスク
ena_tex	タスク例外処理の許可	タスク
sns_tex	タスク例外禁止状態の参照	タスク/非タスク
ref_tex	タスク例外処理の詳細情報の参照	タスク
iref_tex	タスク例外処理の詳細情報の参照	非タスク

def_tex

概要

タスク例外処理ルーチンの定義

C 言語形式

```
ER      def_tex ( ID tskid, T_DTEX *pk_dtex );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
I	T_DTEX * <i>pk_dtex</i> ;	NULL : タスク例外処理ルーチン定義解除 NULL 以外 : タスク例外処理ルーチン定義情報を格納した領域へのポインタ

【タスク例外処理ルーチン定義情報 T_DTEX の構造】

```
typedef struct t_dtex {
    ATR    texatr;          /* タスク例外処理ルーチン属性 */
    FP     texrtn;         /* タスク例外処理ルーチンの実行開始アドレス */
} T_DTEX;
```

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。

tskid で指定されたタスクに対して、*pk_dtex* で指定された内容でタスク例外処理ルーチンを定義します。既にタスク例外処理ルーチンが定義されていた場合は、本サービス・コールによって定義内容が更新されます。

ただし、*pk_dtex* に NULL が指定された場合は、*tskid* で指定されたタスクのタスク例外処理ルーチンの定義を解除し、そのタスクの保留例外要因を 0 クリアし、さらにタスク例外禁止状態にします。

- 1) タスク ID (*tskid*)
タスク例外処理ルーチンを定義するタスク ID を指定します。*tskid*=TSK_SELF (=0) の指定により自タスクの指定になります。
- 2) タスク例外処理ルーチン属性 (*texatr*)
texatr には、TA_HLNG のみを指定できます。
- TA_HLNG (= 0x0000)
タスク例外処理ルーチンの記述言語は、C 言語のみをサポートしています。
- 3) タスク例外処理ルーチンの実行開始アドレス (*texrtn*)
texrtn には、タスク例外処理ルーチンの実行開始アドレスを指定します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_RSATR	-11	予約属性 - <i>texatr</i> != TA_HLNG
E_PAR	-17	パラメータ・エラー - <i>pk_dtex</i> != NULL で, <i>texrtn</i> == NULL
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - <i>pk_dtex</i> != NULL で, 呼び出しタスクからの <i>pk_dtex</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは, 信頼されたドメインに所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

ras_tex
iras_tex

概要

タスク例外処理の要求

C 言語形式

```
ER    ras_tex ( ID tskid, TEXPTN rasptn );
ER    iras_tex ( ID tskid, TEXPTN rasptn );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
I	TEXPTN <i>rasptn</i> ;	要求するタスク例外要因

機能

tskid で指定されたタスクに対して、タスク例外処理を要求します。対象タスクの保留例外要因を、*rasptn* で示された値との論理和に更新します。

本サービス・コール発行後、「[6.2.3 タスク例外処理ルーチンの起動条件](#)」に示す条件がすべて揃った場合には、RI600PX はタスク例外処理ルーチンを起動する処理を行います。

タスク例外処理ルーチン起動時には、保留例外要因は 0 クリアされ、タスク例外は禁止されます。タスク例外処理ルーチンには、引数として 0 クリア前の保留例外要因と、タスクの拡張情報が渡されます。

タスク例外処理ルーチンが終了すると、タスク例外許可状態に変更され、そのタスクはタスク例外処理ルーチン起動直前の地点から実行を再開します。

タスク例外処理ルーチンからのリターン時、RI600PX はタスクのユーザ・スタックにタスクのコンテキストレジスタを保存します。この処理においてユーザ・スタックのオーバー・フローを検出した場合は、[システム・ダウン](#)となります。

タスク例外処理ルーチンで CPU ロック状態に移行した場合、タスク例外処理ルーチンからリターンするまでに CPU ロック状態を解除しなければなりません。CPU ロック状態のままタスク例外処理ルーチンからリターンした場合、[システム・ダウン](#)となります。

割り込み優先レベル (PSW.IPL) は、タスク例外処理ルーチンの起動前後、タスク例外処理ルーチンからのリターン前後で変化しません。割り込み優先レベルがカーネル割り込みマスクレベルより高い状態で、タスク例外処理ルーチンからリターンした場合は、[システム・ダウン](#)となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>rasptn</i> == 0

マクロ	数値	意味
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - <i>iras_tex</i> を発行した際、<i>tskid</i> に TSK_SELF を指定した。
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - タスクから <i>iras_tex</i> を発行した。 - 非タスクから <i>ras_tex</i> を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (<i>ras_tex</i> のみ) <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態不正 <ul style="list-style-type: none"> - <i>tskid</i> のタスクが DORMANT 状態である。 - <i>tskid</i> のタスクにタスク例外処理ルーチンが定義されていない。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <i>tskid</i> のタスクが存在しない。

dis_tex**概要**

タスク例外処理の禁止

C 言語形式

```
ER      dis_tex ( void );
```

パラメータ

なし

機能

自タスクのタスク例外処理を禁止します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_OBJ	-41	オブジェクト状態エラー - 自タスクにタスク例外処理ルーチンが定義されていない。

ena_tex

概要

タスク例外処理の許可

C 言語形式

```
ER    ena_tex ( void );
```

パラメータ

なし

機能

自タスクのタスク例外処理を許可します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスキングレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OBJ	-41	オブジェクト状態エラー <ul style="list-style-type: none"> - 自タスクにタスク例外処理ルーチンが定義されていない。

sns_tex**概要**

タスク例外禁止状態の参照

C 言語形式

```
BOOL      sns_tex ( void );
```

パラメータ

なし

機能

本サービス・コールは、戻り値として、RUNNING 状態のタスクがタスク例外禁止状態の場合は TRUE、そうでない場合は FALSE を返します。詳細を表 19-7 に示します。

表 19-7 sns_tex の戻り値

RUNNING 状態のタスク	RUNNING 状態のタスクのタスク例外処理ルーチン	RUNNING 状態のタスクのタスク例外禁止状態	戻り値	備考
存在する	定義済み	許可状態	FALSE	
		禁止状態	TRUE	
	未定義	禁止状態	TRUE	タスク例外処理ルーチンが未定義の時は、タスク例外禁止状態です。
存在しない	—	—	TRUE	

戻り値

マクロ	数値	意味
TRUE	1	正常終了（タスク例外禁止状態）
FALSE	0	正常終了（タスク例外許可状態）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

ref_tex
iref_tex

概要

タスク例外処理の詳細情報の参照

C 言語形式

```
ER    ref_tex ( ID tskid, T_RTEX *pk_rtex );
ER    iref_tex ( ID tskid, T_RTEX *pk_rtex );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
O	T_RTEX * <i>pk_rtex</i> ;	タスク例外処理詳細情報を格納する領域へのポインタ

【タスク基本情報 T_RTEX の構造】

```
typedef struct t_rtex {
    STAT    texstat;          /* タスク例外処理の状態 */
    TEXPTN  pndptn;         /* 保留例外要因 */
} T_RTEX;
```

機能

tskid で指定されたタスクのタスク例外処理の状態を *pk_rtex* で指定された領域に格納します。

- *texstat*

タスク例外処理の状態が格納されます。

TTEX_ENA : タスク例外処理許可状態
TTEX_DIS : タスク例外処理禁止状態

- *pndptn*

保留されている例外要因パターンが格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>pk_rtex</i> == NULL
E_ID	-18	不正 ID 番号 - <i>tskid</i> < 0 - <i>tskid</i> > VTMAX_TSK - 非タスクから本サービス・コールを発行した際、 <i>tskid</i> に TSK_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_rtex</i> を発行した場合、および非タスクから <i>ref_rtex</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_rtex</i> のみ) - 呼び出しタスクからの <i>pk_rtex</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>tskid</i> のタスクが存在しない。

19.2.4 同期通信機能（セマフォ）

以下に、RI600PX が同期通信機能（セマフォ）として提供しているサービス・コールの一覧を示します。

表 19 - 8 同期通信機能（セマフォ）

サービス・コール名	機能概要	発行有効範囲
cre_sem	セマフォの生成	タスク
acre_sem	セマフォの生成（ID 番号自動割付け）	タスク
del_sem	セマフォの削除	タスク
wai_sem	資源の獲得	タスク
pol_sem	資源の獲得（ポーリング）	タスク
ipol_sem	資源の獲得（ポーリング）	非タスク
twai_sem	資源の獲得（タイムアウト付き）	タスク
sig_sem	資源の返却	タスク
isig_sem	資源の返却	非タスク
ref_sem	セマフォ詳細情報の参照	タスク
iref_sem	セマフォ詳細情報の参照	非タスク

cre_sem acre_sem

概要

セマフォの生成

C 言語形式

```
ER      cre_sem ( ID semid, T_CSEM *pk_csem );
ER_ID   acre_sem ( T_CSEM *pk_csem );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>semid</i> ;	セマフォの ID
I	T_CSEM * <i>pk_csem</i> ;	セマフォ生成情報を格納した領域へのポインタ

【セマフォ生成情報 T_CSEM の構造】

```
typedef struct t_csem {
    ATR      sematr;          /* セマフォ属性 */
    UINT     isemcnt;        /* セマフォ資源数の初期値 */
    UINT     maxsem;         /* セマフォ資源数の最大値 */
} T_CSEM;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
 cre_sem は、指定された *semid* のセマフォを *pk_csem* で指定された内容で生成します。acre_sem は *pk_csem* で指定された内容でセマフォを生成し、生成したセマフォ ID を返します。

1) セマフォ属性 (*sematr*)

sematr には以下を指定できます。

```
sematr := ( TA_TFIFO || TA_TPRI )
```

- TA_TFIFO (= 0x0000)

セマフォの待ちキューの順序を FIFO 順とします。

- TA_TPRI (= 0x0001)

セマフォの待ちキューの順序をタスクの現在優先度順とします。ただし、同じ現在優先度のタスクの中では FIFO 順です。

2) セマフォ資源数の初期値 (*isemcnt*)

セマフォ資源数の初期値を、0 ~ *maxsem* の範囲で指定します。

3) セマフォ資源数の最大値 (*maxsem*)

セマフォ資源数の最大値を、1 ~ TMAX_MAXSEM (= 65535) の範囲で指定します。

戻り値

マクロ	数値	意味
—	正の値	acre_sem の正常終了（生成されたセマフォ ID）
E_OK	0	cre_sem の正常終了
E_RSATR	-11	予約属性 - sematr の bit0 以外のビットが 1
E_PAR	-17	パラメータ・エラー - pk_csem == NULL - maxsem == 0 - maxsem > TMAX_MAXSEM - maxsem < isemcnt
E_ID	-18	不正 ID 番号（cre_sem のみ） - semid ≤ 0 - semid > VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの pk_csem が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOID	-34	空き ID なし（acre_sem のみ）
E_OBJ	-41	オブジェクト状態不正（cre_sem のみ） - semid のセマフォが存在する。

del_sem

概要

セマフォの削除

C 言語形式

```
ER      del_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>semid</i> ;	セマフォの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
semid で指定されたセマフォを削除します。

対象セマフォで [wai_sem](#) または [twai_sem](#) によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、[wai_sem](#) または [twai_sem](#) の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>semid</i> ≤ 0 - <i>semid</i> > VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>semid</i> のセマフォが存在しない。

wai_sem

概要

資源の獲得

C 言語形式

```
ER    wai_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>semid</i> ;	セマフォの ID

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。

ただし、本サービス・コールを発行した際、対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には、資源の獲得は行わず、自タスクを対象セマフォの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（資源獲得待ち状態）へと遷移させます。

なお、資源獲得待ち状態の解除は、以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
sig_sem の発行により、対象セマフォに資源が返却された。	E_OK
isig_sem の発行により、対象セマフォに資源が返却された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
del_sem の発行により、待ち状態を強制的に解除された。	E_DLT

備考 自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は、[セマフォの生成時に指定した順](#)（FIFO 順または現在優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - $semid \leq 0$ - $semid > VTMAX_SEM$

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>semid</i> のセマフォが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <i>rel_wai</i> または <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 - <i>del_sem</i> の発行により、待ち状態を強制的に解除された。

pol_sem ipol_sem

概要

資源の獲得（ポーリング）

C 言語形式

```
ER    pol_sem ( ID semid );
ER    ipol_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>semid</i> ;	セマフォの ID

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。

ただし、本サービス・コールを発行した際、対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には、資源の獲得は行わず、戻り値として E_TMOUT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>semid</i> ≤ 0 - <i>semid</i> > VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ipol_sem を発行した場合、および非タスクから pol_sem を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_NOEXS	-42	オブジェクト未生成 - <i>semid</i> のセマフォが存在しない。
E_TMOUT	-50	ポーリング失敗

twai_sem

概要

資源の獲得（タイムアウト付き）

C 言語形式

```
ER twai_sem ( ID semid, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>semid</i> ;	セマフォの ID
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR ： 永久待ち TMO_POL ： ポーリング 数値： 待ち時間

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 1 を減算）します。

ただし、本サービス・コールを発行した際、対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には、資源の獲得は行わず、自タスクを対象セマフォの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（資源獲得待ち状態）へと遷移させます。

なお、資源獲得待ち状態の解除は、以下の場合に行われます。

資源獲得待ち状態の解除操作	戻り値
sig_sem の発行により、対象セマフォに資源が返却された。	E_OK
isig_sem の発行により、対象セマフォに資源が返却された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>tmout</i> で指定された待ち時間が経過した。	E_TMOUT
del_sem の発行により、待ち状態を強制的に解除された。	E_DLT

備考 1 自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は、[セマフォの生成時に指定した順](#)（FIFO 順または現在優先度順）に行われます。

備考 2 待ち時間 *tmout* に **TMO_FEVR** が指定された際には“[wai_sem](#) と同等の処理”を、**TMO_POL** が指定された際には“[pol_sem](#) と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>tmout</i> < -1 - <i>tmout</i> > (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - <i>semid</i> ≤ 0 - <i>semid</i> > VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>semid</i> のセマフォが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <i>rel_wai</i> または <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 - <i>del_sem</i> の発行により、待ち状態を強制的に解除された。

sig_sem isig_sem

概要

資源の返却

C 言語形式

```
ER    sig_sem ( ID semid );
ER    isig_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>semid</i> ;	セマフォの ID

機能

semid で指定されたセマフォに資源を返却（セマフォ・カウンタに 1 を加算）します。

ただし、本サービス・コールを発行した際、対象セマフォの待ちキューにタスクがキューイングされていた場合には、資源の返却（セマフォ・カウンタの加算処理）は行わず、該当タスク（待ちキューの先頭タスク）に資源を渡します。これにより、該当タスクは、待ちキューから外れ、WAITING 状態（資源獲得待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 RI600PX では、セマフォの資源数として取り得る最大値をセマフォの生成時に指定します。このため、本サービス・コールでは、資源数が最大資源数を超える場合には、資源の返却（セマフォ・カウンタの加算処理）は行わず、戻り値として E_QOVR を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>semid</i> ≤ 0 - <i>semid</i> > VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから isig_sem を発行した。 - 非タスクから sig_sem を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 (sig_sem のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - semid のセマフォが存在しない。
E_QOVR	-43	キューイング・オーバーフロー - 資源数が最大値を超えた。

ref_sem
iref_sem

概要

セマフォ詳細情報の参照

C 言語形式

```
ER    ref_sem ( ID semid, T_RSEM *pk_rsem );
ER    iref_sem ( ID semid, T_RSEM *pk_rsem );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>semid</i> ;	セマフォの ID
O	T_RSEM * <i>pk_rsem</i> ;	セマフォ詳細情報を格納する領域へのポインタ

【セマフォ詳細情報 T_RSEM の構造】

```
typedef struct t_rsem {
    ID    wtskid;          /* 待ちタスクの有無 */
    UINT  semcnt;        /* 現在資源数 */
} T_RSEM;
```

機能

semid で指定されたセマフォのセマフォ詳細情報（待ちタスクの有無、現在資源数など）を *pk_rsem* で指定された領域に格納します。

- *wtskid*
セマフォの待ちキューにタスクがキューイングされているか否かが格納されます。
TSK_NONE : 待ちキューにタスクはキューイングされていない
 その他 : 待ちキューの先頭にキューイングされているタスクの ID
- *semcnt*
セマフォの現在資源数が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>pk_rsem</i> == NULL
E_ID	-18	不正 ID 番号 - <i>semid</i> ≤ 0 - <i>semid</i> > VTMAX_SEM
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_sem</i> を発行した場合、および非タスクから <i>ref_sem</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_sem</i> のみ) - 呼び出しタスクからの <i>pk_rsem</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>semid</i> のセマフォが存在しない。

19.2.5 同期通信機能（イベントフラグ）

以下に、RI600PX が同期通信機能（イベントフラグ）として提供しているサービス・コールの一覧を示します。

表 19 - 9 同期通信機能（イベントフラグ）

サービス・コール名	機能概要	発行有効範囲
cre_flg	イベントフラグの生成	タスク
acre_flg	イベントフラグの生成（ID 番号自動割付け）	タスク
del_flg	イベントフラグの削除	タスク
set_flg	ビット・パターンのセット	タスク
iset_flg	ビット・パターンのセット	非タスク
clr_flg	ビット・パターンのクリア	タスク
iclr_flg	ビット・パターンのクリア	非タスク
wai_flg	ビット・パターンのチェック	タスク
pol_flg	ビット・パターンのチェック（ポーリング）	タスク
ipol_flg	ビット・パターンのチェック（ポーリング）	非タスク
twai_flg	ビット・パターンのチェック（タイムアウト付き）	タスク
ref_flg	イベントフラグ詳細情報の参照	タスク
iref_flg	イベントフラグ詳細情報の参照	非タスク

cre_flg acre_flg

概要

イベントフラグの生成

C 言語形式

```
ER      cre_flg ( ID flgid, T_CFLG *pk_cflg );
ER_ID  acre_flg ( T_CFLG *pk_cflg );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID
I	T_CFLG * <i>pk_cflg</i> ;	イベントフラグ生成情報を格納した領域へのポインタ

【 イベントフラグ生成情報 T_CFLG の構造 】

```
typedef struct t_cflg {
    ATR    flgatr;          /* イベントフラグ属性 */
    FLGPTN iflgptn;       /* 初期ビット・パターン */
} T_CFLG;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

`cre_flg` は、指定された `flgid` のイベントフラグを `pk_cflg` で指定された内容で生成します。`acre_flg` は `pk_cflg` で指定された内容でイベントフラグを生成し、生成したイベントフラグ ID を返します。

1) イベントフラグ属性 (`flgatr`)

`flgatr` には以下を指定できます。

```
flgatr := ( ( TA_TFIFO || TA_TPRI ) | ( TA_WSGL || TA_WMUL ) | [TA_CLR] )
```

- TA_TFIFO (= 0x0000)

イベントフラグの待ちキューの順序を FIFO 順とします。

- TA_TPRI (= 0x0001)

イベントフラグの待ちキューの順序をタスクの現在優先度順とします。ただし、同じ現在優先度のタスクの中では FIFO 順です。

なお、TA_CLR の指定がない場合は、TA_TPRI を指定してもタスクの待ち行列は FIFO 順で管理されます。この振る舞いは、μITRON4.0 仕様の範囲外です。

- TA_WSGL (= 0x0000)

イベントフラグに対し、複数のタスクが待つことを禁止します。

- TA_WMUL (= 0x0002)

イベントフラグに対し、複数のタスクが待つことを許可します。

- TA_CLR (= 0x0004)
wai_flg, pol_flg, ipol_flg, および twai_flg が正常終了するときに、イベントフラグのすべてのビットをクリアします。

- 2) 初期ビット・パターン (iflgptn)
イベントフラグの初期ビット・パターンを指定します。

戻り値

マクロ	数値	意味
—	正の値	acre_flg の正常終了 (生成されたイベントフラグ ID)
E_OK	0	cre_flg の正常終了
E_RSATR	-11	予約属性 - flgatr の bit0, bit1, bit2 以外のビットが 1
E_PAR	-17	パラメータ・エラー - pk_cfg == NULL
E_ID	-18	不正 ID 番号 (cre_flg のみ) - flgid ≤ 0 - flgid > VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの pk_cfg が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOID	-34	空き ID なし (acre_flg のみ)
E_OBJ	-41	オブジェクト状態不正 (cre_flg のみ) - flgid のイベントフラグが存在する。

del_flg**概要**

イベントフラグの削除

C 言語形式

```
ER      del_flg ( ID flgid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
flgid で指定されたイベントフラグを削除します。

対象イベントフラグで [wai_flg](#) または [twai_flg](#) によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、[wai_flg](#) または [twai_flg](#) の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>flgid</i> ≤ 0 - <i>flgid</i> > VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>flgid</i> のイベントフラグが存在しない。

set_flg iset_flg

概要

ビット・パターンのセット

C 言語形式

```
ER    set_flg ( ID flgid, FLGPTN setptn );
ER    iset_flg ( ID flgid, FLGPTN setptn );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID
I	FLGPTN <i>setptn</i> ;	セットするビット・パターン

機能

flgid で指定されたイベントフラグのビット・パターンと *setptn* で指定されたビット・パターンの論理和 OR をとり、その結果を対象イベントフラグにセットします。

そして、待ちキューの順に待ちキューにつながれているタスクの待ち解除条件を満たすかどうかを調べます。待ち解除条件を満たせば、該当タスクを待ちキューから外し、WAITING 状態（イベントフラグ待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。このとき、対象のイベントフラグ属性に [TA_CLR](#) 属性が指定されている場合には、イベントフラグのビット・パターンを 0 クリアし、処理を終了します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>flgid</i> ≤ 0 - <i>flgid</i> > VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから <code>iset_flg</code> を発行した。 - 非タスクから <code>set_flg</code> を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 (set_flg のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - flgid のイベントフラグが存在しない。

clr_flg
iclr_flg

概要

ビット・パターンのクリア

C 言語形式

```
ER      clr_flg ( ID flgid, FLGPTN clrptn );
ER      iclr_flg ( ID flgid, FLGPTN clrptn );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID
I	FLGPTN <i>clrptn</i> ;	クリアするビット・パターン

機能

flgid で指定されたイベントフラグのビット・パターンと *clrptn* で指定されたビット・パターンの論理積 AND をとり、その結果を対象イベントフラグに設定します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>flgid</i> ≤ 0 - <i>flgid</i> > VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iclr_flg</i> を発行した場合、および非タスクから <i>clr_flg</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_NOEXS	-42	オブジェクト未生成 - <i>flgid</i> のイベントフラグが存在しない。

wai_flg

概要

ビット・パターンのチェック

C 言語形式

```
ER      wai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID
I	FLGPTN <i>waiptn</i> ;	要求するビット・パターン
I	MODE <i>wfmode</i> ;	要求条件の指定 TWF_ANDW : AND 待ち TWF_ORW : OR 待ち
O	FLGPTN <i>*p_flgptn</i> ;	条件成立時のビット・パターンを格納する領域へのポインタ

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお、要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には、対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には、自タスクを対象イベントフラグの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移させます。

なお、イベントフラグ待ち状態の解除は、以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
<i>set_flg</i> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<i>iset_flg</i> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>del_flg</i> の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、要求条件 *wfmode* の指定形式を示します。

- *wfmode* == TWF_ANDW

waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* == *TWF_ORW*

waiptn で 1 を設定しているビットのうち、いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1 RI600PX では、イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをイベントフラグの生成時に指定します。このため、すでに待ちタスクがキューイングされているイベントフラグ (*TA_WSGL* 属性) に対して本サービス・コールを発行した場合、RI600PX は要求条件の即時成立／不成立を問わず、戻り値として *E_ILUSE* を返します。

TA_WSGL 属性： キューイング可能なタスクは、1 個

TA_WMUL 属性： キューイング可能なタスクは、複数

備考 2 自タスクを対象イベントフラグ (*TA_WMUL* 属性) の待ちキューにキューイングする際のキューイング方式は、イベントフラグの生成時に指定された順 (FIFO 順または現在優先度順) に行われます。ただし、*TA_CLR* 属性が指定されていない場合は、優先度順の指定の場合でも FIFO 順に行われます。この振る舞いは、 μ ITRON4.0 仕様の範囲外です。

備考 3 対象イベントフラグ (*TA_CLR* 属性) の要求条件が満足した際、RI600PX はビット・パターンのクリア (0 の設定) を行います。

戻り値

マクロ	数値	意味
<i>E_OK</i>	0	正常終了
<i>E_PAR</i>	-17	パラメータ・エラー - <i>waiptn</i> == 0 - 要求条件 <i>wfmode</i> の指定が不正である。 - <i>p_flgptn</i> == NULL
<i>E_ID</i>	-18	不正 ID 番号 - <i>flgid</i> ≤ 0 - <i>flgid</i> > <i>VTMAX_FLG</i>
<i>E_CTX</i>	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
<i>E_MACV</i>	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>p_flgptn</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
<i>E_ILUSE</i>	-28	サービス・コール不正使用 - すでに待ちタスクがキューイングされているイベントフラグ (<i>TA_WSGL</i> 属性) に対して本サービス・コールを発行した。
<i>E_NOEXS</i>	-42	オブジェクト未生成 - <i>flgid</i> のイベントフラグが存在しない。
<i>E_RLWAI</i>	-49	待ち状態の強制解除 - <i>rel_wai</i> または <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。

マクロ	数値	意味
E_DLT	-51	待ちオブジェクトの削除 - <code>del_flg</code> の発行により、待ち状態を強制的に解除された。

pol_flg ipol_flg

概要

ビット・パターンのチェック（ポーリング）

C 言語形式

```
ER    pol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER    ipol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID
I	FLGPTN <i>waiptn</i> ;	要求するビット・パターン
I	MODE <i>wfmode</i> ;	要求条件の指定 TWF_ANDW : AND 待ち TWF_ORW : OR 待ち
O	FLGPTN <i>*p_flgptn</i> ;	条件成立時のビット・パターンを格納する領域へのポインタ

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお、要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には、対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には、戻り値として E_TMOUT を返します。

以下に、要求条件 *wfmode* の指定形式を示します。

- *wfmode* == TWF_ANDW

waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* == TWF_ORW

waiptn で 1 を設定しているビットのうち、いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1 RI600PXでは、イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため、すでに待ちタスクがキューイングされているイベントフラグ (TA_WSGL 属性) に対して本サービス・コールを発行した場合、RI600PX は要求条件の即時成立／不成立を問わず、戻り値として E_ILUSE を返します。

TA_WSGL 属性 : キューイング可能なタスクは、1 個

TA_WMUL 属性 : キューイング可能なタスクは、複数

備考 2 対象イベントフラグ (TA_CLR 属性) の要求条件が満足した際、RI600PX はビット・パターンのクリア (0 の設定) を行います。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>waitpn</i> == 0 - 要求条件 <i>wfmode</i> の指定が不正である。 - <i>p_flgptn</i> == NULL
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <i>flgid</i> ≤ 0 - <i>flgid</i> > VTMAX_FLG
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 <p>備考 タスクから <i>ipol_flg</i> を発行した場合、および非タスクから <i>pol_flg</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。</p>
E_MACV	-26	メモリ・アクセス違反 (<i>pol_flg</i> のみ) <ul style="list-style-type: none"> - 呼び出しタスクからの <i>p_flgptn</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_ILUSE	-28	サービス・コール不正使用 <ul style="list-style-type: none"> - すでに待ちタスクがキューイングされているイベントフラグ (<i>TA_WSGL</i> 属性) に対して本サービス・コールを発行した。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <i>flgid</i> のイベントフラグが存在しない。
E_TMOUT	-50	ポーリング失敗

twai_flg

概要

ビット・パターンのチェック（タイムアウト付き）

C 言語形式

```
ER twai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID
I	FLGPTN <i>waiptn</i> ;	要求するビット・パターン
I	MODE <i>wfmode</i> ;	要求条件の指定 TWF_ANDW : AND 待ち TWF_ORW : OR 待ち
O	FLGPTN <i>*p_flgptn</i> ;	条件成立時のビット・パターンを格納する領域へのポインタ
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR : 永久待ち TMO_POL : ポーリング 数値 : 待ち時間

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお、要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には、対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には、自タスクを対象イベントフラグの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（イベントフラグ待ち状態）へと遷移させます。

なお、イベントフラグ待ち状態の解除は、以下の場合に行われます。

イベントフラグ待ち状態の解除操作	戻り値
<i>set_flg</i> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<i>iset_flg</i> の発行により、対象イベントフラグに要求条件を満足するビット・パターンが設定された。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>tmout</i> で指定された待ち時間が経過した。	E_TMOUT

イベントフラグ待ち状態の解除操作	戻り値
del_flg の発行により、待ち状態を強制的に解除された。	E_DLT

以下に、要求条件 *wfmode* の指定形式を示します。

- *wfmode* == TWF_ANDW
waitptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
- *wfmode* == TWF_ORW
waitptn で 1 を設定しているビットのうち、いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1 RI600PX では、イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをイベントフラグの生成時に指定します。このため、すでに待ちタスクがキューイングされているイベントフラグ (TA_WSGL 属性) に対して本サービス・コールを発行した場合、RI600PX は要求条件の即時成立／不成立を問わず、戻り値として E_ILUSE を返します。

TA_WSGL 属性： キューイング可能なタスクは、1 個

TA_WMUL 属性： キューイング可能なタスクは、複数

備考 2 自タスクを対象イベントフラグ (TA_WMUL 属性) の待ちキューにキューイングする際のキューイング方式は、イベントフラグの生成時に指定された順 (FIFO 順または現在優先度順) に行われます。

ただし、TA_CLR 属性が指定されていない場合は、優先度順の指定の場合でも FIFO 順に行われます。この振る舞いは、μITRON4.0 仕様の範囲外です。

備考 3 対象イベントフラグ (TA_CLR 属性) の要求条件が満足した際、RI600PX はビット・パターンのクリア (0 の設定) を行います。

備考 4 待ち時間 *tmout* に TMO_FEVR が指定された際には“wai_flg と同等の処理”を、TMO_POL が指定された際には“pol_flg と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>waitptn</i> == 0 - 要求条件 <i>wfmode</i> の指定が不正である。 - <i>p_flgptn</i> == NULL - <i>tmout</i> < -1 - <i>tmout</i> > (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <i>flgid</i> ≤ 0 - <i>flgid</i> > VTMAX_FLG
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>p_flgptn</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_ILUSE	-28	サービス・コール不正使用 <ul style="list-style-type: none"> - すでに待ちタスクがキューイングされているイベントフラグ (<i>TA_WSGL</i> 属性) に対して本サービス・コールを発行した。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <i>flgid</i> のイベントフラグが存在しない。
E_RLWAI	-49	待ち状態の強制解除 <ul style="list-style-type: none"> - <i>rel_wai</i> または <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 <ul style="list-style-type: none"> - <i>del_flg</i> の発行により、待ち状態を強制的に解除された。

ref_flg
iref_flg

概要

イベントフラグ詳細情報の参照

C 言語形式

```
ER    ref_flg ( ID flgid, T_RFLG *pk_rflg );
ER    iref_flg ( ID flgid, T_RFLG *pk_rflg );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>flgid</i> ;	イベントフラグの ID
O	T_RFLG * <i>pk_rflg</i> ;	イベントフラグ詳細情報を格納する領域へのポインタ

【 イベントフラグ詳細情報 T_RFLG の構造 】

```
typedef struct t_rflg {
    ID      wtskid;          /* 待ちタスクの有無 */
    FLGPTN flgptn;         /* 現在ビット・パターン */
} T_RFLG;
```

機能

flgid で指定されたイベントフラグのイベントフラグ詳細情報（待ちタスクの有無、現在ビット・パターンなど）を *pk_rflg* で指定された領域に格納します。

- wtskid

イベントフラグの待ちキューにタスクがキューイングされているか否かが格納されます。

TSK_NONE : 待ちキューにタスクはキューイングされていない
 その他 : 待ちキューの先頭にキューイングされているタスクの ID

- flgptn

イベントフラグの現在ビット・パターンが格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>pk_rflg</i> == NULL
E_ID	-18	不正 ID 番号 - <i>flgid</i> ≤ 0 - <i>flgid</i> > VTMAX_FLG
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_flg</i> を発行した場合、および非タスクから <i>ref_flg</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_flg</i> のみ) - 呼び出しタスクからの <i>pk_rflg</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>flgid</i> のイベントフラグが存在しない。

19.2.6 同期通信機能（データ・キュー）

以下に、RI600PX が同期通信機能（データ・キュー）として提供しているサービス・コールの一覧を示します。

表 19 - 10 同期通信機能（データ・キュー）

サービス・コール名	機能概要	発行有効範囲
cre_dtq	データ・キューの生成	タスク
acre_dtq	データ・キューの生成（ID 番号自動割付け）	タスク
del_dtq	データ・キューの削除	タスク
snd_dtq	データの送信	タスク
psnd_dtq	データの送信（ポーリング）	タスク
ipsnd_dtq	データの送信（ポーリング）	非タスク
tsnd_dtq	データの送信（タイムアウト付き）	タスク
fsnd_dtq	データの強制送信	タスク
ifsnd_dtq	データの強制送信	非タスク
rcv_dtq	データの受信	タスク
prcv_dtq	データの受信（ポーリング）	タスク
iprcv_dtq	データの受信（ポーリング）	非タスク
trcv_dtq	データの受信（タイムアウト付き）	タスク
ref_dtq	データ・キュー詳細情報の参照	タスク
iref_dtq	データ・キュー詳細情報の参照	非タスク

cre_dtq
acre_dtq

概要

データ・キューの生成

C 言語形式

```
ER      cre_dtq ( ID dtqid, T_CDTQ *pk_cdtq );
ER_ID   acre_dtq ( T_CDTQ *pk_cdtq );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
I	T_CDTQ * <i>pk_cdtq</i> ;	データ・キュー生成情報を格納した領域へのポインタ

【データ・キュー生成情報 T_CDTQ の構造】

```
typedef struct t_cdtq {
    ATR    dtqatr;          /* データ・キュー属性 */
    UINT   dtqcnt;         /* データ・キュー領域の容量 (データの個数) */
    VP     dtq;            /* データ・キュー領域の先頭アドレス */
} T_CDTQ;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

`cre_dtq` は、指定された `dtqid` のデータ・キューを `pk_cdtq` で指定された内容で生成します。`acre_dtq` は `pk_cdtq` で指定された内容でデータ・キューを生成し、生成したデータ・キュー ID を返します。

1) データ・キュー属性 (`dtqatr`)

`dtqatr` には以下を指定できます。

```
dtqatr := ( TA_TFIFO || TA_TPRI )
```

- TA_TFIFO (= 0x0000)

データ・キューの送信待ちキューの順序を FIFO 順とします。

- TA_TPRI (= 0x0001)

データ・キューの送信待ちキューの順序をタスクの現在優先度順とします。ただし、同じ現在優先度のタスクの中では FIFO 順です。

備考 データ・キューの受信待ちキューは FIFO 順です。

2) データ・キュー領域の容量 (`dtqcnt`)、データ・キュー領域の先頭アドレス (`dtq`)

アプリケーション側で、TSZ_DTQ(`dtqcnt`) バイトのデータ・キュー領域を確保し、その先頭アドレスを `dtq` に指定します。

また、*dtqcnt* に 0 を指定することもできます。この場合、データ・キューにデータを蓄えておくことはできないため、送信側と受信側の先に実行した方が待ち状態になり、他方が行われた時点で待ちが解除される、つまり送信側と受信側が完全に同期した動作となります。なお、*dtqcnt* が 0 の場合は、*dtq* は無視されます。

備考 1 TSZ_DTQ マクロについては、「18.3.2 データ・キュー関連」を参照してください。

備考 2 RI600PX は、データ・キュー領域に関するアクセス権については何も関知しません。通常は、データ・キュー領域はメモリ・オブジェクト以外でかつユーザ・スタック以外の領域に作成してください。メモリ・オブジェクト内にデータ・キュー領域を作成した場合は、そのメモリ・オブジェクトへのオペランド・ライト・アクセスが許可されたタスクが、誤ってデータ・キュー領域を書き換えてしまう危険があります。

備考 3 μ ITRON4.0 仕様には、*dtq* に NULL を指定することで RI600PX がデータ・キュー領域を割り当てる機能がありますが、本サービス・コールはこの機能をサポートしていません。

戻り値

マクロ	数値	意味
—	正の値	acre_dtq の正常終了 (生成されたデータ・キュー ID)
E_OK	0	cre_dtq の正常終了
E_RSATR	-11	予約属性 - <i>dtqatr</i> の bit0 以外のビットが 1
E_PAR	-17	パラメータ・エラー - <i>pk_cdtq</i> == NULL - <i>dtqcnt</i> > 65535 - <i>dtqcnt</i> != 0 かつ <i>dtq</i> + TSZ_DTQ(<i>dtqcnt</i>) > 0x100000000
E_ID	-18	不正 ID 番号 (<i>cre_dtq</i> のみ) - <i>dtqid</i> ≤ 0 - <i>dtqid</i> > VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_cdtq</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOMEM	-33	メモリ不足 - <i>dtqcnt</i> != 0 かつ <i>dtq</i> == NULL
E_NOID	-34	空き ID なし (<i>acre_dtq</i> のみ)
E_OBJ	-41	オブジェクト状態不正 (<i>cre_dtq</i> のみ) - <i>dtqid</i> のデータ・キューが存在する。

del_dtq

概要

データ・キューの削除

C 言語形式

```
ER      del_dtq ( ID dtqid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
dtqid で指定されたデータ・キューを削除します。

対象データ・キューで *snd_dtq*, *tsnd_dtq*, *rcv_dtq*, または *trcv_dtq* によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、*snd_dtq*, *tsnd_dtq*, *rcv_dtq*, または *trcv_dtq* の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>dtqid</i> ≤ 0 - <i>dtqid</i> > VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>dtqid</i> のデータ・キューが存在しない。

snd_dtq

概要

データの送信

C 言語形式

```
ER      snd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
I	VP_INT <i>data</i> ;	送信するデータ

機能

dtqid で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態（データ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がある場合
パラメータ *data* で指定されたデータをデータ・キューに格納します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
自タスクを対象データ・キューの送信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（データ送信待ち状態）へと遷移させます。
なお、データ送信待ち状態の解除は、以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
<i>rcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>prcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>iprcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>trcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>vrst_dtq</i> の発行により、対象データ・キューがリセットされた。	EV_RST
<i>del_dtq</i> の発行により、待ち状態を強制的に解除された。	E_DLT

