お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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。お客様の機器の設計において、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。
5. 輸出に際しては、「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。本資料に記載されている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の目的で使用しないでください。また、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に使用することができません。
6. 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ものですが、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。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されてい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生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   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、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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が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。

User's Manual

RENESAS

M3T-MR32R/4 V.4.00

リファレンスマニュアル

M32R ファミリ用リアルタイムOS

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2004.12

- Microsoft、MS-DOS、Windows**および**Windows NT**は、米国**Microsoft Corporation**の米国およびその他の国における登録商標です。**
- HP-UX**は、米国**Hewlett-Packard Company**のオペレーティングシステムの名称です。**
- Sun 、Java **およびすべての**Java**関連の商標およびロゴは、米国およびその他の国における米国**Sun Microsystems, Inc.**の商標または登録商標です。**
- UNIX**は、**X/Open Company Limited**が独占的にライセンスしている米国ならびに他の国における登録商標です。**
- IBM**および**AT**は、米国**International Business Machines Corporation**の登録商標です。**
- HP 9000**は、米国**Hewlett-Packard Company**の商品名称です。**
- SPARC**および**SPARCstation**は、米国**SPARC International, Inc.**の登録商標です。**
- Intel, Pentium**は、米国**Intel Corporation**の登録商標です。**
- Adobe**および**Acrobat**は、**Adobe Systems Incorporated**（アドビシステムズ社）の登録商標です。**
- Netscape**および**Netscape Navigator**は、米国およびその他の諸国の**Netscape Communications Corporation**社の登録商標です。**
- **その他すべてのブランド名および製品名は個々の所有者の登録商標もしくは商標です。**

安全設計に関するお願い

- **弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。**

本資料ご利用に際しての留意事項

- **本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。**
- **本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは責任を負いません。**
- **本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、予告なしに、本資料に記載した製品又は仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前に株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（**http://www.renesas.com**）などを通じて公開される情報に常にご注意ください。**
- **本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズはその責任を負いません。**
- **本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。**
- **本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へご照会ください。**
- **本資料の転載、複製については、文書による株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズの事前の承諾が必要です。**
- **本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店までご照会ください。**

製品の内容及び本書についてのお問い合わせ先

インストーラが生成する以下のテキストファイルに必要事項を記入の上、ツール技術サポート窓口 support_tool@renesas.com **まで送信ください。**

¥SUPPORT¥**製品名**¥SUPPORT.TXT

株式会社ルネサス ソリューションズ
ツール技術サポート窓口 support_tool@renesas.com
ユーザ登録窓口 regist_tool@renesas.com
ホームページ http://www.renesas.com/jp/tools

はじめに

M3T-MR32R/4(**以下** MR32R **と略す**)は M32R ファミリ用のリアルタイム・オペレーティングシステム[1]です。MR32R はμITRON 仕様[2]に準拠しています。

本マニュアルは MR32R を使用したプログラムの作成手順および作成上の注意事項について説明します。各サービスコールの詳細な使用方法については「MR32R リファレンスマニュアル」を参照してください。

MR32R を使うために必要なこと

MR32R を使用したプログラムを作成するには弊社下記製品を別途御購入して頂く必要があります。

- M32R ファミリクロスツール M3T-CC32R(以下 CC32R と略す)

ドキュメント一覧

MR32R に添付されているドキュメントは以下の３種類あります。

- リリースノート
  ソフトウェアの概要やユーザーズマニュアル、リファレンスマニュアルの訂正などを記載したドキュメントです。
- ユーザーズマニュアル (PDF ファイル)
  MR32R を使用したプログラムの作成手順や作成上の注意事項を記載したドキュメントです。
- リファレンスマニュアル (PDF ファイル)
  MR32R のサービスコールの使用方法や使用例を記述したドキュメントです。
  本マニュアルを読む前に必ずリリースノートをお読みください。

ソフトウェアの使用権

ソフトウェアの使用権はソフトウェア使用権許諾契約書に基づきます。MR32R はお客様の製品開発の目的でのみ使用できます。その他の目的での使用はできませんのでご注意ください。

また、本マニュアルによってソフトウェアの使用権の実施に対する保証及び使用権の実施の許諾を行うものではありません。

---

[1] 以降リアルタイム OS と略します。

[2] μITRON 仕様は、東京大学理学部坂村健博士とその研究室により考案されたものです。したがって、μITRON 仕様の著作権は同氏に属しています。**MR32R** は同氏に承認を得て、μITRON 仕様に基づき製作されたものです。

目次

第 1 章　サービスコールリファレンス　1

1.1.　サービスコールリファレンスの見方　2

サービスコール名　内容(種別)　2

1.2.　タスク管理機能　3

cre_tsk　タスクの生成　3

acre_tsk　タスクの生成(ID 自動割り当て)　3

del_tsk　タスクの削除　6

act_tsk　タスクの起動　8

iact_tsk　タスクの起動(ハンドラ専用)　8

can_act　起動要求カウントのキャンセル　10

ican_act　起動要求カウントのキャンセル(ハンドラ専用)　10

sta_tsk　タスクの起動(起動コード指定)　12

ista_tsk　タスクの起動(起動コード指定,ハンドラ専用)　12

ext_tsk　自タスクの終了　14

exd_tsk　自タスクの終了および削除　16

ter_tsk　タスクの強制終了　17

chg_pri　タスク優先度の変更　18

ichg_pri　タスク優先度の変更(ハンドラ専用)　18

get_pri　タスク優先度の参照　20

iget_pri　タスク優先度の参照(ハンドラ専用)　20

ref_tsk　タスクの状態参照　21

iref_tsk　タスクの状態参照(ハンドラ専用)　21

ref_tst　タスクの状態参照(簡易版)　24

iref_tst　タスクの状態参照(簡易版,ハンドラ専用)　24

1.3.　タスク付属同期機能　26

slp_tsk　起床待ち　26

tslp_tsk　起床待ち(タイムアウト)　26

wup_tsk　タスクの起床　28

iwup_tsk　タスクの起床(ハンドラ専用)　28

can_wup　起床要求のキャンセル　30

ican_wup　起床要求のキャンセル(ハンドラ専用)　30

rel_wai　待ち状態の強制解除　32

irel_wai　待ち状態の強制解除(ハンドラ専用)　32

sus_tsk　強制待ち状態への移行　34

isus_tsk　強制待ち状態への移行(ハンドラ専用)　34

rsm_tsk　強制待ち状態からの再開　36

irsm_tsk　強制待ち状態からの再開(ハンドラ専用)　36

frsm_tsk　強制待ち状態からの強制再開　36

ifrsm_tsk　強制待ち状態からの強制再開(ハンドラ専用)　36

dly_tsk　タスクの遅延　38

1.4.　タスク例外処理ルーチン　40

def_tex　タスク例外処理ルーチンの定義　40

ras_tex　タスク例外処理の要求　42

iras_tex　タスク例外処理の要求(ハンドラ専用)　42

dis_tex　タスク例外処理の禁止　44
ena_tex　タスク例外処理の許可　45
sns_tex　タスク例外禁止状態の参照　46
ref_tex　タスク例外処理の状態参照　47
iref_tex　タスク例外処理の状態参照(ハンドラ専用)　47
1.5. 同期・通信機能(セマフォ)　49
cre_sem　セマフォの生成　49
acre_sem　セマフォの生成(ID 自動割り当て)　49
del_sem　セマフォの削除　51
sig_sem　セマフォ資源の返却　52
isig_sem　セマフォ資源の返却(ハンドラ専用)　52
wai_sem　セマフォ資源の獲得　53
pol_sem　セマフォ資源の獲得(ポーリング)　53
ipol_sem　セマフォ資源の獲得(ハンドラ専用)　53
twai_sem　セマフォ資源の獲得(タイムアウト)　53
ref_sem　セマフォの状態参照　55
iref_sem　セマフォの状態参照(ハンドラ専用)　55
1.6. 同期・通信機能(イベントフラグ)　57
cre_flg　イベントフラグの生成　57
acre_flg　イベントフラグの生成(ID 自動割り当て)　57
del_flg　イベントフラグの削除　59
set_flg　イベントフラグのセット　60
iset_flg　イベントフラグのセット(ハンドラ専用)　60
clr_flg　イベントフラグのクリア　62
iclr_flg　イベントフラグのクリア(ハンドラ専用)　62
wai_flg　イベントフラグ待ち　63
pol_flg　イベントフラグ待ち(ポーリング)　63
ipol_flg　イベントフラグ待ち(ハンドラ専用)　63
twai_flg　イベントフラグ待ち(タイムアウト)　63
ref_flg　イベントフラグの状態参照　66
iref_flg　イベントフラグの状態参照(ハンドラ専用)　66
1.7. 同期・通信機能(データキュー)　68
cre_dtq　データキューの生成　68
acre_dtq　データキューの生成(ID 自動割り当て)　68
del_dtq　データキューの削除　70
snd_dtq　データキューへのデータ送信　71
psnd_dtq　データキューへのデータ送信(ポーリング)　71
ipsnd_dtq　データキューへのデータ送信(ハンドラ専用)　71
tsnd_dtq　データキューへのデータ送信(タイムアウト)　71
fsnd_dtq　データキューへのデータ強制送信　71
ifsnd_dtq　データキューへのデータ強制送信(ハンドラ専用)　71
rcv_dtq　データキューからのデータ受信　74
prcv_dtq　データキューからのデータ受信(ポーリング)　74
iprcv_dtq　データキューからのデータ受信(ハンドラ専用)　74
trcv_dtq　データキューからのデータ受信(タイムアウト)　74
ref_dtq　データキューの状態参照　77
iref_dtq　データキューの状態参照(ハンドラ専用)　77
1.8. 同期・通信機能(メールボックス)　79
cre_mbx　メールボックスの生成　79
acre_mbx　メールボックスの生成(ID 自動割り当て)　79

del_mbx　メールボックスの削除　81
snd_mbx　メールボックスへの送信　82
isnd_mbx　メールボックスへの送信(ハンドラ専用)　82
rcv_mbx　メールボックスからの受信　84
prcv_mbx　メールボックスからの受信(ポーリング)　84
iprcv_mbx　メールボックスからの受信(ハンドラ専用)　84
trcv_mbx　メールボックスからの受信(タイムアウト)　84
ref_mbx　メールボックスの状態参照　86
iref_mbx　メールボックスの状態参照(ハンドラ専用)　86
1.9. 拡張同期・通信機能(メッセージバッファ)　88
cre_mbf　メッセージバッファの生成　88
acre_mbf　メッセージバッファの生成(ID 自動割り当て)　88
del_mbf　メッセージバッファの削除　90
snd_mbf　メッセージバッファへの送信　91
psnd_mbf　メッセージバッファへの送信(ポーリング)　91
tsnd_mbf　メッセージバッファへの送信(タイムアウト)　91
rcv_mbf　メッセージバッファからの受信　93
prcv_mbf　メッセージバッファからの受信(ポーリング)　93
trcv_mbf　メッセージバッファからの受信(タイムアウト)　93
ref_mbf　メッセージバッファの状態参照　95
iref_mbf　メッセージバッファの状態参照(ハンドラ専用)　95
1.10. 拡張同期・通信機能(ランデヴ)　97
cre_por　ランデヴポートの生成　97
acre_por　ランデヴポートの生成(ID 自動割り当て)　97
del_por　ランデヴポートの削除　99
cal_por　ランデヴの呼び出し　100
tcal_por　ランデヴの呼び出し(タイムアウト)　100
acp_por　ランデヴの受付　102
pacp_por　ランデヴの受付(ポーリング)　102
tacp_por　ランデヴの受付(タイムアウト)　102
fwd_por　ランデヴの回送　104
rpl_rdv　ランデヴの終了　106
ref_por　ランデヴポートの状態参照　108
iref_por　ランデヴポートの状態参照(ハンドラ専用)　108
ref_rdv　ランデヴの状態参照　110
iref_rdv　ランデヴの状態参照(ハンドラ専用)　110
1.11. メモリプール管理機能(固定長メモリプール)　112
cre_mpf　固定長メモリプールの生成　112
acre_mpf　固定長メモリプールの生成(ID 自動割り当て)　112
del_mpf　固定長メモリプールの削除　114
get_mpf　固定長メモリブロックの獲得　115
pget_mpf　固定長メモリブロックの獲得(ポーリング)　115
ipget_mpf　固定長メモリブロックの獲得(ハンドラ専用)　115
tget_mpf　固定長メモリブロックの獲得(タイムアウト)　115
rel_mpf　固定長メモリブロックの解放　117
irel_mpf　固定長メモリブロックの解放(ハンドラ専用)　117
ref_mpf　固定長メモリプールの状態参照　118
iref_mpf　固定長メモリプールの状態参照(ハンドラ専用)　118
1.12. メモリプール管理機能(可変長メモリプール)　120
cre_mpl　可変長メモリプールの生成　120

acre_mpl　可変長メモリプールの生成(ID 自動割り当て) 120
del_mpl　可変長メモリプールの削除 122
get_mpl　可変長メモリブロックの獲得 123
pget_mpl　可変長メモリブロックの獲得(ポーリング) 123
tget_mpl　可変長メモリブロックの獲得(タイムアウト) 123
rel_mpl　可変長メモリプールブロックの解放 125
ref_mpl　可変長メモリプールの状態参照 126
iref_mpl　可変長メモリプールの状態参照(ハンドラ専用) 126
1.13. 時間管理機能 128
set_tim　システム時刻の設定 128
iset_tim　システム時刻の設定(ハンドラ専用) 128
get_tim　システム時刻の取得 130
iget_tim　システム時刻の取得(ハンドラ専用) 130
isig_tim　タイムティックの供給 131
1.14. 時間管理機能(周期ハンドラ) 132
cre_cyc　周期ハンドラの生成 132
acre_cyc　周期ハンドラの生成(ID 自動割り当て) 132
del_cyc　周期ハンドラの削除 135
sta_cyc　周期ハンドラの動作開始 136
ista_cyc　周期ハンドラの動作開始(ハンドラ専用) 136
stp_cyc　周期ハンドラの動作停止 137
istp_cyc　周期ハンドラの動作停止(ハンドラ専用) 137
ref_cyc　周期ハンドラの状態参照 138
iref_cyc　周期ハンドラの状態参照(ハンドラ専用) 138
1.15. 時間管理機能(アラームハンドラ) 139
cre_alm　アラームハンドラの生成 139
acre_alm　アラームハンドラの生成(ID 自動割り当て) 139
del_alm　アラームハンドラの削除 141
sta_alm　アラームハンドラの動作開始 142
ista_alm　アラームハンドラの動作開始(ハンドラ専用) 142
stp_alm　アラームハンドラの動作停止 143
istp_alm　アラームハンドラの動作停止(ハンドラ専用) 143
ref_alm　アラームハンドラの状態参照 144
iref_alm　アラームハンドラの状態参照(ハンドラ専用) 144
1.16. システム状態機能 145
rot_rdq　タスク優先順位の回転 145
irot_rdq　タスク優先順位の回転(ハンドラ専用) 145
get_tid　実行中タスク ID の参照 147
iget_tid　実行中タスク ID の参照(ハンドラ専用) 147
loc_cpu　CPU ロック状態への移行 148
iloc_cpu　CPU ロック状態への移行(ハンドラ専用) 148
unl_cpu　CPU ロック状態の解除 150
iunl_cpu　CPU ロック状態の解除(ハンドラ専用) 150
dis_dsp　ディスパッチの禁止 151
ena_dsp　ディスパッチの許可 152
sns_ctx　コンテキストの参照 153
sns_loc　CPU ロック状態の参照 154
sns_dsp　ディスパッチ禁止状態の参照 155
sns_dpn　ディスパッチ保留状態の参照 156
1.17. 割込管理機能 157

def_inh　割込ハンドラの定義　157
idef_inh　割込ハンドラの定義(ハンドラ専用)　157
1.18. システム構成理機能　159
ref_ver　バージョン情報の参照　159
iref_ver　バージョン情報の参照(ハンドラ専用)　159
1.19. 拡張機能　161
vrst_dtq　データキュー領域の初期化　161
vrst_mbx　メールボックス領域の初期化　162
vrst_mbf　メッセージバッファ領域の初期化　163
vrst_mpf　固定長メモリプール領域の初期化　164
vrst_mpl　可変長メモリプール領域の初期化　165

第 2 章　スタック使用量の計算　167
2.1. 各サービスコールのスタック使用量　168
2.2. スタックサイズの算出方法　169
2.2.1. ユーザスタックの算出方法　171
2.2.2. システムスタックの算出方法　172

第 3 章　付録　177
3.1. サービスコール一覧　178
3.2. エラーコード一覧　184
3.3. アセンブリ言語インタフェース　185
3.4. データタイプ　193
3.5. 共通定数と構造体のパケット形式　194

第1章 サービスコールリファレンス

1.1. サービスコールリファレンスの見方

サービスコールリファレンスは、以下の形式で記述しています。

サービスコール名　　内容(種別)

内容には、サービスコール名の処理概要を記載しています。(種別)は、「タイムアウト」の場合は、タイムアウト指定のサービスコール、「ポーリング」は、ポーリングのサービスコール、「ハンドラ専用」は、非タスクコンテキストから発行する場合のサービスコール、「ID 自動割り当て」は、ID 番号をカーネルが割り当てる場合のサービスコールを示します。

■ C 言語 API

C 言語でアプリケーションを記述する際の関数インタフェースを記述しています。アセンブリ言語でアプリケーションを記述する際は、3.3アセンブリ言語インタフェースを参照ください。

■ パラメータ

C 言語でアプリケーションを記述する際の引数の型、内容およびカーネルに渡す構造体の内容について記述しています。

■ リターンパラメータ

C 言語でアプリケーションを記述する際の戻り値の型、内容およびカーネルから返される構造体の内容について記述しています。

■ エラーコード

エラーコード名　　エラーコードの意味

エラーコードの文字列、例えば E_OK などは"itron.h"に"#define"を使って、"mr32r.inc"に ".EQU" を使って定義されています。エラー判定をプログラム中で記述する場合は、この定義された文字列を用いなければなりません。

■ 機能説明

サービスコールの機能を詳細に記述しています。

■ 使用例

サービスコールの使用例を記述しています。

1.2. タスク管理機能

cre_tsk　タスクの生成
acre_tsk　タスクの生成(ID 自動割り当て)

■ C 言語 API

```
ER ercd = cre_tsk( ID tskid, T_CTSK *pk_ctsk);
ER_ID tskid = acre_tsk( T_CTSK *pk_ctsk);
```

■ パラメータ

ID	tskid	生成するタスク ID 番号
T_CTSK	*pk_ctsk	タスク生成情報を格納した構造体へのポインタ

pk_ctsk の内容

```
typedef   struct t_ctsk {
    ATR      tskatr    0     4    タスク属性
    VP_INT   exinf     +4    4    タスクの拡張情報
    FP       task      +8    4    タスクの起動番地
    PRI      itskpri   +12   1    タスクの起動時優先度
    SIZE     stksz     +16   4    タスクのスタック領域のサイズ(バイト数)
    VP       stk       +20   4    タスクのスタック領域の先頭番地
} T_CTSK;
```

■ リターンパラメータ

- cre_tsk の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_tsk の場合

ER_ID	tskid>0	生成したタスクの ID 番号
	tskid<0	エラーコード

■ エラーコード

E_NOID	空き ID なし(acre_tsk 使用時)
E_NOMEM	メモリ不足
E_OBJ	オブジェクト状態が不正(cre_tsk 使用時)

■ 機能説明

cre_tsk サービスコールは、tskid で示されたタスクを生成します。生成するタスクの ID 番号は、ユーザ管理となります。すなわち、指定したタスクを未登録(NON-EXISTENT)状態から休止(DORMANT)状態へ移行します。指定可能な ID 番号の範囲は、1 からコンフィギュレーションファイルの最大項目数定義で設定したユーザシステムの最大タスク数までです。

acre_tsk サービスコールは、カーネルが自動的に割り当てた ID のタスクを生成し、カーネルが割り当てた ID を戻り値として返します。acre_tsk サービスコールにおいて空き ID がない場合は、エラーコードとして E_NOID を返します。

生成するのに必要な情報 pk_ctsk を以下に示します。

◆ **exinf(拡張情報)**
生成するタスクに関する情報を格納するためにユーザが自由に利用できる領域です。exinf の内容に関しては、MR32R は関知しません。拡張情報は、タスク例外発生時、または ext_tsk,ter_tsk サービスコールによってタスクが再起動した際に引数として渡されます。

◆ **tskatr(タスク属性)**
tskatr には、[[__MR_INT||__MR_EXT]|[TA_HLNG||TA_HASM]|[TA_ASM]] を指定できます。カーネルは、TA_HLNG,TA_ASM の値を無視し、C 言語記述、アセンブリ言語の記述を区別しません。

属性	意味
__MR_INT	生成するタスクが使用するスタック領域を内蔵 RAM から確保する
__MR_EXT	生成するタスクが使用するスタック領域を外部 RAM から確保する
TA_HLNG	タスクを高級言語で記述した際に指定します。
TA_ASM	タスクをアセンブリ言語で記述した際に指定します。
TA_ACT	タスク生成後、act_tsk と同様の処理を行い、タスクを実行可能状態に移行します。

◆ **task(タスク起動アドレス)**
生成するタスクの開始アドレスを指定する領域です。C 言語でプログラムを記述する場合は生成するタスク(関数)をプロトタイプ宣言しなければなりません。

◆ **itskpri(タスク起動時優先度)**
生成するタスクが起動された時の優先度を指定する領域です。

◆ **stksz(スタックサイズ)**
生成するタスクが使用するスタックサイズを指定する領域です。

◆ **stk(スタックの先頭アドレス)**
ユーザが確保したスタック領域の先頭アドレスを指定します。この値が NULL の場合は、tskatr の値(__MR_INT,__MR_EXT)に応じてカーネルがスタック領域を割り当てます。NULL でない場合は、tskatr の__MR_INT、__MR_EXT の値は、無視されます。

本サービスコールは、指定したタスクが未登録(NON-EXISTENT)状態の時のみ有効で、その他の状態にあるタスクに対して、本サービスコールを発行した場合は、エラー E_OBJ を返します。本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_task2   2
#define ID_task3   3
void task2(void);      /* プロトタイプ宣言 */
void task3(void);      /* プロトタイプ宣言 */
void task1(void)
{
    T_CTSK ctsk2;
    T_CTSK ctsk3 = { MR_INT, 0, task3,2,200,NULL};
     ER_ID tskid;
     ER ercd;
    ctsk2.tskatr = __MR_EXT;    /* スタックに外部 RAM を指定 */
    ctsk2.task = task2;         /* タスク起動アドレス設定 */
    ctsk2.itskpri = 2;          /* タスク起動時の優先度設定 */
    ctsk2.stksz = 100;          /* タスクが使用するスタックサイズ設定 */
    ctsk2.stk = NULL;

    tskid = acre_tsk( &ctsk2 );
     ercd = cre_tsk( ID_task3, &ctsk3);
          :
}
void task2(void)
{
       :
    ext_tsk();
}
void task3(void)
{
       :
    ext_tsk();
}
```

del_tsk　　タスクの削除

■ C 言語 API

```
ER ercd = del_tsk( ID tskid );
```

■ パラメータ

ID	tskid	削除するタスク ID 番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(tskid のタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態ではないまたは、自タスクを指定)

■ 機能説明

tskid で示されるタスクを削除します。本サービスコールを発行することにより対象となるタスクの状態は、休止(DORMANT)状態から未登録(NON-EXISTENT)状態へ移行します。本サービスコールにより、対象タスクが使用していたスタック領域は解放され、再利用可能な状態となります。

本サービスコールは、対象となるタスクが休止状態の時のみ有効です。したがって、その他の状態 にあるタスクに対して本サービスコールを発行した場合は、エラーE_OBJ を返します。

未登録状態のタスクに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、cre_tsk,acre_tsk によって生成されたタスクに対して有効です。コンフィギュレーションファイルに静的に定義されたタスクに対して本サービスコールを発行した場合の動作は保証されません。本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_task2   2
#define ID_task3   3
void task1()
{
    ER ercd;
            :
    ercd = del_tsk( ID_task2 );
            :
}
void task2()
{
        :
    ext_tsk();
}
```

act_tsk　タスクの起動
iact_tsk　タスクの起動(ハンドラ専用)

■ C 言語 API

```
ER ercd = act_tsk( ID tskid );
ER ercd = iact_tsk( ID tskid );
```

■ パラメータ

ID	tskid	起動するタスク ID 番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(tskid のタスクは存在しない)
E_QOVR	キューイングオーバーフロー(起動要求カウントが 32767 をオーバーフローした)

■ 機能説明

tskid で示されたタスクを起動します。起動したタスクは休止状態から実行可能状態へ移行します。タスク起動時に行われる処理は、以下の通りです。

1 タスクの現在優先度を初期化する。
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 タスク例外処理禁止状態にする。

tskid=TSK_SELF(0)の指定により、自タスクの指定となります。タスクには、タスク生成時に指定したタスクの拡張情報がパラメータとして渡ります。非タスクコンテキストにおいて、tskid に TSK_SELF を指定した場合の動作は保証されません。
対象タスクが休止状態でない場合には、本サービスコールによるタスクの起動要求は、キューイングされます。すなわち、起動要求カウントに1加算されます。起動要求カウントの最大値は、32767 です。起動要求カウントの最大値を越える場合は、エラーコード E_QOVR を返します。
対象タスクが存在しない場合は、エラーコード E_NOEXS を返します。
tskid に TSK_SELF が指定された場合は、自タスクを対象タスクとします。
本サービスコールは、タスクコンテキストからは、act_tsk、非タスクコンテキストからは、iact_tsk を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_task2   2
#define ID_task3   3
void task1()
{
   ER ercd;
       :
   ercd = act_tsk( ID_task2 );
       :
}
void task2()
{
       :
   ext_tsk();
}
```

can_act　起動要求カウントのキャンセル
ican_act　起動要求カウントのキャンセル(ハンドラ専用)

■ C言語API