- 備考1 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は、データの送信要求を行った順に行われます。
- 備考2 自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は、データ・キューの生成時に指定された順（FIFO 順または現在優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $dtqid \leq 0$ - $dtqid > VTMAX_DTQ$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - $dtqid$ のデータ・キューが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 - <code>del_dtq</code> の発行により、待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (<code>vrst_dtq</code>) による待ち解除

psnd_dtq
ipsnd_dtq

概要

データの送信（ポーリング）

C 言語形式

```
ER    psnd_dtq ( ID dtqid, VP_INT data );
ER    ipsnd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
I	VP_INT <i>data</i> ;	送信するデータ

機能

dtqid で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態（データ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がある場合
パラメータ *data* で指定されたデータをデータ・キューに格納します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
戻り値として E_TMOUT を返します。

備考 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は、データの送信要求を行った順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $dtqid \leq 0$ - $dtqid > VTMAX_DTQ$

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから ipsnd_dtq を発行した。 - 非タスクから psnd_dtq を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (psnd_dtq のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - dtqid のデータ・キューが存在しない。
E_TMOUT	-50	ポーリング失敗

tsnd_dtq

概要

データの送信（タイムアウト付き）

C 言語形式

```
ER      tsnd_dtq ( ID dtqid, VP_INT data, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
I	VP_INT <i>data</i> ;	送信するデータ
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

dtqid で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態（データ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がある場合
パラメータ *data* で指定されたデータをデータ・キューに格納します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、データ・キューに空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
自タスクを対象データ・キューの送信待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（データ送信待ち状態）へと遷移させます。
なお、データ送信待ち状態の解除は、以下の場合に行われます。

データ送信待ち状態の解除操作	戻り値
<i>rcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>prcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>iprcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>trcv_dtq</i> の発行により、対象データ・キューに空き領域が確保された。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI

データ送信待ち状態の解除操作	戻り値
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_dtq</code> の発行により、対象データ・キューがリセットされた。	EV_RST
<code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_dtq</code> の発行により、待ち状態を強制的に解除された。	E_DLT

- 備考 1 データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は、データの送信要求を行った順に行われます。
- 備考 2 自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は、データ・キューの生成時に指定された順（FIFO 順または現在優先度順）に行われます。
- 備考 3 待ち時間 `tmout` に `TMO_FEVR` が指定された際には“`snd_dtq` と同等の処理”を、`TMO_POL` が指定された際には“`psnd_dtq` と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <code>tmout</code> < -1 - <code>tmout</code> > (0x7FFFFFFF - <code>TIC_NUME</code>)/ <code>TIC_DENO</code>
E_ID	-18	不正 ID 番号 - <code>dtqid</code> ≤ 0 - <code>dtqid</code> > <code>VTMAX_DTQ</code>
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <code>dtqid</code> のデータ・キューが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 - <code>del_dtq</code> の発行により、待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (<code>vrst_dtq</code>) による待ち解除

fsnd_dtq
ifsnd_dtq

概要

データの強制送信

C 言語形式

```
ER    fsnd_dtq ( ID dtqid, VP_INT data );
ER    ifsnd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
I	VP_INT <i>data</i> ;	送信するデータ

機能

dtqid で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
パラメータ *data* で指定されたデータを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは、受信待ちキューから外れ、WAITING 状態（データ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされていない場合
パラメータ *data* で指定されたデータをデータ・キューに格納します。
データ・キューに空きがない場合は、最古のデータを削除してからデータを格納します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $dtqid \leq 0$ - $dtqid > VTMAX_DTQ$

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した - タスクから ifsnd_dtq を発行した - 非タスクから fsnd_dtq を発行した - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した
E_MACV	-26	メモリ・アクセス違反 (fsnd_dtq のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_ILUSE	-28	サービス・コール不正使用 - データ・キュー領域のデータ数が 0 のデータ・キューに対して本サービス・コールを発行した
E_NOEXS	-42	オブジェクト未生成 - dtqid のデータ・キューが存在しない。

rcv_dtq

概要

データの受信

C 言語形式

```
ER      rcv_dtq ( ID dtqid, VP_INT *p_data );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
O	VP_INT * <i>p_data</i> ;	データを格納する領域へのポインタ

機能

dtqid で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- データ・キューにデータが格納されている場合
データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
送信待ちキューにタスクがキューイングされている場合は、送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち、WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（データ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、データ・キューの容量が 0 の場合のみ生じます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
自タスクを対象データ・キューの受信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。
なお、データ受信待ち状態の解除は、以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
<i>snd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>psnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>ipsnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>tsnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>fsnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>ifsnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI

データ受信待ち状態の解除操作	戻り値
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
del_dtq の発行により、待ち状態を強制的に解除された。	E_DLT

備考 自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は、データの受信要求を行った順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - p_data == NULL
E_ID	-18	不正 ID 番号 - dtqid ≤ 0 - dtqid > VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの p_data が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - dtqid のデータ・キューが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 - del_dtq の発行により、待ち状態を強制的に解除された。

prcv_dtq
iprcv_dtq

概要

データの受信（ポーリング）

C 言語形式

```
ER    prcv_dtq ( ID dtqid, VP_INT *p_data );
ER    iprcv_dtq ( ID dtqid, VP_INT *p_data );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
O	VP_INT * <i>p_data</i> ;	データを格納する領域へのポインタ

機能

dtqid で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- データ・キューにデータが格納されている場合
データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
送信待ちキューにタスクがキューイングされている場合は、送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち、WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（データ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、データ・キューの容量が 0 の場合のみ生じます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
戻り値として E_TMOUT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_data</i> == NULL

マクロ	数値	意味
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - $dtqid \leq 0$ - $dtqid > VTMAX_DTQ$
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - タスクから iprcv_dtq を発行した。 - 非タスクから prcv_dtq を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (prcv_dtq のみ) <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの p_data が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - $dtqid$ のデータ・キューが存在しない。
E_TMOUT	-50	ポーリング失敗

trcv_dtq

概要

データの受信（タイムアウト付き）

C 言語形式

```
ER      trcv_dtq ( ID dtqid, VP_INT *p_data, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
O	VP_INT * <i>p_data</i> ;	データを格納する領域へのポインタ
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

dtqid で指定されたデータ・キューの状況に応じて、以下の処理を行います。

- データ・キューにデータが格納されている場合
データ・キューから最古のデータを取り出して *p_data* で指定された領域に格納します。
送信待ちキューにタスクがキューイングされている場合は、送信待ちキュー先頭タスクの送信データをデータ・キューに格納したのち、WAITING 状態（データ送信待ち状態）から READY 状態へと遷移させます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したデータを *p_data* で指定された領域に格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（データ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、データ・キューの容量が 0 の場合のみ生じます。
- データ・キューにデータが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
自タスクを対象データ・キューの受信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。
なお、データ受信待ち状態の解除は、以下の場合に行われます。

データ受信待ち状態の解除操作	戻り値
<i>snd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>psnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>ipsnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK
<i>tsnd_dtq</i> の発行により、対象データ・キューにデータが送信された。	E_OK

データ受信待ち状態の解除操作	戻り値
<code>fsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>ifsnd_dtq</code> の発行により、対象データ・キューにデータが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_dtq</code> の発行により、待ち状態を強制的に解除された。	E_DLT

備考1 自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は、データの受信要求を行った順に行われます。

備考2 待ち時間 `tmout` に `TMO_FEVR` が指定された際には“`rcv_dtq` と同等の処理”を、`TMO_POL` が指定された際には“`prcv_dtq` と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <code>p_data == NULL</code> - <code>tmout < -1</code> - <code>tmout > (0x7FFFFFFF - TIC_NUME)/TIC_DENO</code>
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <code>dtqid ≤ 0</code> - <code>dtqid > VTMAX_DTQ</code>
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <code>p_data</code> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <code>dtqid</code> のデータ・キューが存在しない。
E_RLWAI	-49	待ち状態の強制解除 <ul style="list-style-type: none"> - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗

マクロ	数値	意味
E_DLT	-51	待ちオブジェクトの削除 - <code>del_dtq</code> の発行により、待ち状態を強制的に解除された。

ref_dtq
iref_dtq

概要

データ・キュー詳細情報の参照

C 言語形式

```
ER    ref_dtq ( ID dtqid, T_RDTQ *pk_rdtq );
ER    iref_dtq ( ID dtqid, T_RDTQ *pk_rdtq );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID
O	T_RDTQ * <i>pk_rdtq</i> ;	データ・キュー詳細情報を格納する領域へのポインタ

【データ・キュー詳細情報 T_RDTQ の構造】

```
typedef struct t_rdtq {
    ID    stskid;          /* データ送信待ちタスクの有無 */
    ID    rtskid;         /* データ受信待ちタスクの有無 */
    UINT  sdtqcnt;       /* 未受信データの総数 */
} T_RDTQ;
```

機能

dtqid で指定されたデータ・キューのデータ・キュー詳細情報（待ちタスクの有無、未受信データの総数など）を *pk_rdtq* で指定された領域に格納します。

- *stskid*
データ・キューの送信待ちキューにタスクがキューイングされているか否かが格納されます。
TSK_NONE : 送信待ちキューにタスクはキューイングされていない
 その他 : 送信待ちキューの先頭にキューイングされているタスクの ID
- *rtskid*
データ・キューの受信待ちキューにタスクがキューイングされているか否かが格納されます。
TSK_NONE : 受信待ちキューにタスクはキューイングされていない
 その他 : 受信待ちキューの先頭にキューイングされているタスクの ID
- *sdtqcnt*
データ・キューのデータ・キュー領域にキューイングされている未受信データの総数が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>pk_rdtq</i> == NULL
E_ID	-18	不正 ID 番号 - <i>dtqid</i> ≤ 0 - <i>dtqid</i> > VTMAX_DTQ
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_dtq</i> を発行した場合、および非タスクから <i>ref_dtq</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_dtq</i> のみ) - 呼び出しタスクからの <i>pk_rdtq</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>dtqid</i> のデータ・キューが存在しない。

19.2.7 同期通信機能（メールボックス）

以下に、RI600PX が同期通信機能（メールボックス）として提供しているサービス・コールの一覧を示します。

表 19 - 11 同期通信機能（メールボックス）

サービス・コール名	機能概要	発行有効範囲
cre_mbx	メールボックスの生成	タスク
acre_mbx	メールボックスの生成（ID 番号自動割付け）	タスク
del_mbx	メールボックスの削除	タスク
snd_mbx	メッセージの送信	タスク
isnd_mbx	メッセージの送信	非タスク
rcv_mbx	メッセージの受信	タスク
prcv_mbx	メッセージの受信（ポーリング）	タスク
iprcv_mbx	メッセージの受信（ポーリング）	非タスク
trcv_mbx	メッセージの受信（タイムアウト付き）	タスク
ref_mbx	メールボックス詳細情報の参照	タスク
iref_mbx	メールボックス詳細情報の参照	非タスク

cre_mbx acre_mbx

概要

メールボックスの生成

C 言語形式

```
ER      cre_mbx ( ID mbxid, T_CMBX *pk_cmbx );
ER_ID   acre_mbx ( T_CMBX *pk_cmbx );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID
I	T_CMBX <i>*pk_cmbx</i> ;	メールボックス生成情報を格納した領域へのポインタ

【メールボックス生成情報 T_CMBX の構造】

```
typedef struct t_cmbx {
    ATR    mbxatr;          /* メールボックス属性 */
    PRI    maxmpri;        /* メッセージ優先度の最大値 */
    VP     mprihd;         /* 将来拡張用 */
} T_CMBX;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
cre_mbx は、指定された *mbxid* のメールボックスを *pk_cmbx* で指定された内容で生成します。**acre_mbx** は *pk_cmbx* で指定された内容でメールボックスを生成し、生成したメールボックス ID を返します。

1) メールボックス属性 (*mbxatr*)

mbxatr には以下を指定できます。

```
mbxatr := ( ( TA_TFIFO || TA_TPRI ) | ( TA_MFIFO || TA_MPRI ) )
```

- TA_TFIFO (= 0x0000)

メールボックスの待ちキューの順序を FIFO 順とします。

- TA_TPRI (= 0x0001)

メールボックスの待ちキューの順序をタスクの現在優先度順とします。ただし、同じ現在優先度のタスクの中では FIFO 順です。

- TA_MFIFO (= 0x0000)

メッセージ・キューの順序を FIFO 順とします。

- TA_MPRI (= 0x0002)

メッセージ・キューの順序をメッセージの優先度順とします。ただし、同じメッセージ優先度のメッセージの中では FIFO 順です。

- 2) メッセージ優先度の最大値 (*maxmpri*)
mbxatr に TA_MPRI を指定した場合は、メッセージ優先度として 1 ~ *maxmpri* を使用できます。
maxmpri に指定可能な範囲は、1 ~ TMAX_MPRI です。
- 3) *mprihd*
将来拡張用のため、単に無視されます。

戻り値

マクロ	数値	意味
—	正の値	acre_mbx の正常終了 (生成されたメールボックス ID)
E_OK	0	cre_mbx の正常終了
E_RSATR	-11	予約属性 - <i>mbxatr</i> の bit0, bit1 以外のビットが 1
E_PAR	-17	パラメータ・エラー - <i>pk_cmbx</i> == NULL - TA_MPRI 指定時 - <i>maxmpri</i> ≤ 0 - <i>maxmpri</i> > TMAX_MPRI
E_ID	-18	不正 ID 番号 (cre_mbx のみ) - <i>mbxid</i> ≤ 0 - <i>mbxid</i> > VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_cmbx</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOID	-34	空き ID なし (acre_mbx のみ)
E_OBJ	-41	オブジェクト状態不正 (cre_mbx のみ) - <i>mbxid</i> のメールボックスが存在する。

del_mbx

概要

メールボックスの削除

C 言語形式

```
ER      del_mbx ( ID mbxid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
mbxid で指定されたメールボックスを削除します。

対象メールボックスで [rcv_mbx](#) または [trcv_mbx](#) によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、[rcv_mbx](#) または [trcv_mbx](#) の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>mbxid</i> ≤ 0 - <i>mbxid</i> > VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>mbxid</i> のメールボックスが存在しない。

snd_mbx isnd_mbx

概要

メッセージの送信

C 言語形式

```
ER      snd_mbx ( ID mbxid, T_MSG *pk_msg );
ER      isnd_mbx ( ID mbxid, T_MSG *pk_msg );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID
I	T_MSG <i>*pk_msg</i> ;	メッセージを格納した領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct {
    VP      *msghead;          /*RI600PX 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct {
    T_MSG   msgque;           /* メッセージヘッダ */
    PRI     msgpri;          /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスに *pk_msg* で指定されたメッセージを送信します。

ただし、本サービス・コールを発行した際、対象メールボックスの待ちキューにタスクがキューイングされていた場合には、メッセージの送信（メッセージのキューイング処理）は行わず、該当タスクにメッセージを渡します。これにより、該当タスクは、待ちキューから外れ、WAITING 状態（メッセージ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 1 メッセージを対象メールボックスのメッセージ・キューにキューイングする際のキューイング方式は、[メールボックスの生成時](#)に指定した順（FIFO 順またはメッセージ優先度順）に行われます。

備考 2 送信したメッセージ（*pk_msg* が指す領域）は、受信されるまでの間に書き換えてはなりません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>pk_msg</i> == NULL - 対象メールボックスが TA_MPRI 属性の場合 : <ul style="list-style-type: none"> - <i>msgpri</i> ≤ 0 - <i>msgpri</i> > TMAX_MPRI
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <i>mbxid</i> ≤ 0 - <i>mbxid</i> > VTMAX_MBX
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - タスクから <i>isnd_mbx</i> を発行した。 - 非タスクから <i>snd_mbx</i> を発行した。 - 「PSW.IPL > カーネル割り込みマスキングレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (<i>snd_mbx</i> のみ) <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからのメッセージ・ヘッダ領域に対するオペランド・ライト・アクセス許可とオペランド・リード・アクセス許可がない。 メッセージ・ヘッダ領域 : <ul style="list-style-type: none"> - TA_MFIFO 属性の場合 : <i>pk_msg</i> から始まる T_MSG 構造体 - TA_MPRI 属性の場合 : <i>pk_msg</i> から始まる T_MSG_PRI 構造体
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <i>mbxid</i> のメールボックスが存在しない。

rcv_mbx

概要

メッセージの受信

C 言語形式

```
ER    rcv_mbx ( ID mbxid, T_MSG **ppk_msg );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID
O	T_MSG <i>**ppk_msg</i> ;	メッセージの先頭アドレスを格納する領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct {
    VP    *msghead;        /*RI600PX 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct {
    T_MSG    msgque;        /* メッセージヘッダ */
    PRI    msgpri;        /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し、その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には、メッセージの受信は行わず、自タスクを対象メールボックスの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbx の発行により、対象メールボックスにメッセージが送信された。	E_OK
isnd_mbx の発行により、対象メールボックスにメッセージが送信された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI

メッセージ受信待ち状態の解除操作	戻り値
del_mbx の発行により、待ち状態を強制的に解除された。	E_DLT

備考 自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は、メールボックスの生成時に指定した順（FIFO 順または現在優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - ppk_msg == NULL
E_ID	-18	不正 ID 番号 - mbxid ≤ 0 - mbxid > VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの ppk_msg が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - mbxid のメールボックスが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または irel_wai の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 - del_mbx の発行により、待ち状態を強制的に解除された。

prcv_mbx iprcv_mbx

概要

メッセージの受信（ポーリング）

C 言語形式

```
ER    prcv_mbx ( ID mbxid, T_MSG **ppk_msg );
ER    iprcv_mbx ( ID mbxid, T_MSG **ppk_msg );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID
O	T_MSG <i>**ppk_msg</i> ;	メッセージの先頭アドレスを格納する領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct {
    VP    *msghead;        /*RI600PX 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct {
    T_MSG    msgque;        /* メッセージヘッダ */
    PRI      msgpri;        /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し、その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には、メッセージの受信は行わず、戻り値として E_TMOUT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>ppk_msg</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mbxid</i> ≤ 0 - <i>mbxid</i> > VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iprcv_mbx</i> を発行した場合、および非タスクから <i>prcv_mbx</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>prcv_mbx</i> のみ) - 呼び出しタスクからの <i>ppk_msg</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mbxid</i> のメールボックスが存在しない。
E_TMOUT	-50	ポーリング失敗

trcv_mbx

概要

メッセージの受信（タイムアウト付き）

C 言語形式

```
ER      trcv_mbx ( ID mbxid, T_MSG **ppk_msg, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID
O	T_MSG <i>**ppk_msg</i> ;	メッセージの先頭アドレスを格納する領域へのポインタ
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct {
    VP      *msghead;          /*RI600PX 管理領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct {
    T_MSG   msgque;          /* メッセージヘッダ */
    PRI     msgpri;         /* メッセージ優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し、その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には、メッセージの受信は行わず、自タスクを対象メールボックスの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbx の発行により、対象メールボックスにメッセージが送信された。	E_OK

メッセージ受信待ち状態の解除操作	戻り値
<code>isnd_mbx</code> の発行により、対象メールボックスにメッセージが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_mbx</code> の発行により、待ち状態を強制的に解除された。	E_DLT

備考 1 自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は、メールボックスの生成時に指定した順（FIFO 順または現在優先度順）に行われます。

備考 2 待ち時間 `tmout` に `TMO_FEVR` が指定された際には“`rcv_mbx` と同等の処理”を、`TMO_POL` が指定された際には“`prcv_mbx` と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <code>tmout < -1</code> - <code>tmout > (0x7FFFFFFF - TIC_NUME)/TIC_DENO</code> - <code>ppk_msg == NULL</code>
E_ID	-18	不正 ID 番号 - <code>mbxid ≤ 0</code> - <code>mbxid > VTMAX_MBX</code>
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <code>ppk_msg</code> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <code>mbxid</code> のメールボックスが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 - <code>del_mbx</code> の発行により、待ち状態を強制的に解除された。

ref_mbx
iref_mbx

概要

メールボックス詳細情報の参照

C 言語形式

```
ER    ref_mbx ( ID mbxid, T_RMBX *pk_rmbx );
ER    iref_mbx ( ID mbxid, T_RMBX *pk_rmbx );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID
O	T_RMBX <i>*pk_rmbx</i> ;	メールボックス詳細情報を格納する領域へのポインタ

【メールボックス詳細情報 T_RMBX の構造】

```
typedef struct t_rmbx {
    ID    wtskid;           /* 待ちタスクの有無 */
    T_MSG *pk_msg;        /* 待ちメッセージの有無 */
} T_RMBX;
```

機能

mbxid で指定されたメールボックスのメールボックス詳細情報（待ちタスクの有無、待ちメッセージの有無など）を *pk_rmbx* で指定された領域に格納します。

- *wtskid*

メールボックスの待ちキューにタスクがキューイングされているか否かが格納されます。

TSK_NONE : 待ちキューにタスクはキューイングされていない
 その他 : 待ちキューの先頭にキューイングされているタスクの ID

- *pk_msg*

メールボックスの待ちキューにメッセージがキューイングされているか否かが格納されます。

NULL : 待ちキューにメッセージはキューイングされていない
 その他 : 待ちキューの先頭にキューイングされているメッセージの先頭アドレス

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <code>pk_rmbx == NULL</code>
E_ID	-18	不正 ID 番号 - <code>mbxid ≤ 0</code> - <code>mbxid > VTMAX_MBX</code>
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <code>iref_mbx</code> を発行した場合、および非タスクから <code>ref_mbx</code> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<code>ref_mbx</code> のみ) - 呼び出しタスクからの <code>pk_rmbx</code> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <code>mbxid</code> のメールボックスが存在しない。

19.2.8 拡張同期通信機能（ミューテックス）

以下に、RI600PX が拡張同期通信機能（ミューテックス）として提供しているサービス・コールの一覧を示します。

表 19 - 12 拡張同期通信機能（ミューテックス）

サービス・コール名	機能概要	発行有効範囲
cre_mtx	ミューテックスの生成	タスク
acre_mtx	ミューテックスの生成（ID 番号自動割付け）	タスク
del_mtx	ミューテックスの削除	タスク
loc_mtx	ミューテックスのロック	タスク
ploc_mtx	ミューテックスのロック（ポーリング）	タスク
tloc_mtx	ミューテックスのロック（タイムアウト付き）	タスク
unl_mtx	ミューテックスのロック解除	タスク
ref_mtx	ミューテックス詳細情報の参照	タスク

cre_mtx acre_mtx

概要

ミューテックスの生成

C 言語形式

```
ER      cre_mtx ( ID mtxid, T_CMTX *pk_cmtx );
ER_ID  acre_mtx ( T_CMTX *pk_cmtx );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mtxid</i> ;	ミューテックスの ID
I	T_CMTX * <i>pk_cmtx</i> ;	ミューテックス生成情報を格納した領域へのポインタ

【 ミューテックス生成情報 T_CMTX の構造 】

```
typedef struct t_cmtx {
    ATR    mtxatr;          /* ミューテックス属性 */
    PRI    ceilpri;        /* 上限優先度 */
} T_CMTX;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
 cre_mtx は、指定された *mtxid* のミューテックスを *pk_cmtx* で指定された内容で生成します。acre_mtx は *pk_cmtx* で指定された内容でミューテックスを生成し、生成したミューテックス ID を返します。

1) ミューテックス属性 (*mtxatr*)

mtxatr には TA_CEILING のみを指定できます。

- TA_CEILING (= 0x0003)

優先度上限プロトコル

詳細は、「8.2.3 簡略化した優先度上限プロトコル」を参照してください。

備考 ミューテックスの待ちキューの順序はタスクの現在優先度順です。ただし、同じ現在優先度のタスクの中では FIFO 順です

2) 上限優先度 (*ceilpri*)

ミューテックスをロックしたタスクの現在優先度は、*ceilpri* に高められます。

ceilpri に指定可能な範囲は、1 ~ TMAX_TPRI です。

戻り値

マクロ	数値	意味
—	正の値	acre_mtx の正常終了（生成されたミューテックス ID）
E_OK	0	cre_mtx の正常終了
E_RSATR	-11	予約属性 - <i>mtxatr</i> != TA_CEILING
E_PAR	-17	パラメータ・エラー - <i>pk_cmtx</i> == NULL - <i>ceilpri</i> ≤ 0 - <i>ceilpri</i> > TMAX_TPRI
E_ID	-18	不正 ID 番号（cre_mtx のみ） - <i>mtxid</i> ≤ 0 - <i>mtxid</i> > VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_cmtx</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOID	-34	空き ID なし（acre_mtx のみ）
E_OBJ	-41	オブジェクト状態不正（cre_mtx のみ） - <i>mtxid</i> のミューテックスが存在する。

del_mtx

概要

ミューテックスの削除

C 言語形式

```
ER      del_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mtxid</i> ;	ミューテックスの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
mtxid で指定されたミューテックスを削除します。

対象ミューテックスをロックしているタスクがある場合、そのロックを解除します。その結果、そのタスクがロックしているミューテックスがなくなった場合には、そのタスクの現在優先度をベース優先度に戻します。対象ミューテックスをロックしていたタスクには、ミューテックスが削除されたことは通知されません。後で [unl_mtx](#) によってミューテックスをロック解除しようとした時点で、エラー E_NOEXS が返されます。

対象ミューテックスで [loc_mtx](#) または [tloc_mtx](#) によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、[loc_mtx](#) または [tloc_mtx](#) の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $mtxid \leq 0$ - $mtxid > VTMAX_MTX$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。

マクロ	数値	意味
E_NOEXS	-42	オブジェクト未生成 - <i>mtxid</i> のミューテックスが存在しない。

loc_mtx

概要

ミューテックスのロック

C 言語形式

```
ER      loc_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mtxid</i> ;	ミューテックスの ID

機能

mtxid で指定されたミューテックスをロックします。

ただし、本サービス・コールを発行した際、対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には、自タスクを対象ミューテックスの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（ミューテックス待ち状態）へと遷移させます。

なお、ミューテックス待ち状態の解除は、以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
<i>unl_mtx</i> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<i>ext_tsk</i> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<i>exd_tsk</i> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<i>ter_tsk</i> の発行により、対象ミューテックスのロック状態が解除された。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>del_mtx</i> の発行により、待ち状態を強制的に解除された。	E_DLT

ミューテックスのロック時には、自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、自タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合は、自タスクの現在優先度は変更しません。

備考 1 自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は、タスクの現在優先度順に行われます。ただし、同じ優先度のタスクの中では FIFO 順に行われます。

備考 2 自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には、戻り値として E_ILUSE を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $mtxid \leq 0$ - $mtxid > VTMAX_MTX$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_ILUSE	-28	サービス・コール不正使用 - 自タスクはすでに対象ミューテックスをロック済み - 上限優先度違反（自タスクのベース優先度 < 対象ミューテックスの上限優先度）
E_NOEXS	-42	オブジェクト未生成 - $mtxid$ のミューテックスが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - rel_wai または $irel_wai$ の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 - del_mtx の発行により、待ち状態を強制的に解除された。

ploc_mtx

概要

ミューテックスのロック（ポーリング）

C 言語形式

```
ER      ploc_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mtxid</i> ;	ミューテックスの ID

機能

mtxid で指定されたミューテックスをロックします。

ただし、本サービス・コールを発行した際、対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には、戻り値として E_TMOUT を返します。

ミューテックスのロック時には、自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、自タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合、自タスクの現在優先度は変更しません。

備考 RI600PX では、対象ミューテックスをロック状態へと遷移させたタスクがロック状態を解除することなく、本サービス・コールを再発行（ミューテックスの多重ロック）した際には、戻り値として E_ILUSE を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>mtxid</i> ≤ 0 - <i>mtxid</i> > VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。

マクロ	数値	意味
E_ILUSE	-28	サービス・コール不正使用 - 自タスクはすでに対象ミューテックスをロック済み - 上限優先度違反（自タスクのベース優先度 < 対象ミューテックスの上限優先度）
E_NOEXS	-42	オブジェクト未生成 - <i>mtxid</i> のミューテックスが存在しない。
E_TMOUT	-50	ポーリング失敗

tloc_mtx

概要

ミューテックスのロック（タイムアウト付き）

C 言語形式

```
ER      tloc_mtx ( ID mtxid, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mtxid</i> ;	ミューテックスの ID
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR ： 永久待ち TMO_POL ： ポーリング 数値： 待ち時間

機能

パラメータ *mtxid* で指定されたミューテックスをロックします。

ただし、本サービス・コールを発行した際、対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には、自タスクを対象ミューテックスの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（ミューテックス待ち状態）へと遷移させます。

なお、ミューテックス待ち状態の解除は、以下の場合に行われます。

ミューテックス待ち状態の解除操作	戻り値
uni_mtx の発行により、対象ミューテックスのロック状態が解除された。	E_OK
ext_tsk の発行により、対象ミューテックスのロック状態が解除された。	E_OK
exd_tsk の発行により、対象ミューテックスのロック状態が解除された。	E_OK
ter_tsk の発行により、対象ミューテックスのロック状態が解除された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <i>tmout</i> で指定された待ち時間が経過した。	E_TMOUT
del_mtx の発行により、待ち状態を強制的に解除された。	E_DLT

ミューテックスのロック時には、自タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、自タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合、自タスクの現在優先度は変更しません。

備考 1 自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は、タスクの現在優先度順に行われます。ただし、同じ優先度のタスクの中では FIFO 順に行われます。

備考2 RI600PXでは、自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には、戻り値としてE_ILUSEを返します。

備考3 待ち時間 *tmout* に `TMO_FEVR` が指定された際には“`loc_mtx` と同等の処理”を、`TMO_POL` が指定された際には“`ploc_mtx` と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>tmout</i> < -1 - <i>tmout</i> > (0x7FFFFFFF - <code>TIC_NUME</code>)/ <code>TIC_DENO</code>
E_ID	-18	不正 ID 番号 - <i>mtxid</i> ≤ 0 - <i>mtxid</i> > <code>VTMAX_MTX</code>
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_ILUSE	-28	サービス・コール不正使用 - 自タスクはすでに対象ミューテックスをロック済み - 上限優先度違反（自タスクのベース優先度 < 対象ミューテックスの上限優先度）
E_NOEXS	-42	オブジェクト未生成 - <i>mtxid</i> のミューテックスが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 - <code>del_mtx</code> の発行により、待ち状態を強制的に解除された。

unl_mtx

概要

ミューテックスのロック解除

C 言語形式

```
ER      unl_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mtxid</i> ;	ミューテックスの ID

機能

mtxid で指定されたミューテックスのロック状態を解除します。その結果、自タスクがロックしているミューテックスがなくなった場合には、自タスクの現在優先度をベース優先度に変更します。

本サービス・コールを発行した際、対象ミューテックスの待ちキューにタスクがキューイングされていた場合には、ミューテックスのロック解除処理後、ただちに該当タスク（待ちキューの先頭タスク）によるミューテックスのロック処理が行われます。このとき、該当タスクは、待ちキューから外れ、WAITING 状態（ミューテックス待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、該当タスクの現在優先度を対象ミューテックスの上限優先度に変更します。ただし、該当タスクがすでに他のミューテックスをロックしており、かつ対象ミューテックスの上限優先度がロック済みのミューテックスの上限優先度以下の場合には、該当タスクの現在優先度は変更しません。

備考 1 ミューテックスのロック解除が可能なタスクは“対象ミューテックスをロックしたタスク”に限られます。このため、自タスクがロックしていないミューテックスに対して本サービス・コールを発行した場合には、戻り値として E_ILUSE を返します。

備考 2 タスクの終了時、そのタスクがロックしていたミューテックスはロック解除されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $mtxid \leq 0$ - $mtxid > VTMAX_MTX$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_ILUSE	-28	サービス・コール不正使用 - 自タスクは対象ミューテックスをロックしていない
E_NOEXS	-42	オブジェクト未生成 - <i>mtxid</i> のミューテックスが存在しない。

ref_mtx

概要

ミューテックス詳細情報の参照

C 言語形式

```
ER      ref_mtx ( ID mtxid, T_RMTX *pk_rmtx );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mtxid</i> ;	ミューテックスの ID
O	T_RMTX * <i>pk_rmtx</i> ;	ミューテックス詳細情報を格納する領域へのポインタ

【 ミューテックス詳細情報 T_RMTX の構造 】

```
typedef struct t_rmtx {
    ID      htsskid;          /* ロックの有無 */
    ID      wtsskid;          /* 待ちタスクの有無 */
} T_RMTX;
```

機能

mtxid で指定されたミューテックスのミューテックス詳細情報（ロックの有無、待ちタスクの有無など）を *pk_rmtx* で指定された領域に格納します。

- *htsskid*

ミューテックスをロックしているタスクが存在しているか否かが格納されます。

TSK_NONE : ロックしているタスクは存在しない
 その他 : ロックしているタスクの ID

- *wtsskid*

ミューテックスの待ちキューにタスクがキューイングされているか否かが格納されます。

TSK_NONE : 待ちキューにタスクはキューイングされていない
 その他 : 待ちキューの先頭にキューイングされているタスクの ID

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>pk_rmtx</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mtxid</i> ≤ 0 - <i>mtxid</i> > VTMAX_MTX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - 呼び出しタスクからの <i>pk_rmtx</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mtxid</i> のミューテックスが存在しない。

19.2.9 拡張同期通信機能（メッセージ・バッファ）

以下に、RI600PX が拡張同期通信機能（メッセージ・バッファ）として提供しているサービス・コールの一覧を示します。

表 19 - 13 拡張同期通信機能（メッセージ・バッファ）

サービス・コール名	機能概要	発行有効範囲
cre_mbf	メッセージ・バッファの生成	タスク
acre_mbf	メッセージ・バッファの生成（ID 番号自動割付け）	タスク
del_mbf	メッセージ・バッファの削除	タスク
snd_mbf	メッセージの送信	タスク
psnd_mbf	メッセージの送信（ポーリング）	タスク
ipsnd_mbf	メッセージの送信（ポーリング）	非タスク
tsnd_mbf	メッセージの送信（タイムアウト付き）	タスク
rcv_mbf	メッセージの受信	タスク
prcv_mbf	メッセージの受信（ポーリング）	タスク
trcv_mbf	メッセージの受信（タイムアウト付き）	タスク
ref_mbf	メッセージ・バッファ詳細情報の参照	タスク
iref_mbf	メッセージ・バッファ詳細情報の参照	非タスク

cre_mbf acre_mbf

概要

メッセージ・バッファの生成

C 言語形式

```
ER      cre_mbf ( ID mbfid, T_CMBF *pk_cmbf );
ER_ID   acre_mbf ( T_CMBF *pk_cmbf );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
I	T_CMBF * <i>pk_cmbf</i> ;	メッセージ・バッファ生成情報を格納した領域へのポインタ

【メッセージ・バッファ生成情報 T_CMBF の構造】

```
typedef struct t_cmbf {
    ATR    mbfattr;          /* メッセージ・バッファ属性 */
    UINT   maxmsz;         /* メッセージの最大サイズ (バイト数) */
    SIZE   mbfsz;          /* メッセージ・バッファ領域のサイズ (バイト数) */
    VP     mbf;             /* メッセージ・バッファ領域の先頭アドレス */
} T_CMBF;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_mbf は、指定された *mbfid* のメッセージ・バッファを *pk_cmbf* で指定された内容で生成します。acre_mbf は *pk_cmbf* で指定された内容でメッセージ・バッファを生成し、生成したメッセージ・バッファ ID を返します。

1) メッセージ・バッファ属性 (*mbfattr*)

mbfattr には TA_TFIFO のみを指定できます。

- TA_TFIFO (= 0x0000)

メッセージ・バッファの送信待ちキューの順序を FIFO 順とします。

備考 メッセージ・バッファの受信待ちキューは FIFO 順です。

2) メッセージの最大サイズ (*maxmsz*)

このメッセージ・バッファに送信するメッセージの最大サイズを指定します。rcv_mbf, prcv_mbf, および trcv_mbf で指定するメッセージ受信領域は、このサイズ以上でなければなりません。

3) メッセージ・バッファ領域のサイズ (*mbfsz*)、メッセージ・バッファ領域の先頭アドレス (*mbf*)

アプリケーション側で、*mbfsz* バイトのメッセージ・バッファ領域を確保し、その先頭アドレスを *mbf* に指定します。また、*mbfsz* に 0 を指定することもできます。この場合、メッセージ・バッファにメッセージを蓄えておくことは

できないため、送信側と受信側の先に実行した方が待ち状態になり、他方が行われた時点で待ちが解除される、つまり送信側と受信側が完全に同期した動作となります。なお、*mbfsz* が 0 の場合は、*mbf* は無視されます。

備考 1 RI600PX は、メッセージ・バッファ領域に関するアクセス権については何も関知しません。通常は、メッセージ・バッファ領域はメモリ・オブジェクト以外でかつユーザ・スタック以外の領域に作成してください。メモリ・オブジェクト内にメッセージ・バッファ領域を作成した場合は、そのメモリ・オブジェクトへのオペランド・ライト・アクセスが許可されたタスクが、誤ってメッセージ・バッファ領域を書き換えてしまう危険があります。

備考 2 μ ITRON4.0 仕様には、*mbf* に NULL を指定することで RI600PX がメッセージ・バッファ領域を割り当てる機能がありますが、本サービス・コールはこの機能をサポートしていません。

戻り値

マクロ	数値	意味
—	正の値	acre_mbf の正常終了（生成されたメッセージ・バッファ ID）
E_OK	0	cre_mbf の正常終了
E_RSATR	-11	予約属性 - <i>mbfatr</i> != TA_TFIFO
E_PAR	-17	パラメータ・エラー - <i>pk_cmbf</i> == NULL - <i>maxmsz</i> == 0, <i>maxmsz</i> > 65528 - 0 < <i>mbfsz</i> < 8, <i>mbfsz</i> > 65532 - <i>mbfsz</i> == 0 かつ <i>mbf</i> + <i>mbfsz</i> > 0x100000000 - <i>mbfsz</i> == 0 かつ <i>mbfsz</i> < <i>maxmsz</i> + VTSZ_MBFTBL
E_ID	-18	不正 ID 番号（cre_mbf のみ） - <i>mbfid</i> ≤ 0 - <i>mbfid</i> > VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_cmbf</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOMEM	-33	メモリ不足 - <i>mbfsz</i> != 0 かつ <i>mbf</i> == NULL
E_NOID	-34	空き ID なし（acre_mbf のみ）
E_OBJ	-41	オブジェクト状態不正（cre_mbf のみ） - <i>mbfid</i> のメッセージ・バッファが存在する。

del_mbf

概要

メッセージ・バッファの削除

C 言語形式

```
ER      del_mbf ( ID mbfid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
mbfid で指定されたメッセージ・バッファを削除します。