```
ER_UINT actcnt = can_act( ID tskid );
ER_UINT actcnt = ican_act( ID tskid );
```

■ パラメータ

ID	tskid	起動するタスクID番号

■ リターンパラメータ

ER_UINT	actcnt > 0	キャンセルされた起動要求カウント
	actcnt <0	エラーコード

■ エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)

■ 機能説明

tskid で示されたタスクにキューイングされていた起動要求回数を求め、その結果をリターンパラメータとして返し、同時にその起動要求を全て無効にします。
tskid=TSK_SELF(0)の指定により、自タスクの指定になります。非タスクコンテキストにおいて、tskid に TSK_SELF を指定した場合の動作は保証されません。
休止状態のタスクを対象として呼び出すこともできます。その場合のリターンパラメータは 0 となります。
本サービスコールは、タスクコンテキストからは、can_act、非タスクコンテキストからは、ican_act を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_task2   2
#define ID_task3   3
void task1()
{
   ER_UINT actcnt;
       :
   actcnt = can_act( ID_task2 );
       :
}
void task2()
{
       :
   ext_tsk();
}
```

sta_tsk　タスクの起動(起動コード指定)
ista_tsk　タスクの起動(起動コード指定,ハンドラ専用)

■ C 言語 API

```
ER ercd = sta_tsk( ID tskid,VP_INT stcd );
ER ercd = ista_tsk ( ID tskid,VP_INT stcd );
```

■ パラメータ

ID	tskid	起動するタスク ID 番号
VP_INT	stcd	タスク起動コード

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(tskid のタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態ではない)

■ 機能説明

tskid で示されたタスクを起動します。すなわち指定したタスクを休止(DORMANT)状態から実行可能(READY)状態もしくは、実行(RUN)状態へ移行します。本サービスコールは、起動要求をキューイングしません。したがって、対象タスクが休止(DORMANT)状態にない場合に発せられた要求に対しては、サービスコール発行タスクにエラーE_OBJ を返します。本サービスコールは、指定したタスクが休止(DORMANT)状態であるときのみ有効です。未登録状態のタスクに対して本サービスコールを発行した場合は、エラーE_NOEXS を返します
起動コード stcd は起動タスクに引数として渡されます。
ter_tsk、ext_tsk などで終了したタスクを再起動した場合、タスクは以下の状態で起動します 。

1 タスクの現在優先度を初期化する。
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 タスク例外処理禁止状態にする。

※PC、PSW と R4,R11,R12,R13,R14 レジスタ以外のレジスタの初期値は不定です。R4 レジスタに起動コードが設定されます。R11、R12、R13 レジスタにベースアドレスが設定されます。R14 レジスタには、ext_tsk サービスコール先頭アドレスが設定されます。

なお、タスクが再起動された場合でも、以前に定義したタスク例外は解除されません。
本サービスコールは、タスクコンテキストからは、sta_tsk、非タスクコンテキストからは、ista_tsk を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task()
{
  ER ercd;
  ercd = sta_tsk( ID_task2, stcd );
        :
}
void task2(int msg)
{
  if(msg == 0)
        :
}
```

ext_tsk　　自タスクの終了

■ C 言語 API

```
ext_tsk();
```

■ パラメータ

なし

■リターンパラメータ

本サービスコールからﾘﾀｰﾝしません

■エラーコード

本サービスコールからﾘﾀｰﾝしません

■ 機能説明

自タスクを終了します。すなわち、自タスクを実行(RUN)状態から休止(DORMANT)状態へ移行します。ただし、自ﾀｽｸに対する起動要求ｶｳﾝﾄが1以上の場合は、起動要求ｶｳﾝﾄを1減じ、再度 act_tsk,iact_tsk の処理と同様の処理を行い、タスクは、休止(DORMANT)状態から実行可能状態(READY)にします。タスクを起動するときにパラメータとして、タスク拡張情報を渡します。

本サービスコールは、ﾀｽｸからのﾘﾀｰﾝで自動的に発行されるようになっています。従って、タスク終了時に明示的に ext_tsk();を記述する必要はありません。

本サービスコールの発行では自ﾀｽｸが以前に獲得していた資源 (セマフォなど)は解放しません。

なお、タスク例外から本サービスコールを発行した場合は、ﾀｽｸ例外に対応したタスクも同時に正常終了します。

本サービスコールはタスクコンテキストでのみ使用可能です。ﾃｨｽﾊﾟｯﾁ禁止状態、CPU ﾛｯｸ状態から本サービスコールを発行した場合、ﾃｨｽﾊﾟｯﾁ禁止状態、CPU ﾛｯｸ状態は解除されます。本サービスコールは、非ﾀｽｸｺﾝﾃｷｽﾄでは使用できません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
     :
   ext_tsk();
}
```

exd_tsk　　自タスクの終了および削除

■ C言語API

```
exd_tsk();
```

■ パラメータ

なし

■リターンパラメータ

本サービスコールからﾘﾀｰﾝません

■エラーコード

本サービスコールからﾘﾀｰﾝしません

■ 機能説明

自タスクを終了し、さらに自タスクを削除します。すなわち、自タスクを実行(RUN)状態から未登録(NON_EXISTENT)状態へ移行します。本サービスコールは、起動要求カウントの有無にかかわらず自タスクを終了させ、削除します。。

本サービスコールはタスクコンテキストでのみ使用可能です。また、本サービスコールは、ディスパッチ禁止状態、CPUロック状態であっても使用可能です。ディスパッチ禁止状態、CPUロック状態から本サービスコールを発行した場合、ディスパッチ禁止状態、CPU ロック状態は解除されます。本サービスコールは、非タスクコンテキストでは使用できません。また、本サービスコールは、cre_tskによって生成されたタスクでのみ発行可能です。コンフィギュレーションファイルで静的に定義されたタスクから本サービスコールを使用した場合の動作は保証されません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1()
{
        :
   cre_tsk( ID_task2, &ctsk2 );
        :
   sta_tsk( ID_task2, 0 );
        :
}
void task2()
{
        :
     exd_tsk();
}
```

ter_tsk　　タスクの強制終了

■ C言語API

```
ER ercd = ter_tsk( ID tskid );
```

■ パラメータ

ID	tskid	終了するタスクID番号

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが休止状態)
E_ILUSE	対象タスクが、自タスクIDまたは、TSK_SELF

■ 機能説明

tskid で示されたタスクを、強制的に終了させます。対象タスクの起動要求カウントが1以上の場合、起動要求カウントを1減じ、再度 act_tsk,iact_tsk の処理と同様の処理を行い、タスクは、休止(DORMANT)状態から実行可能状態(READY)にします。さらに、タスクを起動するときにパラメータとして、タスク拡張情報を渡します。

このサービスコールで自タスク ID、および TSK_SELF(=0)を指定できません。指定した場合は、E_ILUSE を返します。自タスクを終了する場合は ext_tsk サービスコールを使用してください。

自タスクを指定したタスクが待ち状態に入り、何らかの待ち行列 につながれていた場合には、このサービスコールの実行によってその待ち行列から削除されます。しかし、指定したタスクがそれ以前に獲得したセマフォなどは解放されません。

tskidのタスクが待ち状態であり、待ちオブジェクトの属性がTA_TPRIの場合、本サービスコールによって待ち行列が変更され、待ち行列につながれていたタスクの待ち状態を解除することがあります。(get_mpl,tget_mpl,snd_mbf,tsnd_mbf による待ちの場合)

tskid で示されたタスクが休止(DORMANT)状態にある場合は、サービスコールの戻り値としてエラーE_OBJ を返します。

tskidで示されたタスクが未登録(NON-EXISTENT)状態にある場合は、サービスコールの戻り値としてエラーE_NOEXS を返します。

本サービスコールはタスクコンテキストでのみ使用可能です。非タスクコンテキストでは使用できません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
        :
    ter_tsk( ID_main );
        :
    ext_tsk();
}
```

chg_pri　タスク優先度の変更
ichg_pri　タスク優先度の変更(ハンドラ専用)

■ C言語API

```
ER ercd = chg_pri( ID tskid, PRI tskpri );
ER ercd = ichg_pri( ID tskid, PRI tskpri );
```

■ パラメータ

ID	tskid	対象タスクID番号
PRI	tskpri	変更後のタスク優先度

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが休止状態)

■ 機能説明

tskid で示されたタスクの優先度を、tskpri で示される値に変更し、その変更結果に基づいて再スケジューリングを行います。したがって、レディキューにつながれているタスク (実行状態のタスクを含む)、または優先度順の待ち行列の中のタスクに対して本サービスコールが実行された場合、対象タスクはキューの該当優先度の部分の最後尾に移動します。以前と同じ優先度を指定した場合も、同様に、そのキューの最後尾に移動します 。

タスクの優先度は、数の小さい方が高く1が最高優先度です。優先度として指定できる数値は最小値が 1 です。また、優先度の最大値はコンフィギュレーションファイルで指定した優先度の最大値であり、指定可能範囲は1～255です。例えば、コンフィギュレーションファイルで

```
system{
        stack_size    = 0x100;
        priority      = 13;
};
```

の場合は指定できる優先度の範囲は1から13までです。

TSK_SELF が指定された場合は自タスクの優先度を変更します。非タスクコンテキストにおいて、tskid に TSK_SELF を指定した場合の動作は保証されません。TPRI_INI が指定された場合、タスク生成時に指定した初期タスク優先度に変更します。変更したタスク優先度は、タスクの終了もしくは、本サービスコールが再度実行されるまで有効です。

tskidのタスクが待ち状態であり、待ちオブジェクトの属性がTA_TPRIの場合、本サービスコールによって待ち行列が変更され、待ち行列につながれていたタスクの待ち状態を解除することがあります。(snd_mbf による待ちの場合)

tskid で示されたタスクが休止(DORMANT)状態にある場合は、サービスコールの戻り値としてエラーE_OBJ を返します。tskid で示されたタスクが未登録(NON-EXISTENT)状態にある場合は、サービスコールの戻り値としてエラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、chg_pri、非タスクコンテキストからは、ichg_pri を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
        :
    chg_pri( ID_task2, 2 );
        :
}
```

get_pri　タスク優先度の参照
iget_pri　タスク優先度の参照(ハンドラ専用)

■ C言語API

```
ER ercd = get_pri( ID tskid, PRI *p_tskpri );
ER ercd = iget_pri( ID tskid, PRI *p_tskpri );
```

■ パラメータ

ID	tskid	対象タスクID番号
PRI	*p_tskpri	タスク優先度を返す領域へのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが休止状態)

■ 機能説明

tskidで示されたタスクの優先度を、p_tskpriで示される領域に返します。TSK_SELFが指定された場合は自タスクの優先度を参照します。非タスクコンテキストにおいて、tskidにTSK_SELFを指定した場合の動作は保証されません。
tskidで示されたタスクが休止(DORMANT)状態にある場合は、サービスコールの戻り値としてエラーE_OBJを返します。tskidで示されたタスクが未登録(NON-EXISTENT)状態にある場合は、サービスコールの戻り値としてエラーE_NOEXSを返します。
本サービスコールは、タスクコンテキストからは、get_pri、非タスクコンテキストからは、iget_priを使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
        :
    get_pri( ID_task2, &p_tskpri );
        :
}
```

ref_tsk　タスクの状態参照
iref_tsk　タスクの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_tsk( ID tskid, T_RTSK *pk_rtsk );
ER ercd = iref_tsk( ID tskid, T_RTSK *pk_rtsk );
```

■ パラメータ

ID	tskid	対象タスクID番号
T_RTSK	*pk_rtsk	タスク状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

pk_rtskの内容

```
typedef   struct t_rtsk{
    STAT   tskstat  +0   4    タスク状態
    PRI    tskpri   +4   2    タスクの現在優先度
    PRI    tskbpri  +6   2    タスクのベース優先度
    STAT   tskwait  +8   4    待ち要因
    ID     wobjid   +12  2    待ちオブジェクトID
    TMO    lefttmo  +16  4    タイムアウトするまでの時間
    UINT   actcnt   +20  4    起動要求キューイング数
    UINT   wupcnt   +24  4    起床要求キューイング数
    UINT   suscnt   +28  4    強制待ち要求ネスト数
} T_RTSK;
```

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)

■ 機能説明

tskidで示されたタスクの状態を参照し、そのタスクの現在の情報をpk_rtskの指す領域にリターン値として返します。tskidとしてTSK_SELFが指定された場合は、自タスクの状態を参照します。非タスクコンテキストにおいて、tskidにTSK_SELFを指定した場合の動作は保証されません。

tskidで示されたタスクが未登録(NON-EXISTENT)状態にある場合は、サービスコールの戻り値としてエラーE_NOEXSを返します。

◆tskstat(タスク状態)

tskstatには指定したタスクの状態によって次の値が返されます。

- TTS_RUN(0x0001) 実行(RUN)状態
- TTS_RDY(0x0002) 実行可能(READY)状態
- TTS_WAI(0x0004) 待ち(WAIT)状態
- TTS_SUS(0x0008) 強制待ち(SUSPEND)状態
- TTS_WAS(0x000C) 二重待ち(WAIT-SUSPEND)状態
- TTS_DMT(0x0010) 休止(DORMANT)状態

◆tskpri(タスクの現在優先度)

tskpri には、指定したタスクの現在優先度を返します。タスクが休止状態の場合は、tskpriは、不定となります。

◆tskbpri(タスクのベース優先度)

tskpri には、指定したタスクのベース優先度を返します。M3T-MR32R は、ミューテックス機能をサポートしていないため、tskpri と tskbpri は、同じ値となります。また、タスクが休止状態の場合は、tskpriは、不定となります。

◆ **tskwait(タスクの待ち要因)**

対象タスクが待ち状態であるならば次の待ち要因が返されます。各待ち要因の値を以下に示します。タスク状態が、待ち状態(TTS_WAI、または TTS_WAS)以外の場合は、tskwait は不定となります。

- TTW_SLP (0x0001)　　slp_tsk、tslp_tsk による待ち
- TTW_DLY (0x0002)　　dly_tsk による待ち
- TTW_SEM (0x0004)　　wai_sem、twai_sem による待ち
- TTW_FLG (0x0008)　　wai_flg、twai_flg による待ち
- TTW_SDTQ(0x0010)　　snd_dtq、tsnd_dtq による待ち
- TTW_RDTQ(0x0020)　　rcv_dtq,trcv_dtq による待ち
- TTW_MBX (0x0040)　　rcv_mbx、trcv_mbx による待ち
- TTW_SMBF(0x0100)　　snd_mbf、tsnd_mbf による待ち
- TTW_RMBF(0x0200)　　rcv_mbf、trcv_mbf による待ち
- TTW_CAL (0x0400)　　cal_por、tcal_por による待ち
- TTW_ACP (0x0800)　　acp_por、tacp_por による待ち
- TTW_RDV (0x1000)　　ランデヴ終了待ち
- TTW_MPF (0x2000)　　get_mpf,tget_mpf による待ち
- TTW_MPL (0x4000)　　get_mpl,tget_mpl による待ち

◆ **wobjid(待ちオブジェクト ID)**

対象タスクがランデヴ待ち状態以外の待ち状態(TTS_WAI または、TTS_WAS)であるならば、待ち対象オブジェクト ID を返します。それ以外の場合は、wobjid は、不定となります。

◆ **lefttmo(タイムアウトまでの時間)**

対象タスクが待ち状態(TTS_WAI,または TTS_WAS)の場合、タイムアウトするまでの時間を返します。永久待ちの場合は、TMO_FEVR を返します。それ以外の状態の場合は、lefttmo は不定となります。

◆ **actcnt(起動要求カウント)**

現在の起動要求キューイング数を返します。

◆ **wupcnt(起床要求カウント)**

現在の起床要求キューイング数を返します。タスクが休止状態の場合は、wupcnt は、不定となります。

◆ **suscnt(強制待ち要求カウント)**

現在の強制待ち要求ネスト数を返します。タスクが休止状態の場合は、suscnt は、不定となります。

本サービスコールは、タスクコンテキストからは、ref_tsk、非タスクコンテキストからは、iref_tsk を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    T_RTSK rtsk;
        :
    ref_tsk( ID_main, &rtsk );
        :
}
```

ref_tst　タスクの状態参照(簡易版)
iref_tst　タスクの状態参照(簡易版,ハンドラ専用)

■ C言語API

```
ER ercd = ref_tst( ID tskid, T_RTST *pk_rtst );
ER ercd = iref_tst( ID tskid, T_RTST *pk_rtst );
```

■ パラメータ

ID	tskid	対象タスクID番号
T_RTST	*pk_rtst	タスク状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

pk_rtskの内容

```
typedef   struct t_rtst{
    STAT   tskstat  +0   4    タスク状態
    STAT   tskwait  +4   4    待ち要因
} T_RTST;
```

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)

■ 機能説明

tskidで示されたタスクの状態を参照し、そのタスクの現在の情報をpk_rtstが指す領域にリターン値として返します。tskidとしてTSK_SELFが指定された場合は、自タスクの状態を参照します。非タスクコンテキストにおいて、tskidにTSK_SELFを指定した場合の動作は保証されません。
tskidで示されたタスクが未登録(NON-EXISTENT)状態にある場合は、サービスコールの戻り値としてエラーE_NOEXSを返します。

◆tskstat(タスク状態)

tskstatには指定したタスクの状態によって次の値が返されます。

- TTS_RUN(0x0001) 実行(RUN)状態
- TTS_RDY(0x0002) 実行可能(READY)状態
- TTS_WAI(0x0004) 待ち(WAIT)状態
- TTS_SUS(0x0008) 強制待ち(SUSPEND)状態
- TTS_WAS(0x000C) 二重待ち(WAIT-SUSPEND)状態
- TTS_DMT(0x0010) 休止(DORMANT)状態

◆ **tskwait(タスクの待ち要因)**

対象タスクが待ち状態であるならば次の待ち要因が返されます。各待ち要因の値を以下に示します。タスク状態が、待ち状態(TTS_WAI、または TTS_WAS)以外の場合は、tskwait は不定となります。

- TTW_SLP (0x0001)　　slp_tsk、tslp_tsk による待ち
- TTW_DLY (0x0002)　　dly_tsk による待ち
- TTW_SEM (0x0004)　　wai_sem、twai_sem による待ち
- TTW_FLG (0x0008)　　wai_flg、twai_flg による待ち
- TTW_SDTQ(0x0010)　　snd_dtq、tsnd_dtq による待ち
- TTW_RDTQ(0x0020)　　rcv_dtq,trcv_dtq による待ち
- TTW_MBX (0x0040)　　rcv_mbx、trcv_mbx による待ち
- TTW_SMBF(0x0100)　　snd_mbf、tsnd_mbf による待ち
- TTW_RMBF(0x0200)　　rcv_mbf、trcv_mbf による待ち
- TTW_CAL (0x0400)　　cal_por、tcal_por による待ち
- TTW_ACP (0x0800)　　acp_por、tacp_por による待ち
- TTW_RDV (0x1000)　　ランデヴ終了待ち
- TTW_MPF (0x2000)　　get_mpf,tget_mpf による待ち
- TTW_MPL (0x4000)　　get_mpl,tget_mpl による待ち

本サービスコールは、タスクコンテキストからは、ref_tst、非タスクコンテキストからは、iref_tst を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    T_RTST rtst;
            :
    ref_tst( ID_main, &rtst );
            :
}
```

1.3. タスク付属同期機能

slp_tsk　起床待ち
tslp_tsk　起床待ち(タイムアウト)

■ **C 言語 API**

```
ER ercd = slp_tsk();
ER ercd = tslp_tsk( TMO tmout );
```

■ **パラメータ**

- slp_tsk の場合
  なし
- tslp_tsk の場合

TMO	tmout	タイムアウト値

■**リターンパラメータ**

ER	ercd	正常終了(E_OK)またはエラーコード

■**エラーコード**

E_TMOUT	タイムアウト
E_RLWAI	強制待ち解除

■ **機能説明**

自タスクを実行(RUN)状態から起床待ち状態へ移行します。本サービスコール実行による待ち状態は、以下に示す場合に解除されます。

◆ **他タスクおよび割り込みからタスク起床のサービスコール を発行した場合**
この時のエラーコードは、E_OK が返ります。

◆ **他タスクおよび割り込みから待ち状態強制解除のサービスコール を発行した場合**
この時のエラーコードは、E_RLWAI が返ります。

◆ **tmout 経過後、最初のタイムティックが発生した場合(tslp_tsk の場合)**
この時のエラーコードは、E_TMOUT が返ります。

本サービスコールにより待ち(WAIT)状態となっているときに他のタスクから sus_tsk されるとそのタスクの状態は二重待ち(WAIT-SUSPEND)状態になります。この場合はタスク起床のサービスコールにより待ち状態が解除されても、まだ強制待ち状態であり、rsm_tsk の発行まで、タスクの実行は再開されません。

tslp_tsk では、引数に tmout を指定し一定時間自タスクを起床待ち状態に移行させることが出来ます。tmout の単位は、msです。すなわち、tslp_tsk(10);と記述すれば、10ms 間、自タスクが実行(RUN)状態から待ち(WAIT)状態へ移行します。tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定となり、slp_tsk サービスコールと同じ動作を行います。tmout=TMO_POL(=0)を指定した場合、起床要求カウントを 1 減らして実行を継続します。起床要求カウントが 0 の場合は、E_TMOUT を返します。

tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。

本サービスコールはタスクコンテキストからのみ発行してください。非タスクコンテキストから発行した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task()
{
                :
    if( slp_tsk() != E_OK )
        error(“Forced wakeup¥n”);
                :
                :
    if( tslp_tsk( 10 ) == E_TMOUT )
        error(“time out¥n”);
                :
}
```

wup_tsk　タスクの起床
iwup_tsk　タスクの起床(ハンドラ専用)

■ C言語API

```
ER ercd = wup_tsk( ID tskid );
ER ercd = iwup_tsk( ID tskid );
```

■ パラメータ

ID	tskid	対象タスクID番号

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが休止状態)
E_QOVR	キューイングのオーバーフロー

■ 機能説明

tskidで指定したタスクがslp_tskあるいはtslp_tskの実行による待ち(WAIT)状態であれば、待ちを解除して実行可能(READY)状態もしくは実行(RUN)状態に移行します。また、tskid で指定したタスクが二重待ち(WAIT-SUSPEND)状態である時は、待ちのみを解除して強制待ち(SUSPEND)状態に移行します。

対象タスクが休止(DORMANT)状態にある場合に発せられた要求に対しては、サービスコール発行タスクにエラーE_OBJ を返し、対象タスクが未登録状態の場合は、エラーE_NOEXS を返します。tskid に TSK_SELF が指定された場合は、自タスク指定となります。非タスクコンテキストにおいて、tskid に TSK_SELF を指定した場合の動作は保証されません。

slp_tsk あるいは tslp_tsk サービスコール実行による待ち(WAIT)状態もしくは二重待ち(WAIT-SUSPEND)状態にないタスクに対して本サービスコールを行なった場合は、起床要求が蓄積されます。すなわち、起床要求対象タスクの起床要求カウントを1つ増やすことにより起床要求を蓄積します。

起床要求カウントの最大値は32767です。起床要求カウントが32767の時に、これを越えて起床要求を発生させた場合、起床要求カウントは 32767 のままで本サービスコールの発行タスクには、エラーコードE_QOVRを返します。

本サービスコールは、タスクコンテキストからは、wup_tsk、非タスクコンテキストからは、iwup_tskを使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
                :
   if( wup_tsk( ID_main ) != E_OK )
       printf("Can't wakeup main()¥n");
                :
}
```

can_wup　起床要求のキャンセル
ican_wup　起床要求のキャンセル(ハンドラ専用)

■ C言語API

```
ER_UINT wupcnt = can_wup( ID tskid );
ER_UINT wupcnt = ican_wup( ID tskid );
```

■ パラメータ

ID	tskid	対象タスクID番号

■リターンパラメータ

ER_UINT	wupcnt > 0	キャンセルされた起床要求カウント
	wupcnt <0	エラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが休止状態)

■ 機能説明

tskidで示された対象タスクの起床要求カウントを0(ゼロ)クリアします。すなわち、本サービスコール発行以前にwup_tsk、iwup_tskサービスコールにより蓄積されていた起床要求をすべて無効にします。

また、本サービスコールの戻り値として0(ゼロ)クリアする前の起床要求カウント、すなわち無効になった起床要求回数(wupcnt)が返されます。対象タスクが休止(DORMANT)状態に発せられた要求に対しては、サービスコール発行タスクにエラーE_OBJを返し、未登録(NON-EXISTENT)状態の場合は、エラーE_NOEXSを返します。tskidにTSK_SELFが指定された場合は、自タスク指定となります。

本サービスコールは、タスクコンテキストからは、can_wup、非タスクコンテキストからは、ican_wupを使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task()
{
    ER_UINT wupcnt;
              :
    wupcnt = can_wup(ID_main);
    if( wup_cnt  > 0 )
        printf(“wupcnt = %d¥n”,wupcnt);
              :
}
```

rel_wai　待ち状態の強制解除
irel_wai　待ち状態の強制解除(ハンドラ専用)

■ C言語API

```
ER ercd = rel_wai( ID tskid );
ER ercd = irel_wai( ID tskid );
```

■ パラメータ

ID	tskid	対象タスクID番号

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが待ち状態でない)

■ 機能説明

tskid で示されたタスクの待ち状態(強制待ち(SUSPEND)状態を除く)を、強制的に解除し、実行可能(READY)状態あるいは実行(RUN)状態に移行します。強制的に解除されたタスクにはエラーコードE_RLWAIを返します。対象タスクが何らかの待ち行列 につながれていた場合には、本サービスコールの実行によってその待ち行列から削除されます。
二重待ち状態(WAIT-SUSPEND)のタスクに対して、本サービスコールを発行した場合、対象タスクの待ち状態は解除され、強制待ち状態(SUSPEND)に移行します。[3]
tskidのタスクが待ち状態であり、待ちオブジェクトの属性がTA_TPRIの場合、本サービスコールによって待ち行列が変更され、待ち行列につながれていたタスクの待ち状態を解除することがあります。(get_mpl,tget_mpl,snd_mbf,tsnd_mbf による待ちの場合)
対象タスクが待ち状態にない場合は、サービスコール発行タスクにエラーE_OBJ を返します。また、対象タスクが未登録状態(NON-EXISTENT)の場合に、本サービスコールを発行した場合は、エラーE_NOEXSを返します。本サービスコールは、自タスクを指定できません。
本サービスコールは、タスクコンテキストからは、rel_wai、非タスクコンテキストからは、irel_wai を使用してください。