対象メッセージ・バッファで [snd_mbf](#), [tsnd_mbf](#), [rcv_mbf](#), または [trcv_mbf](#) によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、[snd_mbf](#), [tsnd_mbf](#), [rcv_mbf](#), または [trcv_mbf](#) の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>mbfid</i> ≤ 0 - <i>mbfid</i> > VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>mbfid</i> のメッセージ・バッファが存在しない。

snd_mbf

概要

メッセージの送信

C 言語形式

```
ER      snd_mbf ( ID mbfid, VP msg, UINT msgsz );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
I	VP <i>msg</i> ;	送信メッセージへのポインタ
I	UINT <i>msgsz</i> ;	送信メッセージのサイズ (単位: バイト)

機能

mbfid で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは受信待ちキューから外れ、WAITING 状態 (メッセージ受信待ち状態) から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がある場合
msg で指定されたメッセージをメッセージ・バッファに格納します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
減少サイズ = $\text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL}$
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (メッセージ送信待ち状態) へと遷移させます。
なお、メッセージ送信待ち状態の解除は、以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
<i>rcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<i>prcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<i>trcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK

メッセージ送信待ち状態の解除操作	戻り値
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <code>rel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>ter_tsk</code> の発行により、待ち状態を強制的に解除された。 - <code>tsnd_mbf</code> のパラメータ <code>tmout</code> で指定された待ち時間が経過した。 	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_mbf</code> の発行により、対象メッセージ・バッファがリセットされた。	EV_RST
<code>del_mbf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

備考1 メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は、メッセージの送信要求を行った順に行われます。

備考2 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は、FIFO順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <code>msgsz == 0</code> - <code>msgsz ></code> (生成時に指定したメッセージの最大サイズ) - <code>msg == NULL</code>
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <code>mbfid ≤ 0</code> - <code>mbfid > VTMAX_MBF</code>
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <code>msg</code> が示す領域 (先頭: <code>msg</code>, サイズ: <code>msgsz</code>) に対するオペランド・リード・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <code>mbfid</code> のメッセージ・バッファが存在しない。
E_RLWAI	-49	待ち状態の強制解除 <ul style="list-style-type: none"> - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。

マクロ	数値	意味
E_DLT	-51	待ちオブジェクトの削除 - <code>del_mbf</code> の発行により、待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (<code>vrst_mbf</code>) による待ち解除

psnd_mbf
ipsnd_mbf

概要

メッセージの送信（ポーリング）

C 言語形式

```
ER    psnd_mbf ( ID mbfid, VP msg, UINT msgsz );
ER    ipsnd_mbf ( ID mbfid, VP msg, UINT msgsz );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
I	VP <i>msg</i> ;	送信メッセージへのポインタ
I	UINT <i>msgsz</i> ;	送信メッセージのサイズ（単位：バイト）

機能

mbfid で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは受信待ちキューから外れ、WAITING 状態（メッセージ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がある場合
msg で指定されたメッセージをメッセージ・バッファに格納します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
減少サイズ = $up4(msgsz) + VTSZ_MBFTBL$
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
戻り値として E_TMOUT を返します。

備考 メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は、メッセージの送信要求を行った順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>msgsz</i> == 0 - <i>msgsz</i> > (生成時に指定したメッセージの最大サイズ) - <i>msg</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mbfid</i> ≤ 0 - <i>mbfid</i> > VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから <i>ipsnd_mbf</i> を発行した。 - 非タスクから <i>psnd_mbf</i> を発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (<i>psnd_mbf</i> のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>msg</i> が示す領域 (先頭: <i>msg</i> , サイズ: <i>msgsz</i>) に対するオペランド・リード・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mbfid</i> のメッセージ・バッファが存在しない。
E_TMOUT	-50	ポーリング失敗

tsnd_mbf

概要

メッセージの送信（タイムアウト付き）

C 言語形式

```
ER    tsnd_mbf ( ID mbfid, VP msg, UINT msgsz, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
I	VP <i>msg</i> ;	送信メッセージへのポインタ
I	UINT <i>msgsz</i> ;	送信メッセージのサイズ（単位：バイト）
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR ： 永久待ち TMO_POL ： ポーリング 数値： 待ち時間

機能

mbfid で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- 受信待ちキューにタスクがキューイングされている場合
msg で指定されたメッセージを受信待ちキュー先頭のタスクに渡します。これにより、該当タスクは受信待ちキューから外れ、WAITING 状態（メッセージ受信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がある場合
msg で指定されたメッセージをメッセージ・バッファに格納します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
 減少サイズ = $\text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL}$
- 受信待ちキューおよび送信待ちキューにタスクがキューイングされておらず、メッセージ・バッファにメッセージを格納するための空き領域がない場合、または送信待ちキューにタスクがキューイングされている場合
 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ送信待ち状態）へと遷移させます。
 なお、メッセージ送信待ち状態の解除は、以下の場合に行われます。

メッセージ送信待ち状態の解除操作	戻り値
<i>rcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<i>prcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK
<i>trcv_mbf</i> の発行により、対象メッセージ・バッファに空き領域が確保された。	E_OK

メッセージ送信待ち状態の解除操作	戻り値
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <code>rel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。 - <code>ter_tsk</code> の発行により、待ち状態を強制的に解除された。 - <code>tsnd_mbf</code> のパラメータ <code>tmout</code> で指定された待ち時間が経過した。 	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>vrst_mbf</code> の発行により、対象メッセージ・バッファがリセットされた。	EV_RST
パラメータ <code>tmout</code> で指定された待ち時間が経過した。	E_TMOUT
<code>del_mbf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

- 備考 1 メッセージを対象メッセージ・バッファに書き込む際の書き込み方法は、メッセージの送信要求を行った順に行われます。
- 備考 2 自タスクを対象メッセージ・バッファの送信待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考 3 待ち時間 `tmout` に `TMO_FEVR` が指定された際には“`snd_mbf` と同等の処理”を、`TMO_POL` が指定された際には“`psnd_mbf` と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <code>msgsz == 0</code> - <code>msgsz > (生成時に指定したメッセージの最大サイズ)</code> - <code>msg == NULL</code> - <code>tmout < -1</code> - <code>tmout > (0x7FFFFFFF - TIC_NUME)/TIC_DENO</code>
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <code>mbfid ≤ 0</code> - <code>mbfid > VTMAX_MBF</code>
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <code>msg</code> が示す領域（先頭：<code>msg</code>、サイズ：<code>msgsz</code>）に対するオペランド・リード・アクセス許可がない。

マクロ	数値	意味
E_NOEXS	-42	オブジェクト未生成 - <i>mbfid</i> のメッセージ・バッファが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <i>rel_wai</i> または <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 - <i>del_mbf</i> の発行により、待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (<i>vrst_mbf</i>) による待ち解除

rcv_mbf

概要

メッセージの受信

C 言語形式

```
ER_UINT rcv_mbf ( ID mbfid, VP msg );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
O	VP <i>msg</i> ;	メッセージを格納する領域へのポインタ

機能

mbfid で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し、そのメッセージサイズを戻り値として返します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。
増加サイズ = $\text{up4}(\text{戻り値}) + \text{VTSZ_MBFTBL}$
さらに以下を、送信待ちキューにタスクがなくなるか、メッセージを格納できなくなるまで繰り返します。
 - 送信待ちキューにタスクがキューイングされており、キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は、そのメッセージをメッセージ・バッファに格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（メッセージ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
減少サイズ = $\text{up4}(\text{該当タスクが送信したメッセージ・サイズ}) + \text{VTSZ_MBFTBL}$
- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し、そのメッセージサイズを戻り値として返します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（メッセージ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、メッセージ・バッファのサイズが 0 の場合のみ生じます。
- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
<code>snd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK

メッセージ受信待ち状態の解除操作	戻り値
<code>psnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>ipsnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>tsnd_mbf</code> の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
<code>rel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>irel_wai</code> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<code>del_mbf</code> の発行により、待ち状態を強制的に解除された。	E_DLT

備考1 最大メッセージ・サイズは、メッセージ・バッファの生成時に指定します。`msg` の指す領域は、必ず最大メッセージ・サイズ以上としてください。

備考2 自タスクを対象メッセージ・バッファの受信待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。

戻り値

マクロ	数値	意味
—	正の値	正常終了（受信したメッセージのサイズ）
E_PAR	-17	パラメータ・エラー - <code>msg == NULL</code>
E_ID	-18	不正 ID 番号 - <code>mbfid ≤ 0</code> - <code>mbfid > VTMAX_MBF</code>
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <code>msg</code> が示す領域（先頭： <code>msg</code> 、サイズ：生成時に指定した最大メッセージ・サイズ）に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <code>mbfid</code> のメッセージ・バッファが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 - <code>del_mbf</code> の発行により、待ち状態を強制的に解除された。

prcv_mbf

概要

メッセージの受信（ポーリング）

C 言語形式

```
ER_UINT prcv_mbf ( ID mbfid, VP msg );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
O	VP <i>msg</i> ;	メッセージを格納する領域へのポインタ

機能

mbfid で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し、そのメッセージサイズを戻り値として返します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。
増加サイズ = $\text{up4}(\text{戻り値}) + \text{VTSZ_MBFTBL}$
さらに以下を、送信待ちキューにタスクがなくなるか、メッセージを格納できなくなるまで繰り返します。
 - 送信待ちキューにタスクがキューイングされており、キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は、そのメッセージをメッセージ・バッファに格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（メッセージ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
減少サイズ = $\text{up4}(\text{該当タスクが送信したメッセージ・サイズ}) + \text{VTSZ_MBFTBL}$
- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し、そのメッセージサイズを戻り値として返します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（メッセージ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、メッセージ・バッファのサイズが 0 の場合のみ生じます。
- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
戻り値として E_TMOUT を返します。

備考 最大メッセージ・サイズは、メッセージ・バッファの生成時に指定します。*msg* の指す領域は、必ず最大メッセージ・サイズ以上としてください。

戻り値

マクロ	数値	意味
—	正	正常終了（受信したメッセージのサイズ）
E_PAR	-17	パラメータ・エラー - <i>msg</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mbfid</i> ≤ 0 - <i>mbfid</i> > VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>msg</i> が示す領域（先頭： <i>msg</i> , サイズ：生成時に指定した最大メッセージ・サイズ）に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mbfid</i> のメッセージ・バッファが存在しない。
E_TMOUT	-50	ポーリング失敗

trcv_mbf

概要

メッセージの受信（タイムアウト付き）

C 言語形式

```
ER_UINT trcv_mbf ( ID mbfid, VP msg, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
O	VP <i>msg</i> ;	メッセージを格納する領域へのポインタ
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

mbfid で指定されたメッセージ・バッファの状況に応じて、以下の処理を行います。

- メッセージ・バッファにメッセージが格納されている場合
メッセージ・バッファから最古のメッセージを取り出して *msg* で指定された領域に格納し、そのメッセージサイズを戻り値として返します。このとき、対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ増加します。
増加サイズ = $\text{up4}(\text{戻り値}) + \text{VTSZ_MBFTBL}$
さらに以下を、送信待ちキューにタスクがなくなるか、メッセージを格納できなくなるまで繰り返します。
 - 送信待ちキューにタスクがキューイングされており、キュー先頭のタスクが指定したメッセージのサイズだけの空き領域がある場合は、そのメッセージをメッセージ・バッファに格納します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（メッセージ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。また、このとき対象メッセージ・バッファの空き領域は以下の式で算出されるサイズだけ減少します。
減少サイズ = $\text{up4}(\text{該当タスクが送信したメッセージ・サイズ}) + \text{VTSZ_MBFTBL}$
- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされている場合
送信待ちキュー先頭のタスクが指定したメッセージを *msg* で指定された領域に格納し、そのメッセージサイズを戻り値として返します。これにより、該当タスクは、送信待ちキューから外れ、WAITING 状態（メッセージ送信待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
なお、この状況は、メッセージ・バッファのサイズが 0 の場合のみ生じます。
- メッセージ・バッファにメッセージが格納されておらず、送信待ちキューにタスクがキューイングされていない場合
自タスクを対象メッセージ・バッファの受信待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
なお、メッセージ受信待ち状態の解除は、以下の場合に行われます。

メッセージ受信待ち状態の解除操作	戻り値
snd_mbf の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
psnd_mbf の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
ipsnd_mbf の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
tsnd_mbf の発行により、対象メッセージ・バッファにメッセージが送信された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
パラメータ <i>tmout</i> で指定された待ち時間が経過した	E_TMOUT
del_mbf の発行により、待ち状態を強制的に解除された。	E_DLT

- 備考 1 最大メッセージ・サイズは、メッセージ・バッファの生成時に指定します。*msg* の指す領域は、必ず最大メッセージ・サイズ以上としてください。
- 備考 2 自タスクを対象メッセージ・バッファの受信待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考 3 待ち時間 *tmout* に **TMO_FEVR** が指定された際には“rcv_mbf と同等の処理”を、**TMO_POL** が指定された際には“prcv_mbf と同等の処理”を実行します。

戻り値

マクロ	数値	意味
—	正	正常終了（受信したメッセージのサイズ）
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>msg</i> == NULL - <i>tmout</i> < -1 - <i>tmout</i> > (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <i>mbfid</i> ≤ 0 - <i>mbfid</i> > VTMAX_MBF
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>msg</i> が示す領域（先頭：<i>msg</i>、サイズ：生成時に指定した最大メッセージ・サイズ）に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <i>mbfid</i> のメッセージ・バッファが存在しない。

マクロ	数値	意味
E_RLWAI	-49	待ち状態の強制解除 - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 - <code>del_mbf</code> の発行により、待ち状態を強制的に解除された。

ref_mbf iref_mbf

概要

メッセージ・バッファ詳細情報の参照

C 言語形式

```
ER    ref_mbf ( ID mbfid, T_RMBF *pk_rmbf );
ER    iref_mbf ( ID mbfid, T_RMBF *pk_rmbf );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID
O	T_RMBF * <i>pk_rmbf</i> ;	メッセージ・バッファ詳細情報を格納する領域へのポインタ

【メッセージ・バッファ詳細情報 T_RMBF の構造】

```
typedef struct t_rmbf {
    ID    stskid;          /* メッセージ送信待ちタスクの有無 */
    ID    rtskid;         /* メッセージ受信待ちタスクの有無 */
    UINT  msgcnt;        /* 未受信メッセージの総数 */
    SIZE  fmbfsz;       /* 空き領域のサイズ */
} T_RMBF;
```

機能

mbfid で指定されたメッセージ・バッファのメッセージ・バッファ詳細情報（待ちタスクの有無、未受信データの総数など）を *pk_rmbf* で指定された領域に格納します。

- *stskid*

メッセージ・バッファの送信待ちキューにタスクがキューイングされているか否かが格納されます。

TSK_NONE : 送信待ちキューにタスクはキューイングされていない
 その他 : 送信待ちキューの先頭にキューイングされているタスクの ID

- *rtskid*

メッセージ・バッファの受信待ちキューにタスクがキューイングされているか否かが格納されます。

TSK_NONE : 受信待ちキューにタスクはキューイングされていない
 その他 : 受信待ちキューの先頭にキューイングされているタスクの ID

- *msgcnt*

メッセージ・バッファに格納されている未受信メッセージ総数が格納されます。

- *fmbfsz*

メッセージ・バッファの空き領域のサイズ（単位：バイト）が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>pk_rmbf</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mbfid</i> ≤ 0 - <i>mbfid</i> > VTMAX_MBF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_mbf</i> を発行した場合、および非タスクから <i>ref_mbf</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_mbf</i> のみ) - 呼び出しタスクからの <i>pk_rmbf</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mbfid</i> のメッセージ・バッファが存在しない。

19.2.10 メモリ・プール管理機能（固定長メモリ・プール）

以下に、RI600PX がメモリ・プール管理機能（固定長メモリ・プール）として提供しているサービス・コールの一覧を示します。

表 19 - 14 メモリ・プール管理機能（固定長メモリ・プール）

サービス・コール名	機能概要	発行有効範囲
cre_mpf	固定長メモリ・プールの生成	タスク
acre_mpf	固定長メモリ・プールの生成（ID 番号自動割付け）	タスク
del_mpf	固定長メモリ・プールの削除	タスク
get_mpf	固定長メモリ・ブロックの獲得	タスク
pget_mpf	固定長メモリ・ブロックの獲得（ポーリング）	タスク
ipget_mpf	固定長メモリ・ブロックの獲得（ポーリング）	非タスク
tget_mpf	固定長メモリ・ブロックの獲得（タイムアウト付き）	タスク
rel_mpf	固定長メモリ・ブロックの返却	タスク
irel_mpf	固定長メモリ・ブロックの返却	非タスク
ref_mpf	固定長メモリ・プール詳細情報の参照	タスク
iref_mpf	固定長メモリ・プール詳細情報の参照	非タスク

cre_mpf acre_mpf

概要

固定長メモリ・プールの生成

C 言語形式

```
ER      cre_mpf ( ID mpfid, T_CMPF *pk_cmpf );
ER_ID  acre_mpf ( T_CMPF *pk_cmpf );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID
I	T_CMPF * <i>pk_cmpf</i> ;	固定長メモリ・プール生成情報を格納した領域へのポインタ

【 固定長メモリ・プール生成情報 T_CMPF の構造 】

```
typedef struct t_cmpf {
    ATR    mpfatr;          /* 固定長メモリ・プール属性 */
    UINT   blkcnt;         /* 獲得可能なメモリ・ブロック数 */
    UINT   blkksz;        /* メモリ・ブロックのサイズ (バイト数) */
    VP     mpf;           /* 固定長メモリ・プール領域の先頭アドレス */
    VP     mpfmb;        /* 固定長メモリ・プール管理領域の先頭アドレス */
} T_CMPF;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_mpf は、指定された *mpfid* の固定長メモリ・プールを *pk_cmpf* で指定された内容で生成します。acre_mpf は *pk_cmpf* で指定された内容で固定長メモリ・プールを生成し、生成した固定長メモリ・プール ID を返します。

1) 固定長メモリ・プール属性 (*mpfatr*)

mpfatr には以下を指定できます。

```
mpfatr := ( TA_TFIFO || TA_TPRI )
```

- TA_TFIFO (= 0x0000)

固定長メモリ・プールの待ちキューの順序を FIFO 順とします。

- TA_TPRI (= 0x0001)

固定長メモリ・プールの待ちキューの順序をタスクの現在優先度順とします。ただし、同じ現在優先度のタスクの中では FIFO 順です。

2) 獲得可能なメモリ・ブロック数 (*blkcnt*)、メモリ・ブロックのサイズ (*blkksz*)、固定長メモリ・プール領域の先頭アドレス (*mpf*)

アプリケーション側で、TSZ_MPF(*blkcnt*, *blksz*) バイトの固定長メモリ・プール領域を確保し、その先頭アドレスを *mpf* に指定します。

備考 1 TSZ_MPF マクロについては、「18.3.3 固定長メモリ・プール関連」を参照してください。

備考 2 RI600PX は、固定長メモリ・プール領域に関するアクセス権については何も関知しません。タスクが固定長メモリ・プールから獲得したメモリ・ブロックにアクセスするには、固定長メモリ・プール領域は適切にアクセス許可が設定されたメモリ・オブジェクト内に確保する必要があります。また、RI600PX は固定長メモリ・プール領域内に管理テーブルを生成します。この管理テーブルがアプリケーションによって書き換えられた場合、システムの正常な動作は保証されません。

備考 3 μ ITRON4.0 仕様には、*mpf* に NULL を指定することで RI600PX が固定長メモリ・プール領域を割り当てる機能がありますが、本サービス・コールはこの機能をサポートしていません。

備考 4 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は、以下を守ってください。

- 固定長メモリ・プール生成時に指定するメモリ・ブロック・サイズを、目的のアライメント数の倍数とする。
- 固定長メモリ・プール領域の先頭アドレスを、目的のアライメント数のアドレスとする。

3) 固定長メモリ・プール管理領域の先頭アドレス (*mpfmb*)

アプリケーション側で、TSZ_MPFMB(*blkcnt*, *blksz*) バイトの固定長メモリ・プール管理領域を確保し、その先頭アドレスを *mpfmb* に指定します。

備考 1 TSZ_MPFMB マクロについては、「18.3.3 固定長メモリ・プール関連」を参照してください。

備考 2 RI600PX は、固定長メモリ・プール管理領域に関するアクセス権については何も関知しません。通常は、固定長メモリ・プール管理領域はメモリ・オブジェクト以外でかつユーザ・スタック以外の領域に作成してください。メモリ・オブジェクト内に固定長メモリ・プール管理領域を作成した場合は、そのメモリ・オブジェクトへのオペランド・ライト・アクセスが許可されたタスクが、誤って固定長メモリ・プール管理領域を書き換えてしまう危険があります。

戻り値

マクロ	数値	意味
—	正の値	acre_mpf の正常終了 (生成された固定長メモリ・プール ID)
E_OK	0	cre_mpf の正常終了
E_RSATR	-11	予約属性 - <i>mpfatr</i> の bit0 以外のビットが 1
E_PAR	-17	パラメータ・エラー - <i>pk_cmpf</i> == NULL - <i>blkcnt</i> == 0, <i>blkcnt</i> > 65535 - <i>blksz</i> == 0, <i>blksz</i> > 65535 - TSZ_MPF(<i>blkcnt</i> , <i>blksz</i>) > VTMAX_AREASIZE - <i>mpf</i> + TSZ_MPF(<i>blkcnt</i> , <i>blksz</i>) > 0x100000000
E_ID	-18	不正 ID 番号 (cre_mpf のみ) - <i>mpfid</i> ≤ 0 - <i>mpfid</i> > VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_cmpf</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 <ul style="list-style-type: none"> - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOMEM	-33	メモリ不足 <ul style="list-style-type: none"> - <i>mpf</i> == NULL - <i>mpfmb</i> == NULL
E_NOID	-34	空き ID なし (<i>acre_mpf</i> のみ)
E_OBJ	-41	オブジェクト状態不正 (<i>cre_mpf</i> のみ) <ul style="list-style-type: none"> - <i>mpfid</i> の固定長メモリ・プールが存在する。

del_mpf

概要

固定長メモリ・プールの削除

C 言語形式

```
ER      del_mpf ( ID mpfid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
mpfid で指定された固定長メモリ・プールを削除します。

対象固定長メモリ・プールで [get_mpf](#) または [tget_mpf](#) によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、[get_mpf](#) または [tget_mpf](#) の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>mpfid</i> ≤ 0 - <i>mpfid</i> > VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>mpfid</i> の固定長メモリ・プールが存在しない。

get_mpf

概要

固定長メモリ・ブロックの獲得

C 言語形式

```
ER      get_mpf ( ID mpfid, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID
O	VP <i>*p_blk</i> ;	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し、その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には、固定長メモリ・ブロックの獲得は行わず、自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお、固定長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
rel_mpf の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
irel_mpf の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
rel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
irel_wai の発行により、待ち状態を強制的に解除された。	E_RLWAI
vrst_mpf の発行により、対象固定長メモリ・プールがリセットされた。	EV_RST
del_mpf の発行により、待ち状態を強制的に解除された。	E_DLT

備考 1 自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、[固定長メモリ・プールの生成](#)時に指定した順（FIFO 順または現在優先度順）に行われます。

備考 2 獲得したメモリ・ブロックの内容は不定です。

備考 3 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は、以下を守ってください。

- 固定長メモリ・プール生成時に指定するメモリ・ブロック・サイズを、目的のアライメント数の倍数とする。
- 固定長メモリ・プール領域の先頭アドレスを、目的のアライメント数のアドレスとする。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_blk</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mpfid</i> ≤ 0 - <i>mpfid</i> > VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>p_blk</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mpfid</i> の固定長メモリ・プールが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <i>rel_wai</i> または <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 - <i>del_mpf</i> の発行により、待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (<i>vrst_mpf</i>) による待ち解除

pget_mpf ipget_mpf

概要

固定長メモリ・ブロックの獲得（ポーリング）

C 言語形式

```
ER    pget_mpf ( ID mpfid, VP *p_blk );
ER    ipget_mpf ( ID mpfid, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID
O	VP <i>*p_blk</i> ;	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し、その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールから固定長メモリ・ブロックを獲得できなかった（空き固定長メモリ・ブロックが存在しなかった）場合には、固定長メモリ・ブロックの獲得は行わず、戻り値として E_TMOUT を返します。

備考 1 獲得したメモリ・ブロックの内容は不定です。

備考 2 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は、以下を守ってください。

- 固定長メモリ・プール生成時に指定するメモリ・ブロック・サイズを、目的のアライメント数の倍数とする。
- 固定長メモリ・プール領域の先頭アドレスを、目的のアライメント数のアドレスとする。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_blk</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mpfid</i> ≤ 0 - <i>mpfid</i> > VTMAX_MPF

マクロ	数値	意味
E_CTX	-25	<p>コンテキスト・エラー</p> <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 <p>備考 タスクから ipget_mpf を発行した場合、および非タスクから get_mpf を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。</p>
E_MACV	-26	<p>メモリ・アクセス違反 (pget_mpfのみ)</p> <ul style="list-style-type: none"> - 呼び出しタスクからの <i>p_blk</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	<p>オブジェクト未生成</p> <ul style="list-style-type: none"> - <i>mpfid</i> の固定長メモリ・プールが存在しない。
E_TMOUT	-50	ポーリング失敗

tget_mpf

概要

固定長メモリ・ブロックの獲得（タイムアウト付き）

C 言語形式

```
ER      tget_mpf ( ID mpfid, VP *p_blk, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID
O	VP <i>*p_blk</i> ;	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し、その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には、固定長メモリ・ブロックの獲得は行わず、自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお、固定長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

固定長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<i>rel_mpf</i> の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
<i>irel_mpf</i> の発行により、対象固定長メモリ・プールに固定長メモリ・ブロックが返却された。	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>vrst_mpf</i> の発行により、対象固定長メモリ・プールがリセットされた。	EV_RST
<i>tmout</i> で指定された待ち時間が経過した。	E_TMOUT
<i>del_mpf</i> の発行により、待ち状態を強制的に解除された。	E_DLT

備考 1 自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、[固定長メモリ・プールの生成時](#)に指定した順（FIFO 順または現在優先度順）に行われます。

備考 2 獲得したメモリ・ブロックの内容は不定です。

- 備考3 メモリ・ブロックのアライメント数は 1 です。これより大きいアライメント数のメモリ・ブロックを獲得したい場合は、以下を守ってください。
- 固定長メモリ・プール生成時に指定するメモリ・ブロック・サイズを、目的のアライメント数の倍数とする。
 - 固定長メモリ・プール領域の先頭アドレスを、目的のアライメント数のアドレスとする。
- 備考4 待ち時間 *tmout* に `TMO_FEVR` が指定された際には“`get_mpf` と同等の処理”を、`TMO_POL` が指定された際には“`pget_mpf` と同等の処理”を実行します。

戻り値

マクロ	数値	意味
<code>E_OK</code>	0	正常終了
<code>E_PAR</code>	-17	パラメータ・エラー <ul style="list-style-type: none"> - <code>p_blk == NULL</code> - <code>tmout < -1</code> - <code>tmout > (0x7FFFFFFF - TIC_NUME)/TIC_DENO</code>
<code>E_ID</code>	-18	不正 ID 番号 <ul style="list-style-type: none"> - <code>mpfid ≤ 0</code> - <code>mpfid > VTMAX_MPF</code>
<code>E_CTX</code>	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
<code>E_MACV</code>	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <code>p_blk</code> が示す領域に対するオペランド・ライト・アクセス許可がない。
<code>E_NOEXS</code>	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <code>mpfid</code> の固定長メモリ・プールが存在しない。
<code>E_RLWAI</code>	-49	待ち状態の強制解除 <ul style="list-style-type: none"> - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
<code>E_TMOUT</code>	-50	待ち時間が経過した、またはポーリング失敗
<code>E_DLT</code>	-51	待ちオブジェクトの削除 <ul style="list-style-type: none"> - <code>del_mpf</code> の発行により、待ち状態を強制的に解除された。
<code>EV_RST</code>	-127	オブジェクト・リセット (<code>vrst_mpf</code>) による待ち解除

rel_mpf
irel_mpf

概要

固定長メモリ・ブロックの返却

C 言語形式

```
ER    rel_mpf ( ID mpfid, VP blk );
ER    irel_mpf ( ID mpfid, VP blk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID
I	VP <i>blk</i> ;	固定長メモリ・ブロックの先頭アドレス

機能

mpfid で指定された固定長メモリ・プールに *blk* で指定された固定長メモリ・ブロックを返却します。

ただし、本サービス・コールを発行した際、対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には、固定長メモリ・ブロックの返却は行わず、該当タスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。これにより、該当タスクは、待ちキューから外れ、WAITING 状態（固定長メモリ・ブロック獲得待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>blk</i> == NULL - <i>blk</i> が不正
E_ID	-18	不正 ID 番号 - $mpfid \leq 0$ - $mpfid > VTMAX_MPF$

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irel_mpf を発行した。 - 非タスクから rel_mpf を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (rel_mpf のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>mpfid</i> の固定長メモリ・プールが存在しない。

ref_mpf
iref_mpf

概要

固定長メモリ・プール詳細情報の参照

C 言語形式

```
ER    ref_mpf ( ID mpfid, T_RMPF *pk_rmpf );
ER    iref_mpf ( ID mpfid, T_RMPF *pk_rmpf );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID
O	T_RMPF * <i>pk_rmpf</i> ;	固定長メモリ・プール詳細情報を格納する領域へのポインタ

【 固定長メモリ・プール詳細情報 T_RMPF の構造 】

```
typedef struct t_rmpf {
    ID    wtskid;          /* 待ちタスクの有無 */
    UINT  fblkcnt;        /* 空きメモリ・ブロックの総数 */
} T_RMPF;
```

機能

mpfid で指定された固定長メモリ・プールの固定長メモリ・プール詳細情報（待ちタスクの有無、空き固定長メモリ・ブロックの総数など）を *pk_rmpf* で指定された領域に格納します。

- *wtskid*

固定長メモリ・プールの待ちキューにタスクがキューイングされているか否かが格納されます。

TSK_NONE : 待ちキューにタスクはキューイングされていない
 その他 : 待ちキューの先頭にキューイングされているタスクの ID

- *fblkcnt*

空きメモリ・ブロックの総数が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>pk_rmpf</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mpfid</i> ≤ 0 - <i>mpfid</i> > VTMAX_MPF
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_mpf</i> を発行した場合、および非タスクから <i>ref_mpf</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_mpf</i> のみ) - 呼び出しタスクからの <i>pk_rmpf</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mpfid</i> の固定長メモリ・プールが存在しない。

19.2.11 メモリ・プール管理機能（可変長メモリ・プール）

以下に、RI600PX がメモリ・プール管理機能（可変長メモリ・プール）として提供しているサービス・コールの一覧を示します。

表 19 - 15 メモリ・プール管理機能（可変長メモリ・プール）

サービス・コール名	機能概要	発行有効範囲
cre_mpl	可変長メモリ・プールの生成	タスク
acre_mpl	可変長メモリ・プールの生成（ID 番号自動割付け）	タスク
del_mpl	可変長メモリ・プールの削除	タスク
get_mpl	可変長メモリ・ブロックの獲得	タスク
pget_mpl	可変長メモリ・ブロックの獲得（ポーリング）	タスク
ipget_mpl	可変長メモリ・ブロックの獲得（ポーリング）	非タスク
tget_mpl	可変長メモリ・ブロックの獲得（タイムアウト付き）	タスク
rel_mpl	可変長メモリ・ブロックの返却	タスク
ref_mpl	可変長メモリ・プール情報詳細の参照	タスク
iref_mpl	可変長メモリ・プール詳細情報の参照	非タスク

cre_mpl acre_mpl

概要

可変長メモリ・プールの生成

C 言語形式

```
ER      cre_mpl ( ID mplid, T_CMPL *pk_cmpl );
ER_ID   acre_mpl ( T_CMPL *pk_cmpl );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID
I	T_CMPL * <i>pk_cmpl</i> ;	可変長メモリ・プール生成情報を格納した領域へのポインタ

【可変長メモリ・プール生成情報 T_CMPL の構造】

```
typedef struct t_cmpl {
    ATR      mplatr;          /* 可変長メモリ・プール属性 */
    SIZE     mplsz;          /* 可変長メモリ・プール領域のサイズ (バイト数) */
    VP       mpl;           /* 可変長メモリ・プール領域の先頭アドレス */
    UINT     maxblksz;      /* 最大メモリ・ブロック・サイズ (バイト数) */
} T_CMPL;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

`cre_mpl` は、指定された `mplid` の可変長メモリ・プールを `pk_cmpl` で指定された内容で生成します。`acre_mpl` は `pk_cmpl` で指定された内容で可変長メモリ・プールを生成し、生成した可変長メモリ・プール ID を返します。

1) 可変長メモリ・プール属性 (`mplatr`)

`mplatr` には `TA_TFIFO` のみを指定できます。

- `TA_TFIFO` (= 0x0000)

可変長メモリ・プールの待ちキューの順序を FIFO 順とします。

2) 可変長メモリ・プール領域のサイズ (`mplsz`)、可変長メモリ・プール領域の先頭アドレス (`mpl`)

アプリケーション側で、`mplsz` バイトの可変長メモリ・プール領域を確保し、その先頭アドレスを `mpl` に指定します。`mpl` は 4 バイト境界でなければなりません。

備考 1 RI600PX は、可変長メモリ・プール領域に関するアクセス権については何も関知しません。タスクが可変長メモリ・プールから獲得したメモリ・ブロックにアクセスするには、可変長メモリ・プール領域は適切にアクセス許可が設定されたメモリ・オブジェクト内に確保する必要があります。また、RI600PX は可変長メモリ・プール領域内に管理テーブルを生成します。この管理テーブルがアプリケーションによって書き換えられた場合、システムの正常な動作は保証されません。

備考2 μ ITRON4.0 仕様には、*mpl* に NULL を指定することで RI600PX が可変長メモリ・プール領域を割り当てる機能がありますが、本サービス・コールはこの機能をサポートしていません。

備考3 獲得するメモリ・ブロックのアライメント数は、4 です。

3) 最大メモリ・ブロック・サイズ (*maxblksz*)

maxblksz にはメモリ・ブロックの最大サイズを指定します。実際に獲得可能メモリ・ブロックの最大サイズは、*maxblksz* より大きくなる場合があります。詳細は、「9.3.1 可変長メモリ・ブロックのサイズ」を参照してください。

戻り値

マクロ	数値	意味
—	正の値	<i>acre_mpl</i> の正常終了 (生成された可変長メモリ・プール ID)
E_OK	0	<i>cre_mpl</i> の正常終了
E_RSATR	-11	予約属性 - <i>mplatr</i> != TA_TFIFO
E_PAR	-17	パラメータ・エラー - <i>pk_cmpl</i> == NULL - <i>mplsz</i> < 24, <i>mplsz</i> > VTMAX_AREASIZE - <i>maxblksz</i> == 0, <i>maxblksz</i> > 0x0BFFFFFF4 - <i>maxblksz</i> に対して <i>mplsz</i> が小さすぎる。 - <i>mpl</i> + <i>mplsz</i> > 0x100000000 - <i>mpl</i> が 4 バイト境界でない
E_ID	-18	不正 ID 番号 (<i>cre_mpl</i> のみ) - <i>mplid</i> ≤ 0 - <i>mplid</i> > VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_cmpl</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOMEM	-33	メモリ不足 - <i>mpl</i> == NULL
E_NOID	-34	空き ID なし (<i>acre_mpl</i> のみ)
E_OBJ	-41	オブジェクト状態不正 (<i>cre_mpl</i> のみ) - <i>mplid</i> の可変長メモリ・プールが存在する。

del_mpl

概要

可変長メモリ・プールの削除

C 言語形式

```
ER    del_mpl ( ID mplid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
mplid で指定された可変長メモリ・プールを削除します。

対象可変長メモリ・プールで [get_mpl](#) または [tget_mpl](#) によって待っているタスクがある場合は、そのタスクの待ち状態を解除し、[get_mpl](#) または [tget_mpl](#) の戻り値として E_DLT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>mplid</i> ≤ 0 - <i>mplid</i> > VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスケレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>mplid</i> の可変長メモリ・プールが存在しない。

get_mpl

概要

可変長メモリ・ブロックの獲得

C 言語形式

```
ER      get_mpl ( ID mplid, UINT blksz, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID
I	UINT <i>blksz</i> ;	可変長メモリ・ブロックの要求サイズ (単位: バイト)
O	VP <i>*p_blk</i> ;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し、その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった (要求サイズ分の連続する空き領域が存在しなかった) 場合には、可変長メモリ・ブロックの獲得は行わず、自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態から WAITING 状態 (可変長メモリ・ブロック獲得待ち状態) へと遷移させます。

なお、可変長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<i>rel_mpl</i> の発行により、対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <i>rel_wai</i> の発行により、待ち状態を強制的に解除された。 - <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。 - <i>ter_tsk</i> の発行により、待ち状態を強制的に解除された。 - <i>tget_mpl</i> のパラメータ <i>tmout</i> で指定された待ち時間が経過した。 	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>vrst_mpl</i> の発行により、対象可変長メモリ・プールがリセットされた。	EV_RST
<i>del_mpl</i> の発行により、待ち状態を強制的に解除された。	E_DLT

備考 1 実際に獲得されるメモリ・ブロックのサイズについては、「9.3.1 可変長メモリ・ブロックのサイズ」を参照してください。

- 備考2 自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考3 獲得したメモリ・ブロックの内容は不定です。
- 備考4 獲得するメモリ・ブロックのアライメント数は、可変長メモリ・プールの生成方法によって異なります。
- システム・コンフィギュレーション・ファイルで生成した場合
獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには、[可変長メモリ・プール情報 \(variable_memorypool\[\]\)](#) のメモリ・プール領域に付与するセクション名 (`mpl_section`) に個別のセクション名を指定し、リンク時にそのセクションを 4 バイト境界アドレスに配置してください。
 - `cre_mpl` または `acre_mpl` サービスコールで生成した場合
獲得するメモリ・ブロックのアライメント数は 4 です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <code>blksz == 0</code> - <code>blksz</code> が獲得可能な最大サイズを超えている。 - <code>p_blk == NULL</code>
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <code>mplid ≤ 0</code> - <code>mplid > VTMAX_MPL</code>
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <code>p_blk</code> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <code>mplid</code> の可変長メモリ・プールが存在しない。
E_RLWAI	-49	待ち状態の強制解除 <ul style="list-style-type: none"> - <code>rel_wai</code> または <code>irel_wai</code> の発行により、待ち状態を強制的に解除された。
E_DLT	-51	待ちオブジェクトの削除 <ul style="list-style-type: none"> - <code>del_mpl</code> の発行により、待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (<code>vrst_mpl</code>) による待ち解除

pget_mpl ipget_mpl

概要

可変長メモリ・ブロックの獲得（ポーリング）

C 言語形式

```
ER    pget_mpl ( ID mplid, UINT blkksz, VP *p_blk );
ER    ipget_mpl ( ID mplid, UINT blkksz, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID
I	UINT <i>blkksz</i> ;	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	VP <i>*p_blk</i> ;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mplid で指定された可変長メモリ・プールから *blkksz* で指定されたサイズの可変長メモリ・ブロックを獲得し、その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には、可変長メモリ・ブロックの獲得は行わず、戻り値として E_TMOUT を返します。