[3] すなわち、強制待ち状態は、本サービスコールにより待ちは解除されません。強制待ち状態は、rsm_tsk,irsm_tsk,frsm_tsk,ifrsm_tsk サービスコールによって待ちが解除されます。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
            :
    if( rel_wai( ID_main ) != E_OK )
        error("Can't rel_wai main()¥n");
         :
}
```

sus_tsk　強制待ち状態への移行
isus_tsk　強制待ち状態への移行(ハンドラ専用)

■ C言語API

```
ER ercd = sus_tsk( ID tskid );
ER ercd = isus_tsk( ID tskid );
```

■ パラメータ

ID	tskid	対象タスクID番号

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが休止状態)
E_QOVR	キューイングのオーバーフロー

■ 機能説明

tskid で示されたタスクの実行を中断させ、強制待ち(SUSPEND)状態へ移行します。tskid で示されたタスクが待ち状態にある場合は、2重待ち状態に移行します。tskid に TSK_SELF(=0)の指定により、自タスクの指定になります。

強制待ち状態は、rsm_tsk,irsm_tsk,frsm_tsk,ifrsm_tsk サービスコールの発行によって解除されます。tskid で示された対象タスクが休止(DORMANT)状態にある場合は、サービスコールの戻り値としてエラーE_OBJ を返し、未登録状態(NON-EXISTENT)にある場合は、エラーE_NOEXS を返します。

本サービスコールの実行により強制待ち要求はネストされます。強制待ち要求のネスト数の最大値は、32767 です。強制待ち要求のネスト数が、32767 のタスクに対して、本サービスコールを発行した場合、強制待ち要求ネスト数は 32767 のままで、本サービスコールを発行したタスクに対してエラーE_QOVR を返します。

本サービスコールで自タスクを指定することはできません。

本サービスコールは、タスクコンテキストからは、sus_tsk、非タスクコンテキストからは、isus_tsk を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task()
{
                :
    if( sus_tsk( ID_main ) != E_OK )
        printf(“Can’t suspend task main()¥n”);
                :
}
```

rsm_tsk 強制待ち状態からの再開
irsm_tsk 強制待ち状態からの再開(ハンドラ専用)
frsm_tsk 強制待ち状態からの強制再開
ifrsm_tsk 強制待ち状態からの強制再開(ハンドラ専用)

■ C言語API

```
ER ercd = rsm_tsk( ID tskid );
ER ercd = irsm_tsk( ID tskid );
ER ercd = frsm_tsk( ID tskid );
ER ercd = ifrsm_tsk( ID tskid );
```

■ パラメータ

ID	tskid	対象タスクID番号

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskidのタスクは存在しない)
E_OBJ	オブジェクト状態が不正(tskidのタスクが強制待ち状態でない)

■ 機能説明

tskid で示されたタスクが sus_tsk サービスコールによって中断されている場合、対象タスクの強制待ち状態を解除し、実行を再開します。このとき、対象タスクはレディキューの最後尾につながれます。対象タスクの強制待ち要求ネスト数が1以上の場合は、強制待ち要求ネスト数を1減算します。その結果が0になった場合に強制待ち状態を解除します。

frsm_tsk,ifrsm_tsk サービスコールの場合は、強制待ち要求ネスト数が 1 以上の場合であっても強制待ち要求ネスト数を 0 にクリアし、強制待ち状態を解除します。

対象タスクが、2重待ち状態の場合は、待ち状態に移行させます。

対象タスクが強制待ち(SUSPEND)状態にない場合(休止(DORMANT)状態を含む)に発せられた要求に対してはサービスコール発行タスクにエラーE_OBJ を返します。

なお、対象タスクが未登録状態の場合に、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、rsm_tsk･frsm_tsk、非タスクコンテキストからは、irsm_tsk･ifrsm_tsk を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task1()
{
                :
    if( rsm_tsk( ID_main ) != E_OK )
        printf(“Can’t resume main()¥n”);
                :
                :
    if(frsm_tsk( ID_task2 ) != E_OK )
        printf(“Can’t forced resume task2()¥n”);
                :
}
```

dly_tsk　　タスクの遅延

■ C言語API

```
ER ercd = dly_tsk( TMO tmout );
```

■ パラメータ

TMO	tmout	遅延時間

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_RLWAI	強制待ち解除

■ 機能説明

自タスクの実行を、tmout で指定した時間だけ一時的に停止し、実行(RUN)状態から待ち状態へ移行します。具体的には、tmout で指定した時間経過後の最初のタイムティックで待ち状態が解除されます。そのため、tmout に0が指定された場合は、いったん待ち状態に移行し、最初のタイムティックで待ち状態が解除されます。

本サービスコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本サービスコールを発行したタスクは、タイムアウト待ち行列からはずされ、レディキューに接続されます。

◆ **tmout 経過後、最初のタイムティックが発生した場合**

この時のエラーコードは、E_OK を返します。

◆ **tmout の時間が経過する前に前に rel_wai、irel_wai サービスコールを発行した場合**

この時のエラーコードは、E_RLWAI を返します。

なお、遅延時間中に wup_tsk、iwup_tsk サービスコールが発行されても、待ち解除とはなりません。

tmout の単位は、msです。すなわち、dly_tsk(50);と記述すれば、50ms 間、自タスクが実行(RUN)状態から遅延待ち状態へ移行します。

tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。

本サービスコールはタスクコンテキストからのみ発行してください。非タスクコンテキストから発行した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task()
{
    if( dly_tsk( 10 ) != E_OK );
        printf(“Forced wakeup¥n”);
              :
}
```

1.4. タスク例外処理ルーチン

def_tex　　タスク例外処理ルーチンの定義

■ C 言語 API

```
ER ercd = def_tex( ID tskid, T_DTEX *pk_dtex );
```

■ パラメータ

ID	tskid	生成するタスク ID 番号
T_DTEX	*pk_dtex	タスク例外処理ルーチン定義情報を格納したパケットへのポインタ

pk_dtex の内容

```
typedef   struct t_dtex {
    ATR     texatr   0     4     タスク例外処理ルーチン属性
    FP      texrtn   +4    4     タスク例外処理ルーチンの起動番地
} T_DTEX;
```

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_NOEXS	未登録状態(tskid のタスクは存在しない)

■ 機能説明

tskid で示されたタスク例外処理ルーチンを pk_dtex で示された内容で定義します。tskid=TSK_SELF の指定により自タスクの指定になります。

◆ texatr(タスク例外処理ルーチン属性)

タスク例外処理ルーチンの属性を設定します。texatr には、[TA_HLNG || TA_HASM] を指定できます。カーネルは、TA_HLNG,TA_ASM の値を無視し、C 言語記述、アセンブリ言語の記述を区別しません。

属性	意味
TA_HLNG	タスク例外を高級言語で記述した際に指定します。
TA_ASM	タスク例外をアセンブリ言語で記述した際に指定します。

◆ texrtn(タスク例外処理ルーチン起動番地)

texrtn には、タスク例外処理ルーチンの先頭アドレスを指定します。

def_tex サービスコールでは、pk_dtex＝NULL(0)と指定した場合には tskid のタスク例外処理ルーチンの定義を解除します。この時、タスクの保留例外要因を 0 クリアし、タスクをタスク例外処理禁止状態に移行します。すでにタスク例外処理ルーチンが定義されていた場合は、以前の定義を解除し新しい定義に置き換えます。この時、保留例外要因のクリアとタスク例外処理の禁止は行いません。

本サービスコールはタスクコンテキストからのみ発行してください。非タスクコンテキストから発行した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task()
{
    T_DTEX pk_dtex;
    ER ercd;
            :
    pk_dtex.texrtn = (FP)texrtn1;
    ercd = def_tex( ID_main, &pk_dtex );
            :
}
```

ras_tex　タスク例外処理の要求
iras_tex　タスク例外処理の要求(ハンドラ専用)

■ C言語API

```
ER ercd = ras_tex( ID tskid, TEXPTN rasptn);
ER ercd = iras_tex( ID tskid, TEXPTN rasptn );
```

■ パラメータ

ID	tskid	生成するタスクID番号
TEXPTN	rasptn	要求するタスク例外処理のタスク例外要因

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_OBJ	オブジェクト状態が不正(tskidのタスクが休止状態、またはタスク例外が定義されていない)
E_NOEXS	未登録状態(tskidのタスクは存在しない)

■ 機能説明

tskidで示されたタスクに対して、rasptnで指定されるタスク例外要因によって、タスク例外処理を要求します。つまり、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(OR)に更新します。tskid=TSK_SELFの指定により自タスクの指定になります。ただし、非タスクコンテキストにおいて、tskid に TSK_SELF を指定した場合の動作は保証されません。本サービスコールにより、タスク例外処理ルーチンを起動する条件が揃った場合には、タスク例外処理ルーチンを起動する処理を行います。

tskid で示された対象タスクが休止(DORMANT)状態にある場合は、サービスコールの戻り値としてエラーE_OBJ を返し、未登録状態(NON-EXISTENT)にある場合は、エラーE_NOEXS を返します。rasptn に 0 が指定された場合であっても、エラーコードE_PAR は返しません。

本サービスコールは、タスクコンテキストからは、ras_tex、非タスクコンテキストからは、iras_tex を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task()
{
    TEXPTN texptn;
    ER ercd;
            :
    texptn= 0x00000001;
    ercd = ras_tex( ID_task2, texptn );
            :
}
```

dis_tex　　タスク例外処理の禁止

■ C言語API

```
ER ercd = dis_tex();
```

■ パラメータ

なし

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_OBJ	オブジェクト状態が不正 (自タスクに例外処理ルーチンが定義されていない)

■ 機能説明

自タスクをタスク例外処理禁止状態に移行させます。自タスクに例外処理ルーチンが定義されていない場合は、E_OBJ を返します。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task()
{
            :
    dis_tex();
            :
}
```

ena_tex　　タスク例外処理の許可

■ C言語API

```
ER ercd = ena_tex();
```

■ パラメータ

なし

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■エラーコード

E_OBJ	オブジェクト状態が不正 (自タスクに例外処理ルーチンが定義されていない)

■ 機能説明

自タスクをタスク例外処理許可状態に移行させます。自タスクに例外処理ルーチンが定義されていない場合は、E_OBJ を返します。
本サービスコールにより、タスク例外処理ルーチンを起動する条件がそろった場合は、タスク例外処理ルーチンを起動する処理を行います。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task()
{
            :
    ena_tex();
            :
}
```

sns_tex　　タスク例外禁止状態の参照

■ C 言語 API

```
ER ercd = sns_tex();
```

■ パラメータ

なし

■リターンパラメータ

BOOL	state	タスク例外禁止状態

■エラーコード

なし

■ 機能説明

実行状態のタスクが、タスク例外禁止状態の場合に TRUE、タスク例外許可状態の場合に FALSE を返します。実行状態のタスクとは、タスクコンテキストから呼び出した場合は自タスクであり、非タスクコンテキストから呼び出した場合は非タスクコンテキストに移行する直前に実行していたタスクです。非タスクコンテキストから呼び出された場合で、実行状態のタスクがないときには TRUE を返します。

タスク例外処理ルーチンが定義されていないタスクは、タスク例外処理禁止状態に保たれているため、実行状態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場合には、このサービスコールは TRUE を返します。

本サービスコールは、タスクコンテキスト、非タスクコンテキストにおいて使用可能です。また、ディスパッチ禁止状態、CPU ロック状態においても使用可能です。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task()
{
            :
    if( sns_tex() == TRUE )
        printf(" Task Exception disable¥n");
            :
}
```

ref_tex　タスク例外処理の状態参照
iref_tex　タスク例外処理の状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_tex( ID tskid, T_RTEX *pk_rtex );
ER ercd = iref_tex( ID tskid, T_RTEX *pk_rtex );
```

■ パラメータ

ID	tskid	対象タスクID番号
T_RTEX	*pk_rtex	タスク例外処理状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RTEX	*pk_rtex	タスク例外処理状態を返すパケットへのポインタ

pk_rtexの内容

```
typedef   struct t_rtex{
    STAT    texstat   +0   4    タスク例外処理の状態
    TEXPTN  pndptn    +4   4    保留例外要因
} T_RTEX;
```

■エラーコード

E_OBJ	オブジェクト状態が不正(対象タスクが休止状態、もしくは対象タスクに例外処理ルーチンが定義されていない)
E_NOEXS	未登録状態(tskidのタスクは存在しない)

■ 機能説明

tskidで示されたタスクのタスク例外処理に関する状態を参照し、pk_rtexが指す領域に返します。

texstatには、対象タスクがタスク例外許可状態かタスク例外処理禁止状態かによって次のいずれか
の値を返します。

- TTEX_ENA(0x00000000)タスク例外許可状態
- TTEX_DIS(0x00000001)タスク例外禁止状態

pndptnには、対象タスクの保留例外要因を返します。処理されていない例外処理要求がないときには、pndptnには0を返します。tskid=TSK_SELFの指定により自タスクの指定になります。

本サービスコールは、タスクコンテキストからは、ref_tex、非タスクコンテキストからは、iref_texを使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task()
{
    T_RTEX pk_rtex;
    ER ercd;
            :
    ercd = ref_tex( ID_main, &pk_rtex );
            :
}
```

1.5. 同期･通信機能(セマフォ)

cre_sem　セマフォの生成
acre_sem　セマフォの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_sem( ID semid, T_CSEM *pk_csem );
ER_ID semid = acre_sem( T_CSEM *pk_csem );
```

■ **パラメータ**

ID	semid	生成するセマフォ ID 番号(cre_sem 使用時)
T_CSEM	*pk_csem	セマフォ生成情報を格納した構造体へのポインタ

pk_csem の内容

```
typedef   struct t_csem {
    ATR     sematr   0    4    セマフォ属性
    UINT    isemcnt  +4   4    セマフォ資源数の初期値
    UINT    maxsem   +8   4    セマフォ資源数の最大値
} T_CSEM;
```

■ **リターンパラメータ**

- cre_sem の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_sem の場合

ER_ID	semid>0	生成したセマフォの ID 番号
	semid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_sem 使用時)
E_OBJ	オブジェクト状態が不正 (semid のセマフォが存在(cre_sem 使用時))

■ **機能説明**

cre_sem サービスコールは、semid で示されたセマフォを pk_csem の内容で生成します。acre_sem サービスコールは、未使用のセマフォ ID をセマフォ ID として割り当て、pk_csem の内容でセマフォを生成します。割り当てた ID 番号は、戻り値 semid として返します。生成するセマフォの情報 pk_csem について以下に示します。

◆ **sematr(セマフォ属性)**

sematr には、(TA_TFIFO||TA_TPRI)を指定できます。

属性	意味
TA_TFIFO	待ちタスクのキューイングは FIFO 順
TA_TPRI	待ちタスクのキューイングは優先度順

◆ **isemcnt**

セマフォ生成時の対象セマフォカウンタの初期値をこの領域にセットします。この領域には、0 から 0x7FFFH までの値が設定できます。

◆ **maxsem**

対象セマフォカウンタの最大値をこの領域にセットします。この領域には、0 から 7FFFH までの値が設定できます。

cre_sem サービスコールにおいて、すでに存在するセマフォに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_sem サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。

cre_sem サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大セマフォ数までです。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_sem1 1
void task1()
{
  T_CSEM csem;
  ER ercd;
    csem.isemcnt = 0xff;      /* セマフォカウンタの初期値 */
    csem.maxsem = 0x7fffff;  /* セマフォカウンタの最大値 */
    ercd = cre_sem( ID_sem1, &csem );
              :
}
```

del_sem　　セマフォの削除

■ C言語API

```
ER ercd = del_sem( ID semid );
```

■ パラメータ

ID	semid	削除するセマフォID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(semidのセマフォは存在しない)

■ 機能説明

semidで示されたセマフォを削除します。削除されたセマフォは、再度、同じID番号で新しいセマフォを生成することができます。削除するセマフォに対して、条件成立を待っているタスクが存在しても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、セマフォ待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコードE_DLTが返されます。

存在しないセマフォに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、cre_sem,acre_sem によって生成されたセマフォに対して有効です。コンフィギュレーションファイルに静的に定義されたセマフォに対して本サービスコールを発行した場合の動作は保証されません。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_sem2 2
void task1()
{
      :
    del_sem( ID_sem2 );
      :
}
```

sig_sem　セマフォ資源の返却
isig_sem　セマフォ資源の返却(ハンドラ専用)

■ C言語API

```
ER ercd = sig_sem( ID semid );
ER ercd = isig_sem( ID semid );
```

■ パラメータ

ID	semid	返却するセマフォID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(semidのセマフォは存在しない)
E_QOVR	キューイングオーバーフロー

■ 機能説明

semidで示されたセマフォに対して、資源を1つ返却します。
対象セマフォの待ち行列にタスクがつながれている場合には、行列の先頭タスクを実行可能(READY)状態へ移行します。一方、待ち行列にタスクがつながれていない場合には、そのセマフォの計数値を1だけ増やします 。セマフォの計数値がcre_semあるいはコンフィギュレーションファイルで指定した最大値 (maxsem)を越えて資源の返却(sig_sem、isig_sem サービスコール)をおこなうとセマフォの計数値はそのままで、サービスコール発行タスクにエラーE_QOVRを返します。
存在しないセマフォに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。
本サービスコールは、タスクコンテキストからは、sig_sem、非タスクコンテキストからは、isig_semを使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
                :
     if( sig_sem( ID_sem ) == E_QOVR )
       error("Overflow¥n");
                :
}
```

wai_sem　セマフォ資源の獲得
pol_sem　セマフォ資源の獲得(ポーリング)
ipol_sem　セマフォ資源の獲得(ハンドラ専用)
twai_sem　セマフォ資源の獲得(タイムアウト)

■ C 言語 API

```
ER ercd = wai_sem( ID semid );
ER ercd = pol_sem( ID semid );
ER ercd = ipol_sem( ID semid );
ER ercd = twai_sem( ID semid, TMO tmout );
```

■ パラメータ

ID	semid	返却するセマフォ ID 番号
TMO	tmout	タイムアウト値(twai_sem の場合)

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(semid のセマフォは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

semid で示されたセマフォから、資源を一つ獲得する操作を行います。
そのセマフォの計数値が 1 以上の場合には、計数値を 1 だけ減じてサービスコール発行タスクは実行を継続します。一方、セマフォの計数値が 0 の場合には、wai_sem,twai_sem サービスコール呼び出しタスクは、そのセマフォ待ち行列につなぎます。semid のセマフォ属性が TA_TFIFO の場合は、待ち行列に FIFO 順でタスクをつなぎます。TA_TPRI の場合は、待ち行列に優先度順でタスクをつなぎます。pol_sem,ipol_sem サービスコールでは、直ちにリターンし、エラーE_TMOUT を返します。
twai_sem サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pol_sem と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、wai_sem サービスコールと同じ動作をします。
wai_sem,twai_sem サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、sig_sem、isig_sem サービスコールが発行され、待ち解除条件が満足された場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_sem サービスコールによって待ち状態の対象となっているセマフォが削除された場合**
  この場合、エラーコードは、E_DLT を返します。

存在しないセマフォに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返しま

す。
タスクコンテキストにおいては、wai_sem,twai_sem,pol_sem サービスコール、非タスクコンテキストにおいては、ipol_sem を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
                :
   if( wai_sem( ID_sem ) != E_OK )
       printf("Forced wakeup¥n");
                :
                :
   if( pol_sem( ID_sem ) != E_OK )
       printf("Forced wakeup¥n");
                :
                :
     if( twai_sem( ID_sem, 10 ) != E_OK )
         printf("Forced wakeup"n");
                :
}
```

ref_sem　セマフォの状態参照
iref_sem　セマフォの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_sem( ID semid, T_RSEM *pk_rsem );
ER ercd = iref_sem( ID semid, T_RSEM *pk_rsem );
```

■ パラメータ

ID	semid	対象セマフォID番号
T_RSEM	*pk_rsem	セマフォ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RSEM	*pk_rsem	セマフォ状態を返すパケットへのポインタ

pk_rsemの内容

```
typedef   struct t_rsem{
    ID     wtskid   +0   2    待ちタスクID
    UINT   semcnt   +4   4    現在のセマフォカウンタ値
} T_RSEM;
```

■エラーコード

E_NOEXS	未登録状態(semidのセマフォは存在しない)

■ 機能説明

semidで示されたセマフォの各種の状態を返します。

- **wtskid**
  wtskidには待ち行列の先頭タスクのID番号を返します。
  待ちタスクの無い場合はTSK_NONEを返します。
- **semcnt**
  semcntには、現在のセマフォカウント値を返します。

存在しないセマフォに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、ref_sem、非タスクコンテキストからは、iref_semを使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  T_RSEM   rsem;
      :
    ref_sem( ID_sem ,&rsem );
      :
}
```

1.6. 同期･通信機能(イベントフラグ)

cre_flg　　イベントフラグの生成
acre_flg　　イベントフラグの生成(ID 自動割り当て)

■ C 言語 API

```
ER ercd = cre_flg( ID flgid, T_CFLG *pk_cflg );
ER_ID flgid = acre_flg( T_CFLG *pk_cflg );
```

■ パラメータ

ID	flgid	生成するｲﾍﾞﾝﾄﾌﾗｸﾞ ID 番号(cre_flg 使用時)
T_CFLG	*pk_cflg	ｲﾍﾞﾝﾄﾌﾗｸﾞ生成情報を格納した構造体へのポインタ

pk_cflg の内容

```
typedef   struct t_cflg {
    ATR     flgatr   0     4    ｲﾍﾞﾝﾄﾌﾗｸﾞ属性
    FLGPTN  iflgptn  +4    4    ｲﾍﾞﾝﾄﾌﾗｸﾞの初期値
} T_CFLG;
```

■ リターンパラメータ

- cre_flg の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_flg の場合

ER_ID	flgid>0	生成したｲﾍﾞﾝﾄﾌﾗｸﾞの ID 番号
	flgid<0	エラーコード

■ エラーコード

E_NOID	空き ID なし(acre_flg 使用時)
E_OBJ	ｵﾌﾞｼﾞｪｸﾄ状態が不正 (flgid のｲﾍﾞﾝﾄﾌﾗｸﾞが存在(cre_flg 使用時))

■ 機能説明

cre_flg ｻｰﾋﾞｽｺｰﾙは、flgid で示されたｲﾍﾞﾝﾄﾌﾗｸﾞを pk_cflg の内容で生成します。
acre_flg ｻｰﾋﾞｽｺｰﾙは、未使用のｲﾍﾞﾝﾄﾌﾗｸﾞ ID をｲﾍﾞﾝﾄﾌﾗｸﾞ ID として割り当て、pk_cflg の内容でｲﾍﾞﾝﾄﾌﾗｸﾞを生成します。割り当てた ID 番号を、戻り値 flgid として返します。生成するｲﾍﾞﾝﾄﾌﾗｸﾞの情報 pk_cflg について以下に示します。

◆ **flgatr(ｲﾍﾞﾝﾄﾌﾗｸﾞ属性)**

flgatr には、((TA_TFIFO||TA_TPRI)|(TA_WSGL||TA_WMUL)|[TA_CLR])を指定できます。
TA_WSGL 属性のｲﾍﾞﾝﾄﾌﾗｸﾞでは、待つことの出来るﾀｽｸは一つのみになります。この場合は、TA_TFIFO と TA_TPRI のどちらの属性を指定しても動作は同じになります。
TA_CLR 属性が指定された場合、ﾀｽｸをｲﾍﾞﾝﾄﾌﾗｸﾞ待ち状態から解除する際にｲﾍﾞﾝﾄﾌﾗｸﾞのすべてのﾋﾞｯﾄﾊﾟﾀｰﾝをｸﾘｱします。

属性	意味
TA_TFIFO	待ちﾀｽｸのｷｭｰｲﾝｸﾞは FIFO 順
TA_TPRI	待ちﾀｽｸのｷｭｰｲﾝｸﾞは優先度順
TA_WSGL	複数ﾀｽｸの待ちを禁止
TA_WMUL	複数ﾀｽｸの待ちを許可
TA_CLR	ｸﾘｱ指定

◆ **iflgptn**

ｲﾍﾞﾝﾄﾌﾗｸﾞ生成時の対象ｲﾍﾞﾝﾄﾌﾗｸﾞのﾋﾞｯﾄﾊﾟﾀｰﾝをこの領域にｾｯﾄします。
cre_flg ｻｰﾋﾞｽｺｰﾙにおいて、すでに存在するｲﾍﾞﾝﾄﾌﾗｸﾞに対して、本ｻｰﾋﾞｽｺｰﾙ

を発行した場合は、エラーE_OBJ を返します。acre_flg サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。
cre_flg サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大イベントフラグ数までです。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_flg1 1
#define ID_flg2 2
void task1()
{
  T_CFLG cflg1;
  T_CFLG cflg2={TA_TFIFO,0xfff};
  ER ercd;
  ER_ID flgid;
            :
  cflg1.iflgptn = 0xff;    /* イベントフラグの初期ビットパターン設定 */
  ercd = cre_flg( ID_flg1, &cflg1 );
            :
  flgid = acre_flg( &cflg2 );
}
```

del_flg　　イベントフラグの削除

■ C言語API