- 備考 1 実際に獲得されるメモリ・ブロックのサイズについては、「[9.3.1 可変長メモリ・ブロックのサイズ](#)」を参照してください。
- 備考 2 獲得したメモリ・ブロックの内容は不定です。
- 備考 3 獲得するメモリ・ブロックのアライメント数は、可変長メモリ・プールの生成方法によって異なります。
- システム・コンフィギュレーション・ファイルで生成した場合
獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには、[可変長メモリ・プール情報 \(variable_memorypool\[\]\)](#) のメモリ・プール領域に付与するセクション名 (*mpl_section*) に個別のセクション名を指定し、リンク時にそのセクションを 4 バイト境界アドレスに配置してください。
 - *cre_mpl* または *acre_mpl* サービスコールで生成した場合
獲得するメモリ・ブロックのアライメント数は 4 です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>blksz</i> == 0 - <i>blksz</i> が獲得可能な最大サイズを超えている。 - <i>p_blk</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mplid</i> ≤ 0 - <i>mplid</i> > VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>ipget_mpl</i> を発行した場合、および非タスクから <i>pget_mpl</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>pget_mpl</i> のみ) - 呼び出しタスクからの <i>p_blk</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mplid</i> の可変長メモリ・プールが存在しない。
E_TMOUT	-50	ポーリング失敗

tget_mpl

概要

可変長メモリ・ブロックの獲得（タイムアウト付き）

C 言語形式

```
ER      tget_mpl ( ID mplid, UINT blksz, VP *p_blk, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID
I	UINT <i>blksz</i> ;	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	VP <i>*p_blk</i> ;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ
I	TMO <i>tmout</i> ;	待ち時間（単位：ミリ秒） TMO_FEVR： 永久待ち TMO_POL： ポーリング 数値： 待ち時間

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し、その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし、本サービス・コールを発行した際、対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には、可変長メモリ・ブロックの獲得は行わず、自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち、RUNNING 状態からタイムアウト付きの WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお、可変長メモリ・ブロック獲得待ち状態の解除は、以下の場合に行われます。

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<i>rel_mpl</i> の発行により、対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された。	E_OK
送信待ちキュー先頭のタスクが、以下のいずれかによって待ち状態を強制的に解除された。 <ul style="list-style-type: none"> - <i>rel_wai</i> の発行により、待ち状態を強制的に解除された。 - <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。 - <i>ter_tsk</i> の発行により、待ち状態を強制的に解除された。 - <i>tget_mpl</i> のパラメータ <i>tmout</i> で指定された待ち時間が経過した。 	E_OK
<i>rel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>irel_wai</i> の発行により、待ち状態を強制的に解除された。	E_RLWAI
<i>vrst_mpl</i> の発行により、対象可変長メモリ・プールがリセットされた。	EV_RST

可変長メモリ・ブロック獲得待ち状態の解除操作	戻り値
<i>tmout</i> で指定された待ち時間が経過した。	E_TMOUT
<i>del_mpl</i> の発行により、待ち状態を強制的に解除された。	E_DLT

- 備考 1 実際に獲得されるメモリ・ブロックのサイズについては、「9.3.1 可変長メモリ・ブロックのサイズ」を参照してください。
- 備考 2 自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は、FIFO 順に行われます。
- 備考 3 獲得したメモリ・ブロックの内容は不定です。
- 備考 4 獲得するメモリ・ブロックのアライメント数は、可変長メモリ・プールの生成方法によって異なります。
- システム・コンフィギュレーション・ファイルで生成した場合
獲得するメモリ・ブロックのアライメント数は 1 です。アライメント数を 4 とするには、可変長メモリ・プール情報 (*variable_memorypool[]*) のメモリ・プール領域に付与するセクション名 (*mpl_section*) に個別のセクション名を指定し、リンク時にそのセクションを 4 バイト境界アドレスに配置してください。
 - *cre_mpl* または *acre_mpl* サービスコールで生成した場合
獲得するメモリ・ブロックのアライメント数は 4 です。
- 備考 5 待ち時間 *tmout* に *TMO_FEVR* が指定された際には“*get_mpl* と同等の処理”を、*TMO_POL* が指定された際には“*pget_mpl* と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>blksz</i> == 0 - <i>blksz</i> が獲得可能な最大サイズを超えている。 - <i>p_blk</i> == NULL - <i>tmout</i> < -1 - <i>tmout</i> > (0x7FFFFFFF - <i>TIC_NUME</i>)/<i>TIC_DENO</i>
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <i>mplid</i> ≤ 0 - <i>mplid</i> > <i>VTMAX_MPL</i>
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - ディスパッチ禁止状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>p_blk</i> が示す領域に対するオペランド・ライト・アクセス許可がない。

マクロ	数値	意味
E_NOEXS	-42	オブジェクト未生成 - <i>mplid</i> の可変長メモリ・プールが存在しない。
E_RLWAI	-49	待ち状態の強制解除 - <i>rel_wai</i> または <i>irel_wai</i> の発行により、待ち状態を強制的に解除された。
E_TMOUT	-50	待ち時間が経過した、またはポーリング失敗
E_DLT	-51	待ちオブジェクトの削除 - <i>del_mpl</i> の発行により、待ち状態を強制的に解除された。
EV_RST	-127	オブジェクト・リセット (<i>vrst_mpl</i>) による待ち解除

rel_mpl

概要

可変長メモリ・ブロックの返却

C 言語形式

```
ER    rel_mpl ( ID mplid, VP blk );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID
I	VP <i>blk</i> ;	可変長メモリ・ブロックの先頭アドレス

機能

mplid で指定された可変長メモリ・プールに *blk* で指定された可変長メモリ・ブロックを返却します。

可変長メモリ・ブロックを返却したあと、対象可変長メモリ・プールの待ちキューにキューイングされているタスクをキューの先頭から調べていき、待ちタスクが要求するサイズのメモリを割り当てられる場合はメモリを割り当てます。この動作を待ちキューにタスクがなくなるか、メモリが割り当てられなくなるまで繰り返します。これにより、メモリを獲得できたタスクは、待ちキューから外れ、WAITING 状態（可変長メモリ・ブロック獲得待ち状態）から READY 状態へ、または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 RI600PX では、*blk* に関して簡易的なエラー検出しか行っていません。*blk* には、必ず正しい値を指定してください。*blk* が不正でかつエラー検出されない場合、以後の動作は保証されません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>blk</i> == NULL - <i>blk</i> が不正
E_ID	-18	不正 ID 番号 - $mplid \leq 0$ - $mplid > VTMAX_MPL$

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>mplid</i> の可変長メモリ・プールが存在しない。

ref_mpl
iref_mpl

概要

可変長メモリ・プール詳細情報の参照

C 言語形式

```
ER    ref_mpl ( ID mplid, T_RMPL *pk_rmpl );
ER    iref_mpl ( ID mplid, T_RMPL *pk_rmpl );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID
O	T_RMPL * <i>pk_rmpl</i> ;	可変長メモリ・プール詳細情報を格納する領域へのポインタ

【可変長メモリ・プール詳細情報 T_RMPL の構造】

```
typedef struct t_rmpl {
    ID    wtskid;          /* 待ちタスクの有無 */
    SIZE  fmplsz;        /* 空き領域の合計サイズ */
    UINT  fblksz;        /* 獲得可能なメモリ・ブロックの最大サイズ */
} T_RMPL;
```

機能

mplid で指定された可変長メモリ・プールの可変長メモリ・プール詳細情報（待ちタスクの有無、空き可変長メモリ・ブロックの合計サイズなど）を *pk_rmpl* で指定された領域に格納します。

- *wtskid*
可変長メモリ・プールの待ちキューにタスクがキューイングされているか否かが格納されます。
TSK_NONE : 待ちキューにタスクはキューイングされていない
 その他 : 待ちキューの先頭にキューイングされているタスクの ID
- *fmplsz*
可変長メモリ・プールの空き領域の合計サイズ（単位：バイト）が格納されます。
- *fblksz*
獲得可能なメモリ・ブロックの最大サイズ（単位：バイト）が格納されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>pk_rmpl</i> == NULL
E_ID	-18	不正 ID 番号 - <i>mplid</i> ≤ 0 - <i>mplid</i> > VTMAX_MPL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_mpl</i> を発行した場合、および非タスクから <i>ref_mpl</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_mpl</i> のみ) - 呼び出しタスクからの <i>pk_rmpl</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>mplid</i> の可変長メモリ・プールが存在しない。

19.2.12 時間管理機能

以下に、RI600PX が時間管理機能として提供しているサービス・コールの一覧を示します。

表 19 - 16 時間管理機能

サービス・コール名	機能概要	発行有効範囲
set_tim	システム時刻の設定	タスク
iset_tim	システム時刻の設定	非タスク
get_tim	システム時刻の参照	タスク
iget_tim	システム時刻の参照	非タスク
cre_cyc	周期ハンドラの生成	タスク
acre_cyc	周期ハンドラの生成 (ID 番号自動割付け)	タスク
del_cyc	周期ハンドラの削除	タスク
sta_cyc	周期ハンドラの動作開始	タスク
ista_cyc	周期ハンドラの動作開始	非タスク
stp_cyc	周期ハンドラの動作停止	タスク
istp_cyc	周期ハンドラの動作停止	非タスク
ref_cyc	周期ハンドラ詳細情報の参照	タスク
iref_cyc	周期ハンドラ詳細情報の参照	非タスク
cre_alm	アラーム・ハンドラの生成	タスク
acre_alm	アラーム・ハンドラの生成 (ID 番号自動割付け)	タスク
del_alm	アラーム・ハンドラの削除	タスク
sta_alm	アラーム・ハンドラの動作開始	タスク
ista_alm	アラーム・ハンドラの動作開始	非タスク
stp_alm	アラーム・ハンドラの動作停止	タスク
istp_alm	アラーム・ハンドラの動作停止	非タスク
ref_alm	アラーム・ハンドラ詳細情報の参照	タスク
iref_alm	アラーム・ハンドラ詳細情報の参照	非タスク

set_tim iset_tim

概要

システム時刻の設定

C 言語形式

```
ER    set_tim ( SYSTIM *p_sysstim );
ER    iset_tim ( SYSTIM *p_sysstim );
```

パラメータ

I/O	パラメータ	説明
I	SYSTIM *p_sysstim;	システム時刻情報を格納した領域へのポインタ

【システム時刻情報 SYSTIM の構造】

```
typedef struct  systim {
    UH    utime;          /* システム時刻 (上位 16 ビット) */
    UW    ltime;          /* システム時刻 (下位 32 ビット) */
} SYSTIM;
```

機能

システム時刻 (単位: ミリ秒) を *p_sysstim* で指定された時間に変更します。

備考 システム時刻を変更しても、それ以前に行われた時間管理要求 (タスクのタイムアウト, *dly_tsk* によるタスクの遅延, 周期ハンドラ, およびアラーム・ハンドラ) が発生する実時刻は変化しません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_sysstim</i> == NULL

マクロ	数値	意味
E_CTX	-25	<p>コンテキスト・エラー</p> <ul style="list-style-type: none">- CPU ロック状態から本サービス・コールを発行した。- 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 <p>備考 タスクから <code>iset_tim</code> を発行した場合、および非タスクから <code>set_tim</code> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。</p>
E_MACV	-26	<p>メモリ・アクセス違反 (<code>set_tim</code> のみ)</p> <ul style="list-style-type: none">- 呼び出しタスクからの <code>p_sysstim</code> が示す領域に対するオペランド・リード・アクセス許可がない。

get_tim iget_tim

概要

システム時刻の参照

C 言語形式

```
ER    get_tim ( SYSTIM *p_system );
ER    iget_tim ( SYSTIM *p_system );
```

パラメータ

I/O	パラメータ	説明
O	SYSTIM *p_system;	システム時刻情報を格納する領域へのポインタ

【システム時刻情報 SYSTIM の構造】

```
typedef struct    systim {
    UH            utime;        /* システム時刻 (上位 16 ビット) */
    UW            ltime;        /* システム時刻 (下位 32 ビット) */
} SYSTIM;
```

機能

システム時刻 (単位: ミリ秒) を *p_system* で指定された領域に格納します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_system</i> == NULL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iget_tim</i> を発行した場合、および非タスクから <i>get_tim</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 (get_tim のみ) - 呼び出しタスクからの <i>p_systim</i> が示す領域に対するオペランド・ライト・アクセス許可がない。

cre_cyc acre_cyc

概要

周期ハンドラの生成

C 言語形式

```
ER      cre_cyc ( ID cycid, T_CCYC *pk_ccyc );
ER_ID   acre_cyc ( T_CCYC *pk_ccyc );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>cycid</i> ;	周期ハンドラの ID
I	T_CCYC * <i>pk_ccyc</i> ;	周期ハンドラ生成情報を格納した領域へのポインタ

【周期ハンドラ生成情報 T_CCYC の構造】

```
typedef struct t_ccyc {
    ATR      cycatr;          /* 周期ハンドラ属性 */
    VP_INT   exinf;         /* 拡張情報 */
    FP      cycchr;         /* 周期ハンドラの実行開始アドレス */
    RELTIM   cyctim;        /* 起動周期 (ミリ秒) */
    RELTIM   cycphs;        /* 起動位相 (ミリ秒) */
} T_CCYC;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_cyc は、指定された *cycid* の周期ハンドラを *pk_ccyc* で指定された内容で生成します。acre_cyc は *pk_ccyc* で指定された内容で周期ハンドラを生成し、生成した周期ハンドラ ID を返します。

1) 周期ハンドラ属性 (*cycatr*)

cycatr には以下を指定できます。

```
cycatr := ( TA_HLNG | [ TA_STA ] | [ TA_PHS ] )
```

- TA_HLNG (= 0x0000)

周期ハンドラの記述言語は、C 言語のみをサポートしています。

- TA_STA (= 0x0002)

TA_STA を指定すると周期ハンドラは動作状態 (STA 状態) となり、指定しないと停止状態 (STP 状態) となります。

- TA_PHS (= 0x0004)

TA_PHS を指定した場合、周期ハンドラの動作を開始する時に周期ハンドラの起動位相を保存して、次に起動すべき時刻を決定します。TA_PHS が指定されていない場合は、次に起動する時刻は *sta_cyc*、*ista_cyc* が呼び出された時刻から *cyctim* 後となります。

「10.6.5 周期ハンドラの動作開始」も参照してください。

- 2) 拡張情報 (*exinf*)
exinf は、周期ハンドラに引数として渡されます。*exinf* は、ユーザが生成する周期ハンドラに関する情報を設定するなどの目的で自由に使用できます。
- 3) 周期ハンドラの実行開始アドレス (*cychdr*)
cychdr には、周期ハンドラの実行開始アドレスを指定します。
- 4) 起動周期 (*cyctim*)、起動位相 (*cycphs*)
cyctim には起動周期 (ミリ秒) を指定します。
cycphs には、本サービス・コール呼出し時点からの最初の起動までの時間 (ミリ秒) を指定します。TA_STA、TA_PHS 両方の指定がない場合、*cycphs* は無視されます。

戻り値

マクロ	数値	意味
—	正の値	acre_cyc の正常終了 (生成された周期ハンドラ ID)
E_OK	0	cre_cyc の正常終了
E_RSATR	-11	予約属性 - <i>cycatr</i> の bit1, bit2 以外のビットが 1
E_PAR	-17	パラメータ・エラー - <i>pk_ccyc</i> == NULL - <i>cychdr</i> == NULL - <i>cyctim</i> == 0, <i>cyctim</i> > (0x7FFFFFFF - TIC_NUME)/TIC_DENO - <i>cyctim</i> < <i>cycphs</i>
E_ID	-18	不正 ID 番号 (<i>cre_cyc</i> のみ) - <i>cycid</i> ≤ 0 - <i>cycid</i> > VTMAX_CYH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_ccyc</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOID	-34	空き ID なし (<i>acre_cyc</i> のみ)
E_OBJ	-41	オブジェクト状態不正 (<i>cre_cyc</i> のみ) - <i>cycid</i> の周期ハンドラが存在する。

del_cyc

概要

周期ハンドラの削除

C 言語形式

```
ER    del_cyc ( ID cycid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>cycid</i> ;	周期ハンドラの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
cycid で指定された周期ハンドラを削除します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>cycid</i> ≤ 0 - <i>cycid</i> > VTMAX_CYH
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>cycid</i> の周期ハンドラが存在しない。

sta_cyc
ista_cyc

概要

周期ハンドラの動作開始

C 言語形式

```
ER    sta_cyc ( ID cycid );
ER    ista_cyc ( ID cycid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>cycid</i> ;	周期ハンドラの ID

機能

cycid で指定された周期ハンドラの動作状態を停止状態（STP 状態）から動作状態（STA 状態）へと遷移させます。これにより、対象周期ハンドラは、RI600PX の起動対象となります。

なお、本サービス・コールの発行から 1 回目の起動要求が発行されるまでの相対時間間隔は、[周期ハンドラの生成時に TA_PHS 属性を指定しているか否か](#)により異なります。詳細は、「[10.6.5 周期ハンドラの動作開始](#)」を参照してください。

- TA_PHS 属性を指定した場合
起動位相および起動周期にしたがって、起動タイミングが設定されます。
ただし、対象周期ハンドラの動作状態が開始状態の場合には、本サービス・コールを発行しても何も処理は行わず、エラーとしても扱いません
- TA_PHS 属性を指定しない場合
本サービス・コールの発行時点を基点に、起動周期にしたがって起動タイミングが設定されます。
なお、起動タイミング設定処理については、対象周期ハンドラの動作状態に関係なく実行されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>cycid</i> ≤ 0 - <i>cycid</i> > VTMAX_CYH

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ista_cyc を発行した場合、および非タスクから sta_cyc を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_NOEXS	-42	オブジェクト未生成 - <i>cycid</i> の周期ハンドラが存在しない。

stp_cyc istp_cyc

概要

周期ハンドラの動作停止

C 言語形式

```
ER    stp_cyc ( ID cycid );
ER    istp_cyc ( ID cycid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>cycid</i> ;	周期ハンドラの ID

機能

cycid で指定された周期ハンドラの動作状態を動作状態 (STA 状態) から停止状態 (STP 状態) へと遷移させます。これにより、本サービス・コールの発行から *sta_cyc* または *ista_cyc* が発行されるまでの間、対象周期ハンドラは、RI600PX の起動対象から除外されます。

備考 本サービス・コールでは、停止要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、対象周期ハンドラの動作状態が停止状態 (STP 状態) へと遷移していた場合には、何も処理は行わず、エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>cycid</i> ≤ 0 - <i>cycid</i> > <i>VTMAX_CYH</i>
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>istp_cyc</i> を発行した場合、および非タスクから <i>stp_cyc</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。

マクロ	数値	意味
E_NOEXS	-42	オブジェクト未生成 - <i>cycid</i> の周期ハンドラが存在しない。

ref_cyc
iref_cyc

概要

周期ハンドラ詳細情報の参照

C 言語形式

```
ER    ref_cyc ( ID cycid, T_RCYC *pk_rcyc );
ER    iref_cyc ( ID cycid, T_RCYC *pk_rcyc );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>cycid</i> ;	周期ハンドラの ID
O	T_RCYC * <i>pk_rcyc</i> ;	周期ハンドラ詳細情報を格納する領域へのポインタ

【周期ハンドラ詳細情報 T_RCYC の構造】

```
typedef struct t_rcyc {
    STAT    cycstat;          /* 現在状態 */
    RELTIM  lefttim;        /* 残り時間 */
} T_RCYC;
```

機能

cycid で指定された周期ハンドラの周期ハンドラ詳細情報（現在状態、残り時間など）を *pk_rcyc* で指定された領域に格納します。

- *cycstat*

周期ハンドラの現在状態が格納されます。

TCYC_STP : 停止状態 (STP 状態)

TCYC_STA : 動作状態 (STA 状態)

- *lefttim*

周期ハンドラが次に起動するまでの残り時間（単位：ミリ秒）が格納されます。対象周期ハンドラが停止状態の場合、*lefttim* は不定値となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>pk_rcyc</i> == NULL
E_ID	-18	不正 ID 番号 - <i>cycid</i> ≤ 0 - <i>cycid</i> > VTMAX_CYH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_cyc</i> を発行した場合、および非タスクから <i>ref_cyc</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_cyc</i> のみ) - 呼び出しタスクからの <i>pk_rcyc</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>cycid</i> の周期ハンドラが存在しない。

cre_alm
acre_alm

概要

アラーム・ハンドラの生成

C 言語形式

```
ER      cre_alm ( ID almid, T_CALM *pk_calm );
ER_ID  acre_alm ( T_CALM *pk_calm );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>almid</i> ;	アラーム・ハンドラの ID
I	T_CALM * <i>pk_calm</i> ;	アラーム・ハンドラ生成情報を格納した領域へのポインタ

【 アラーム・ハンドラ生成情報 T_CALM の構造 】

```
typedef struct t_calm {
    ATR    almatr;          /* アラーム・ハンドラ属性 */
    VP_INT exinf;          /* 拡張情報 */
    FP     almhdr;        /* アラーム・ハンドラの実行開始アドレス */
} T_CALM;
```

機能

本サービス・コールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
cre_alm は、指定された *almid* のアラーム・ハンドラを *pk_calm* で指定された内容で生成します。**acre_alm** は *pk_calm* で指定された内容でアラーム・ハンドラを生成し、生成したアラーム・ハンドラ ID を返します。

- 1) アラーム・ハンドラ属性 (*almatr*)
almatr には、TA_HLNG のみを指定できます。
 - TA_HLNG (= 0x0000)
 アラーム・ハンドラの記述言語は、C 言語のみをサポートしています。
- 2) 拡張情報 (*exinf*)
exinf は、アラーム・ハンドラに引数として渡されます。*exinf* は、ユーザが生成するアラーム・ハンドラに関する情報を設定するなどの目的で自由に使用できます。
- 3) アラーム・ハンドラの実行開始アドレス (*almhdr*)
almhdr には、アラーム・ハンドラの実行開始アドレスを指定します。

戻り値

マクロ	数値	意味
—	正の値	acre_alm の正常終了 (生成されたアラーム・ハンドラ ID)
E_OK	0	cre_alm の正常終了
E_RSATR	-11	予約属性 - <i>almatr</i> != TA_HLNG
E_PAR	-17	パラメータ・エラー - <i>pk_calm</i> == NULL - <i>almhdr</i> == NULL
E_ID	-18	不正 ID 番号 (cre_alm のみ) - <i>almid</i> ≤ 0 - <i>almid</i> > VTMAX_CYH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_calm</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOID	-34	空き ID なし (acre_alm のみ)
E_OBJ	-41	オブジェクト状態不正 (cre_alm のみ) - <i>almid</i> のアラーム・ハンドラが存在する。

del_alm

概要

アラーム・ハンドラの削除

C 言語形式

```
ER      del_alm ( ID almid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>almid</i> ;	アラーム・ハンドラの ID

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
almid で指定されたアラーム・ハンドラを削除します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $almid \leq 0$ - $almid > VTMAX_CYH$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、 信頼されたドメイン に所属していない。
E_NOEXS	-42	オブジェクト未生成 - <i>almid</i> のアラーム・ハンドラが存在しない。

sta_alm
ista_alm

概要

アラーム・ハンドラの動作開始

C 言語形式

```
ER    sta_alm ( ID almid, RELTIM almtim );
ER    ista_alm ( ID almid, RELTIM almtim );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>almid</i> ;	アラーム・ハンドラの ID
I	RELTIM <i>almtim</i> ;	起動時刻（単位：ミリ秒）

機能

almid で指定されたアラーム・ハンドラの起動時刻を、本サービス・コールが呼び出された時刻から *almtim* ミリ秒後に設定し、動作状態（STA 状態）にします。これにより、対象アラーム・ハンドラは、RI600PX の起動対象となります。

備考 1 *almtim* に 0 を指定すると、次の基本クロック割り込み時にアラーム・ハンドラが起動されます。

備考 2 対象アラーム・ハンドラがすでに動作状態の場合でも、本サービス・コールは起動時刻を再設定します。以前の起動時刻の設定は無効となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>almtim</i> > (0x7FFFFFFF - TIC_NUME)/TIC_DENO
E_ID	-18	不正 ID 番号 - <i>almid</i> ≤ 0 - <i>almid</i> > VTMAX_ALH

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから ista_alm を発行した場合、および非タスクから stp_alm を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_NOEXS	-42	オブジェクト未生成 - <i>almid</i> のアラーム・ハンドラが存在しない。

stp_alm istp_alm

概要

アラーム・ハンドラの動作停止

C 言語形式

```
ER      stp_alm ( ID almid );
ER      istp_alm ( ID almid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>almid</i> ;	アラーム・ハンドラの ID

機能

almid で指定されたアラーム・ハンドラの動作状態を動作状態（STA 状態）から停止状態（STP 状態）へと遷移させます。これにより、本サービス・コールの発行から *sta_alm* または *ista_alm* が発行されるまでの間、対象アラーム・ハンドラは、RI600PX の起動対象から除外されます。

備考 本サービス・コールでは、停止要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、対象アラーム・ハンドラの動作状態が停止状態（STP 状態）へと遷移していた場合には、何も処理は行わず、エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>almid</i> ≤ 0 - <i>almid</i> > VTMAX_ALH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>istp_alm</i> を発行した場合、および非タスクから <i>stp_alm</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。

マクロ	数値	意味
E_NOEXS	-42	オブジェクト未生成 - <i>almid</i> のアラーム・ハンドラが存在しない。

ref_alm iref_alm

概要

アラーム・ハンドラ詳細情報の参照

C 言語形式

```
ER    ref_alm ( ID almid, T_RALM *pk_ralm );
ER    iref_alm ( ID almid, T_RALM *pk_ralm );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>almid</i> ;	アラーム・ハンドラの ID
O	T_RALM * <i>pk_ralm</i> ;	アラーム・ハンドラ詳細情報を格納する領域へのポインタ

【アラーム・ハンドラ詳細情報 T_RALM の構造】

```
typedef struct t_alm {
    STAT    almstat;          /* 現在状態 */
    RELTIM  lefttim;        /* 残り時間 */
} T_RALM;
```

機能

almid で指定されたアラーム・ハンドラのアラーム・ハンドラ詳細情報（現在状態、残り時間など）を *pk_ralm* で指定された領域に格納します。

- *almstat*

アラーム・ハンドラの現在状態が格納されます。

TALM_STP : 停止状態 (STP 状態)

TALM_STA : 動作状態 (STA 状態)

- *lefttim*

アラーム・ハンドラが次に起動するまでの残り時間（単位：ミリ秒）が格納されます。対象アラーム・ハンドラが停止状態の場合、*lefttim* は不定値となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>pk_ralm</i> == NULL
E_ID	-18	不正 ID 番号 - <i>almid</i> ≤ 0 - <i>almid</i> > VTMAX_ALH
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_alm</i> を発行した場合、および非タスクから <i>ref_alm</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_alm</i> のみ) - 呼び出しタスクからの <i>pk_ralm</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - <i>almid</i> のアラーム・ハンドラが存在しない。

19.2.13 システム状態管理機能

以下に、RI600PX がシステム状態管理機能として提供しているサービス・コールの一覧を示します。

表 19 - 17 システム状態管理機能

サービス・コール名	機能概要	発行有効範囲
rot_rdq	タスクの優先順位の回転	タスク
irotd_rdq	タスクの優先順位の回転	非タスク
get_tid	RUNNING 状態のタスクの参照	タスク
iget_tid	RUNNING 状態のタスクの参照	非タスク
loc_cpu	CPU ロック状態への移行	タスク
iloc_cpu	CPU ロック状態への移行	非タスク
unl_cpu	CPU ロック状態の解除	タスク
iunl_cpu	CPU ロック状態の解除	非タスク
dis_dsp	ディスパッチの禁止	タスク
ena_dsp	ディスパッチの許可	タスク
sns_ctx	コンテキスト種別の参照	タスク, 非タスク
sns_loc	CPU ロック状態の参照	タスク, 非タスク
sns_dsp	ディスパッチ禁止状態の参照	タスク, 非タスク
sns_dpn	ディスパッチ保留状態の参照	タスク, 非タスク
vsys_dwn	システム・ダウン	タスク, 非タスク
ivsys_dwn	システム・ダウン	タスク, 非タスク
vsta_knl	RI600PX の起動	タスク, 非タスク
ivsta_knl	RI600PX の起動	タスク, 非タスク

rot_rdq
irotd_rdq

概要

タスクの優先順位の回転

C 言語形式

```
ER    rot_rdq ( PRI tskpri );
ER    irot_rdq ( PRI tskpri );
```

パラメータ

I/O	パラメータ	説明
I	PRI <i>tskpri</i> ;	タスクの優先度 TPRI_SELF : 自タスクのベース優先度 数値 : タスクの優先度

機能

tskpri で指定された優先度に対応したレディ・キューの先頭タスクを最後尾につなぎかえ、タスクの実行順序を明示的に変更します。

- 備考 1 本サービス・コールでは、レディ・キューの対象優先度にタスクが 1 つもキューイングされていなかった場合、何も処理は行わず、エラーとしても扱いません。
- 備考 2 本サービス・コールを周期ハンドラなどから一定周期で発行することにより、ラウンドロビン・スケジューリングを実現することができます。
- 備考 3 RI600PX におけるレディ・キューは、優先度をキーとしたハッシュ・テーブルであり、実行可能な状態 (RUNNING 状態または READY 状態) へと遷移したタスクが FIFO 順でキューイングされます。このため、スケジューラは、起動された際にレディ・キューの優先度高位から検出処理を実行し、キューイングされているタスクを検出した場合には、該当優先度の先頭タスクに CPU の利用権を与えることにより、RI600PX のスケジューリング方式 (優先度方式、FCFS 方式) を実現しています。
- 備考 4 ミューテックスをロック中のタスクの現在優先度は、ベース優先度と異なる場合があります。この場合、そのタスクが本サービス・コールで *tskpri* に **TPRI_SELF** を指定しても、自タスクが属する現在優先度のレディ・キューを回転することはできません。
- 備考 5 現在優先度とベース優先度については、「[8.2.2 現在優先度とベース優先度](#)」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー - <i>tskpri</i> < 0 - <i>tskpri</i> > TMAX_TPRI - irot_rdq を発行した際、 <i>tskpri</i> に TPRI_SELF を指定した。
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから irot_rdq を発行した。 - 非タスクから rot_rdq を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (rot_rdq のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。

get_tid iget_tid

概要

RUNNING 状態のタスクの参照

C 言語形式

```
ER    get_tid ( ID *p_tskid );
ER    iget_tid ( ID *p_tskid );
```

パラメータ

I/O	パラメータ	説明
O	ID *p_tskid;	タスク ID を格納する領域へのポインタ

機能

RUNNING 状態のタスクの ID を *p_tskid* で指定された領域に格納します。

RUNNING 状態のタスクが存在しない場合 (IDLE 状態) には, *p_tskid* で指定された領域に **TSK_NONE** を格納します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_tskid</i> == NULL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iget_tid</i> を発行した場合, および非タスクから <i>get_tid</i> を発行した場合, コンテキスト・エラーは検出されず, システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>get_tid</i> のみ) - 呼び出しタスクからの <i>p_tskid</i> が示す領域に対するオペランド・ライト・アクセス許可がない。

loc_cpu
iloc_cpu

概要

CPU ロック状態への移行

C 言語形式

```
ER      loc_cpu ( void );
ER      iloc_cpu ( void );
```

パラメータ

なし

機能

システムを CPU ロック状態へ移行させます。

CPU ロック状態では、タスクのスケジューリングは禁止され、カーネル管理割り込みもマスクされます。つまり、カーネル管理外割り込みハンドラを除くすべての処理プログラムに対して、排他的に処理を行うことができます。

ただし、CPU ロック状態では、発行可能なサービス・コールは以下に制限されます。

発行可能なサービス・コール	機能概要
ext_tsk	自タスクの終了 (CPU ロック状態は解除されます)
exd_tsk	自タスクの終了と削除 (CPU ロック状態は解除されます)
sns_tex	タスク例外禁止状態の参照
loc_cpu , iloc_cpu	CPU ロック状態への移行
unl_cpu , iunl_cpu	CPU ロック状態の解除
sns_loc	CPU ロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_ctx	コンテキスト種別の参照
sns_dpn	ディスパッチ保留状態の参照
vsys_dwn , ivsys_dwn	システム・ダウン

CPU ロック状態は、[unl_cpu](#)、[iunl_cpu](#)、[ext_tsk](#)、および [exd_tsk](#) によって解除されます。

備考 1 本サービス・コールの発行により変更した CPU ロック状態の解除は、本サービス・コールを発行した処理プログラムが終了する以前に行う必要があります。

備考 2 本サービス・コールでは、ロック要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、システム状態種別が CPU ロック状態へと変更されていた場合には、何も処理は行わず、エラーとしても扱いません。

備考 3 RI600PX では、一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため、本サービス・コールの発行により、該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には、時間管理機能が正常に動作しなくなる場合があります。

備考4 “カーネル管理割り込み”については、「12.1 割り込みの種類」を参照してください。

備考5 `chg_ims` によって割り込みマスクを0以外に変更している間は、`loc_cpu` は `E_ILUSE` エラーを返します。

戻り値

マクロ	数値	意味
<code>E_OK</code>	0	正常終了
<code>E_CTX</code>	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <code>iloc_cpu</code> を発行した場合、および非タスクから <code>loc_cpu</code> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
<code>E_ILUSE</code>	-28	サービス・コール不正使用 - タスクで <code>chg_ims</code> によって割り込みマスクを0以外に変更している状態から本サービス・コールを発行した。

unl_cpu
iunl_cpu

概要

CPU ロック状態の解除

C 言語形式

```
ER    unl_cpu ( void );
ER    iunl_cpu ( void );
```

パラメータ

なし

機能

システムを CPU ロック解除状態へ移行させます。

備考 1 本サービス・コールでは、解除要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、システム状態種別が非 CPU ロック状態へと変更されていた場合には、何も処理は行わず、エラーとしても扱いません。

備考 2 本サービス・コールでは、`dis_dsp` の発行により変更されたディスパッチ禁止状態の解除処理は行われません。

備考 3 `ext_tsk` および `exd_tsk` によっても、CPU ロック状態が解除されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから unl_cpu を発行した。 - タスクから iunl_cpu を発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 (unl_cpu のみ) <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。

dis_dsp

概要

ディスパッチの禁止

C 言語形式

```
ER    dis_dsp ( void );
```

パラメータ

なし

機能

システムをディスパッチ禁止状態へ移行させます。

ディスパッチ禁止状態では、タスクのスケジューリングが禁止されるため、他のタスクに対して排他的に処理を行うことができます。

ディスパッチ禁止状態へ遷移する操作は、以下の通りです。

- [dis_dsp](#)
- [chg_ims](#) によって割り込みマスク (PSW.IPL) を 0 以外に変更

また、ディスパッチ禁止状態を解除する操作は、以下の通りです。

- [ena_dsp](#)
- [ext_tsk](#)
- [exd_tsk](#)
- [chg_ims](#) によって割り込みマスク (PSW.IPL) を 0 に変更

備考 1 本サービス・コールの発行により変更したディスパッチ禁止状態の解除は、本サービス・コールを発行したタスクが DORMANT 状態へと遷移する以前に行う必要があります。

備考 2 本サービス・コールでは、禁止要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、システム状態種別がディスパッチ禁止状態へと変更されていた場合には、何も処理は行わず、エラーとしても扱いません。

備考 3 ディスパッチ禁止状態の間に“自タスクを状態遷移させる可能性のあるサービス・コール ([wai_sem](#), [wai_flg](#) など)”を発行した場合には、要求条件の即時成立／不成立を問わず、戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

ena_dsp

概要

ディスパッチの許可

C 言語形式

```
ER    ena_dsp ( void );
```

パラメータ

なし

機能

システムをディスパッチ許可状態へ移行させます。
ディスパッチ禁止状態へ移行する操作は、以下の通りです。

- [dis_dsp](#)
- [chg_ims](#) によって割り込みマスク (PSW.IPL) を 0 以外に変更

また、ディスパッチ許可状態へ移行する操作は、以下の通りです。

- [ena_dsp](#)
- [ext_tsk](#)
- [exd_tsk](#)
- [chg_ims](#) によって割り込みマスク (PSW.IPL) を 0 に変更

備考 1 本サービス・コールでは、許可要求のキューイングが行われません。このため、すでに本サービス・コールが発行され、システム状態種別がディスパッチ許可状態へと変更されていた場合には、何も処理は行わず、エラーとしても扱いません。

備考 2 ディスパッチ禁止状態の間に“自タスクを状態遷移させる可能性のあるサービス・コール ([wai_sem](#), [wai_flg](#) など)”を発行した場合には、要求条件の即時成立／不成立を問わず、戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。

sns_ctx

概要

コンテキスト種別の参照

C 言語形式

```
BOOL    sns_ctx ( void );
```

パラメータ

なし

機能

本サービス・コールは、本サービス・コールを発行した処理プログラムのコンテキスト種別（非タスク・コンテキスト、タスク・コンテキスト）を調べます。本サービス・コールは、戻り値として、非タスク・コンテキストの場合は TRUE、タスク・コンテキストの場合は FALSE を返します

戻り値

マクロ	数値	意味
TRUE	1	正常終了（非タスク・コンテキスト）
FALSE	0	正常終了（タスク・コンテキスト）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

sns_loc**概要**

CPU ロック状態の参照

C 言語形式

```
BOOL    sns_loc ( void );
```

パラメータ

なし

機能

本サービス・コールは、CPU ロック状態か否かを調べます。本サービス・コールは、戻り値として、CPU ロック状態の場合は TRUE、CPU ロック解除状態の場合は FALSE を返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了 (CPU ロック状態)
FALSE	0	正常終了 (非 CPU ロック状態)
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

sns_dsp

概要

ディスパッチ禁止状態の参照

C 言語形式

```
BOOL      sns_dsp ( void );
```

パラメータ

なし

機能

本サービス・コールは、ディスパッチ禁止状態か否かを調べます。本サービス・コールは、戻り値として、ディスパッチ禁止状態の場合は TRUE、ディスパッチ許可状態の場合は FALSE を返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（ディスパッチ禁止状態）
FALSE	0	正常終了（ディスパッチ許可状態）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

sns_dpn

概要

ディスパッチ保留状態の参照

C 言語形式

```
BOOL    sns_dpn ( void );
```

パラメータ

なし

機能

本サービス・コールは、ディスパッチ保留状態か否かを調べます。本サービス・コールは、戻り値として、ディスパッチ保留状態の場合は TRUE、ディスパッチ保留状態でない場合は FALSE を返します。

なお、以下のいずれかの条件を満たすときをディスパッチ保留状態と呼びます。

- ディスパッチ禁止状態
- CPU ロック状態
- ハンドラなど、PSW.IPL>0 の状態

戻り値

マクロ	数値	意味
TRUE	1	正常終了（ディスパッチ保留状態）
FALSE	0	正常終了（非ディスパッチ保留状態）
E_CTX	-25	コンテキスト・エラー - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した

vsys_dwn
ivsys_dwn

概要

システム・ダウン

C 言語形式

```
void vsys_dwn ( W type, VW inf1, VW inf2, VW inf3 );
void ivsys_dwn ( W type, VW inf1, VW inf2, VW inf3 );
```

パラメータ

I/O	パラメータ	説明
I	W <i>type</i> ;	エラー種別
I	VW <i>inf1</i> ;	システム・ダウン情報 1
I	VW <i>inf2</i> ;	システム・ダウン情報 2
I	VW <i>inf3</i> ;	システム・ダウン情報 3

機能

システム・ダウン・ルーチン (`_RI_sys_dwn_()`) に制御を渡します。

type には、エラー種別として発生したエラーに対応した値 (1 ~ 0x7FFFFFFF) を設定してください。なお、0 以下の値は RI600PX によって予約されています。

本サービス・コールからリターンすることはありません。

パラメータ仕様の詳細は、「[15.2.2 システム・ダウン・ルーチンのパラメータ](#)」を参照してください。

本サービス・コールは μ ITRON4.0 仕様外の機能です。

備考 RI600PX 内部で異常を検出した場合にも、システム・ダウン・ルーチンが呼び出されます。

戻り値

なし

vsta_knl
ivsta_knl

概要

RI600PX の起動

C 言語形式

```
void    vsta_knl ( void );  
void    ivsta_knl ( void );
```

パラメータ

なし

機能

RI600PX を起動します。

本サービス・コールからリターンすることはありません。

本サービス・コールを発行するときは、必ず以下を満たす状態で無ければなりません。

- 全割り込みが受け付けられないこと（例えば、PSW.I == 0）
- CPU の動作モードがスーパーバイザ・モード（PSW.PM == 0）であること

本サービス・コールの処理概要を、以下に示します。

- 1) ISP レジスタを、SI セクションの最終アドレス +1 に初期化
- 2) INTB レジスタを、cfg600px によって生成された可変ベクタ・テーブル（INTERRUPT_VECTOR セクション）先頭アドレスに初期化
- 3) システム時刻を 0 に初期化
- 4) システム・コンフィギュレーション・ファイルで定義された各種オブジェクトの生成
ここで異常を検出した場合は、[システム・ダウン](#)となります。
- 5) MPU（Memory Protection Unit）を初期化
ここで異常を検出した場合は、[システム・ダウン](#)となります。
- 6) 基本クロック用タイマの初期化（[基本クロック用タイマ初期化ルーチン](#)（`_RI_init_cmt_knl()`）の呼び出し）
- 7) スケジューラに制御を移す

本サービス・コールは μ ITRON4.0 仕様外の機能です。

戻り値

なし

19.2.14 割り込み管理機能

以下に、RI600PX が割り込み管理機能として提供しているサービス・コールの一覧を示します。

表 19 - 18 割り込み管理機能

サービス・コール名	機能概要	発行有効範囲
chg_ims	割り込みマスク・パターンの変更	タスク
ichg_ims	割り込みマスク・パターンの変更	非タスク
get_ims	割り込みマスク・パターンの参照	タスク
iget_ims	割り込みマスク・パターンの参照	非タスク

chg_ims ichg_ims

概要

割り込みマスクの変更

C 言語形式

```
ER      chg_ims ( IMASK imask );
ER      ichg_ims ( IMASK imask );
```

パラメータ

I/O	パラメータ	説明
I	IMASK <i>imask</i> ;	割り込みマスク値

機能

PSW レジスタの IPL ビットを、*imask* で指定された値に変更します。*imask* には、0 ~ 15 の値を指定できます。
chg_ims サービス・コールでは、*imask* に 0 以外を指定するとシステムはディスパッチ禁止状態に移行 ([dis_dsp](#) と等価) し、*imask* に 0 を指定するとシステムはディスパッチ許可状態に移行 ([ena_dsp](#) と等価) します。

一方、ichg_ims では、ディスパッチ禁止 / 許可状態の遷移はありません。

PSW レジスタの IPL ビットを [カーネル割り込みマスクレベル \(system_IPL\)](#) を超える値に変更している間は、一部のサービス・コールの発行が禁止されます。

発行可能なサービス・コール	機能概要
chg_ims , ichg_ims	割り込みマスクの変更
get_ims , iget_ims	割り込みマスクの参照
vsys_dwn , ivsys_dwn	システム・ダウン
vsta_knl , ivsta_knl	RI600PX の起動

備考 1 非タスクでは、割り込みマスクを起動時より下げてはなりません。

備考 2 chg_ims の発行によって変更したディスパッチ禁止状態の解除は、本サービス・コールを発行したタスクが DORMANT 状態へと遷移する以前に行う必要があります。

備考 3 ディスパッチ禁止状態の間に“自タスクを状態遷移させる可能性のあるサービス・コール ([wai_sem](#), [wai_flg](#) など)”を発行した場合には、要求条件の即時成立 / 不成立を問わず、戻り値として E_CTX を返します。

備考 4 RI600PX では、一定周期で発生する基本クロック用タイマ割り込みを利用して [時間管理機能](#) を実現しています。このため、本サービス・コールの発行により、該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には、[時間管理機能](#) が正常に動作しなくなる場合があります。

備考 5 タスクが chg_ims によって PSW.IPL を 0 以外に変更している間は、[ena_dsp](#) を呼び出さないようにしてください。[ena_dsp](#) を呼び出すと、その時点でディスパッチ許可状態に遷移します。タスク・ディスパッチが発生すると、PSW はディスパッチ先のタスクの状態に更新されるので、意図せずに IPL 値が下がってしまうことがあります。

備考6 「12.8 マスカブル割り込みの受け付け禁止」も参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>imask</i> > 15
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - タスクから <i>ichg_ims</i> を発行した。 - 非タスクから <i>chg_ims</i> を発行した。
E_MACV	-26	メモリ・アクセス違反 (<i>chg_ims</i> のみ) - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。

get_ims iget_ims

概要

割り込みマスクの参照

C 言語形式

```
ER    get_ims ( IMASK *p_imask );
ER    iget_ims ( IMASK *p_imask );
```

パラメータ

I/O	パラメータ	説明
O	IMASK *p_imask;	割り込みマスク値を格納する領域へのポインタ

機能

PSW レジスタの IPL ビット値を、*p_imask* で指定された領域に格納します。

備考 1 本サービス・コールでは、コンテキスト・エラーは検出されません。

備考 2 コンパイラが提供する以下の組み込み関数を使用すると、本サービス・コールよりも高速に PSW レジスタの IPL ビット値を取得することができます。組み込み関数の詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX コーディング編」を参照してください。

- get_ipi() : PSW レジスタの IPL ビットの参照
- get_psw() : PSW レジスタの参照

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>p_imask</i> == NULL
E_MACV	-26	メモリ・アクセス違反 (get_ims のみ) - 呼び出しタスクからの <i>p_imask</i> が示す領域に対するオペランド・ライト・アクセス許可がない。

19.2.15 システム構成管理機能

以下に、RI600PX がシステム構成管理機能として提供しているサービス・コールの一覧を示します。

表 19 - 19 システム構成管理機能

サービス・コール名	機能概要	発行有効範囲
ref_ver	バージョン情報の参照	タスク
iref_ver	バージョン情報の参照	非タスク

ref_ver
iref_ver

概要

バージョン情報の参照

C 言語形式

```
ER    ref_ver ( T_RVER *pk_rver );
ER    iref_ver ( T_RVER *pk_rver );
```

パラメータ

I/O	パラメータ	説明
O	T_RVER *pk_rver;	バージョン情報を格納する領域へのポインタ

【バージョン情報 T_RVER の構造】

```
typedef struct t_rver {
    UH maker;           /* カーネルのメーカー・コード */
    UH prid;            /* カーネルの識別番号 */
    UH spver;           /* ITRON 仕様のバージョン番号 */
    UH prver;           /* カーネルのバージョン番号 */
    UH prno[4];         /* カーネル製品の管理情報 */
} T_RVER;
```

機能

RI600PX のバージョン情報を *pk_rver* で指定された領域に格納します。

- maker

maker は、このカーネルを作ったメーカーを表します。RI600PX では、ルネサスエレクトロニクスを意味する 0x011B が返ります。

なお、*maker* と同じ値がカーネル構成マクロ `TKERNEL_MAKER` に定義されています。

- prid

prid は、カーネルや VLSI の種類を区別する番号を表します。RI600PX では、0x0004 が返ります。

なお、*prid* と同じ値がカーネル構成マクロ `TKERNEL_PRID` に定義されています。

- spver

spver は、カーネルの準拠する仕様を表します。RI600PX では、0x5403 が返ります。

なお、*spver* と同じ値がカーネル構成マクロ `TKERNEL_SPVER` に定義されています。

- prver

prver は、カーネルのバージョン番号を表します。

例えば、カーネルのバージョンが V1.02.03 の場合は 0x0123 となります。

なお、*prver* と同じ値がカーネル構成マクロ `TKERNEL_PRVER` に定義されています。

- *prno*

prno は、製品管理情報や製品番号などを表します。RI600PX では、すべての *prno*[] に 0x0000 が返ります

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>pk_rver</i> == NULL
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。 備考 タスクから <i>iref_ver</i> を発行した場合、および非タスクから <i>ref_ver</i> を発行した場合、コンテキスト・エラーは検出されず、システムの正常な動作は保証されません。
E_MACV	-26	メモリ・アクセス違反 (<i>ref_ver</i> のみ) - 呼び出しタスクからの <i>pk_rver</i> が示す領域に対するオペランド・ライト・アクセス許可がない。

19.2.16 オブジェクト・リセット機能

以下に、RI600PX がオブジェクト・リセット機能として提供しているサービス・コールの一覧を示します。

表 19 - 20 オブジェクト・リセット機能

サービス・コール名	機能概要	発行有効範囲
vrst_dtq	データ・キューのリセット	タスク
vrst_mbx	メールボックスのリセット	タスク
vrst_mbf	メッセージ・バッファのリセット	タスク
vrst_mpf	固定長メモリ・プールのリセット	タスク
vrst_mpl	可変長メモリ・プールのリセット	タスク

vrst_dtq**概要**

データ・キューのリセット

C 言語形式

```
ER      vrst_dtq ( ID dtqid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>dtqid</i> ;	データ・キューの ID

機能

dtqid で指定されたデータ・キューをリセットします。

データ・キューに蓄えられていたデータは破棄されます。また、データ送信を待っていたタスクの待ち状態は解除され、そのタスクには戻り値として EV_RST が返されます。

備考 1 本サービス・コールでは、データ受信を待っていたタスクの待ち状態は解除されません。

備考 2 本サービス・コールは μ ITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $dtqid \leq 0$ - $dtqid > VTMAX_DTQ$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>dtqid</i> のデータ・キューが存在しない。

vrst_mbx

概要

メールボックスのリセット

C 言語形式

```
ER      vrst_mbx ( ID mbxid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbxid</i> ;	メールボックスの ID

機能

mbxid で指定されたメールボックスをリセットします。
メールボックスに蓄えられていたメッセージは、RI600PX の管理から外れます。

備考 1 本サービス・コールでは、メッセージ受信を待っていたタスクの待ち状態は解除されません。

備考 2 本サービス・コールは μ ITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - <i>mbxid</i> \leq 0 - <i>mbxid</i> > VTMAX_MBX
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>mbxid</i> のメールボックスが存在しない。

vrst_mbf

概要

メッセージ・バッファのリセット

C 言語形式

```
ER      vrst_mbf ( ID mbfid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mbfid</i> ;	メッセージ・バッファの ID

機能

mbfid で指定されたメッセージ・バッファをリセットします。

メッセージ・バッファに蓄えられていたメッセージは破棄されます。また、メッセージ送信を待っていたタスクの待ち状態は解除され、そのタスクには戻り値として EV_RST が返されます。

備考 1 本サービス・コールでは、メッセージ受信を待っていたタスクの待ち状態は解除されません。

備考 2 本サービス・コールは μ ITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $mbfid \leq 0$ - $mbfid > VTMAX_MBF$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>mbfid</i> のデータ・キューが存在しない。

vrst_mpf

概要

固定長メモリ・プールのリセット

C 言語形式

```
ER      vrst_mpf ( ID mpfid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mpfid</i> ;	固定長メモリ・プールの ID

機能

mpfid で指定された固定長メモリ・プールをリセットします。

固定長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され、戻り値として EV_RST が返されます

備考 1 すでに獲得されていた固定長メモリ・ブロックはすべて、対象固定長メモリ・プールに返却されます。このため、本サービス・コール以降はそれらの固定長メモリ・ブロックにアクセスしてはなりません。

備考 2 本サービス・コールは μ ITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $mpfid \leq 0$ - $mpfid > VTMAX_MPF$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>mpfid</i> の固定長メモリ・プールが存在しない。

vrst_mpl

概要

可変長メモリ・プールのリセット

C 言語形式

```
ER      vrst_mpl ( ID mplid );
```

パラメータ

I/O	パラメータ	説明
I	ID <i>mplid</i> ;	可変長メモリ・プールの ID

機能

mplid で指定された可変長メモリ・プールをリセットします。

可変長メモリ・ブロックの獲得を待っていたタスクの待ち状態は解除され、戻り値として EV_RST が返されます。

備考 1 すでに獲得されていた可変長メモリ・ブロックはすべて、対象可変長メモリ・プールに返却されます。このため、本サービス・コール以降はそれらの可変長メモリ・ブロックにアクセスしてはなりません。

備考 2 本サービス・コールは μ ITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	不正 ID 番号 - $mplid \leq 0$ - $mplid > VTMAX_MPL$
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 - <i>mplid</i> の可変長メモリ・プールが存在しない。

19.2.17 メモリ・オブジェクト保護機能

以下に、RI600PX がメモリ・オブジェクト保護機能として提供しているサービス・コールの一覧を示します。

表 19 - 21 メモリ・オブジェクト保護機能

サービス・コール名	機能概要	発行有効範囲
ata_mem	メモリ・オブジェクトの登録	タスク
det_mem	メモリ・オブジェクトの登録解除	タスク
sac_mem	メモリ・オブジェクトのアクセス許可ベクタの変更	タスク
vprb_mem	メモリ領域に対するアクセス権のチェック	タスク
ref_mem	メモリ・オブジェクトの詳細状態の参照	タスク

ata_mem

概要

メモリ・オブジェクトの登録

C 言語形式

```
ER      ata_mem ( T_AMEM *pk_amem , ACVCT *p_acvct );
```

パラメータ

I/O	パラメータ	説明
I	T_AMEM *pk_amem;	メモリ・オブジェクト登録情報を格納した領域へのポインタ
I	ACVCT *p_acvct;	アクセス許可ベクタを格納した領域へのポインタ

【メモリ・オブジェクト登録情報 T_AMEM の構造】

```
typedef struct t_amem {
    ATR    mematr;          /* メモリ・オブジェクト属性 */
    VP     base;           /* メモリ・オブジェクトの先頭アドレス */
    SIZE   size;           /* メモリ・オブジェクトのサイズ (バイト数) */
} T_AMEM;
```

【アクセス許可ベクタ ACVCT の構造】

```
typedef struct acvct {
    ACPTN  acptn1;         /* オペランド・リード・アクセス許可パターン */
    ACPTN  acptn2;         /* オペランド・ライト・アクセス許可パターン */
    ACPTN  acptn3;        /* 実行アクセス許可パターン */
} ACVCT;
```

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。

base で示されたアドレスから *size* バイトの領域を、*p_acvct* で指定されたアクセス許可ベクタを持つメモリ・オブジェクトとして登録します。

アクセス許可パターンは、ビット N-1 がドメイン ID#N に所属するタスクからのアクセス許可を示し、1 が許可、0 が禁止を示します。

指定するメモリ・オブジェクト領域は、以下を満たす必要があります。

- A) 先頭アドレス (*base*) は 16 バイト境界であること。そうでない場合は E_PAR エラーを返します。
- B) サイズ (*size*) は 16 の整数倍であること。そうでない場合は E_PAR エラーを返します。
- C) ユーザ・スタックおよびメモリ・オブジェクトと重なっていないこと。そうでない場合、エラーは検出されず、システムの正常な動作は保証されません。

備考1 アクセス許可ベクタを設定するマクロとして、以下が用意されています。

- TACT_SRW

すべてのアクセスがすべてのドメインに許可されることを示すアクセス許可ベクタを返します。初期代入文の右辺にのみ記述可能。

- TACT_SRO

オペランド・ライト・アクセスはすべてのドメインに禁止され、オペランド・リードアクセスおよび実行アクセスはすべてのドメインに許可されていることを示すアクセス許可ベクタを返します。初期代入文の右辺にのみ記述可能。

- ACVCT TACT_PRW(ID domid)

すべてのアクセス（オペランド・リード、オペランド・ライト、実行）が、*domid* で指定されたドメインにのみ許可されていることを示すアクセス許可ベクタを返します。初期代入文の右辺にのみ記述可能。

- ACVCT TACT_PRO(ID domid)

オペランド・リード・アクセスおよび実行アクセスが *domid* で指定されたドメインにのみ許可され、オペランド・ライト・アクセスがすべてのドメインに禁止されていることを示すアクセス許可ベクタを返します。初期代入文の右辺にのみ記述可能。

- ACVCT TACT_SRPW(ID domid)

オペランド・リード・アクセスおよび実行アクセスがすべてのドメインにのみ許可され、オペランド・ライト・アクセスが *domid* で指定されたドメインにのみ許可されていることを示すアクセス許可ベクタを返します。初期代入文の右辺にのみ記述可能。

備考2 アクセス許可パターンを設定するマクロとして、以下が用意されています。

- TACP_SHARED

すべてのドメインがアクセスできることを示すアクセス許可パターンを返します。

- ACPTN TACP(ID domid)

domid で指定されたドメインからのみアクセス可能であることを示すアクセス許可パターンを返します。

備考3 メモリ・オブジェクト属性 (*mematr*) は、単に無視されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>pk_amem</i> == NULL - <i>base</i> が 16 バイト境界でない。 - <i>size</i> が 16 の整数倍でない。 - <i>p_acvct</i> == NULL - <i>acptn1</i> == <i>acptn2</i> == <i>acptn3</i> == 0 - <i>acptn1</i>, <i>acptn2</i>, または <i>acptn3</i> の最大ドメイン ID (<i>VTMAX_DOMAIN</i>) より大きなドメイン ID に対応するビットが 1。 - <i>size</i> == 0 - <i>base</i> + <i>size</i> > 0x100000000
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。

マクロ	数値	意味
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_amem</i> が示す領域に対するオペランド・リード・アクセス許可がない。 - 呼び出しタスクからの <i>p_acvct</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。 - あるドメインにアクセス許可されるメモリ・オブジェクトが7個を超える。
E_OBJ	-41	オブジェクト状態不正 - 既に <i>base</i> が同じメモリ・オブジェクトが登録されている。

det_mem

概要

メモリ・オブジェクトの登録解除

C 言語形式

```
ER    det_mem ( VP base );
```

パラメータ

I/O	パラメータ	説明
I	VP <i>base</i> ;	メモリ・オブジェクトの先頭アドレス

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
先頭アドレスが *base* のメモリ・オブジェクトの登録を解除します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - 非タスクから本サービス・コールを発行した。 - CPU ロック状態から本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_OACV	-27	オブジェクト・アクセス違反 <ul style="list-style-type: none"> - 呼出しタスクは、信頼されたドメインに所属していない。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - 先頭アドレスが <i>base</i> のメモリ・オブジェクトが存在しない。

sac_mem

概要

メモリ・オブジェクトのアクセス許可ベクタの変更

C 言語形式

```
ER      sac_mem ( VP base, ACVCT *p_acvct );
```

パラメータ

I/O	パラメータ	説明
I	VP <i>base</i> ;	メモリ・オブジェクトの先頭アドレス
I	ACVCT <i>*p_acvct</i> ;	アクセス許可ベクタを格納した領域へのポインタ

【アクセス許可ベクタ ACVCT の構造】

```
typedef struct acvct {
    ACPTN acptn1;      /* オペランド・リード・アクセス許可パターン */
    ACPTN acptn2;      /* オペランド・ライト・アクセス許可パターン */
    ACPTN acptn3;      /* 実行アクセス許可パターン */
} ACVCT;
```

機能

本サービス・コールは、[信頼されたドメイン](#)に所属するタスクからのみ呼び出せます。
先頭アドレスが *base* のメモリ・オブジェクトのアクセス許可ベクタを、*p_acvct* で指定された内容に変更します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <i>p_acvct</i> == NULL - <i>acptn1</i> == <i>acptn2</i> == <i>acptn3</i> == 0 - <i>acptn1</i>, <i>acptn2</i>, または <i>acptn3</i> の最大ドメイン ID (VTMAX_DOMAIN) より大きなドメイン ID に対応するビットが 1。

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>p_acvct</i> が示す領域に対するオペランド・リード・アクセス許可がない。
E_OACV	-27	オブジェクト・アクセス違反 - 呼出しタスクは、信頼されたドメインに所属していない。 - あるドメインにアクセス許可されるメモリ・オブジェクトが7個を超える。
E_NOEXS	-42	オブジェクト未生成 - 先頭アドレスが <i>base</i> のメモリ・オブジェクトが存在しない。

vprb_mem

概要

メモリ領域に対するアクセス権のチェック

C 言語形式

```
ER_BOOL vprb_mem ( VP base, SIZE size, ID tskid, MODE pmmode );
```

パラメータ

I/O	パラメータ	説明
I	VP <i>base</i> ;	先頭アドレス
I	SIZE <i>size</i> ;	サイズ (バイト数)
I	ID <i>tskid</i> ;	タスクの ID TSK_SELF : 自タスク 数値 : タスクの ID
I	MODE <i>pmmode</i> ;	アクセス・モード

機能

base で指定されたアドレスから *size* バイトの領域について、*tskid* で指定されたタスクが *pmmode* で指定されたアクセスが許可されているかをチェックします。許可されている場合は TRUE、許可されていない場合は FALSE を返します。
pmmode には、以下を指定できます。

```
pmmode := ( TPM_READ | TPM_WRITE | TPM_EXEC )
```

- **TPM_READ** (= 0x0001)
オペランド・リード・アクセス権をチェックします。
- **TPM_WRITE** (= 0x0002)
オペランド・ライト・アクセス権をチェックします。
- **TPM_EXEC** (= 0x0004)
実行アクセス権をチェックします。

備考 本サービス・コールは μ ITRON4.0 仕様外の機能です。

戻り値

マクロ	数値	意味
TRUE	1	正常終了 (アクセス権あり)
FALSE	0	正常終了 (アクセス権なし)

マクロ	数値	意味
E_PAR	-17	パラメータ・エラー <ul style="list-style-type: none"> - <code>size == 0</code> - <code>pmmode == 0</code>, <code>pmmode</code> の bit0 ~ 2 以外のビットが 1
E_ID	-18	不正 ID 番号 <ul style="list-style-type: none"> - <code>tskid < 0</code> - <code>tskid > VTMAX_TSK</code>
E_CTX	-25	コンテキスト・エラー <ul style="list-style-type: none"> - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 <ul style="list-style-type: none"> - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。
E_NOEXS	-42	オブジェクト未生成 <ul style="list-style-type: none"> - <code>tskid</code> のタスクが存在しない。

ref_mem

概要

メモリ・オブジェクトの詳細情報の参照

C 言語形式

```
ER    ref_mem ( VP base, T_RMEM *pk_rmem );
```

パラメータ

I/O	パラメータ	説明
I	VP <i>base</i> ;	メモリ・オブジェクトの先頭アドレス
O	T_RMEM * <i>pk_rmem</i> ;	メモリ・オブジェクト詳細情報を格納する領域へのポインタ

【アクセス許可ベクタ ACVCT の構造】

```
typedef struct acvct {
    ACPTN acptn1;    /* オペランド・リード・アクセス許可パターン */
    ACPTN acptn2;    /* オペランド・ライト・アクセス許可パターン */
    ACPTN acptn3;    /* 実行アクセス許可パターン */
} ACVCT;
```

【メモリ・オブジェクト詳細情報 T_RMEM の構造】

```
typedef struct t_rmem {
    ACVCT acvct;    /* アクセス許可ベクタ */
} T_RMEM;
```

機能

先頭アドレスが *base* のメモリ・オブジェクトの状態を *pk_rmem* で指定された領域に格納します。

アクセス許可パターンは、ビット N-1 がドメイン ID#N からのアクセス許可を示し、1 が許可、0 が禁止を示します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータ・エラー - <i>pk_rmem</i> == NULL

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した。 - 非タスクから本サービス・コールを発行した。 - 「PSW.IPL > カーネル割り込みマスクレベル」の状態から本サービス・コールを発行した。
E_MACV	-26	メモリ・アクセス違反 - スタック・ポインタが呼出しタスクのユーザ・スタック領域の範囲外を指している。 - 呼び出しタスクからの <i>pk_rmem</i> が示す領域に対するオペランド・ライト・アクセス許可がない。
E_NOEXS	-42	オブジェクト未生成 - 先頭アドレスが <i>base</i> のメモリ・オブジェクトが存在しない。

第20章 システム・コンフィギュレーション・ファイル

本章では、RI600PX に提供するデータを保持した情報ファイルを生成する際に必要となるシステム・コンフィギュレーション・ファイルの記述方法について解説しています。

20.1 概 要

以下に、システム・コンフィギュレーション・ファイルの表記方法を示します。

- コメント

“//”（連続する2個のスラッシュ）から行末までの部分がコメントとして扱われます。

- 数値

数値は、以下の形式で入力できます。ただし、0xFFFFFFFF を超える数値を指定してはなりません。

16進数： 数値の先頭に“0x”か“0X”を付加します。または、数値の最後に'h'か'H'を付加します。後者の場合かつ先頭が英文字(A～F)で始まる場合は先頭に必ず'0'を付加してください。なおここで使用する数値表現で英文字(A～F)は大文字・小文字を識別しません。

10進数： “23”のように整数のみで表します。ただし'0'で始めることはできません。

8進数： 数値の先頭に'0'を付加するか数値の最後に'O'もしくは'o'を付加します。

2進数： 数値の最後に'B'または'b'を付加します。ただし'0'で始めることはできません。

- 演算子

数値に対して、以下の演算子を使用できます。

表 20 - 1 演算子

演算子	優先度	演算方向
()	高	左から右
- (単項マイナス)		右から左
*, /, %		左から右
+, - (二項マイナス)	低	左から右

- シンボル

シンボルは数字、英大文字、英小文字、'_' (アンダースコア) より構成され、数字以外の文字で始まる文字列で表されます。

- 関数名

関数名は、数字、英大文字、英小文字、'_' (アンダースコア)、'\$' (ドル記号) より構成される数字以外の文字で始まり、“()”で終わる文字列で表されます。

アセンブリ言語で記述したモジュールを指定する場合は、その先頭ラベルを'_'で始まるように命名し、'_'を除いたものを関数名として指定してください。

- 周波数

周波数は、数字と'.' (ピリオド) から構成され“MHz”で終わる文字列で表されます。小数点以下は6桁が有効です。なお周波数は10進数のみで記述可能です。

20.2 デフォルト・システム・コンフィギュレーション・ファイル

多くの定義項目では、ユーザが記述を省略した場合にデフォルト・システム・コンフィギュレーション・ファイルの内容が補われます。デフォルト・システム・コンフィギュレーション・ファイルは、環境変数 LIB600 で指定されるフォルダにある default.cfg です。なお、このファイルを編集してはなりません。

20.3 コンフィギュレーション情報（静的 API）

システム・コンフィギュレーション・ファイルに記述するデータ（コンフィギュレーション情報）は、以下の通りです。

- システム情報 (system)
- 基本クロック割り込み情報 (clock)
- 最大 ID (maxdefine)
- ドメイン定義 (domain)
- メモリ・オブジェクト定義 (memory_object[])
- タスク情報 (task[])
- セマフォ情報 (semaphore[])
- イベントフラグ情報 (flag[])
- データ・キュー情報 (dataqueue[])
- メールボックス情報 (mailbox[])
- ミューテックス情報 (mutex[])
- メッセージ・バッファ情報 (message_buffer[])
- 固定長メモリ・プール情報 (memorypool[])
- 可変長メモリ・プール情報 (variable_memorypool[])
- 周期ハンドラ情報 (cyclic_hand[])
- アラーム・ハンドラ情報 (alarm_hand[])
- 可変ベクタ情報 (interrupt_vector[])
- 固定ベクタ／例外ベクタ情報 (interrupt_fvector[])

20.4 システム情報 (system)

ここでは、RI600PX システム全般に関する情報を定義します。

なお、システム情報として定義可能な数は、1 個に限られます。また、システム情報の定義は省略できません。

形式

<> 内は、ユーザが記述する部分を示します。

```
system {
    stack_size = <1. システム・スタック・サイズ (stack_size) >;
    priority   = <2. タスク優先度の最大値 (priority) >;
    system_IPL = <3. カーネル割り込みマスクレベル (system_IPL) >;
    message_pri = <4. メッセージ優先度の最大値 (message_pri) >;
    tic_deno   = <5. 基本クロック割り込み周期の分母 (tic_deno) >;
    tic_num   = <6. 基本クロック割り込み周期の分子 (tic_num) >;
    context   = <7. タスク・コンテキスト・レジスタ (context) >;
};
```

1) システム・スタック・サイズ (stack_size)

- 説明
サービス・コール処理および割り込み処理で使用するスタックサイズの合計を定義します。
- 定義形式
数値
- 定義範囲
8 以上の 4 の倍数
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 0x800) を適用

2) タスク優先度の最大値 (priority)

- 説明
タスク優先度の最大値を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 32) を適用
- **TMAX_TPRI**
cfg600pxは、本設定値を定義したマクロ**TMAX_TPRI**を、システム情報ヘッダ・ファイルkernel_id.hに出力します。

3) カーネル割り込みマスクレベル (system_IPL)

- 説明
カーネルのクリティカルセクション実行時の割り込みマスクレベル (PSW レジスタの IPL 値) を定義します。これよりも高いレベルの割り込みは、カーネル管理外割り込みの扱いとなります。カーネル管理外割り込みとカーネル管理割り込みについては、「[12.1 割り込みの種類](#)」を参照してください。
- 定義形式
数値
- 定義範囲
1 ~ 15

- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は7）を適用
- [VTKNL_LVL](#)
cfg600pxは、本設定値を定義したマクロ[VTKNL_LVL](#)を、システム情報ヘッダ・ファイルkernel_id.hに出力します。

4) メッセージ優先度の最大値 (*message_pri*)

- 説明
メールボックス機能で使用するメッセージの優先度の最大値を定義します。なお、メールボックス機能を使用しない場合は、本項目は意味を持ちません。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は255）を適用
- [TMAX_MPRI](#)
cfg600pxは、本設定値を定義したマクロ[TMAX_MPRI](#)を、システム情報ヘッダ・ファイルkernel_id.hに出力します。

5) 基本クロック割り込み周期の分母 (*tic_deno*)

- 説明
基本クロック割り込み周期時間は、次の式で算出されます。なお、*tic_deno*と*tic_num*の少なくともひとつは1でなければなりません。
$$\text{基本クロック割り込み周期時間 (単位: ミリ秒)} = \text{tic_num} / \text{tic_deno}$$
- 定義形式
数値
- 定義範囲
1 ~ 100
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は1）を適用
- 備考
tic_num, *tic_deno*の設定に関わらず、サービス・コールで扱う時間の単位は常にミリ秒になります。*tic_num*, *tic_deno*によって、RI600PXが管理する時間の精度を規定することになります。
- [TIC_DENO](#)
cfg600pxは、本設定値を定義したマクロ[TIC_DENO](#)を、システム情報ヘッダ・ファイルkernel_id.hに出力します。

6) 基本クロック割り込み周期の分子 (*tic_num*)

- 説明
前項を参照してください。
- 定義形式
数値
- 定義範囲
1 ~ 65535
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は1）を適用
- [TIC_NUME](#)
cfg600pxは、本設定値を定義したマクロ[TIC_NUME](#)を、システム情報ヘッダ・ファイルkernel_id.hに出力します。

7) タスク・コンテキスト・レジスタ (context)

- 説明
タスクが使用するレジスタ・セットを定義します。ここでの設定は、全タスクに適用されます。
- 定義形式
シンボル
- 定義範囲
以下から選択してください。

表 20 - 2 system.context

設定値	CPU		FPU	DSP
	PSW, PC, R0 ~ R7, R14, R15	R8 ~ R13	FPSW	アキュムレータ ^a
"NO"	保証する	保証する	保証しない	保証しない
"FPSW"	保証する	保証する	保証する	保証しない
"ACC"	保証する	保証する	保証しない	保証する
"FPSW,ACC"	保証する	保証する	保証する	保証する
"MIN"	保証する	保証しない	保証しない	保証しない
"MIN,FPSW"	保証する	保証しない	保証する	保証しない
"MIN,ACC"	保証する	保証しない	保証しない	保証する
"MIN,FPSW,ACC"	保証する	保証しない	保証する	保証する

- a. コンパイラオプション“-isa=rxv2”指定時は ACC0 レジスタと ACC1 レジスタ, その他の場合は ACC0 レジスタ (RXv2 アーキテクチャの場合) または ACC レジスタ (RXv1 アーキテクチャの場合) です。

備考 コンパイラ・オプション“-isa”は、コンパイラ CC-RX V2.01 以降でサポートされています。

- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は NO）を適用
- 備考
必ず「[20.5 system.context の注意事項](#)」も参照してください。

20.5 system.context の注意事項

ここでは、system.context に関する注意事項を説明します。

20.5.1 FPU, DSP に関する注意事項

system.context に設定すべき値は、FPU, DSP をどのように扱うかによって異なります。

以降では、以下の各ケースについて、system.context の推奨値を示します。推奨以外の設定にした場合は、推奨設定に比べてわずかに RI600PX の性能が悪化します。

- 1) FPU および DSP (アキュムレータ) を搭載した MCU を使用する場合
該当 MCU : RX600 シリーズなど
- 2) FPU を搭載せず、かつ DSP (アキュムレータ) を搭載する MCU を使用する場合
該当 MCU : RX200 シリーズなど
- 3) FPU を搭載し、かつ DSP (アキュムレータ) を搭載しない MCU を使用する場合
該当 MCU : 本マニュアル作成時点では、これに該当する MCU は存在しません。
- 4) FPU も DSP (アキュムレータ) も搭載しない MCU を使用する場合
該当 MCU : 本マニュアル作成時点では、これに該当する MCU は存在しません。

備考 コンパイラが浮動小数点演算命令を出力するのは、-cpu=rx600 かつ -fpu オプション指定時のみです。
アセンブラで“chkfpu”オプションを指定すると、浮動小数点演算命令の記述を Warning として検出します。
また、コンパイラが DSP 機能命令を出力することはありません。
アセンブラで“chkdsp”オプションを指定すると、DSP 機能命令の記述を Warning として検出します。

- 1) FPU および DSP (アキュムレータ) を搭載した MCU を使用する場合

表 20 - 3 FPU および DSP (アキュムレータ) を搭載した MCU を使用する場合

タスクの命令使用状況		system.context の推奨
浮動小数点演算命令	DSP 機能命令	
あり	あり	“FPSW” と “ACC” を含む設定が必須
	なし	“FPSW” を含む設定が必須で、かつ “ACC” を含まない設定を推奨
なし	あり	“ACC” を含む設定が必須で、かつ “FPSW” を含まない設定を推奨
	なし	“FPSW”, “ACC” を含まない設定を推奨

20.5.2 コンパイラの“fint_register”, “base”, および“pid” オプションとの関係

system.context に、“MIN” を含む指定を行うことで、RI600PX は R8 ~ R13 をタスク・コンテキストとして保存しないようになり、高速化が図れるようになっています。

system.context にこれらの設定をしても良いケースは、コンパイラの“fint_register”, “base”, および“pid” オプションで R8 ~ R13 をすべて使用する指定をした場合のみです。

その他の場合で system.context に前述の設定をした場合、RI600PX は正常に動作しません。

【 良い例 】

例 1 : -fint_register=4 -base=rom=R8 -base=ram=R9

例 2 : -fint_register=3 -base=rom=R8 -base=ram=R9 -base=0x80000=R10

【 悪い例 】

例 3 : “-fint_register”, “-base”, “-pid” の指定なし

例 4 : -fint_register=4

例 5 : -base=rom=R8 -base=ram=R9

例 6 : -fint_register=3 -base=rom=R8 -base=ram=R9

20.6 基本クロック割り込み情報 (clock)

ここでは、基本クロック割り込みに関する情報を定義します。cfg600px は、本定義に基づいて、基本クロック用タイマ初期化関数 (`_RI_init_cmt()`) が記述された `ri_cmt.h` を出力します。

なお、基本クロック割り込み情報として定義可能な数は、1個に限られます。

形式

<>内は、ユーザが記述する部分を示します。

```
clock {
    timer      = <1. 基本クロック用タイマ・チャンネルの選択 (timer) >;
    template   = <2. テンプレート・ファイル (template) >;
    timer_clock = <3. CMT 入力周波数 (timer_clock) >;
    IPL        = <4. 基本クロック割り込み優先レベル (IPL) >;
};
```

1) 基本クロック用タイマ・チャンネルの選択 (timer)

- 説明
基本クロックに使用するタイマのチャンネルを定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれかから選択してください。

表 20 - 7 clock.timer

設定値	説明
CMT0	可変ベクタ 28 に割り当てられた CMT チャンネル 0 を使用する。
CMT1	可変ベクタ 29 に割り当てられた CMT チャンネル 1 を使用する。
CMT2	可変ベクタ 30 に割り当てられた CMT チャンネル 2 を使用する。
CMT3	可変ベクタ 31 に割り当てられた CMT チャンネル 3 を使用する。
OTHER	上記以外のタイマを使用する。この場合、ユーザがタイマ初期化ルーチンを作成する必要があります。
NOTIMER	基本クロック割り込みを使用しない。

備考 1 CMT (Compare Match Timer) は、RX MCU に標準的に搭載されているタイマです。

備考 2 使用する RX MCU が CMT チャンネル 2, 3 をサポートしていない場合は、"CMT2" および "CMT3" を選択しないでください。また、CMT チャンネル 2, 3 に割り当てられた可変ベクタが表 20 - 7 と異なる RX MCU を使用する場合も、"CMT2" および "CMT3" を選択しないでください。

例えば、RX111 は CMT チャンネル 2, 3 をサポートしていません。また、RX64M では、CMT チャンネル 2, 3 に割り当てられた可変ベクタは、30, 31 ではありません。

- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は "CMT0") を適用

2) テンプレート・ファイル (*template*)