```
ER ercd = del_flg( ID flgid );
```

■ パラメータ

ID	flgid	削除するイベントフラグID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(flgidのタスクは存在しない)

■ 機能説明

flgidで示されたイベントフラグを削除します。削除されたイベントフラグは、再度、同じID番号で新しいイベントフラグを生成することができます。削除するイベントフラグに対して、条件成立を待っているタスクが存在しても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、イベントフラグ待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコードE_DLTが返されます。

存在しないイベントフラグに対して、本サービスコールを発行した場合は、エラー E_NOEXSを返します。

本サービスコールは、cre_flg,acre_flg によって生成されたイベントフラグに対して有効です。コンフィギュレーションファイルに静的に定義されたイベントフラグに対して本サービスコールを発行した場合の動作は保証されません。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_flg2 2
void task1()
{
      :
    del_flg( ID_flg2 );
      :
    ext_tsk();
}
```

set_flg　　イベントフラグのセット
iset_flg　　イベントフラグのセット(ハンドラ専用)

■ C言語API

```
ER ercd = set_flg( ID flgid, FLGPTN setptn );
ER ercd = iset_flg( ID flgid, FLGPTN setptn );
```

■ パラメータ

ID	flgid	返却するイベントフラグID番号
FLGPTN	setptn	セットするビットパターン

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(flgidのタスクは存在しない)

■ 機能説明

flgidで示される32ビットのイベントフラグのうち、setptnで示されているビットをセットします。つまり、flgidで示されるイベントフラグの値に対してsetptnの論理和(OR)をとります。イベントフラグ値の変更の結果、wai_flg、twai_flg サービスコールによってそのイベントフラグを待っていたタスクの待ち解除の条件を満たすようになれば、そのタスクの待ちを解除して実行可能(READY)状態もしくは実行(RUN)状態へ移行します 。

待ち解除条件は、待ち行列の先頭から順に評価されます。イベントフラグ属性として、TA_WMULが指定されている場合、イベントフラグは、一回の set_flg,iset_flg サービスコール発行で、複数タスクが同時に待ち解除することができます。また、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は、条件を満たしたタスクの待ち状態を解除した後、イベントフラグのすべてのビットパターンをクリアし、サービスコールの処理を終了します。

setptnの全ビットを0とした場合は、対象イベントフラグに対して何の操作も行いませんが、エラーとはなりません。なお、存在しないイベントフラグに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、set_flg、非タスクコンテキストからは、iset_flg を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
                :
   set_flg( ID_flg,(UINT)0xff00 );
                :
   ext_tsk();
}
```

clr_flg　イベントフラグのクリア
iclr_flg　イベントフラグのクリア(ハンドラ専用)

■ C言語API

```
ER ercd = clr_flg( ID flgid, FLGPTN clrptn );
ER ercd = iclr_flg( ID flgid, FLGPTN clrptn );
```

■ パラメータ

ID	flgid	返却するイベントフラグID番号
FLGPTN	clrptn	クリアするビットパターン

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(flgidのイベントフラグは存在しない)

■ 機能説明

flgidで示される32ビットイベントフラグのうち、対応するclrptnの0になっているビットをクリアします。つまり、flgid で示されるイベントフラグビットパターンを、clrptn 値との論理積(AND)に更新します。clrptn の全ビットを 1 とした場合、イベントフラグに対して何の操作も行なわないことになりますが、エラーにはなりません。
存在しないイベントフラグに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。
本サービスコールは、タスクコンテキストからは、clr_flg、非タスクコンテキストからは、iclr_flg を使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
      :
  clr_flg( ID_flg, (UINT)0xff00 );
      :
}
```

wai_flg　イベントフラグ待ち
pol_flg　イベントフラグ待ち(ポーリング)
ipol_flg　イベントフラグ待ち(ハンドラ専用)
twai_flg　イベントフラグ待ち(タイムアウト)

■ C言語API

```
ER ercd = wai_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN
*p_flgptn );
ER ercd = pol_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN
*p_flgptn );
ER ercd = ipol_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN
*p_flgptn );
ER ercd = twai_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn,
TMO tmout );
```

■ パラメータ

ID	flgid	返却するイベントフラグ ID 番号
FLGPTN	waiptn	待ちビットパターン
MODE	wfmode	待ちモード
FLGPTN	*p_flgptn	待ち解除時のビットパターンを返す領域へのポインタ
TMO	tmout	タイムアウト値(twai_flg の場合)

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
FLGPTN	*p_flgptn	待ち解除時のビットパターンを返す領域へのポインタ

■ エラーコード

E_NOEXS	未登録状態(flgid のイベントフラグは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
E_ILUSE	サービスコール不正使用 (TA_WSGL 属性のイベントフラグに待ちタスクが存在)

■ 機能説明

flgid で示されるイベントフラグにおいて、waiptn で指定したビットが wfmode で示される待ち解除条件にしたがってセットされるのを待ちます。p_flgptn の指す領域には、待ち解除されるときのイベントフラグのビットパターンを返します。

対象イベントフラグが TA_WSGL 属性の場合、既に他のタスクが待っている場合は、エラー E_ILUSE を返します。

本サービスコール呼び出し時にすでに待ち解除条件を満たしている場合は、すぐにリターンし、E_OK を返します。待ち解除条件を満たしていない時、wai_flg,twai_flg サービスコールの場合は、イベントフラグ待ち行列につながれます。その際、flgid のイベントフラグ属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。pol_flg,ipol_flg サービスコールの場合は、直ちにリターンし、エラー E_TMOUT を返します。

twai_flg サービスコールの場合は、tmout に待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。

tmout に TMO_POL(=0)を指定した場合は、タイムアウト値として 0 を指定したことを示し、pol_flg と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定

で、wai_flg サービスコールと同じ動作になります。

wai_flg,twai_flg サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、待ち解除条件が成立した場合**
  この時のエラーコードは、E_OK を返します。
- **待ち解除条件が満たされないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この時のエラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この時のエラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_flg サービスコールによって待ち状態の対象となっているイベントフラグが削除された場合**
  この時のエラーコードは、E_DLT を返します。

wfmode の指定方法および各モードの意味は以下のとおりです。

wfmode(待ちモード)	意味
TWF_ANDW	waiptn で指定したビットが全てセットされるのを待つ(AND 待ち)。
TWF_ORW	waiptn で指定したビットのいずれかがセットされるの待つ(OR 待ち)。

タスクコンテキストにおいては、wai_flg,twai_flg,pol_flg サービスコール、非タスクコンテキストにおいては、ipol_flg を使用してください。

■ 使用例

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  UINT flgptn;
            :
    if(wai_flg(&flgptn,ID_flg2,(UINT)0x0ff0,TWF_ANDW)!=E_OK)
      error("Wait Released¥n");
            :
            :
    if(pol_flg(&flgptn,ID_flg2,(UINT)0x0ff0,TWF_ORW)!=E_OK)
      printf("Not set EventFlag¥n");
            :
            :
    if( twai_flg(&flgptn, ID_flg2,(UINT)0x0ff0, TWF_ANDW, 5) != E_OK )
      error("Wait Released¥n");
            :
}
```

ref_flg　イベントフラグの状態参照
iref_flg　イベントフラグの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_flg( ID flgid, T_RFLG *pk_rflg );
ER ercd = iref_flg( ID flgid, T_RFLG *pk_rflg );
```

■ パラメータ

ID	flgid	対象イベントフラグ ID 番号
T_RFLG	*pk_rflg	イベントフラグ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RFLG	*pk_rflg	イベントフラグ状態を返すパケットへのポインタ

pk_rflg の内容

```
typedef   struct t_rflg{
    ID      wtskid   +0   2    待ちタスク ID
    FLGPTN  flgptn   +4   4    現在のイベントフラグビットパターン
} T_RFLG;
```

■エラーコード

E_NOEXS	未登録状態(flgid のイベントフラグは存在しない)

■ 機能説明

flgid で示されたイベントフラグの各種の状態を返します。

- **wtskid**
  wtskid には待ち行列の先頭タスクの ID 番号を返します。
  待ちタスクの無い場合は TSK_NONE を返します。
- **flgptn**
  flgptn には、現在のイベントフラグビットパターンを返します。

存在しないイベントフラグに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。
本サービスコールは、タスクコンテキストからは、ref_flg、非タスクコンテキストからは、iref_flg を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    T_RFLG rflg;
    ref_flg( ID_flg, &rflg );
              :
}
```

1.7. 同期･通信機能(データキュー)

cre_dtq　データキューの生成
acre_dtq　データキューの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_dtq( ID dtqid, T_CDTQ *pk_cdtq );
ER_ID dtqid = acre_dtq( T_CDTQ *pk_cdtq );
```

■ **パラメータ**

ID	dtqid	生成するデータキューID 番号(cre_dtq 使用時)
T_CDTQ	*pk_cdtq	データキュー生成情報を格納した構造体へのポインタ

pk_cdtqの内容

```
typedef   struct t_cdtq {
    ATR     dtqatr   0     4     データキュー属性
    UINT    dtqcnt   +4    4     データキュー領域のサイズ(データ個数
    VP      dtq      +8    4     データキュー領域の先頭アドレス
} T_CDTQ;
```

■ **リターンパラメータ**

- cre_dtq の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_dtq の場合

ER_ID	dtqid>0	生成したデータキューの ID 番号
	dtqid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_dtq 使用時)
E_OBJ	オブジェクト状態が不正 (dtqid のデータキューが存在(cre_dtq 使用時))
E_NOMEM	空きメモリ無し

■ **機能説明**

cre_dtq サービスコールは、dtqid で示されたデータキューを pk_cdtq の内容で生成します。acre_dtq サービスコールは、未使用のデータキューID をデータキューID として割り当て、pk_cdtq の内容でデータキューを生成します。割り当てた ID 番号を、戻り値 dtqid として返します。生成するデータキューの情報 pk_cdtq について以下に示します。

◆ **dtqatr(データキュー属性)**

dtqatr には、((TA_TFIFO||TA_TPRI) | [__MR_INT || __MR_EXT])を指定できます。

データキューのデータ受信待ち行列は、dtqatr の値にかかわらず常に FIFO 待ちとなります。また、データキューに送信されるデータは優先度を持たず、FIFO で管理されます。

属性	意味
TA_TFIFO	待ちタスクのキューイングは FIFO 順
TA_TPRI	待ちタスクのキューイングは優先度順
__MR_INT	データキュー領域を内蔵 RAM から確保する
__MR_EXT	データキュー領域を外部 RAM から確保する

◆ **dtqcnt**

データキュー領域に蓄積可能なデータの数を指定します。dtqcnt に 0 を指定することの可能です。この場合、データ送信タスクとデータ受信タスクは完全に同期した動作となります。

◆**dtq**

ユーザが確保したデータキュー領域の先頭アドレスを指定します。この値が NULL の場合は、dtqatr の値(__MR_INT,__MR_EXT)に応じてカーネルがデータキュー領域を割り当てます。NULL でない場合は、dtqatr の__MR_INT、__MR_EXT の値は、無視されます。

cre_dtq サービスコールにおいて、すでに存在するデータキューに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_dtq サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。
cre_dtq サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大データキュー数までです。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_dtq1 1
void task1()
{
   T_CDTQ setdtq;
              :
     setdtq.dtqatr = __MR_EXT;
     setdtq.dtqcnt = 10;   /* データ個数 */
     setdtq.dtq = NULL;
     cre_dtq( ID_dtq1, &setdtq );
              :
     ext_tsk();
}
```

del_dtq　　データキューの削除

■ C言語API

```
ER ercd = del_dtq( ID dtqid );
```

■ パラメータ

ID	dtqid	削除するデータキューID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(dtqidのデータキューは存在しない)

■ 機能説明

dtqid で示されたデータキューを削除します。削除されたデータキューは、データキュー領域が解放され、再度、同じ ID 番号で新しいデータキューを生成することができます。削除するデータキューに対して、データ受信を待っているタスク、データ送信を待っているタスクが存在しても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、データ受信待ちもしくはデータ送信待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコード E_DLT が返されます。

存在しないデータキューに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、cre_dtq,acre_dtq によって生成されたデータキューに対して有効です。コンフィギュレーションファイルに静的に定義されたデータキューに対して本サービスコールを発行した場合の動作は保証されません。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_dtq2 2
void task1(void)
{
      :
    del_dtq( ID_dtq 2 );
      :
    ext_tsk();
}
```

snd_dtq	**データキューへのデータ送信**
psnd_dtq	**データキューへのデータ送信(ポーリング)**
ipsnd_dtq	**データキューへのデータ送信(ハンドラ専用)**
tsnd_dtq	**データキューへのデータ送信(タイムアウト)**
fsnd_dtq	**データキューへのデータ強制送信**
ifsnd_dtq	**データキューへのデータ強制送信(ハンドラ専用)**

■ C言語API

```
ER ercd = snd_dtq( ID dtqid, VP_INT data );
ER ercd = psnd_dtq( ID dtqid, VP_INT data );
ER ercd = ipsnd_dtq( ID dtqid, VP_INT data );
ER ercd = tsnd_dtq( ID dtqid, VP_INT data, TMO tmout );
ER ercd = fsnd_dtq( ID dtqid, VP_INT data );
ER ercd = ifsnd_dtq( ID dtqid, VP_INT data );
```

■ パラメータ

ID	dtqid	送信対象のデータキューID番号
TMO	tmout	タイムアウト値(tsnd_dtqの場合)
VP_INT	data	送信するデータ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(dtqidのデータキューは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
E_ILUSE	サービスコール不正使用(dtqcntが0のデータキューに対してfsnd_dtq,ifsnd_dtqを発行)
EV_RST	データキュー領域クリアによって待ち状態が解除された