- 説明

CMT (コンペアマッチタイマ) の初期化関数が定義されたテンプレート・ファイルを指定します。

timer に "NOTIMER" または "OTHER" を指定した場合、本指定は単に無視されます。

テンプレート・ファイルは、RI600PX によって提供されます。テンプレートファイルは、今後のバージョンで追加される場合があります。各テンプレート・ファイルがサポートする MCU は、リリースノートを参照してください。

テンプレートによっては、CMT1, CMT2, CMT3 のいずれかがサポートされない場合があります。*timer* にそのテンプレートで未対応の CMT チャンネルを指定した場合、*cfg600px* はエラーを報告しませんが、*cfg600px* が出力する *ri_cmt.h* をインクルードするファイルのコンパイル時にエラーになります。

- 定義形式

シンボル

- 定義範囲

—

- 省略時の扱い

デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は "rx630.tpl") を適用

3) CMT 入力周波数 (*timer_clock*)

- 説明

CMT に供給されるクロックの周波数を定義します。PCLK (周辺モジュールクロック) の周波数を指定してください。

- 定義形式

周波数

- 定義範囲

—

- 省略時の扱い

デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 25MHz) を適用

4) 基本クロック割り込み優先レベル (*IPL*)

- 説明

基本クロック用タイマの割り込み優先レベルを定義します。

- 定義形式

数値

- 定義範囲

1 ~ システム情報 (*system*) のカーネル割り込みマスクレベル (*system_IPL*)

- 省略時の扱い

デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 4) を適用

- *VTIM_LVL*

cfg600px は、本設定値を定義したマクロ *VTIM_LVL* を、システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

20.7 最大 ID (maxdefine)

ここでは、各オブジェクトの最大 ID を定義します。また、オブジェクトを動的に生成するサービス・コールを使用するには、対応するオブジェクトについて本定義を行う必要があります。

各オブジェクトの最大 ID を定義したマクロが、システム情報ヘッダ・ファイル kernel_id.h に出力されます（「18.2 定数マクロ」参照）

形式

<>内は、ユーザが記述する部分を示します。

```
maxdefine {
    max_task      = <1. 最大タスク ID (max_task) >;
    max_sem       = <2. 最大セマフォ ID (max_sem) >;
    max_flag     = <3. 最大イベントフラグ ID (max_flag) >;
    max_dtq      = <4. 最大データ・キュー ID (max_dtq) >;
    max_mbx      = <5. 最大メールボックス ID (max_mbx) >;
    max_mtx      = <6. 最大ミューテックス ID (max_mtx) >;
    max_mbf      = <7. 最大メッセージ・バッファ ID (max_mbf) >;
    max_mpf      = <8. 最大固定長メモリ・プール ID (max_mpf) >;
    max_mpl      = <9. 最大可変長メモリ・プール ID (max_mpl) >;
    max_cyh      = <10. 最大周期ハンドラ ID (max_cyh) >;
    max_alh      = <11. 最大アラーム・ハンドラ ID (max_alh) >;
    max_domain   = <12. 最大ドメイン ID (max_domain) >;
};
```

1) 最大タスク ID (max_task)

- 説明

本定義により、cre_tsk, acre_tsk, del_tsk, exd_tsk, def_tex を使用できます。
使用できるタスク ID の範囲は以下の通りです。

- 最小値：1
- 最大値：max_task 設定値, task[] で指定された ID 番号, task[] 定義数, の中の最大値

- 定義形式 数値

- 定義範囲 1 ~ 255

- 省略時の扱い

cre_tsk, acre_tsk, del_tsk, def_tex は E_NOSPT を返します。また、exd_tsk はシステム・ダウンとなります。
使用できるタスク ID の範囲は以下の通りです。

- 最小値：1
- 最大値：task[] で指定された ID 番号, task[] 定義数, の中の最大値

- VTMAX_TSK

cfg600px は、最大値を定義したマクロ VTMAX_TSK を、システム情報ヘッダ・ファイル kernel_id.h に出力します。

2) 最大セマフォ ID (max_sem)

- 説明

本定義により、cre_sem, acre_sem, del_sem を使用できます。
使用できるセマフォ ID の範囲は以下の通りです。

- 最小値：1
- 最大値：max_sem 設定値, semaphore[] で指定された ID 番号, semaphore[] 定義数, の中の最大値

- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cre_sem, acre_sem, del_sem は E_NOSPT を返します。
使用できるセマフォ ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : semaphore[] で指定された ID 番号, semaphore[] 定義数, の中の最大値
- [VTMAX_SEM](#)
cfg600px は, 最大値を定義したマクロ [VTMAX_SEM](#) を, システム情報ヘッダ・ファイル kernel_id.h に出力します。

3) 最大イベントフラグ ID (*max_flag*)

- 説明
本定義により, cre_flg, acre_flg, del_flg を使用できます。
使用できるイベントフラグ ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : *max_flag* 設定値, flag[] で指定された ID 番号, flag[] 定義数, の中の最大値
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cre_flg, acre_flg, del_flg は E_NOSPT を返します。
使用できるイベントフラグ ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : flag[] で指定された ID 番号, flag[] 定義数, の中の最大値
- [VTMAX_FLG](#)
cfg600px は, 最大値を定義したマクロ [VTMAX_FLG](#) を, システム情報ヘッダ・ファイル kernel_id.h に出力します。

4) 最大データ・キュー ID (*max_dtq*)

- 説明
本定義により, cre_dtq, acre_dtq, del_dtq を使用できます。
使用できるデータ・キュー ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : *max_dtq* 設定値, dataqueue[] で指定された ID 番号, dataqueue[] 定義数, の中の最大値
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cre_dtq, acre_dtq, del_dtq は E_NOSPT を返します。
使用できるデータ・キュー ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : dataqueue[] で指定された ID 番号, dataqueue[] 定義数, の中の最大値
- [VTMAX_DTQ](#)
cfg600px は, 最大値を定義したマクロ [VTMAX_DTQ](#) を, システム情報ヘッダ・ファイル kernel_id.h に出力します。

5) 最大メールボックス ID (*max_mbx*)

- 説明

本定義により、*cre_mbx*、*acre_mbx*、*del_mbx* を使用できます。
使用できるメールボックス ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *max_mbx* 設定値、*mailbox[]* で指定された ID 番号、*mailbox[]* 定義数、の中の最大値

- 定義形式

数値

- 定義範囲

1 ~ 255

- 省略時の扱い

cre_mbx、*acre_mbx*、*del_mbx* は E_NOSPT を返します。
使用できるメールボックス ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *mailbox[]* で指定された ID 番号、*mailbox[]* 定義数、の中の最大値

- **VTMAX_MBX**

cfg600px は、最大値を定義したマクロ **VTMAX_MBX** を、システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

6) 最大ミューテックス ID (*max_mtx*)

- 説明

本定義により、*cre_mtx*、*acre_mtx*、*del_mtx* を使用できます。
使用できるミューテックス ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *max_mtx* 設定値、*mutex[]* で指定された ID 番号、*mutex[]* 定義数、の中の最大値

- 定義形式

数値

- 定義範囲

1 ~ 255

- 省略時の扱い

cre_mtx、*acre_mtx*、*del_mtx* は E_NOSPT を返します。
使用できるミューテックス ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *mutex[]* で指定された ID 番号、*mutex[]* 定義数、の中の最大値

- **VTMAX_MTX**

cfg600px は、最大値を定義したマクロ **VTMAX_MTX** を、システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

7) 最大メッセージ・バッファ ID (*max_mbf*)

- 説明

本定義により、*cre_mbf*, *acre_mbf*, *del_mbf* を使用できます。
使用できるメッセージ・バッファ ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *max_mbf* 設定値, *message_buffer[]* で指定された ID 番号, *message_buffer[]* 定義数, の中の最大値

- 定義形式

数値

- 定義範囲

1 ~ 255

- 省略時の扱い

cre_mbf, *acre_mbf*, *del_mbf* は E_NOSPT を返します。
使用できるメッセージ・バッファ ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *message_buffer[]* で指定された ID 番号, *message_buffer[]* 定義数, の中の最大値

- VTMAX_MBF

cfg600px は, 最大値を定義したマクロ *VTMAX_MBF* を, システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

8) 最大固定長メモリ・プール ID (*max_mpf*)

- 説明

本定義により、*cre_mpf*, *acre_mpf*, *del_mpf* を使用できます。
使用できる固定長メモリ・プール ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *max_mpf* 設定値, *memorypool[]* で指定された ID 番号, *memorypool[]* 定義数, の中の最大値

- 定義形式

数値

- 定義範囲

1 ~ 255

- 省略時の扱い

cre_mpf, *acre_mpf*, *del_mpf* は E_NOSPT を返します。
使用できる固定長メモリ・プール ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *memorypool[]* で指定された ID 番号, *memorypool[]* 定義数, の中の最大値

- VTMAX_MPF

cfg600px は, 最大値を定義したマクロ *VTMAX_MPF* を, システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

9) 最大可変長メモリ・プール ID (*max_mpl*)

- 説明

本定義により、*cre_mpl*, *acre_mpl*, *del_mpl* を使用できます。
使用できる可変長メモリ・プール ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *max_mpl* 設定値, *variable_memorypool[]* で指定された ID 番号, *variable_memorypool[]* 定義数, の中の最大値

- 定義形式
数値- 定義範囲
1 ~ 255

- 省略時の扱い

cre_mpl, *acre_mpl*, *del_mpl* は E_NOSPT を返します。
使用できる可変長メモリ・プール ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *variable_memorypool[]* で指定された ID 番号, *variable_memorypool[]* 定義数, の中の最大値

- [VTMAX_MPL](#)

cfg600px は, 最大値を定義したマクロ [VTMAX_MPL](#) を, システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

10) 最大周期ハンドラ ID (*max_cyh*)

- 説明

本定義により、*cre_cyc*, *acre_cyc*, *del_cyc* を使用できます。
使用できる周期ハンドラ ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *max_cyh* 設定値, *cyclic_hand[]* で指定された ID 番号, *cyclic_hand[]* 定義数, の中の最大値

- 定義形式
数値- 定義範囲
1 ~ 255

- 省略時の扱い

cre_cyc, *acre_cyc*, *del_cyc* は E_NOSPT を返します。
使用できる周期ハンドラ ID の範囲は以下の通りです。

- 最小値 : 1

- 最大値 : *cyclic_hand[]* で指定された ID 番号, *cyclic_hand[]* 定義数, の中の最大値

- [VTMAX_CYH](#)

cfg600px は, 最大値を定義したマクロ [VTMAX_CYH](#) を, システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

11) 最大アラーム・ハンドラ ID (*max_alh*)

- 説明
本定義により、*cre_alm*, *acre_alm*, *del_alm* を使用できます。
使用できるアラーム・ハンドラ ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : *max_alh* 設定値, *alarm_hand[]* で指定された ID 番号, *alarm_hand[]* 定義数, の中の最大値
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cre_alm, *acre_alm*, *del_alm* は E_NOSPT を返します。
使用できるアラーム・ハンドラ ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : *alarm_hand[]* で指定された ID 番号, *alarm_hand[]* 定義数, の中の最大値
- [VTMAX_ALH](#)
cfg600px は, 最大値を定義したマクロ [VTMAX_ALH](#) を, システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

12) 最大ドメイン ID (*max_domain*)

- 説明
使用できるドメイン ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : *max_domain* 設定値, *domain[]* で指定された ID 番号, の中の最大値
- 定義形式
数値
- 定義範囲
1 ~ 15
- 省略時の扱い
使用できるドメイン ID の範囲は以下の通りです。
 - 最小値 : 1
 - 最大値 : *domain[]* で指定された ID 番号の中の最大値
- [VTMAX_DOMAIN](#)
cfg600px は, 最大値を定義したマクロ [VTMAX_DOMAIN](#) を, システム情報ヘッダ・ファイル *kernel_id.h* に出力します。

20.8 ドメイン定義 (domain)

ここでは、ドメインを定義します。本定義を省略したドメインIDのドメインは、trust = NOの扱いとなります。

形式

<>内は、ユーザが記述する部分を示します。

```
domain[ <1. ID 番号> ] {  
    trust          = <2. 信頼されたドメイン (trust)>;  
};
```

1) ID 番号

- 説明
ドメイン ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 15
- 省略時の扱い
省略不可

2) 信頼されたドメイン (trust)

- 説明
信頼されたドメインかどうかを定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
YES : 信頼されたドメインとする。
NO : 信頼されたドメインとしない。
- 省略時の扱い
省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は“NO”）を適用

20.9 メモリ・オブジェクト定義 (memory_object[])

ここでは、各メモリ・オブジェクトを定義します。
必ず「3.11 メモリ・マップ設計」も参照してください。

形式

<>内は、ユーザが記述する部分を示します。

```
memory_object[ ] {
    start_address = <1. メモリ・オブジェクトの先頭アドレス (start_address) >;
    end_address   = <2. メモリ・オブジェクトの終端アドレス (end_address) >;
    acptn1       = <3. アクセス許可パターン (acptn1, acptn2, acptn3) >;
    acptn2       = <3. アクセス許可パターン (acptn1, acptn2, acptn3) >;
    acptn3       = <3. アクセス許可パターン (acptn1, acptn2, acptn3) >;
};
```

1) メモリ・オブジェクトの先頭アドレス (start_address)

- 説明
メモリ・オブジェクトの先頭アドレスを、数値またはセクション名で定義します。
セクション名を指定した場合、そのセクションはリンク時に16バイト境界アドレスに配置する必要があるため、リンク時に“aligned_section”リンク・オプションを指定してください。
数値で指定する場合は、16の倍数でなければなりません。
- 定義形式
シンボルまたは数値
- 定義範囲
数値の場合は、0 ~ 0xFFFFFFFF0で、かつ16の倍数
- 省略時の扱い
省略不可

2) メモリ・オブジェクトの終端アドレス (end_address)

- 説明
メモリ・オブジェクトの終端アドレスを、数値またはセクション名で定義します。
セクション名を指定した場合、そのセクションの終端アドレスを16の倍数+15に切り上げたアドレスをメモリ・オブジェクトの終端アドレスと扱います。リンク時にこのセクションの終端アドレスが16の倍数+15以外となる場合は、セクション終端アドレス+1 ~ 16の倍数+15に切り上げたアドレスまでの範囲に他のセクションを配置してはなりません。リンク時に、このセクションの後続セクションに“aligned_section”リンク・オプションを指定することで、この条件が満たされるようになります。
数値で指定する場合は、16の倍数+15でなければなりません。
- 定義形式
シンボルまたは数値
- 定義範囲
数値の場合は、0x0000000F ~ 0xFFFFFFFFで、かつの16の倍数+15
- 省略時の扱い
省略不可

3) アクセス許可パターン (*acptn1*, *acptn2*, *acptn3*)

- 説明

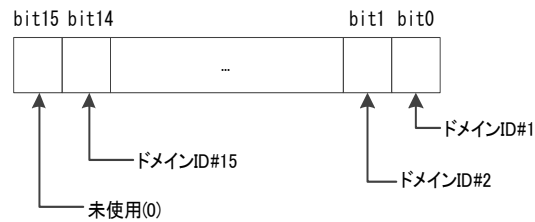
オペランド・リード・アクセス (*acptn1*) , オペランド・ライト・アクセス (*acptn2*) , 実行アクセス (*acptn3*) のアクセス許可パターンを, シンボルまたは数値で定義します。

シンボルとしては, TACP_SHARED (全ドメインにアクセスを許可) のみを指定できます。

ドメインごとに許可/禁止を個別に指定するには, 図 20-1 に従って数値で指定します。

最大ドメイン ID を超えるドメイン ID に対するビットがセットされていた場合, *cfg600px* はエラーを検出せず, RI600PX 起動時にシステム・ダウンとなります。

図 20-1 アクセス許可パターン



- 定義形式

シンボルまたは数値

- 定義範囲

シンボル : TACP_SHARED (全ドメインにアクセスを許可)

数値 : 0 ~ 0x7FFF

- 省略時の扱い

デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は TACP_SHARED) を適用

20.10 タスク情報 (task[])

ここでは、各タスクを定義します。

形式

<> 内は、ユーザが記述する部分を示します。

```
task[ <1. ID 番号> ] {
    name           = <2. ID 名称 (name) >;
    entry_address = <3. タスクの開始アドレス (entry_address) >;
    stack_size    = <4. ユーザ・スタック・サイズ (stack_size) >;
    stack_section = <5. ユーザ・スタック領域に付与するセクション名 (stack_section) >;
    priority      = <6. 起動時優先度 (priority) >;
    initial_start = <7. TA_ACT 属性 (initial_start) >;
    exinf         = <8. 拡張情報 (exinf) >;
    texrtn       = <9. タスク例外処理ルーチンの開始アドレス (texrtn) >;
    domain_num    = <10. 所属ドメイン ID (domain_num) >;
};
```

1) ID 番号

- 説明
タスク ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (name)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称> <ID 番号>
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) タスクの開始アドレス (entry_address)

- 説明
タスクの実行開始関数を定義します。
- 定義形式
関数名
- 定義範囲
—
- 省略時の扱い
省略不可

4) ユーザ・スタック・サイズ (*stack_size*)

- 説明
ユーザ・スタック・サイズを定義します。
- 定義形式
数値
- 定義範囲
以下に示す値以上の16の倍数

表 20 - 8 ユーザ・スタック・サイズの下限值

system.context 設定	コンパイラ・オプション“-isa”	下限値
NO	—	68
FPSW	—	72
ACC	“-isa=rxv2”	92
	“-isa=rxv1” またはオプション“-isa”の指定なし	76
FPSW,ACC	“-isa=rxv2”	96
	“-isa=rxv1” またはオプション“-isa”の指定なし	80
MIN	—	44
MIN,FPSW	—	48
MIN,ACC	“-isa=rxv2”	68
	“-isa=rxv1” またはオプション“-isa”の指定なし	52
MIN,FPSW,ACC	“-isa=rxv2”	72
	“-isa=rxv1” またはオプション“-isa”の指定なし	56

備考 コンパイラ・オプション“-isa”は、コンパイラ CC-RX V2.01 以降でサポートされています。

- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は256）を適用

5) ユーザ・スタック領域に付与するセクション名 (*stack_secion*)

- 説明
ユーザ・スタック領域に付与するセクション名を定義します。
cfg600px は、*stack_secion* で指定されたセクションに *stack_size* で指定されたサイズのユーザ・スタック領域を生成します。このセクションのセクション属性は“DATA”，アライメント数は4です。リンク時には、このセクションをRAM領域に配置してください。ただし、0番地に配置してはなりません。
また、このセクションは16バイト境界アドレスに配置する必要があるため、リンク時に“aligned_section”リンク・オプションを指定してください。
- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時はSURI_STACK）を適用

6) 起動時優先度 (*priority*)

- 説明
タスクの起動時の優先度を定義します。

- 定義形式
数値
- 定義範囲
1～システム情報 (system) のタスク優先度の最大値 (priority)
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 1) を適用

7) TA_ACT 属性 (*initial_start*)

- 説明
タスクの初期状態を定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 - ON : TA_ACT 属性を指定する (初期状態を READY 状態とする)
 - OFF : TA_ACT 属性を指定しない (初期状態を DORMANT 状態とする)
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は OFF) を適用

8) 拡張情報 (*exinf*)

- 説明
タスクの拡張情報を定義します。
- 定義形式
数値
- 定義範囲
0～0xFFFFFFFF
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 0) を適用
- 備考
タスクが TA_ACT 属性, *act_tsk* または *iact_tsk* によって起動された場合, タスクに拡張情報が渡されます。

9) タスク例外処理ルーチンの開始アドレス (*texrtn*)

- 説明
タスク例外処理ルーチンの実行開始関数を定義します。タスク例外処理ルーチンを定義しない場合は, 本定義を行わないでください。
- 定義形式
関数名
- 定義範囲
—
- 省略時の扱い
タスク例外処理ルーチンは定義されません。

10) 所属ドメイン ID (*domain_num*)

- 説明
タスクが所属するドメイン ID を定義します。
- 定義形式
数値
- 定義範囲
1～15
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 1) を適用

20.11 セマフォ情報 (semaphore[])

ここでは、各セマフォを定義します。

形式

<> 内は、ユーザが記述する部分を示します。

```
semaphore[ <1. ID 番号 > ] {  
    name           = <2. ID 名称 (name) >;  
    max_count      = <3. 最大資源数 (max_count) >;  
    initial_count  = <4. 初期資源数 (initial_count) >;  
    wait_queue     = <5. 待ちキュー属性 (wait_queue) >;  
};
```

1) ID 番号

- 説明
セマフォ ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (*name*)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称 > <ID 番号 >
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) 最大資源数 (*max_count*)

- 説明
セマフォの最大資源数を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 65535
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 1) を適用

4) 初期資源数 (*initial_count*)

- 説明
セマフォの初期資源数を指定します。
- 定義形式
数値
- 定義範囲
0 ~ *max_count*
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は1）を適用

5) 待ちキュー属性 (*wait_queue*)

- 説明
待ちキューの属性を定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 - TA_TFIFO : FIFO 順
 - TA_TPRI : タスクの現在優先度順
ただし、同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

20.12 イベントフラグ情報 (flag[])

ここでは、各イベントフラグを定義します。

形式

<> 内は、ユーザが記述する部分を示します。

```
flag[ <1. ID 番号> ] {  
    name           = <2. ID 名称 (name) >;  
    initial_pattern = <3. 初期ビット・パターン (initial_pattern) >;  
    wait_multi     = <4. 複数待ちの許可属性 (wait_multi) >;  
    clear_attribute = <5. クリア属性 (clear_attribute) >;  
    wait_queue     = <6. 待ちキュー属性 (wait_queue) >;  
};
```

1) ID 番号

- 説明
イベントフラグ ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (*name*)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称> <ID 番号>
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) 初期ビット・パターン (*initial_pattern*)

- 説明
イベントフラグの初期ビット・パターンを定義します。
- 定義形式
数値
- 定義範囲
0 ~ 0xFFFFFFFF
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 0) を適用

4) 複数待ちの許可属性 (*wait_multi*)

- 説明
複数タスク待ちの許可に関する属性を定義します。

- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 - TA_WSGL : 複数タスクの待ちを許可しない
 - TA_WMUL : 複数タスクの待ちを許可する
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_WSGL）を適用

5) クリア属性 (*clear_attribute*)

- 説明
イベントフラグのクリア属性 (TA_CLR) を定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 - NO : TA_CLR 属性を指定しない
 - YES : TA_CLR 属性を指定する
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は NO）を適用

6) 待ちキュー属性 (*wait_queue*)

- 説明
待ちキューの属性を定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
ただし、TA_CLR 属性が指定されていない場合 (*clear_attribute* = NO) は、TA_TPRI を指定した場合でも、待ちキューは FIFO 順に管理されます。この振る舞いは、 μ ITRON4.0 仕様の範囲外です。
 - TA_TFIFO : FIFO 順
 - TA_TPRI : タスクの現在優先度順
ただし、同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

20.13 データ・キュー情報 (dataqueue[])

ここでは、各データ・キュー定義します。

形式

<>内は、ユーザが記述する部分を示します。

```
dataqueue[ <1. ID 番号 > ] {  
    name           = <2. ID 名称 (name) >;  
    buffer_size    = <3. データ数 (buffer_size) >;  
    wait_queue     = <4. 待ちキュー属性 (wait_queue) >;  
};
```

1) ID 番号

- 説明
データ・キュー ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (*name*)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称> <ID 番号>
```
- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) データ数 (*buffer_size*)

- 説明
データ・キューに格納可能なデータ数を定義します。
- 定義形式
数値
- 定義範囲
0 ~ 65535
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 0) を適用

4) 待ちキュー属性 (*wait_queue*)

- 説明
送信待ちキューの属性を定義します。
なお、受信待ちキューは常に FIFO 順で管理されます。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 - TA_TFIFO : FIFO 順
 - TA_TPRI : タスクの現在優先度順
ただし、同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

20.14 メールボックス情報 (mailbox[])

ここでは、各メールボックスを定義します。

形式

<>内は、ユーザが記述する部分を示します。

```
mailbox[ <1. ID 番号> ] {  
    name           = <2. ID 名称 (name) >;  
    wait_queue     = <3. 待ちキュー属性 (wait_queue) >;  
    message_queue  = <4. メッセージのキューイング属性 (message_queue) >;  
    max_pri        = <5. 最大メッセージ優先度 (max_pri) >;  
};
```

1) ID 番号

- 説明
メールボックス ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (name)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称> <ID 番号>
```
- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) 待ちキュー属性 (*wait_queue*)

- 説明
待ちキューの属性を定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 - TA_TFIFO : FIFO 順
 - TA_TPRI : タスクの現在優先度順
ただし、同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は TA_TFIFO) を適用

4) メッセージのキューイング属性 (*message_queue*)

- 説明
メッセージのキューイング属性を定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 - TA_MFIFO : メッセージの送信要求を行った順
 - TA_MPRI : メッセージの優先度順
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は TA_MFIFO) を適用

5) 最大メッセージ優先度 (*max_pri*)

- 説明
message_queue に TA_MPRI を指定した場合、1 ~ *max_pri* のメッセージ優先度を使用できます。
なお、*message_queue* に TA_MFIFO を指定した場合、本項目は単に無視されます。
- 定義形式
数値
- 定義範囲
1 ~ システム情報 (system) のタスク優先度の最大値 (priority)
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 1) を適用

20.15 ミューテックス情報 (mutex[])

ここでは、各ミューテックスを定義します。

形式

<>内は、ユーザが記述する部分を示します。

```
mutex[ <1. ID番号> ] {  
    name           = <2. ID名称 (name)>;  
    ceilpri        = <3. 上限優先度 (ceilpri)>;  
};
```

1) ID番号

- 説明
ミューテックス ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID名称 (name)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID名称> <ID番号>
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) 上限優先度 (ceilpri)

- 説明
RI600PX のミューテックスは、[簡略化した優先度上限プロトコル](#)を採用しています。ここには、上限優先度を定義します。
- 定義形式
数値
- 定義範囲
1 ~ [システム情報 \(system\) のタスク優先度の最大値 \(priority\)](#)
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 1) を適用

20.16 メッセージ・バッファ情報 (message_buffer[])

ここでは、各メッセージ・バッファを定義します。

形式

<> 内は、ユーザが記述する部分を示します。

```
message_buffer[ <1. ID 番号> ] {  
    name           = <2. ID 名称 (name) >;  
    mbf_size       = <3. バッファ・サイズ (mbf_size) >;  
    mbf_section    = <4. メッセージ・バッファ領域に付与するセクション名 (mbf_section) >;  
    max_msgsiz    = <5. 最大メッセージ・サイズ (max_msgsiz) >;  
};
```

1) ID 番号

- 説明
メッセージ・バッファ ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (name)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称> <ID 番号>
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) バッファ・サイズ (mbf_size)

- 説明
メッセージ・バッファのサイズを、バイト単位で義します。
- 定義形式
数値
- 定義範囲
0, または 8 ~ 65532 の範囲の 4 の倍数
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 0) を適用

4) メッセージ・バッファ領域に付与するセクション名 (*mbf_section*)

- 説明

メッセージ・バッファ領域に付与するセクション名を定義します。
mbf_size > 0 の場合、*cfg600px* は *mbf_section* で指定されたセクションに *mbf_size* で指定されたサイズのメッセージ・バッファ領域を生成します。このセクションのセクション属性は "DATA", アライメント数は 4 です。リンク時には、このセクションを RAM 領域に配置してください。
リンク時に、このセクションを 0 番地に配置してはなりません。

- 定義形式

シンボル

- 定義範囲

—

- 省略時の扱い

デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は BURI_HEAP）を適用

5) 最大メッセージ・サイズ (*max_msgsz*)

- 説明

最大メッセージ・サイズをバイト単位で指定します。
mbf_size > 0 の場合は、*max_msgsz* は (*mbf_size* - 4) 以下でなければなりません。

- 定義形式

数値

- 定義範囲

1 ~ 65528

- 省略時の扱い

デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 4）を適用

20.17 固定長メモリ・プール情報 (memorypool[])

ここでは、各固定長メモリ・プールを定義します。

形式

<>内は、ユーザが記述する部分を示します。

```
memorypool[ <1. ID 番号 > ] {  
    name           = <2. ID 名称 (name) >;  
    siz_block      = <3. 固定長メモリ・ブロック・サイズ (siz_block) >;  
    num_block      = <4. 固定長メモリ・ブロック数 (num_block) >;  
    section        = <5. メモリ・プール領域に付与するセクション名 (section) >;  
    wait_queue     = <6. 待ちキュー属性 (wait_queue) >;  
};
```

1) ID 番号

- 説明
固定メモリ・プール ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (name)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称 > <ID 番号 >
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) 固定長メモリ・ブロック・サイズ (siz_block)

- 説明
固定長メモリ・ブロック・サイズを、バイト単位で定義します。
- 定義形式
数値
- 定義範囲
1 ~ 65535
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 256) を適用

4) 固定長メモリ・ブロック数 (num_block)

- 説明
固定長メモリ・ブロック数を定義します。

- 定義形式
数値
- 定義範囲
1 ~ 65535
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 1）を適用

5) メモリ・プール領域に付与するセクション名 (*section*)

- 説明
固定長メモリ・プール領域に付与するセクション名を定義します。
cfg600px は、*section* で指定されたセクションに *siz_block* × *num_block* で算出されたサイズの固定長メモリ・プール領域を生成します。このセクションのセクション属性は "DATA", アライメント数は 4 です。リンク時には、このセクションを RAM 領域に配置してください。
リンク時に、このセクションを 0 番地に配置してはなりません。
- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は BRI_HEAP）を適用

6) 待ちキュー属性 (*wait_queue*)

- 説明
待ちキューの属性を定義します。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
 TA_TFIFO : FIFO 順
 TA_TPRI : タスクの現在優先度順
 ただし、同じ現在優先度のタスクの中では FIFO 順
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は TA_TFIFO）を適用

20.18 可変長メモリ・プール情報 (variable_memorypool[])

ここでは、各可変長メモリ・プールを定義します。

形式

<>内は、ユーザが記述する部分を示します。

```
variable_memorypool[ <1. ID 番号 > ] {  
    name           = <2. ID 名称 (name) >;  
    heap_size      = <3. 可変長メモリ・プールのサイズ (heap_size) >;  
    max_memsize    = <4. 可変長メモリ・ブロック・サイズの上限 (max_memsize) >;  
    mpl_section    = <5. メモリ・プール領域に付与するセクション名 (mpl_section) >;  
};
```

1) ID 番号

- 説明
可変メモリ・プール ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (*name*)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称 > <ID 番号 >
```
- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) 可変長メモリ・プールのサイズ (*heap_size*)

- 説明
可変長メモリ・プールのサイズを、バイト単位で定義します。
- 定義形式
数値
- 定義範囲
24 ~ 0x10000000
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値 (出荷時は 1024) を適用

4) 可変長メモリ・ブロック・サイズの上限 (*max_memsize*)

- 説明
可変長メモリ・ブロックサイズの上限をバイト単位で定義します。
- 定義形式
数値
- 定義範囲
1 ~ 0xBFFFFFF4
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は 36）を適用
- 補足
可変長メモリ・ブロック・サイズについては、「9.3.1 可変長メモリ・ブロックのサイズ」を参照してください。

5) メモリ・プール領域に付与するセクション名 (*mpl_section*)

- 説明
可変長メモリ・プール領域に付与するセクション名を定義します。
cfg600px は、*mpl_section* で指定されたセクションに *heap_size* バイトの可変長メモリ・プール領域を生成します。このセクションのセクション属性は "DATA"、アライメント数は 4 です。リンク時には、このセクションを RAM 領域に配置してください。
リンク時に、このセクションを 0 番地に配置してはなりません。
- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は BRI_HEAP）を適用

20.19 周期ハンドラ情報 (cyclic_hand[])

ここでは、各周期ハンドラを定義します。

形式

<>内は、ユーザが記述する部分を示します。

```
cyclic_hand[ <1. ID 番号> ] {
    name           = <2. ID 名称 (name) >;
    entry_address  = <3. 周期ハンドラの開始アドレス (entry_address) >;
    interval_counter = <4. 起動周期 (interval_counter) >;
    start          = <5. 初期状態 (start) >;
    phs_counter    = <6. 起動位相 (phs_counter) >;
    phsatr         = <7. TA_PHS 属性 (phsatr) >;
    exinf         = <8. 拡張情報 (exinf) >;
};
```

1) ID 番号

- 説明
周期ハンドラ ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (name)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称> <ID 番号>
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) 周期ハンドラの開始アドレス (entry_address)

- 説明
周期ハンドラの実行開始関数を定義します。
- 定義形式
関数名
- 定義範囲
—
- 省略時の扱い
省略不可

4) 起動周期 (interval_counter)

- 説明
起動周期をミリ秒単位で定義します。
 - 定義形式
数値
 - 定義範囲
1 ~ (0x7FFFFFFF - system.tic_num) / system.tic_deno
 - 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は1）を適用
- 5) 初期状態 (*start*)
- 説明
周期ハンドラの初期状態を定義します。
 - 定義形式
シンボル
 - 定義範囲
以下のいずれか。
 - OFF : 停止状態とする (TA_STA 属性なし)
 - ON : 動作状態とする (TA_STA 属性あり)
 - 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時はOFF）を適用
- 6) 起動位相 (*phs_counter*)
- 説明
起動位相をミリ秒単位で定義します。
 - 定義形式
数値
 - 定義範囲
0 ~ *interval_counter*
 - 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は0）を適用
- 7) TA_PHS 属性 (*phsattr*)
- 説明
起動位相に関する属性を定義します。
 - 定義形式
シンボル
 - 定義範囲
以下のいずれか。
 - OFF : 起動位相を保存しない (TA_PHS 属性なし)
 - ON : 起動位相を保存する (TA_PHS 属性あり)
 - 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時はOFF）を適用
- 8) 拡張情報 (*exinf*)
- 説明
周期ハンドラの拡張情報を定義します。
 - 定義形式
数値
 - 定義範囲
0 ~ 0xFFFFFFFF

- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は0）を適用
- 備考
周期ハンドラには、拡張情報が渡されます。

20.20 アラーム・ハンドラ情報 (alarm_hand[])

ここでは、各アラーム・ハンドラを定義します。

形式

<>内は、ユーザが記述する部分を示します。

```
alarm_hand[ <1. ID 番号> ] {  
    name           = <2. ID 名称 (name)>;  
    entry_address  = <3. アラーム・ハンドラの開始アドレス (entry_address)>;  
    exinf          = <4. 拡張情報 (exinf)>;  
};
```

1) ID 番号

- 説明
アラーム・ハンドラ ID 番号を定義します。
- 定義形式
数値
- 定義範囲
1 ~ 255
- 省略時の扱い
cfg600px が ID 番号を自動的に割り当てます。

2) ID 名称 (name)

- 説明
ID 名称を定義します。指定された ID 名称は、システム情報ヘッダ・ファイル (kernel_id.h) に以下の形式で出力されます。

```
#define <ID 名称> <ID 番号>
```

- 定義形式
シンボル
- 定義範囲
—
- 省略時の扱い
省略不可

3) アラーム・ハンドラの開始アドレス (entry_address)

- 説明
アラーム・ハンドラの実行開始関数を定義します。
- 定義形式
関数名
- 定義範囲
—
- 省略時の扱い
省略不可

4) 拡張情報 (*exinf*)

- 説明
アラーム・ハンドラの拡張情報を定義します。
- 定義形式
数値
- 定義範囲
0 ~ 0xFFFFFFFF
- 省略時の扱い
デフォルト・システム・コンフィギュレーション・ファイルの設定値（出荷時は0）を適用
- 備考
アラーム・ハンドラには、拡張情報が渡されます。

20.21 可変ベクタ情報 (interrupt_vector[])

ここでは、RX MCU の可変ベクタに設定する割り込みハンドラを定義します。

未定義の割り込みが発生した場合は、システム・ダウンとなります。

なお、cfg600px はここで定義した割り込みに関する割り込み制御レジスタや、割り込み要因等の初期設定のコードは生成しません。初期設定は、アプリケーションで実装いただく必要があります。

備考 ベクタ番号 1 ~ 8 は RI600PX が使用するため、定義しないでください。また、MCU 仕様で予約となっているベクタには定義しないでください。

形式

<> 内は、ユーザが記述する部分を示します。

```
interrupt_vector[ <1. ベクタ番号 > ] {
    entry_address = <2. 割り込みハンドラの開始アドレス (entry_address) >;
    os_int        = <3. カーネル管理割り込みの指定 (os_int) >;
    pragma_switch = <4. pragma ディレクティブに渡すスイッチ (pragma_switch) >;
};
```

1) ベクタ番号

- 説明
ベクタ番号を定義します。
- 定義形式
数値
- 定義範囲
0 ~ 255
- 省略時の扱い
省略不可

2) 割り込みハンドラの開始アドレス (entry_address)

- 説明
割り込みハンドラの実行開始関数を定義します。
- 定義形式
関数名
- 定義範囲
—
- 省略時の扱い
省略不可

3) カーネル管理割り込みの指定 (os_int)

- 説明
この割り込みがカーネル管理割り込みかどうかを定義します。
カーネル割り込みマスクレベル (system_IPL) 以下の割り込み優先レベルの割り込みはカーネル管理割り込み、それ以外の割り込みはカーネル管理外割り込みとして定義する必要があります。
なお、カーネル割り込みマスクレベル (system_IPL) が 15 の場合は、すべての可変ベクタ割り込みはカーネル管理割り込みとする必要があります。
- 定義形式
シンボル
- 定義範囲
以下のいずれか。
YES : カーネル管理割り込み

NO : カーネル管理外割り込み

- 省略時の扱い
省略不可

4) pragma ディレクティブに渡すスイッチ (*pragma_switch*)

- 説明

cfg600px は、*entry_address* で指定された関数を割り込み関数として扱う #pragma interrupt ディレクティブを、システム情報ヘッダ・ファイル kernel_id.h に出力します。この pragma ディレクティブに渡すスイッチを指定します。

- 定義形式
シンボル

- 定義範囲

以下を指定できます。複数指定する場合は、カンマで区切ってください。なお、“ACC”と“NOACC”を同時に指定することはできません。

E : 多重割り込みを許可する“enable”スイッチを渡します

F : 高速割り込みを指定する“fint”スイッチを渡します。なお、高速割り込みは必ずカーネル管理外割り込み (os_int=NO) としなければなりません。

S : 割り込みハンドラで使用するレジスタ数を制限する“save”スイッチを渡します。

ACC : 割り込みハンドラで ACC レジスタを保證する“acc”スイッチを渡します。

NOACC : 割り込みハンドラで ACC レジスタを保證しない“no_acc”スイッチを渡します。

- 省略時の扱い

何もスイッチを渡しません。

備考1 ACC レジスタの保證については、以下を参照してください。

表 20 - 9 ACC レジスタの保證

pragma_switch の設定	コンパイラの“-save_acc”オプション	
	なし	あり
“ACC”, “NOACC”ともに無し	“acc”, “no_acc” いずれのスイッチも渡されません。ACC は保證されません。	“acc”, “no_acc” いずれのスイッチも渡されません。ACC は保證されます。
“ACC”あり	“acc”スイッチが渡されます。ACC は保證されます。	
“NOACC”あり	“no_acc”スイッチが渡されます。ACC は保證されません。	

備考2 基本クロック用タイマ・チャネルの選択 (timer) に“CMT0”, “CMT1”, “CMT2”, または“CMT3”のいずれかを指定した場合は、暗黙的に以下の指定で interrupt_vector[] が定義がなされたものと扱います。

- ベクタ番号

- “CMT0” : 28

- “CMT1” : 29

- “CMT2” : 30

- “CMT3” ; 31

- *entry_address* : RI600PX 内の基本クロック割り込み処理ルーチンの開始アドレス

- *os_int* : YES

- *pragma_switch* : E,ACC

20.22 固定ベクタ／例外ベクタ情報 (interrupt_fvector[])

ここでは、RXv1 アーキテクチャの固定ベクタ・テーブル (0xFFFFFFF80 ~ 0xFFFFFFFF 番地) / RXv2 アーキテクチャの例外ベクタ・テーブルを定義します。

MCUによっては、固定ベクタ・テーブル／例外ベクタ・テーブルには、割り込みハンドラのアドレスだけでなく、エンディアン選択レジスタなども含まれています。

固定ベクタ／例外ベクタの割り込み要因は、すべてカーネル管理外割り込みの扱いとなります。

RI600PXでは、ベクタ・アドレスに対して表 20 - 10 のようにベクタ番号を割り当てています。本定義を行わないベクタの扱いも表 20 - 10 に示します。

なお、固定ベクタ・テーブル／例外ベクタ・テーブルの内容は MCU 毎に異なります。詳細は、使用する MCU のハードウェア・マニュアルを参照してください。

また、cfg600px はここで定義した割り込みに関する割り込み制御レジスタや、割り込み要因等の初期設定のコードは生成しません。初期設定は、アプリケーションで実装いただく必要があります。

表 20 - 10 固定ベクタ・テーブル／例外ベクタ・テーブル

ベクタ・アドレス ^a	ベクタ番号	要因の例 (MCU 毎に異なります)	定義省略時の扱い
0xFFFFFFF80	0	エンディアン選択レジスタ	コンパイラのendianオプションに応じて、以下が設定されます。 - “-endian=little” の場合 0xFFFFFFFF - “-endian=big” の場合 0xFFFFFFF8
0xFFFFFFF84	1	(予約領域)	0xFFFFFFFF
0xFFFFFFF88	2	オプション機能選択レジスタ 1	
0xFFFFFFF8C	3	オプション機能選択レジスタ 0	
0xFFFFFFF90	4	(予約領域)	
0xFFFFFFF94	5	(予約領域)	
0xFFFFFFF98	6	(予約領域)	
0xFFFFFFF9C	7	ROM コード・プロテクト (フラッシュ・メモリ)	
0xFFFFFFFA0	8	オンチップ・デバッガ ID コード・プロテクト (フラッシュ・メモリ)	
0xFFFFFFFA4	9		
0xFFFFFFFA8	10		
0xFFFFFFAAC	11		
0xFFFFFFFBC	12	(予約領域)	
0xFFFFFFFBC4	13	(予約領域)	
0xFFFFFFFBC8	14	(予約領域)	
0xFFFFFFFBC	15	(予約領域)	

ベクタ・アドレス ^a	ベクタ番号	要因の例 (MCU 毎に異なります)	定義省略時の扱い
0xFFFFF0C0	16	(予約領域)	システム・ダウン
0xFFFFF0C4	17	(予約領域)	
0xFFFFF0C8	18	(予約領域)	
0xFFFFF0CC	19	(予約領域)	
0xFFFFF0D0	20	特権命令例外	
0xFFFFF0D4	21	アクセス例外	アクセス例外ハンドラ ^b
0xFFFFF0D8	22	(予約領域)	システム・ダウン
0xFFFFF0DC	23	未定義命令例外	
0xFFFFF0E0	24	(予約領域)	
0xFFFFF0E4	25	浮動小数点例外	
0xFFFFF0E8	26	(予約領域)	
0xFFFFF0EC	27	(予約領域)	
0xFFFFF0F0	28	(予約領域)	
0xFFFFF0F4	29	(予約領域)	
0xFFFFF0F8	30	ノンマスカブル割り込み	
0xFFFFF0FC	31	リセット	

- a. 表 20 - 10 に記載のベクタ・アドレスは、RXv1 アーキテクチャの固定ベクタ・テーブルのアドレスです。RXv2 アーキテクチャの例外ベクタ・テーブルのアドレスは、EXTB レジスタによって決まります。EXTB レジスタの初期値は、RXv1 アーキテクチャの固定ベクタ・テーブルのアドレスと同じです。2.6.4 節の「FIX_INTERRUPT_VECTOR セクション」も参照してください。
- b. ベクタ 21(アクセス例外)にハンドラを定義しないでください。定義すると、アクセス例外ハンドラが起動されなくなります。

形式

<>内は、ユーザが記述する部分を示します。

```
interrupt_fvector[ <1. ベクタ番号 > ] {
    entry_address = <2. 割り込みハンドラの開始アドレス (entry_address) >;
    pragma_switch = <3. pragma ディレクティブに渡すスイッチ (pragma_switch) >;
};
```

1) ベクタ番号

- 説明
ベクタ番号を定義します。
- 定義形式
数値
- 定義範囲
0 ~ 31

- 省略時の扱い
省略不可

2) 割り込みハンドラの開始アドレス (*entry_address*)

- 説明
割り込みハンドラの実行開始関数、または固定ベクタ／例外ベクタへの設定値を定義します。
- 定義形式
関数名または数値
- 定義範囲
数値の場合は 0 ~ 0xFFFFFFFF
- 省略時の扱い
省略不可

3) pragma ディレクティブに渡すスイッチ (*pragma_switch*)

- 説明
cfg600px は、*entry_address* で指定された関数を、割り込み関数として kernel_id.h に #pragma interrupt ディレクティブを出力します。この pragma ディレクティブに渡すスイッチを指定します。
なお、*entry_address* に数値を指定した場合、およびベクタ番号 31 (リセット) については、#pragma interrupt ディレクティブは出力されません。
- 定義形式
シンボル
- 定義範囲
以下を指定できます。複数指定する場合は、カンマで区切ってください。なお、“ACC” と “NOACC” を同時に指定することはできません。
 - S : 割り込みハンドラで使用するレジスタ数を制限する “save” スイッチを渡します。
 - ACC : 割り込みハンドラで ACC レジスタを保証する “acc” スイッチを渡します。
 - NOACC : 割り込みハンドラで ACC レジスタを保証しない “no_acc” スイッチを渡します。
- 省略時の扱い
何もスイッチを渡しません。
- 備考
ACC レジスタの扱いについては、[表 20 - 9](#) を参照してください。

20.23 RAM 使用量の算出

RI600PX が使用／管理する RAM 領域は、その用途により、以下の4種類のセクションに大別されます。

- BRI_RAM セクション：RI600PX の管理データ、システム・コンフィギュレーション・ファイルで生成されたデータ・キュー領域、システム・コンフィギュレーション・ファイルでセクション指定を省略して生成されたメッセージ・バッファ領域
- RRI_RAM セクション：RI600PX の管理データ（動的生成使用時）
- BURI_HEAP セクション：システム・コンフィギュレーション・ファイルでセクション指定を省略して生成された固定長メモリ・プール領域および可変長メモリ・プール領域
- SURI_STACK セクション：システム・コンフィギュレーション・ファイルでセクション指定を省略して生成されたタスクのユーザ・スタック領域
- SI セクション：システム・スタック領域

20.23.1 BRI_RAM, RRI_RAM セクション

BRI_RAM および RRI_RAM セクションは、RI600PX の管理データが割り付けられるセクションです。

表 20 - 11 に、BRI_RAM および RRI_RAM セクションのメモリ容量計算式（単位：バイト）を示します。なお、実際のサイズは、境界調整のために表 20 - 11 で算出される値よりも大きくなる場合があります。

表 20 - 11 BRI_RAM および RRI_RAM セクションのメモリ容量計算式

種別	セクション	メモリ容量計算式（単位：バイト）
システム管理 ブロック	BRI_RAM	$28 + 4 \times \text{down}((TMAX_TPRI - 1) / 32 + 1) + TMAX_TPRI$ $+ VTMAX_SEM + VTMAX_FLG + 2 \times VTMAX_DTQ + VTMAX_MBX$ $+ VTMAX_MTX + 2 \times VTMAX_MBF + VTMAX_MPF + VTMAX_MPL$ $+ 57 \times VTMAX_DOMAIN$
	RRI_RAM ^a	$4 + VTMAX_SEM + VTMAX_FLG + 2 \times VTMAX_DTQ + VTMAX_MBX$ $+ VTMAX_MTX + 2 \times VTMAX_MBF + VTMAX_MPF + VTMAX_MPL$
タスク管理 ブロック	BRI_RAM	$28 \times VTMAX_TSK$
	RRI_RAM ^b	$24 \times VTMAX_TSK$
セマフォ管理 ブロック ^c	BRI_RAM	$4 \times VTMAX_SEM + \text{down}(VTMAX_SEM / 8 + 1)$
	RRI_RAM ^d	$4 \times VTMAX_SEM$
イベントフラグ 管理ブロック ^e	BRI_RAM	$8 \times VTMAX_FLG + 2 \times \text{down}(VTMAX_FLG / 8 + 1)$
	RRI_RAM ^f	$4 \times VTMAX_FLG$
データ・キュー 管理ブロック ^g	BRI_RAM	$6 \times VTMAX_DTQ + \text{down}(VTMAX_DTQ / 8 + 1) + DTQ_ALLSIZE$
	RRI_RAM ^h	$8 \times VTMAX_DTQ$
メールボックス 管理ブロック ⁱ	BRI_RAM	$8 \times VTMAX_MBX + 2 \times \text{down}(VTMAX_MBX / 8 + 1)$
	RRI_RAM ^j	$VTMAX_MBX$
ミューテックス 管理ブロック ^k	BRI_RAM	$VTMAX_MTX + \text{down}(VTMAX_MTX / 8 + 1)$
	RRI_RAM ^l	$VTMAX_MTX$
メッセージ・ バッファ 管理ブロック ^m	BRI_RAM	$16 \times VTMAX_MBF + MBF_ALLSIZE$
	RRI_RAM ⁿ	$8 \times VTMAX_MBF$

種別	セクション	メモリ容量計算式 (単位: バイト)
固定長メモリ・ プール管理 ブロック ^o	BRI_RAM	$8 \times VTMAX_MPF + \text{down}(VTMAX_MPF / 8 + 1)$ $+ \sum (\text{memorypool}[\text{ }].\text{num_block} / 8 + 1)$
	RRI_RAM ^p	$12 \times VTMAX_MPF$
可変長メモリ・ プール管理 ブロック ^q	BRI_RAM	$208 \times VTMAX_MPL$
	RRI_RAM ^r	$20 \times VTMAX_MPL$
周期ハンドラ 管理ブロック ^s	BRI_RAM	$8 \times VTMAX_CYH$
	RRI_RAM ^t	$20 \times VTMAX_CYH$
アラーム・ハンド ラ管理ブロック ^u	BRI_RAM	$8 \times VTMAX_ALH$
	RRI_RAM ^v	$8 \times VTMAX_ALH$

- a. 最大セマフォ ID (`max_sem`)、最大イベントフラグ ID (`max_flag`)、最大データ・キュー ID (`max_dtq`)、最大メールボックス ID (`max_mbx`)、最大ミューテックス ID (`max_mtx`)、最大メッセージ・バッファ ID (`max_mbf`)、最大固定長メモリ・プール ID (`max_mpf`)、最大可変長メモリ・プール ID (`max_mpl`) のすべてが 0 または未定義の場合、この領域のサイズは 4 バイトとなります。
- b. 最大タスク ID (`max_task`) の定義がある場合のみ、生成されます。
- c. `VTAX_SEM` が 0 の場合、これらの領域は生成されません。
- d. 最大セマフォ ID (`max_sem`) の定義がある場合のみ、生成されます。
- e. `VTMAX_FLG` が 0 の場合、これらの領域は生成されません。
- f. 最大イベントフラグ ID (`max_flag`) の定義がある場合のみ、生成されます。
- g. `VTMAX_DTQ` が 0 の場合、これらの領域は生成されません。
- h. 最大データ・キュー ID (`max_dtq`) の定義がある場合のみ、生成されます。
- i. `VTMAX_MBX` が 0 の場合、これらの領域は生成されません。
- j. 最大メールボックス ID (`max_mbx`) の定義がある場合のみ、生成されます。
- k. `VTMAX_MTX` が 0 の場合、これらの領域は生成されません。
- l. 最大ミューテックス ID (`max_mtx`) の定義がある場合のみ、生成されます。
- m. `VTMAX_MBF` が 0 の場合、これらの領域は生成されません。
- n. 最大メッセージ・バッファ ID (`max_mbf`) の定義がある場合のみ、生成されます。
- o. `VTMAX_MPF` が 0 の場合、これらの領域は生成されません。
- p. 最大固定長メモリ・プール ID (`max_mpf`) の定義がある場合のみ、生成されます。
- q. `VTMAX_MPL` が 0 の場合、これらの領域は生成されません。
- r. 最大可変長メモリ・プール ID (`max_mpl`) の定義がある場合のみ、生成されます。
- s. `VTMAX_CYH` が 0 の場合、これらの領域は生成されません。
- t. 最大周期ハンドラ ID (`max_cyh`) の定義がある場合のみ、生成されます。
- u. `VTMAX_ALH` が 0 の場合、これらの領域は生成されません。
- v. 最大アラーム・ハンドラ ID (`max_alh`) の定義がある場合のみ、生成されます。

備考 メモリ容量計算式のキーワードは、以下に示した意味を持ちます。

VTMAX_TPRI : 最大タスク優先度です。

cfg600px は、システム情報 (system) のタスク優先度の最大値 (priority) 設定値を定義したマクロ `VTMAX_TPRI` を、システム情報ヘッダ・ファイル `kernel_id.h` に出力します。

VTMAX_TSK : 最大タスク ID です。

cfg600px は、最大タスク ID を定義したマクロ `VTMAX_TSK` を、システム情報ヘッダ・ファイル `kernel_id.h` に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。

VTMAX_SEM : 最大セマフォ ID です。

cfg600px は、最大セマフォ ID を定義したマクロ `VTMAX_SEM` を、システム情報ヘッダ・ファイル `kernel_id.h` に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。

VTMAX_FLG : 最大イベントフラグ ID です。

cfg600px は、最大イベントフラグ ID を定義したマクロ `VTMAX_FLG` を、システム情

報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。

- VTMAX_DTQ** : 最大データ・キュー ID 数です。
cfg600px は、最大データ・キュー ID を定義したマクロ **VTMAX_FLG** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- DTQ_ALLSIZE** : システム・コンフィギュレーション・ファイルで生成したデータ・キュー領域のサイズの合計です。具体的には、以下によって算出されます。
 $\sum \text{dataqueue[]}.buffer_size \times 4$
ただし、この計算結果が 0 になる場合は、**DTQ_ALLSIZE** は 4 です。
- VTMAX_MBX** : 最大メールボックス ID です。
cfg600px は、最大メールボックス ID を定義したマクロ **VTMAX_MBX** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- VTMAX_MTX** : 最大ミューテックス ID です。
cfg600px は、最大ミューテックス ID を定義したマクロ **VTMAX_MTX** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- VTMAX_MBF** : 最大メッセージ・バッファ ID です。
cfg600px は、最大メッセージ・バッファ ID を定義したマクロ **VTMAX_MBF** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- MBF_ALLSIZE** : システム・コンフィギュレーション・ファイルで、“mbf_section” を省略して生成したメッセージ・バッファ領域のサイズの合計です。具体的には、以下によって算出されます。
 $\sum \text{message_buffer[]}.mbf_size$
- VTMAX_MPF** : 最大固定長メモリ・プール ID です。
cfg600px は、最大固定長メモリ・プール ID を定義したマクロ **VTMAX_MPF** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- VTMAX_MPL** : 最大可変長メモリ・プール ID です。
cfg600px は、最大可変長メモリ・プール ID を定義したマクロ **VTMAX_MPL** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- VTMAX_CYH** : 最大周期ハンドラ ID です。
cfg600px は、最大周期ハンドラ ID を定義したマクロ **VTMAX_CYH** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- VTMAX_ALH** : 最大アラーム・ハンドラ ID です。
cfg600px は、最大アラーム・ハンドラ ID を定義したマクロ **VTMAX_ALH** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。
- VTMAX_DOMAIN** : 最大ドメイン ID です。
cfg600px は、最大ドメイン ID を定義したマクロ **VTMAX_DOMAIN** を、システム情報ヘッダ・ファイル kernel_id.h に出力します。詳細は、「20.7 最大 ID (maxdefine)」を参照してください。

20.23.2 BURI_HEAP セクション

BURI_HEAP セクションは、固定長メモリ・プール領域および可変長メモリ・プール領域が割り付けられるセクションです。なお、固定長メモリ・プールおよび可変長メモリ・プールの定義時に、それぞれの領域をユーザ指定のセクションに割り当てることもできます。

BURI_HEAP セクションのサイズは、以下の合計です。なお、アプリケーション側で BURI_HEAP セクションにデータを生成した場合は、そのサイズも加算してください。

- 固定長メモリ・プール領域の合計サイズ

固定長メモリ・プール情報 (memorypool[]) で “section” を省略した固定長メモリ・プール定義について、以下の式で算出されます。

$$\sum (\text{memorypool}[\text{ }].\text{siz_block} \times \text{memorypool}[\text{ }].\text{num_block})$$

- 可変長メモリ・プール領域の合計サイズ

可変長メモリ・プール情報 (variable_memorypool[]) で “mpl_section” を省略した可変長メモリ・プール定義について、以下の式で算出されます。

$$\sum \text{variable_memorypool}[\text{ }].\text{heap_size}$$

20.23.3 SURI_STACK セクション

SURI_STACK セクションは、各タスクのユーザ・スタック領域が割り付けられるセクションです。なお、ユーザ・スタック領域は、システム・コンフィギュレーション・ファイルでの定義時にユーザ定義のセクションに割り当てることもできます。

SURI_STACK セクションのサイズは、タスク情報 (task[]) で “stack_section” を省略したタスク定義について、以下の式で算出されます。なお、アプリケーション側で SURI_STACK セクションにデータを生成した場合は、そのサイズも加算してください。

$$\sum \text{task}[\text{ }].\text{stack_size}$$

備考 スタック使用量の見積りについては、「付録D スタック使用量の算出」を参照してください。

20.23.4 SI セクション

SI セクションは、システム・スタック領域が割り付けられるセクションです。

SI セクションのサイズは、システム情報 (system) のシステム・スタック・サイズ (stack_size) です。

備考 スタック使用量の見積りについては、「付録D スタック使用量の算出」を参照してください。

20.24 記述例

以下に、システム・コンフィギュレーション・ファイルの記述例を示します。

備考 RI600PX では、システム・コンフィギュレーション・ファイルのサンプル・ソース・ファイルを提供しています。

```
// System Definition
system{
    stack_size = 1024;
    priority   = 10;
    system_IPL = 14;
    message_pri = 1;
    tic_deno   = 1;
    tic_num    = 1;
    context    = FPSW,ACC;
};

// System Clock Definition -----
clock{
    timer      = CMT0;
    template   = rx630.tpl;
    timer_clock = 25MHz;
    IPL        = 13;
};

// Number of object -----
maxdefine{
    max_task    = 10;
    max_sem     = 1;
    // max_flag  = ;
    max_dtq     = 1;
    // max_mbx   = ;
    // max_mtx   = ;
    // max_mbf   = ;
    // max_mpf   = ;
    max_mpl     = 1;
    // max_cyh   = ;
    // max_alh   = ;
    max_domain  = 3;
};

// Trusted domain
domain[1] {
    trust      = YES;
};

// Memory Object Definition : Master domain data
memory_object[1]{
    start_address = BU_MASTERDOM;
    end_address   = RU_MASTERDOM_2;
    acptn1       = 0x0001;
    acptn2       = 0x0001;
    acptn3       = 0;
};

// Memory Object Definition : App-domain A data
memory_object[2]{
    start_address = BU_DOM_A;
    end_address   = RU_DOM_A_2;
    acptn1       = 0x0002;
    acptn2       = 0x0002;
    acptn3       = 0;
};
```



```
// Memory Object Definition : App-domain B data
memory_object[3]{
    start_address = BU_DOM_B;
    end_address   = RU_DOM_B_2;
    acptn1        = 0x0004;
    acptn2        = 0x0004;
    acptn3        = 0;
};

// Memory Object Definition : Shared data
memory_object[4]{
    start_address = BURI_HEAP;
    end_address   = RU_SH_2;
    acptn1        = TACP_SHARED;
    acptn2        = TACP_SHARED;
    acptn3        = 0;
};

// Memory Object Definition : Master domain code and const
memory_object[5]{
    start_address = PU_MASTERDOM;
    end_address   = DU_MASTERDOM_2;
    acptn1        = 0x0001;
    acptn2        = 0;
    acptn3        = 0x0001;
};

// Memory Object Definition : App-domain A code and const
memory_object[6]{
    start_address = PU_DOM_A;
    end_address   = DU_DOM_A_2;
    acptn1        = 0x0002;
    acptn2        = 0;
    acptn3        = 0x0002;
};

// Memory Object Definition : App-domain B code and const
memory_object[7]{
    start_address = PU_DOM_B;
    end_address   = DU_DOM_B_2;
    acptn1        = 0x0004;
    acptn2        = 0;
    acptn3        = 0x0004;
};

// Memory Object Definition : Shared code and const
memory_object[8]{
    start_address = PU_SH;
    end_address   = DU_SH_2;
    acptn1        = TACP_SHARED;
    acptn2        = 0;
    acptn3        = TACP_SHARED;
};
```

```

// Task Definition -----
task[]{
    name           = ID_MASTERDOMTASK;
    entry_address  = MasterDom_Task();
    initial_start  = ON;
    stack_size     = 256;
    priority       = 1;
    // stack_section = SURI_STACK;
    exinf          = 1;
    // texrtn        = ;
    domain_num     = 1;
};

// Semaphore Definition -----
// semaphore[]{
//     name           = ID_SEM1;
//     wait_queue    = TA_TFIFO;
//     max_count     = 1;
//     initial_count = 1;
// };

// Eventflag Definition -----
// flag[]{
//     name           = ID_FLG1;
//     initial_pattern = 0;
//     wait_queue    = TA_TFIFO;
//     wait_multi    = TA_WSGL;
//     clear_attribute = NO;
// };

// Data Queue Definition -----
// dataqueue[]{
//     name           = ID_DTQ1;
//     buffer_size   = 4;
//     wait_queue    = TA_TFIFO;
// };

// Mailbox Definition -----
// mailbox[]{
//     name           = ID_MBX1;
//     wait_queue    = TA_TFIFO;
//     message_queue = TA_MFIFO;
//     max_pri       = 1;
// };

// Mutex definition -----
// mutex[]{
//     name           = ID_MTX1;
//     ceilpri       = 1;
// };

// Message Buffer Definition -----
// message_buffer[]{
//     name           = ID_MBF1;
//     mbf_size      = 128;
//     mbf_section   = BRI_RAM;
//     max_msgsz     = 16;
//     wait_queue    = TA_TFIFO;
// };

```

```

// Fixed-sized Memory Pool Definition -----
// memorypool[]{
//     name           = ID_MPF1;
//     section        = BURI_HEAP;
//     num_block      = 1;
//     siz_block      = 0x100;
//     wait_queue     = TA_TFIFO;
// };
// Variable-sized Memory Pool Definition -----
// variable_memorypool[]{
//     name           = ID_MPL1;
//     mpl_section    = BURI_HEAP;
//     heap_size      = 1024;
//     max_memsize    = 36;
// };

// Cyclic Handler Definition -----
cyclic_hand[] {
    name           = ID_CYC1;
    entry_address  = cyhl();
    interval_counter = 10;
    start          = ON;
    phsatr         = OFF;
    phs_counter    = 10;
    exinf          = 1;
};

// Alarm Handler Definition -----
alarm_hand[] {
    name           = ID_ALM1;
    entry_address  = alhl();
    exinf          = 1;
};

// Relocatable Vector Definition -----
// interrupt_vector[64]{
//     os_int         = YES;
//     entry_address  = inh64();
//     pragma_switch  = E;
// };

// Fixed Vector Definition -----
// interrupt_fvector[0] { // MDES register (address : 0xFFFFF80)
//     entry_address  = AUTO_ENDIAN;
// };

// interrupt_fvector[1] { // Reserved (address : 0xFFFFF84)
//     entry_address  = 0xFFFFFFFF;
// };

// interrupt_fvector[2] { // OFS1 register (address : 0xFFFFF88)
//     entry_address  = 0xFFFFFFFF;
// };

// interrupt_fvector[3] { // OFS0 register (address : 0xFFFFF8C)
//     entry_address  = 0xFFFFFFFF;
// };

// interrupt_fvector[4] { // Reserved (address : 0xFFFFF90)
//     entry_address  = 0xFFFFFFFF;
// };

```

```
// interrupt_fvector[5] { // Reserved (address : 0xFFFFFFFF96)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[6] { // Reserved (address : 0xFFFFFFFF98)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[7] { // ROM code protect (address : 0xFFFFFFFF9C)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[8] { // ID coce protect (address : 0xFFFFFFFFA0)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[9] { // ID coce protect (address : 0xFFFFFFFFA4)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[10] { // ID coce protect (address : 0xFFFFFFFFA8)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[11] { // ID coce protect (address : 0xFFFFFFFFAC)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[12] { // Reserved (address : 0xFFFFFFFFA0)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[13] { // Reserved (address : 0xFFFFFFFFA4)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[14] { // Reserved (address : 0xFFFFFFFFA8)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[15] { // Reserved (address : 0xFFFFFFFFAC)
//     entry_address = 0xFFFFFFFF;
// };

// interrupt_fvector[30] { // NMI (address : 0xFFFFFFFFF8)
//     entry_address = NMI_handler();
//     pragma_switch = ;
// };

// interrupt_fvector[31] { // Reset (address : 0xFFFFFFFFFC)
//     entry_address = PowerON_Reset_PC();
// };
```

第21章 コンフィギュレータ cfg600px

本章では、コンフィギュレータ cfg600px について解説しています。

21.1 概要

RI600PX が提供している機能を利用したシステム（ロード・モジュール）を構築する場合、RI600PX に提供するデータを保持した情報ファイルが必要となります。

基本的に情報ファイルは、規定された形式のデータ羅列であるため、各種エディタを用いて記述することは可能です。しかし、情報ファイルは、記述性／可読性の面で劣ったものとなっているため、記述に際してはかなりの時間と労力を必要とします。

そこで、RI600PX では、記述性／可読性の面で優れたシステム・コンフィギュレーション・ファイルから情報ファイルへと変換するユーティリティ・ツール（コンフィギュレータ cfg600px）を提供しています。

cfg600px は、システム・コンフィギュレーション・ファイルを入力ファイルとして読み込んだあと、情報ファイルを出力します。

以下に、cfg600px が出力する情報ファイルについて示します。

- システム情報ヘッダ・ファイル (kernel_id.h)
システム・コンフィギュレーション・ファイルに記述されたオブジェクト名（タスク名、セマフォ名など）と ID の対応付けを保持した情報ファイルです。アプリケーションからインクルードします。
- サービス・コール定義ファイル (kernel_sysint.h)
サービス・コールを INT 命令を用いて呼び出すための宣言が記述されたファイルです。kernel.h からインクルードされます。
- ROM 定義ファイル (kernel_rom.h), RAM 定義ファイル (kernel_ram.h)
RI600PX の管理データが記述されたファイルです。これらのファイルは、ブート処理ファイルでのみ、インクルードする必要があります。詳細は、「[17.2 ブート処理ファイル\(ユーザ・OWN・コーディング部\)](#)」を参照してください。
- システム定義ファイル (ri600.inc)
システム定義ファイルは、mkritblpx が出力するテーブル・ファイル (ritable.src) からインクルードされます。
- ベクタ・テーブル・テンプレート・ファイル (vector.tpl)
ベクタ・テーブル・テンプレート・ファイルは、mkritblpx への入力ファイルとなります。
- CMT タイマ定義ファイル (ri_cmt.h)
[基本クロック割り込み情報 \(clock\)](#) の [基本クロック用タイマ・チャネルの選択 \(timer\)](#) に CMT0, CMT, CMT2, CMT3 のいずれかを指定した場合は、[テンプレート・ファイル \(template\)](#) で指定されたテンプレートファイルが環境変数 LIB600 から検索され、ri_cmt.h にリネームされて出力されます。CMT タイマ定義ファイルは、基本クロック用タイマ初期化ルーチンで使用します。詳細は、「[10.9 基本クロック用タイマ初期化ルーチン \(_RI_init_cmt_knl\(\)\)](#)」を参照してください。

21.2 cfg600px の起動

21.2.1 コマンド・ラインからの起動

事前に、環境変数“LIB600”を、“<ri_root>%lib600”に設定する必要があります。

以下に、cfg600px をコマンド・ラインから起動する際の起動方法を示します。

ただし、入力例中の“C>”はコマンド・プロンプトを、“△”はスペース・キーの入力を、“[Enter]”はエンター・キーの入力を表しています。

また、“[]”で囲まれたオプションは、省略可能なオプションであることを表しています。

```
C> cfg600px.exe △ [-U] △ [-v] △ [-V] △ file [Enter]
```

出力ファイルは、カレント・フォルダに生成されます。

以下に、各オプションの詳細を示します。

- -U

未定義割り込みが発生した時にはシステム・ダウンとなりますが、本オプションを指定すると、発生した割り込みのベクタ番号がシステム・ダウン・ルーチンに渡されるようになる（「[第15章 システム・ダウン](#)」を参照）ため、デバッグに役立ちます。ただし、RI600PX のコード・サイズが約 1.5kB 増加します。

- -v

コマンドのオプションの説明と詳細なバージョンを表示します。

- -V

cfg600px が生成するファイルの作成状況を表示します。

- file

cfg600px への入力ファイル名（システム・コンフィギュレーション・ファイル名）を指定します。拡張子を省略した場合は、拡張子“.cfg”を補って解釈します。

備考 <ri_root> は、RI600PX のインストール・フォルダを表しています。

デフォルトでは、“C:%Program Files%Renesas Electronics\CubeSuite+%RI600PX” となります。

21.2.2 CubeSuite+ からの起動

プロパティ パネルの [[システム・コンフィギュレーション・ファイル関連情報](#)] タブで設定した内容に基づき、CubeSuite+ のビルド時に起動されます。

第22章 テーブル生成ユーティリティ mkritblpx

本章では、テーブル生成ユーティリティ mkritblpx について解説しています。

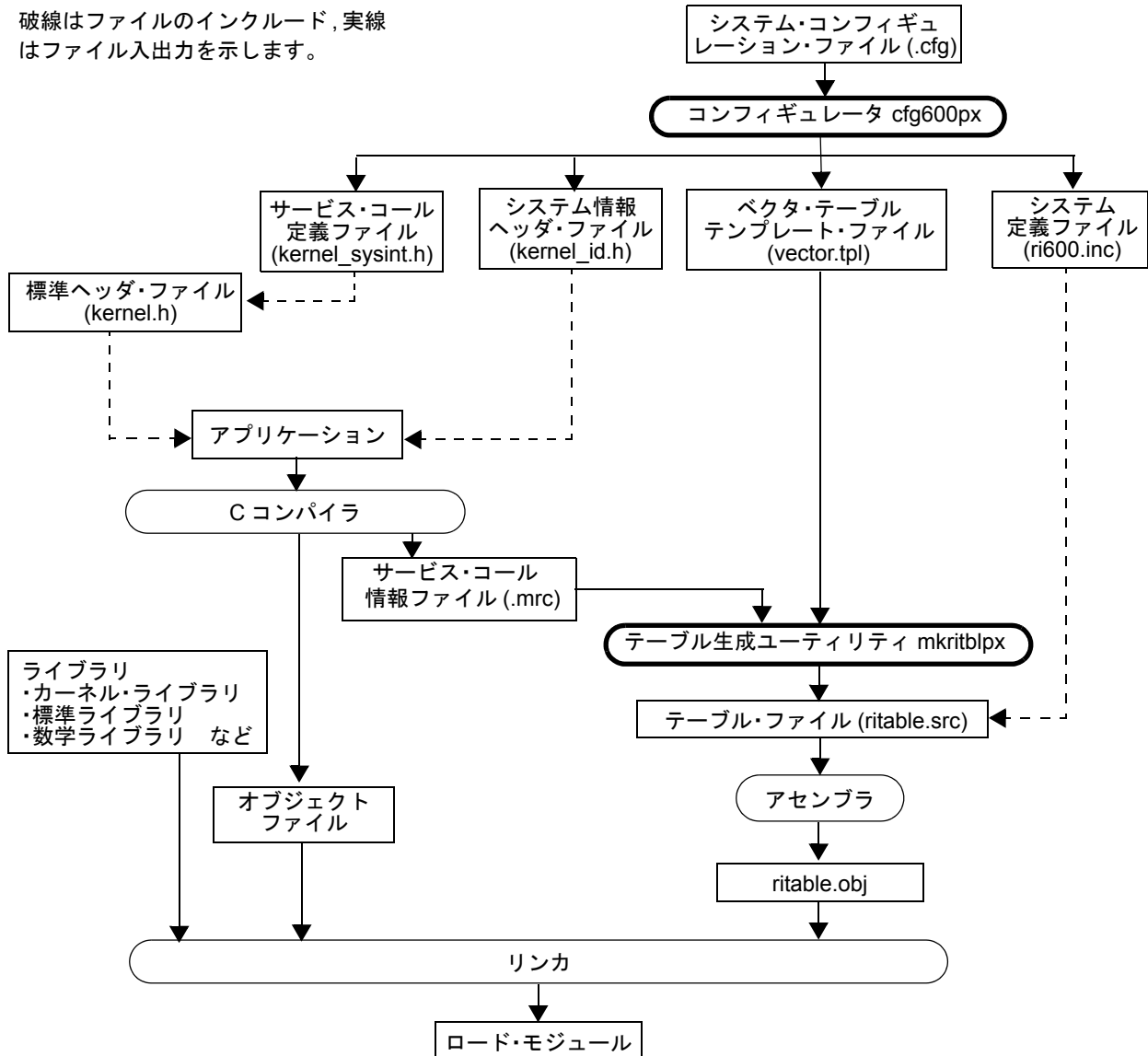
22.1 概要

mkritblpx は、アプリケーションで使用しているサービス・コール情報を収集して、サービス・コール・テーブルと割り込みベクタ・テーブルを生成するコマンド・ライン・ツールです。

アプリケーションのコンパイルによって、アプリケーション内で使用しているサービス・コール情報がサービス・コール情報ファイル (.mrc) に出力されます。mkritblpx は、これらのサービス・コール情報ファイルを入力として、システムで使用するサービス・コールだけがリンクされるようにサービス・コール・テーブルを生成します。

また、mkritblpx は cfg600px が出力したベクタ・テーブル・テンプレート・ファイル (vector.tpl) とサービス・コール情報ファイルを元に、割り込みベクタ・テーブルを生成します。

図 22 - 1 mkritblpx の概要



22.2 mkritblpx の起動

22.2.1 コマンド・ラインからの起動

事前に、環境変数“LIB600”を、“<ri_root>%lib600”に設定する必要があります。

以下に、mkritblpx をコマンド・ラインから起動する際の起動方法を示します。

ただし、入力例中の“C>”はコマンド・プロンプトを、“△”はスペース・キーの入力を、“[Enter]”はエンター・キーの入力を表しています。

また、“[]”で囲まれたオプションは、省略可能なオプションであることを表しています。

```
C> mkritblpx.exe △ [path] [Enter]
```

出力ファイルは、カレント・フォルダに生成されます。

以下に、各オプションの詳細を示します。

- *path*

サービス・コール情報ファイル、またはサービス・コール情報ファイルを検索するフォルダのパスを指定します。なお、フォルダのパスを指定した場合、下位フォルダは検索されません。

mkritblpx は、*path* 指定の有無に関わらず、カレント・フォルダを検索対象とします。

備考 <ri_root> は、RI600PX のインストール・フォルダを表しています。

デフォルトでは、“C:%Program Files%Renesas Electronics%CubeSuite+%RI600PX”となります。

22.2.2 CubeSuite+ からの起動

プロパティ パネルの [システム・コンフィギュレーション・ファイル関連情報] タブで設定した内容に基づき、CubeSuite+ のビルド時に起動されます。

22.3 注意事項

「2.6.1 サービス・コール情報ファイルと“-ri600_preinit_mrc”コンパイラ・オプション」を参照してください。

付録 A ウィンドウ・リファレンス

本付録では、コンフィギュレータ `cfg600px` とテーブル生成ユーティリティ `mkritblpx` の起動オプションを統合開発環境 CubeSuite+ から設定する際に必要となるウィンドウ／パネルについて説明しています。

A.1 説明

以下に、ウィンドウ／パネルの一覧を示します。

表 A - 1 ウィンドウ／パネルの一覧

ウィンドウ／パネル名	機能概要
メイン・ウィンドウ	CubeSuite+ を起動した際、最初にオープンするウィンドウ
プロジェクト・ツリー パネル	プロジェクトの構成要素のツリー表示
プロパティ パネル	プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード、システム・コンフィギュレーション・ファイル等について、詳細情報の表示、および設定の変更

メイン・ウィンドウ

概要

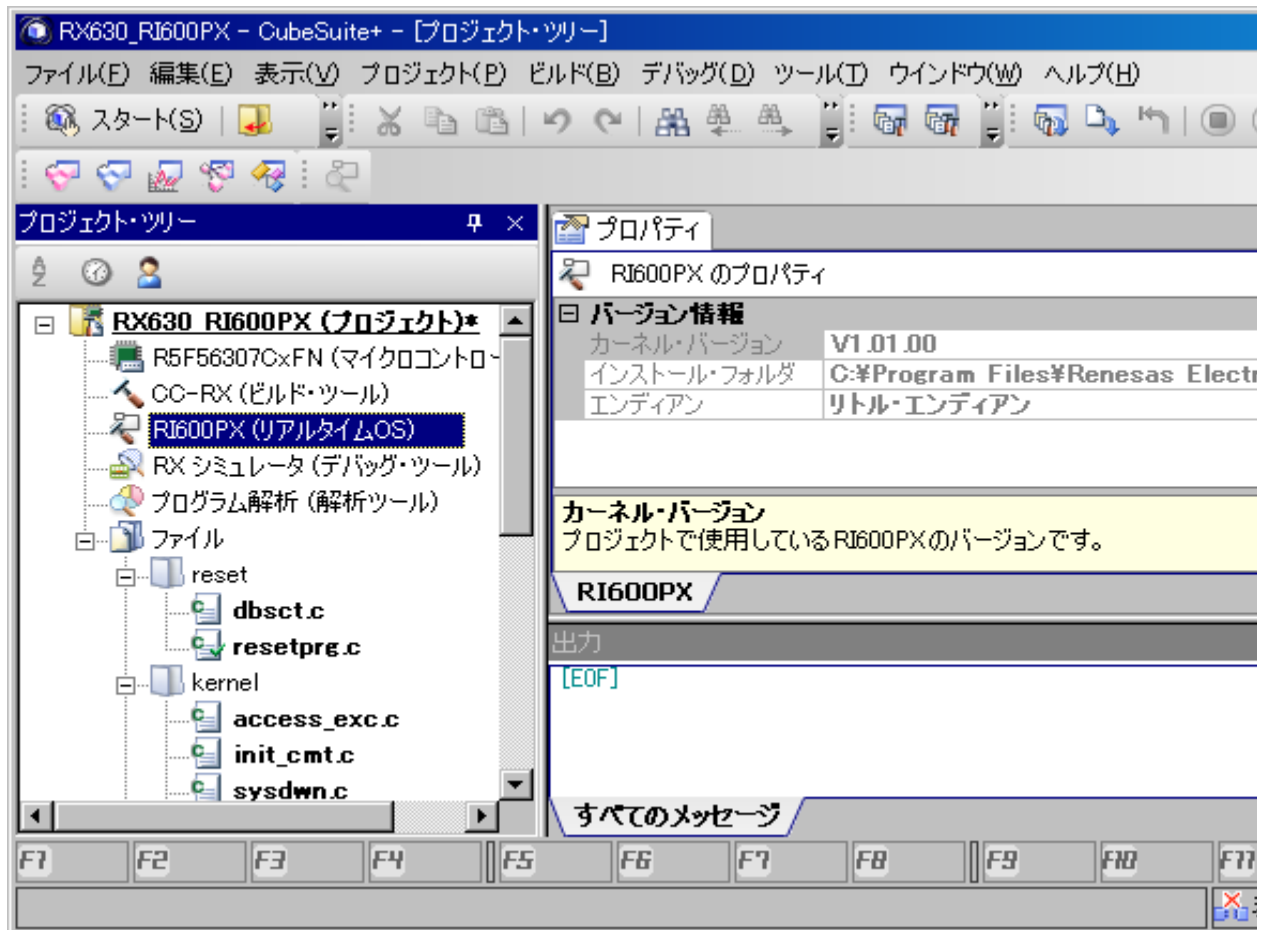
CubeSuite+ を起動した際、最初にオープンするウィンドウです。

ビルドを行う際は、本ウィンドウからユーザ・プログラムの実行制御、および各パネルのオープンを行います。

なお、本ウィンドウは、次の方法でオープンすることができます。

- Windows の [スタート] → [すべてのプログラム] → [Renesas Electronics CubeSuite+] → [CubeSuite+] を選択

表示イメージ



機能

1) メニューバー

リアルタイム OS 関連のメニューを示します。

なお、各メニューから引き出される項目は、ユーザ設定 ダイアログでカスタマイズすることができます。

- [表示]


リアルタイム OS	リアルタイム OS の各ツールを起動するためのカスケード・メニューを表示します。
リソース情報	リアルタイム OS リソース情報 パネルをオープンします。 なお、本メニューは、デバッグ・ツールと切断時は無効となります。

2) ツールバー

リアルタイム OS 関連のボタン群を示します。

なお、ツールバー上のボタンは、ユーザ設定 ダイアログでカスタマイズすることができます。また、同ダイアログにより、新規にツールバーを作成することもできます。

- リアルタイム OS ツールバー

	リアルタイム OS リソース情報 パネルをオープンします。 なお、本ボタンは、デバッグ・ツールと切断時は無効となります。
---	---

3) パネル表示エリア

以下のパネルを表示するエリアです。

- プロジェクト・ツリー パネル
- プロパティ パネル
- 出力 パネル

表示内容の詳細については、各パネルの項を参照してください。

備考 出力 パネルについての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

プロジェクト・ツリーパネル

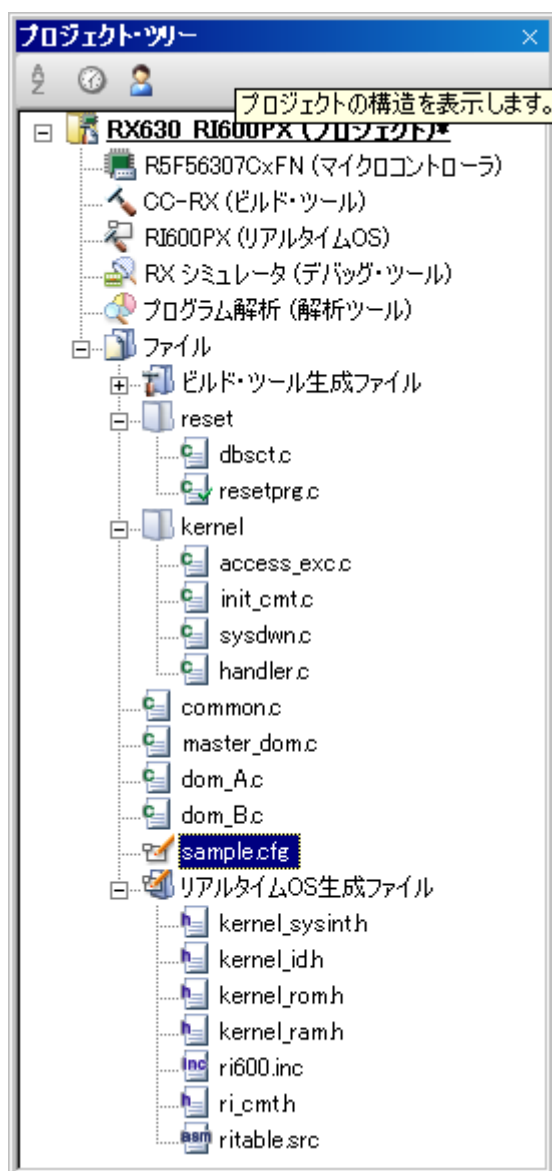
概要

プロジェクトを構成するリアルタイム OS ノード、システム・コンフィギュレーション・ファイル等の構成要素をツリー表示します。

なお、本パネルは、次の方法でオープンすることができます。

- [表示] メニュー→ [プロジェクト・ツリー] を選択

表示イメージ



機能

1) プロジェクト・ツリー エリア

プロジェクトの構成要素を以下のノードでツリー表示します。

ノード	説明
RI600PX (リアルタイム OS) ("リアルタイム OS ノード" と呼びます。)	使用するリアルタイム OS です。
xxx.cfg	システム・コンフィギュレーション・ファイルです。
リアルタイム OS 生成ファイル ("リアルタイム OS 生成ファイル・ノード" と呼びます。)	<p>システム・コンフィギュレーション・ファイル追加時に作成されるノードで、以下の情報ファイルが直下に表示されます。</p> <ul style="list-style-type: none"> - システム情報ヘッダ・ファイル (kernel_id.h) - サービス・コール定義ファイル (kernel_sysint.h) - ROM 定義ファイル (kernel_rom.h) - RAM 定義ファイル (kernel_ram.h) - システム定義ファイル (ri600.inc) - CMT タイマ定義ファイル (ri_cmt.h) - テーブル・ファイル (ritable.src) <p>本ノード、および本ノードに表示されているファイルを直接削除することはできません。 システム・コンフィギュレーション・ファイルをプロジェクトから外した場合、本ノード、および本ノードに表示されているファイルは表示されなくなります。</p>

コンテキスト・メニュー

- 1) リアルタイム OS ノード, リアルタイム OS 生成ファイル・ノードを選択している場合

プロパティ	選択しているノードのプロパティを プロパティ パネル に表示します。
-------	---

- 2) システム・コンフィギュレーション・ファイル, 情報ファイルを選択している場合

アセンブル	選択しているアセンブラ・ソース・ファイルをアセンブルします。 なお、本メニューは、システム情報テーブル・ファイル, エントリ・ファイルを選択している場合のみ表示されます。 ただし、ビルド・ツールが実行中の場合は無効となります。
開く	ファイルの拡張子に割り当てられたアプリケーションで選択しているファイルを開きます。 なお、本メニューは、複数のファイルを選択している場合は無効となります。
内部エディタで開く ...	エディタ パネルで選択しているファイルを開きます。 なお、本メニューは、複数のファイルを選択している場合は無効となります。
アプリケーションを指定して開く ...	プログラムから開く ダイアログを開き、指定したアプリケーションで選択しているファイルを開きます。 ただし、複数のファイルを選択している場合は無効となります。
エクスプローラでフォルダを開く	選択しているファイルが存在しているフォルダをエクスプローラで開きます。
追加	プロジェクトにファイル, カテゴリ・ノードを追加するためのカスケード・メニューを表示します。
既存のファイルを追加 ...	既存のファイルを追加 ダイアログを開き、選択したファイルをプロジェクトに追加します。
新しいファイルを追加 ...	ファイル追加 ダイアログを開き、選択した種類でファイルを作成し、プロジェクトに追加します。
新しいカテゴリを追加	選択しているファイルと同じレベルにカテゴリ・ノードを追加し、カテゴリ名が編集可能な状態になります。 なお、本メニューは、ビルド・ツールが実行中の場合、およびカテゴリのネスト数が20の場合は無効となります。
プロジェクトから外す	選択しているファイルをプロジェクトから外します。 ファイル自体はファイル・システム上からは削除されません。 なお、本メニューは、ビルド・ツールが実行中の場合は無効となります。
コピー	選択しているファイルをクリップ・ボードにコピーします。 ファイル名を編集の場合は、選択している文字列をクリップ・ボードにコピーします。
貼り付け	本メニューは常に無効です。
名前の変更	選択しているファイルの名前が編集可能な状態になります。 実際のファイル名も変更されます。
プロパティ	選択しているファイルのプロパティを プロパティ パネル に表示します。

プロパティ パネル

概要

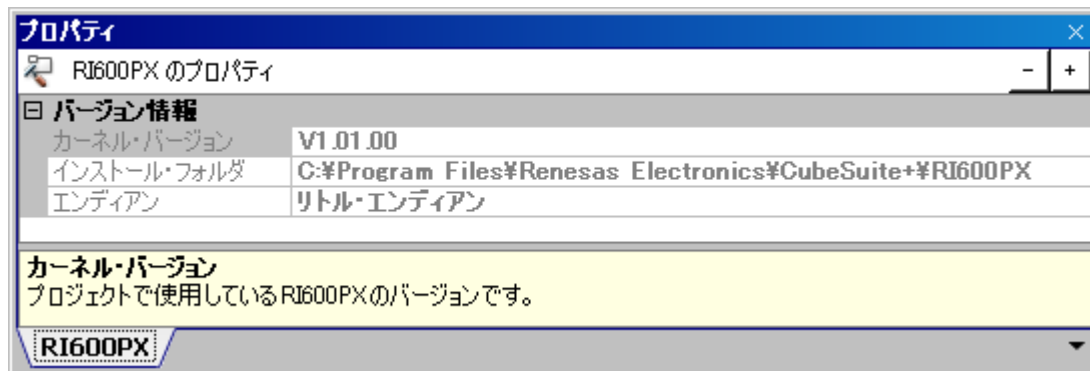
プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード、システム・コンフィギュレーション・ファイル等について、カテゴリ別に詳細情報の表示、および設定の変更を行います。

なお、本パネルは、次の方法でオープンすることができます。

- プロジェクト・ツリー パネル上において、リアルタイム OS ノード、システム・コンフィギュレーション・ファイル等を選択したのち、[表示] メニュー→ [プロパティ] を選択、またはコンテキスト・メニュー→ [プロパティ] を選択

備考 すでにプロパティ パネルがオープンしている場合、プロジェクト・ツリー パネル上において、リアルタイム OS ノード、システム・コンフィギュレーション・ファイル等を選択することで、選択した項目の詳細情報を表示します。

表示イメージ



機能

1) 対象名エリア

プロジェクト・ツリー パネルで選択しているノードの名称を表示します。
複数のノードを選択している場合、本エリアは空欄となります。

2) 詳細情報表示／変更エリア

プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード、システム・コンフィギュレーション・ファイル等の詳細情報を、カテゴリ別のリスト形式で表示し、設定の変更を直接行うことができるエリアです。

☐マークは、そのカテゴリ内に含まれているすべての項目が展開表示されていることを示し、また、☒マークは、カテゴリ内の項目が折りたたみ表示されていることを示します。展開／折りたたみ表示の切り替えは、このマークのクリック、またはカテゴリ名のダブルクリックにより行うことができます。

カテゴリ、およびそれに含まれる項目の表示内容／設定方法についての詳細は、該当するタブの項を参照してください。

3) プロパティの説明エリア

詳細情報表示／変更エリアで選択したカテゴリや項目の簡単な説明を表示します。

4) タブ選択エリア

タブを選択することにより、詳細情報を表示するカテゴリが切り替わります。

本パネルには、次のタブが存在します（各タブ上における表示内容／設定方法についての詳細は、該当するタブの項を参照してください）。

- プロジェクト・ツリー パネルでリアルタイム OS ノードを選択している場合
 - [RI600PX] タブ
- プロジェクト・ツリー パネルでシステム・コンフィギュレーション・ファイルを選択している場合
 - [システム・コンフィギュレーション・ファイル関連情報] タブ
 - [ファイル情報] タブ
- プロジェクト・ツリー パネルでリアルタイム OS 生成ファイル・ノードを選択している場合
 - [カテゴリ情報] タブ
- プロジェクト・ツリー パネルでシステム情報テーブル・ファイル、エン트리・ファイルを選択している場合
 - [ビルド設定] タブ
 - [個別アセンブル・オプション] タブ
 - [ファイル情報] タブ
- プロジェクト・ツリー パネルでシステム情報ヘッダ・ファイルを選択している場合
 - [ファイル情報] タブ

備考 1 [ファイル情報] タブ、[カテゴリ情報] タブ、[ビルド設定] タブ、[個別アセンブル・オプション] タブについての詳細は、「CubeSuite+ 統合開発環境 ユーザーズマニュアル RX ビルド編」を参照してください。

備考 2 プロジェクト・ツリー パネルで複数の構成要素を選択している場合は、その構成要素に共通するタブのみ表示されます。プロパティの値の変更は、選択している複数の構成要素に共通に反映されます。

[編集] メニュー（プロパティ パネル専用部分）

元に戻す	直前に行ったプロパティの値の編集作業を取り消します。
切り取り	プロパティの値を編集中の場合、選択している文字列を切り取ってクリップ・ボードに移動します。
コピー	選択しているプロパティの値文字列をクリップ・ボードにコピーします。
貼り付け	プロパティの値を編集中の場合、クリップ・ボードの内容を挿入します。
削除	プロパティの値を編集中の場合、選択している文字列を削除します。
すべて選択	プロパティの値を編集中の場合、選択しているプロパティの値文字列をすべて選択します。

コンテキスト・メニュー

元に戻す	直前に行ったプロパティの値の編集作業を取り消します。
切り取り	プロパティの値を編集中の場合、選択している文字列を切り取ってクリップ・ボードに移動します。

コピー	選択しているプロパティの値文字列をクリップ・ボードにコピーします。
貼り付け	プロパティの値を編集の場合、クリップ・ボードの内容を挿入します。
削除	プロパティの値を編集の場合、選択している文字列を削除します。
すべて選択	プロパティの値を編集の場合、選択しているプロパティの値文字列をすべて選択します。
デフォルトに戻す	選択している項目の設定値をプロジェクトに設定しているデフォルト値に戻します。 ただし、[個別アセンブル・オプション] タブにおいては、全体オプションの設定値に戻します。
すべてデフォルトに戻す	現在表示しているタブの設定値をすべてプロジェクトに設定しているデフォルト値に戻します。 ただし、[個別アセンブル・オプション] タブにおいては、全体オプションの設定値に戻します。

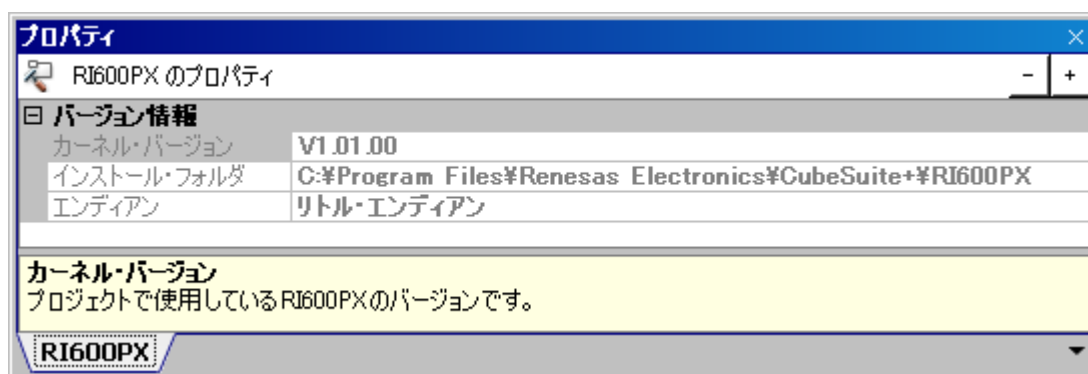
[RI600PX] タブ

概要

本タブでは、使用する RI600PX に対して、次に示すカテゴリごとに詳細情報の表示を行います。

- バージョン情報

表示イメージ



機能

1) [バージョン情報]

RI600PX のバージョンに関する詳細情報の表示を行います。

カーネル・バージョン	使用する RI600PX のバージョンを表示します。	
	デフォルト	インストールしている RI600PX のバージョン
	変更方法	変更不可
インストール・フォルダ	使用する RI600PX がインストールされているフォルダを絶対パスで表示します。	
	デフォルト	使用する RI600PX がインストールされているフォルダ
	変更方法	変更不可
エンディアン	プロジェクトで設定しているエンディアンを表示します。 ビルド・ツールの [エンディアンの選択] プロパティで選択しているレジスタ・モードと同じ値が表示されます。	
	デフォルト	ビルド・ツールでのプロパティで選択しているエンディアン
	変更方法	変更不可

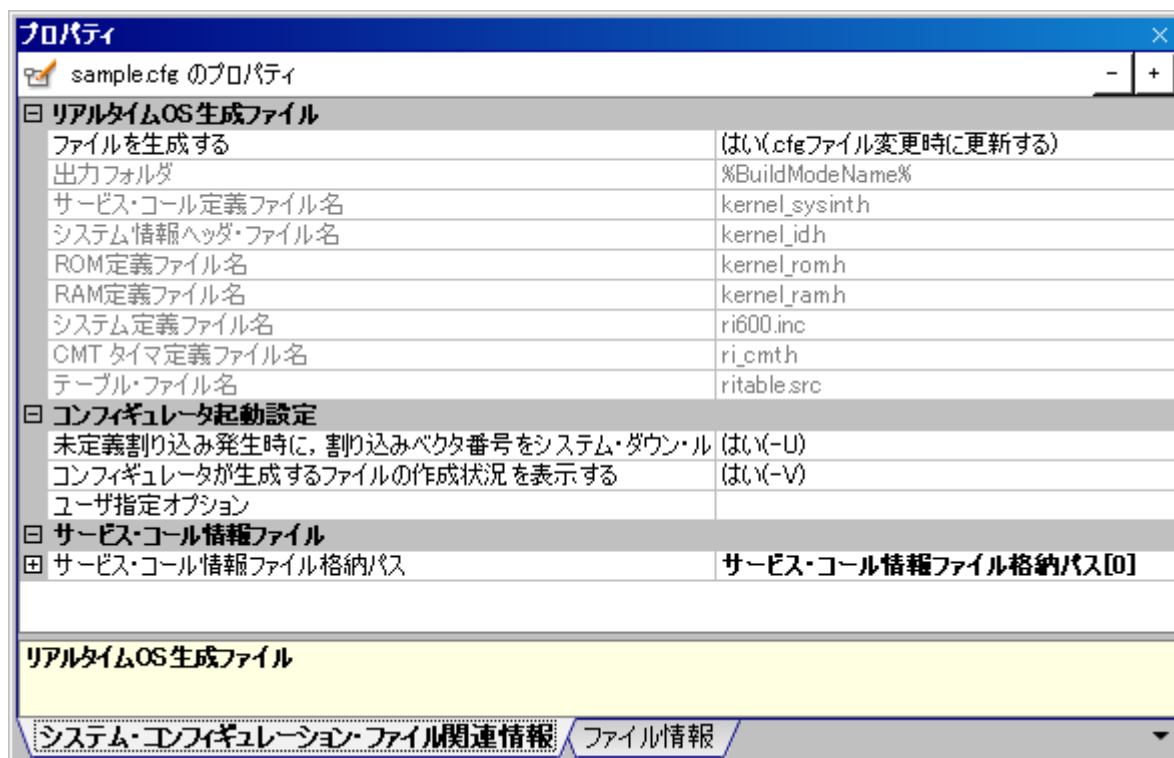
[システム・コンフィギュレーション・ファイル関連情報] タブ

概要

本タブでは、使用するシステム・コンフィギュレーション・ファイルに対して、次に示すカテゴリごとに詳細情報の表示、および設定の変更を行います。

- リアルタイム OS 生成ファイル
- コンフィギュレータ起動設定
- サービス・コール情報ファイル

表示イメージ



機能

- 1) [リアルタイム OS 生成ファイル]

RI600PX 生成ファイルに関する詳細情報の表示、および設定の変更を行います。

ファイルを生成する	リアルタイム OS 生成ファイルを生成するかどうか、およびシステム・コンフィギュレーション・ファイルを変更した場合にリアルタイム OS 生成ファイルを更新するかどうかを選択します。				
	デフォルト	はい (.cfg ファイル変更時に更新する)			
	変更方法	ドロップダウン・リストによる選択			
	指定可能値	<table border="1"> <tr> <td>はい (.cfg ファイル変更時に更新する)</td> <td>リアルタイム OS 生成ファイルを新規生成し、プロジェクト・ツリーに表示します。 リアルタイム OS 生成ファイルがすでに存在するときにシステム・コンフィギュレーション・ファイルを変更した場合は、リアルタイム OS 生成ファイルを更新します。</td> </tr> <tr> <td>いいえ (プロジェクトに登録しない)</td> <td>リアルタイム OS 生成ファイルの生成を行わず、プロジェクト・ツリーにも表示しません。 システム情報テーブル・ファイルが存在するときに本項目を選択しても、ファイル自体の削除は行いません。</td> </tr> </table>	はい (.cfg ファイル変更時に更新する)	リアルタイム OS 生成ファイルを新規生成し、プロジェクト・ツリーに表示します。 リアルタイム OS 生成ファイルがすでに存在するときにシステム・コンフィギュレーション・ファイルを変更した場合は、リアルタイム OS 生成ファイルを更新します。	いいえ (プロジェクトに登録しない)
はい (.cfg ファイル変更時に更新する)	リアルタイム OS 生成ファイルを新規生成し、プロジェクト・ツリーに表示します。 リアルタイム OS 生成ファイルがすでに存在するときにシステム・コンフィギュレーション・ファイルを変更した場合は、リアルタイム OS 生成ファイルを更新します。				
いいえ (プロジェクトに登録しない)	リアルタイム OS 生成ファイルの生成を行わず、プロジェクト・ツリーにも表示しません。 システム情報テーブル・ファイルが存在するときに本項目を選択しても、ファイル自体の削除は行いません。				
出力フォルダ	リアルタイム OS 生成ファイルを出力するフォルダを表示します。				
	デフォルト	%BuildModeName%			
	変更方法	変更不可			
サービス・コール定義ファイル名	cfg600px が生成するサービス・コール定義ファイル名を表示します。				
	デフォルト	kernel_sysint.h			
	変更方法	変更不可			
システム情報ヘッダ・ファイル名	cfg600px が生成するシステム情報ヘッダ・ファイル名を表示します。				
	デフォルト	kernel_id.h			
	変更方法	変更不可			
ROM 定義ファイル	cfg600px が生成する ROM 定義ファイル名を表示します。				
	デフォルト	kernel_rom.h			
	変更方法	変更不可			
RAM 定義ファイル	cfg600px が生成する RAM 定義ファイル名を表示します。				
	デフォルト	kernel_ram.h			
	変更方法	変更不可			
システム定義ファイル	cfg600px が生成するシステム定義ファイル名を表示します。				
	デフォルト	ri600.inc			
	変更方法	変更不可			
CMT タイマ定義ファイル名	cfg600px が生成する CMT タイマ定義ファイル名を表示します。				
	デフォルト	ri_cmt.h			
	変更方法	変更不可			
テーブル・ファイル名	mkritblpx が生成するテーブル・ファイル名を表示します。				
	デフォルト	ritable.src			
	変更方法	変更不可			

2) [コンフィギュレータ起動設定]

コンフィギュレータ `cfg600px` の起動オプションを設定します。

未定義割り込み発生時に、割り込みベクタ番号をシステム・ダウン・ルーチンに渡す	未定義割り込みが発生した時にはシステム・ダウンとなりますが、“はい”を選択すると、発生した割り込みのベクタ番号がシステム・ダウン・ルーチンに渡されるようになるため、デバッグに役立ちます。ただし、カーネルのコードサイズが約 1.5kB 増加します。 システム・ダウンについては、「第 15 章 システム・ダウン」を参照してください。	
	デフォルト	はい (-U)
	変更方法	ドロップダウン・リストによる選択
	指定可能値	はい (-U) 未定義割り込み発生時に、割り込みベクタ番号をシステム・ダウン・ルーチンに渡します。 いいえ 未定義割り込み発生時に、割り込みベクタ番号をシステム・ダウン・ルーチンに渡しません。
コンフィギュレータが生成するファイルの作成状況を表示する	cfg600px が生成するファイルの作成状況を表示するかどうかを設定します。	
	デフォルト	はい (-V)
	変更方法	ドロップダウン・リストによる選択
	指定可能値	はい (-V) コンフィギュレータが生成するファイルの作成状況を表示します。 いいえ コンフィギュレータが生成するファイルの作成状況を表示しません。
ユーザ指定オプション	cfg600px に渡すユーザ任意のオプションを設定します。	
	デフォルト	—
	変更方法	テキスト・ボックスによる直接入力
	指定可能値	259 文字までの文字列

3) [サービス・コール情報ファイル]

テーブル生成ユーティリティ `mkritblpx` に入力するサービス・コール情報ファイルの検索パスを設定します。アプリケーションのコンパイルによって生成されるサービス・コール情報ファイルが格納されたフォルダを、漏れなく指定してください。

サービス・コール情報ファイル格納パス	アプリケーションのコンパイルによって生成されるサービス・コール情報ファイル (.mrc) が格納されたフォルダのパス、およびファイル名を設定します。 フォルダ・パスを指定した場合、そのサブ・フォルダは検索対象にはなりません。相対パスで指定した場合は、プロジェクト・フォルダを基点とします。絶対パスで指定した場合は、プロジェクト・フォルダを基点とした相対パスに変換されます（ドライブが異なる場合を除く）。 複数のパスを指定する場合は、改行で区切ります。 なお、本指定に関わらず、プロジェクト・フォルダのパスは <code>mkritblpx</code> に渡されます。 以下のプレース・ホルダを使用できます。 %BuildModeName% : ビルド・モード名に置換します。	
	変更方法	[...] ボタンをクリックし、パス編集ダイアログによる編集
	指定可能値	259 文字までの文字列 なお、拡張子が指定されていない場合、および “.mrc” 以外の拡張子を指定した場合は、フォルダとして扱います。

- 備考 1 サービス・コール情報ファイルについては、「[2.6.1 サービス・コール情報ファイルと“-ri600_preinit_mrc”コンパイラ・オプション](#)」を参照してください。
- 備考 2 コンパイラの外部変数アクセス最適化オプションを使用する場合、CubeSuite+ は 1 回目のコンパイルによって生成されるオブジェクト・ファイルやサービス・コール情報ファイルを格納するフォルダを自動的に生成し、そのフォルダを暗黙的に検索パスに設定します。
- 備考 3 サービス・コール情報ファイルは、コンパイル時にオブジェクト・ファイルと同じフォルダに出力されます。オブジェクト・ファイルの出力フォルダを変更する操作を行った場合は、本項目を適切に変更してください。

付録 B 浮動小数点演算機能

コンパイラが浮動小数点演算命令を出力するのは、`-fpu` オプション指定時のみです。アセンブラで `-chkfpu` オプションを指定すると、浮動小数点演算命令の記述を Warning として検出します。

B.1 タスクおよびタスク例外処理ルーチンで浮動小数点演算命令を使用する場合

システム情報 (system) のタスク・コンテキスト・レジスタ (context) に、“FPSW” を含む設定を行ってください。これにより、FPSW レジスタはタスク毎に独立に管理されるようになります。

タスクおよびタスク例外処理ルーチン起動時の FPSW レジスタは、使用するコンパイラ・オプションに応じた値に初期化されます。詳細は、「4.2.4 タスク内での処理」および「6.2.2 タスク例外処理ルーチン内での処理」を参照してください。

B.2 ハンドラで浮動小数点演算命令を使用する場合

ハンドラは、明示的に FPSW レジスタを保証する必要があります。

また、ハンドラ起動時の FPSW レジスタは不定です。以下のようにして、FPSW レジスタの保証と初期化を行ってください。

```
#include <machine.h> // コンパイラ提供の組み込み関数 get_fpsw(), set_fpsw() を使用する
// ために、machine.h をインクルード

#include "kernel.h"
#include "kernel_id.h"
void handler (void)
{
    unsigned long old_fpsw; // FPSW レジスタを退避するための変数を宣言
    old_fpsw = get_fpsw (); // FPSW レジスタを退避
    set_fpsw (0x00000100); // 必要なら FPSW を初期化
    /* 浮動小数点演算処理 */
    set_fpsw (old_fpsw); // FPSW レジスタを復帰
}
```

付録C DSP機能

DSP機能をサポートしたMCUを使用する場合は、ACCレジスタ（アキュムレータ）の扱いについて注意が必要です。具体的には、ACCレジスタを更新するDSP機能命令（下記）を使用する場合は、後述について留意してください。

- RXv1 / RXv2 アーキテクチャ共通命令
MACHI, MACLO, MULHI, MULLO, RACW, MVTACHI, MVTACLO
- RXv2 アーキテクチャ命令
EMACA, EMSBA, EMULA, MACLH, MSBHI, MSBLH, MSBLO, MULLH, MVTACGU, RACL, RDAACL, RDACW

なお、コンパイラがこれらの命令を生成することはありません。また、アセンブラで `-chkdsp` オプションを指定すると、DSP機能命令の記述を Warning として検出します。

C.1 タスクおよびタスク例外処理ルーチンでDSP機能命令を使用する場合

システム情報（system）のタスク・コンテキスト・レジスタ（context）に、“ACC”を含む設定を行ってください。これにより、ACCレジスタはタスク毎に独立に管理されるようになります。

C.2 割り込みハンドラでのACCレジスタの保証

アプリケーション内に前述のDSP機能命令を使用するタスクおよび割り込みハンドラがひとつでもある場合は、すべての割り込みハンドラがACCレジスタを保証する必要があります。その方法として、以下の2つがあります。また、表 20-9 も参照してください。

- 1) コンパイラ・オプション `-save_acc` を使用する。
- 2) すべての割り込みハンドラの定義（可変ベクタ情報（`interrupt_vector[]`）、および固定ベクタ／例外ベクタ情報（`interrupt_fvector[]`））際に、`pragma_switch` に“ACC”を指定する。

付録 D スタック使用量の算出

スタックがオーバーフローすると、システムの動作は不定となるので、本章を参考にスタックがオーバーフローしないようにしてください。

D.1 スタックの種類

スタックには、ユーザ・スタックとシステム・スタックの2種類があります。スタック使用量の算出方法は、ユーザ・スタックとシステム・スタックで異なります。

- ユーザ・スタック

タスクのスタックを、ユーザ・スタックと呼びます。

システム・コンフィギュレーション・ファイルの[タスク情報 \(task\[\]\)](#) でタスクを生成するときには、サイズとユーザ・スタック領域を割り当てるセクションを指定します。

[cre_tsk](#) または [acre_tsk](#) でタスクを生成するときには、サイズとユーザ・スタック領域の先頭アドレスを指定します。

- システム・スタック

各種ハンドラとカーネルが共通に使用するスタックで、システムにひとつだけ存在します。システム・スタック・サイズは、[システム情報 \(system\)](#) の[システム・スタック・サイズ \(stack_size\)](#) に指定します。システム・スタックのセクション名はSIです。

D.2 Call Walker

CubeSuite+ には、スタック算出ユーティリティである Call Walker が付属しています。Call Walker を使用すると、各関数ツリーで消費されるスタックサイズを確認することができます。

D.3 ユーザ・スタック使用量の算出

各タスクのユーザ・スタックの使用量は、以下の式で算出された値を 16 の倍数に切り上げた値です。

$$\text{ユーザ・スタックの使用量} = \text{treesz_task} + \text{ctxtsz_task} + \text{treesz_tex} + \text{ctxtsz_tex}$$

- *treesz_task*

タスク開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。

- *treesz_tex*

タスク例外処理ルーチン開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。タスク例外処理ルーチンを使用しない場合は、*treesz_tex* は 0 です。

- *ctxtsz_task*, *ctxsz_tex*

タスクのコンテキスト・レジスタのサイズです。*ctxtsz_task* はタスク用、*ctxsz_tex* はタスク例外処理ルーチン用です。タスク例外処理ルーチンを使用しない場合は、*ctxsz_tex* は 0 です。

タスクのコンテキスト・レジスタのサイズは、システム情報 (system) のタスク・コンテキスト・レジスタ (context) の設定によって異なります。表 D-1 を参照してください。

表 D-1 タスク・コンテキスト・レジスタ・サイズ

system.context 設定	コンパイラ・オプション“-isa”	タスク・コンテキスト・レジスタ・サイズ (バイト)
NO	—	68
FPSW	—	72
ACC	“-isa=rxv2”	92
	“-isa=rxv1” またはオプション“-isa”の指定なし	76
FPSW,ACC	“-isa=rxv2”	96
	“-isa=rxv1” またはオプション“-isa”の指定なし	80
MIN	—	44
MIN,FPSW	—	48
MIN,ACC	“-isa=rxv2”	68
	“-isa=rxv1” またはオプション“-isa”の指定なし	52
MIN,FPSW,ACC	“-isa=rxv2”	72
	“-isa=rxv1” またはオプション“-isa”の指定なし	56

D.4 システム・スタック使用量の算出

システム・スタックを最も多く消費するのは、サービス・コール処理中に割り込みが発生、さらに多重割り込みが発生した場合です。すなわち、システム・スタックの必要量（最大サイズ）は以下の計算式で算出することができます。

$$\begin{aligned} \text{システム・スタックの使用量} = & \text{svcsz} \\ & + \sum_{k=1}^{15} \text{inthdrsz}_k \\ & + \text{sysdwnsz} \end{aligned}$$

- svcsz

すべての処理プログラムで使用しているサービス・コールの中での最大の使用量です。svcsz の値は RI600PX のバージョンによって異なります。製品添付のリリースノートを参照してください。

- inthdrsz

各割り込みハンドラ開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。 k は、割り込み優先レベルです。同じ優先レベルの割り込みが複数ある場合は、それらのハンドラの中で最大のサイズを inthdrsz_k としてください。

なお、基本クロック用割り込みハンドラ（割り込み優先レベルは基本クロック割り込み情報（clock）の基本クロック割り込み優先レベル（IPL）で指定します）の使用量は、以下の 3 つのサイズの最大値となります。clocksz1, clocksz2, clocksz3 については、リリースノートを参照してください。

なお、基本クロック用タイマを使用しない場合（clock.timer=NOTIMER）は、基本クロック用割り込みハンドラが使用するサイズを加算する必要はありません。

- clocksz1 + cycsz
- clocksz2 + almsz
- clocksz3

- cycsz

周期ハンドラ開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。周期ハンドラが複数ある場合は、それらのハンドラの中で最大のサイズを cycsz としてください。

- almsz

アラーム・ハンドラ開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）です。アラーム・ハンドラが複数ある場合は、それらのハンドラの中で最大のサイズを almsz としてください。

- sysdwnsz

システム・ダウン・ルーチン開始関数を起点とする関数ツリーで消費されるサイズ（Call Walker 表示サイズ）+ 40 としてください。システム・ダウン・ルーチンに遷移するケースがない場合は、sysdwnsz を 0 として計算してください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2012.04.01	—	初版発行
1.01	2013.09.20 (RI600PX V1.02.00)	22	「2.6.2 ブート処理ファイルのコンパイラ・オプション」の説明を詳細化
		23	RXv2 アーキテクチャのサポートに伴い、カーネル・ライブラリの構成を変更
		26, 217, 225, 529 など	RXv2 アーキテクチャのサポートに伴い、FIX_INTERRUPT_VECTOR セクションおよび EXTB レジスタに関する説明を追加・変更。 また、全般に「固定ベクタ」を「固定ベクタ／例外ベクタ」に置換。
		27	「2.6.5 初期化データ・セクション」を追加
		41, 66, 559	タスクおよびタスク例外処理ルーチン処理開始時の FPSW の仕様を変更
		218	RXv2 アーキテクチャのサポートに伴い、“-isa”, “-cpu” オプションについて追記
		221, 460	RI600PX 起動に関する説明を改善
		222	「17.4 セクション初期化関数 (_INITSCT())」の説明を改善
		230	RI600PX V1.02.00 へのリビジョン・アップに伴い、TKERNEL_PRVER の定義値を 0x0120 に変更
		489	RXv2 アーキテクチャのサポートに伴い、表 20 - 2 を変更
		493	表 20 - 7 の説明を改善
		505	RXv2 アーキテクチャのサポートに伴い、表 20 - 8 を変更
		560	ACC を更新する命令に、RXv2 アーキテクチャ専用命令を追記
		560	すべての割り込みハンドラが明示的に ACC を保証する方法の記載を削除
562	RXv2 アーキテクチャのサポートに伴い、表 D - 1 を変更		

RI600PX リアルタイム・オペレーティング・システム
ユーザーズマニュアル コーディング編

発行年月日 2012年4月1日 Rev.1.00
2013年9月20日 Rev.1.01

発行 ルネサス エレクトロニクス株式会社
〒211-8668 神奈川県川崎市中原区下沼部 1753



ルネサスエレクトロニクス株式会社

■営業お問合せ窓口

<http://www.renesas.com>

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社 〒100-0004 千代田区大手町2-6-2 (日本ビル)

(03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：<http://japan.renesas.com/contact/>

RI600PX

RENESAS

ルネサスエレクトロニクス株式会社

R20UT0964JJ0101