■ 機能説明

dtqidで示されたデータキューに対して、dataで示された4バイトのデータを送信します。対象データキューに受信待ちタスクが存在する場合は、データキューにデータを格納せず、受信待ち行列の先頭タスクにデータを送信し、そのタスクの受信待ち状態を解除します。
一方、既にデータで一杯になったデータキューに対して、snd_dtq,tsnd_dtq を発行した場合、これらのサービスコールを発行したタスクは、実行状態からデータ送信待ち状態に移行し、データキューの空きを待つ送信待ち行列につながれます。その際、dtqid のデータキュー属性がTA_TFIFOの場合は、FIFO順で待ち行列にタスクをつなぎ、TA_TPRIの場合は、優先度順でタスクをつなぎます。psnd_dtq,ipsnd_dtq の場合は、直ちにリターンし、エラーE_TMOUTを返します。
tsnd_dtq サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmoutに指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義のtick_denoの値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、psnd_dtq と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、snd_dtq サービスコールと同じ動作をします。
受信待ちタスクがなく、データキュー領域も一杯でない場合、送信したデータはデータキューに格納されます。
snd_dtq,tsnd_dtq サービスコール実行による待ち状態は、以下に示す場合に解除されま

す。

- **tmout の時間が経過する前に、rcv_dtq,trcv_dtq,prcv_dtq,iprcv_dtq サービスコールが発行され、待ち解除条件が満足された場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_dtq サービスコールによって待ち状態の対象となっているデータキューが削除された場合**
  この場合、エラーコードは、E_DLT を返します。
- **他のタスクから発行した vrst_dtq サービスコールによって待ち状態の対象となっているデータキューが初期化された場合**
  この場合、エラーコードは、EV_RST を返します。

fsnd_dtq,ifsnd_dtq の場合は、データキューの先頭(最古)のデータを削除し、送信データをデータキュー末尾に格納します。データキュー領域がデータで一杯になっていない場合は、fsnd_dtq,ifsnd_dtq は、snd_dtq と同じ動作を行います。
存在しないデータキューに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。
タスクコンテキストにおいては、snd_dtq,tsnd_dtq,psnd_dtq,fsnd_dtq サービスコール、非タスクコンテキストにおいては、ipsnd_dtq,ifsnd_dtq を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
VP_INT data[10];
void task(void)
{
            :
    if( snd_dtq( ID_dtq, data[0]) == E_RLWAI ){
      error("Forced released¥n");
  }
      :
            :
    if( psnd_dtq( ID_dtq, data[1])== E_TMOUT ){
      error("Timeout¥n");
  }
      :
            :
    if( tsnd_dtq( ID_dtq, data[2], 10 ) != E_ TMOUT ){
      error("Timeout ¥n");
  }
      :
            :
    if( fsnd_dtq( ID_dtq, data[3]) != E_OK ){
      error("error¥n");
  }
      :
}
```

rcv_dtq　データキューからのデータ受信
prcv_dtq　データキューからのデータ受信(ポーリング)
iprcv_dtq　データキューからのデータ受信(ハンドラ専用)
trcv_dtq　データキューからのデータ受信(タイムアウト)

■ C言語API

```
ER ercd = rcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = prcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = iprcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = trcv_dtq( ID dtqid, VP_INT *p_data, TMO tmout );
```

■ パラメータ

ID	dtqid	受信対象のデータキューID番号
TMO	tmout	タイムアウト値(trcv_dtqの場合)
VP_INT	*p_data	受信データを格納する領域先頭へのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
VP_INT	*p_data	受信データを格納する領域先頭へのポインタ

■ エラーコード

E_NOEXS	未登録状態(semidのデータキューは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

dtqidで示されたデータキューから、データを受信し、p_dataの指す領域に格納します。対象データキューにデータが存在する場合は、その先頭の(最古の)データを受信します。この結果、データキュー領域に空きが発生するため、送信待ち行列につながれているタスクは、その送信待ち状態が解除され、データキュー領域へのデータを送信します。

データキューにデータが存在せず、データ送信待ちタスクが存在する場合(データキュー領域の容量が 0 の場合)、データ送信待ち行列先頭タスクのデータを受信します。この結果、そのデータ送信待ちタスクの待ち状態は解除されます。

一方、データキュー領域にデータが格納されていないデータキューに対して、rcv_dtq,trcv_dtq を発行した場合、これらのサービスコールを発行したタスクは、実行状態からデータ受信待ち状態に移行し、データ受信待ち行列につながれます。このとき、受信待ち行列へは、FIFO 順でつながれます。prcv_dtq,iprcv_dtq の場合は、直ちにリターンし、エラーE_TMOUT を返します。

trcv_dtq サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、prcv_dtq と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、rcv_dtq サービスコールと同じ動作をします。

rcv_dtq,trcv_dtq サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、rcv_dtq,trcv_dtq,prcv_dtq,iprcv_dtq サービスコールが発行され、待ち解除条件が満足された場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。

- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_dtq サービスコールによって待ち状態の対象となっているデータキューが削除された場合**
  この場合、エラーコードは、E_DLT を返します。

存在しないデータキューに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。
タスクコンテキストにおいては、rcv_dtq,trcv_dtq,prcv_dtq サービスコール、非タスクコンテキストにおいては、iprcv_dtq を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  VP_INT data;
            :
    if( rcv_dtq( ID_dtq, &data ) != E_RLWAI )
        error("forced wakeup¥n");
            :
    if( prcv_dtq( ID_dtq, &data ) != E_TMOUT )
        error("Timeout¥n");
            :
    if( trcv_dtq( ID_dtq, &data, 10 ) != E_DLT )
        error("Dataqueue deleted¥n");
            :
}
```

ref_dtq　データキューの状態参照
iref_dtq　データキューの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_dtq( ID dtqid, T_RDTQ *pk_rdtq );
ER ercd = iref_dtq( ID dtqid, T_RDTQ *pk_rdtq );
```

■ パラメータ

ID	dtqid	対象データキューID番号
T_RDTQ	*pk_rdtq	データキュー状態を返すパケットへのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RDTQ	*pk_rdtq	データキュー状態を返すパケットへのポインタ

pk_rdtqの内容

```
typedef   struct t_rdtq{
    ID      stskid   +0   2    送信待ちタスクID
    ID      rtskid   +0   2    受信待ちタスクID
    UINT    sdtqcnt  +4   4    データキューに入っているデータ数
} T_RDTQ;
```

■ エラーコード

E_NOEXS	未登録状態(dtqidのデータキューは存在しない)

■ 機能説明

dtqidで示されたデータキューの各種の状態を返します。

- **stskid**
  stskidには送信待ち行列の先頭タスクのID番号を返します。待ちタスクの無い場合はTSK_NONEを返します。
- **rtskid**
  rtskidには受信待ち行列の先頭タスクのID番号を返します。待ちタスクの無い場合はTSK_NONEを返します。
- **sdtqcnt**
  sdtqcntには、データキュー領域に格納されているデータ個数を返します。

存在しないデータキューに対して、本サービスコールを発行した場合は、エラーE_NOEXSを返します。
本サービスコールは、タスクコンテキストからは、ref_dtq、非タスクコンテキストからは、iref_dtqを使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  T_RDTQ rdtq;
      :
  ref_dtq( ID_dtq ,&rdtq );
      :
}
```

1.8. 同期･通信機能(メールボックス)

cre_mbx　メールボックスの生成
acre_mbx　メールボックスの生成(ID 自動割り当て)

■ C 言語 API

```
ER ercd = cre_mbx( ID mbxid, T_CMBX *pk_cmbx );
ER_ID mbxid = acre_mbx( T_CMBX *pk_cmbx );
```

■ パラメータ

ID	mbxid	生成するメールボックス ID 番号(cre_mbx 使用時)
T_CMBX	*pk_cmbx	メールボックス生成情報を格納した構造体へのポインタ

pk_cmbx の内容

```
typedef   struct t_cmbx {
    ATR     mbxatr   0     4    メイルボックス属性
    PRI     maxmpri  +4    2    メッセージ優先度の最大値
    VP      mprihd   +8    4    優先度別メッセージキューヘッダの先頭アドレス
} T_CMBX;
```

■ リターンパラメータ

- cre_mbx の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_mbx の場合

ER_ID	mbxid>0	生成したメールボックスの ID 番号
	mbxid<0	エラーコード

■ エラーコード

E_NOID	空き ID なし(acre_mbx 使用時)
E_OBJ	オブジェクト状態が不正 (mbxid のデータキューが存在(cre_mbx 使用時))

■ 機能説明

cre_mbx サービスコールは、mbxid で示されたメールボックスを pk_cmbx の内容で生成します。acre_mbx サービスコールは、未使用のメールボックス ID をメールボックス ID として割り当て、pk_cmbx の内容でメールボックスを生成します。割り当てた ID 番号を、戻り値 mbxid として返します。生成するメールボックスの情報 pk_cmbx について以下に示します。

◆ **mbxatr(データキュー属性)**

mbxatr には、((TA_TFIFO||TA_TPRI) | (TA_MFIFO || TA_MPRI))を指定できます。

属性	意味
TA_TFIFO	待ちタスクのキューイングは FIFO 順
TA_TPRI	待ちタスクのキューイングは優先度順
TA_MPRI	メッセージキューのキューイングは優先度順
TA_MFIFO	メッセージキューのキューイングは FIFO 順

◆ **maxmpri**

メッセージ優先度の最大値を設定します。この値は、M3T-MR32R では使用しません。

◆ **mprihd**

優先度別メッセージキューヘッダの先頭アドレスを設定します。この値は、M3T-MR32R では使用しません。

cre_mbx サービスコールにおいて、すでに存在するメールボックスに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_mbx サービスコールにおいて空きIDが存

在しない場合は、エラーE_NOID を返します。
cre_mbx サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大メールボックス数までです。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mbx1 1
void task1()
{
    T_CMBX pk_cmbx;
            :
    pk_cmbx.mbxatr = TA_TFIFO;
    pk_cmbx.mbxpri = 5;   /* メッセージ優先度最大値 */
    cre_mbx( ID_mbx1, &pk_cmbx );
             :
    ext_tsk();
}
```

del_mbx　　メールボックスの削除

■ C言語API

```
ER ercd = del_mbx( ID mbxid );
```

■ パラメータ

ID	mbxid	削除するメールボックスID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mbxidのメールボックスは存在しない)

■ 機能説明

mbxid で示されたメールボックスを削除します。削除されたメールボックスは、再度、同じ ID 番号で新しいメールボックスを生成することができます。削除するメールボックスに対して、メッセージ受信を待っているタスクが存在しても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、メッセージ受信待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコードE_DLTが返されます。

存在しないメールボックスに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、cre_mbx,acre_mbx によって生成されたメールボックスに対して有効です。コンフィギュレーションファイルに静的に定義されたメールボックスに対して本サービスコールを発行した場合の動作は保証されません。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mbx2 2
void task1(void)
{
      :
    del_mbx( ID_mbx2 );
      :
    ext_tsk();
}
```

snd_mbx　メールボックスへの送信
isnd_mbx　メールボックスへの送信(ハンドラ専用)

■ C言語API

```
ER ercd = snd_mbx( ID mbxid, T_MSG *pk_msg );
ER ercd = isnd_mbx( ID mbxid, T_MSG *pk_msg );
```

■ パラメータ

ID	mbxid	送信対象のメールボックスID番号
T_MSG	*pk_msg	送信するメッセージ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ メッセージパケットの構造

<<メールボックスのメッセージヘッダ>>

```
typedef   struct t_msg{
    VP      msghead   +0    4     カーネル管理領域
} T_MSG;
```

<<メールボックスの優先度付きメッセージヘッダ>>

```
typedef   struct t_msg_pri{
    T_MSG   msgque    +0    4     メッセージヘッダ
    PRI     msgpri    +4    2     メッセージ優先度
} T_MSG_PRI;
```

■ エラーコード

E_NOEXS	未登録状態(mbxidのメールボックスは存在しない)

■ 機能説明

mbxid で示されたメールボックスに pk_msg で示されたメッセージを送信します。すでに対象メールボックスにメッセージの受信を待つタスクが存在していれば、待ち行列先頭のタスクに送信したメッセージが渡され、そのタスクの待ち状態が解除されます。
TA_MFIFO 属性のメールボックスにメッセージを送る場合は、以下の例に示すように先頭にT_MSG 構造体を付加した形式で、メッセージを作成してください。
TA_MPRI 属性のメールボックスにメッセージを送る場合は、以下の例に示すように先頭にT_MSG_PRI 構造体を付加した形式で、メッセージを作成してください。
TA_MFIFO, TA_MPRI いずれの属性の場合もメッセージは RAM 領域に作成してください。
T_MSG の領域はカーネルが使用するため、送信後は書き換えてはなりません。メッセージ送信後、メッセージが受信される前にこの領域を書き換えた場合の動作は保証されません。
タスクコンテキストにおいては、snd_mbx サービスコール、非タスクコンテキストにおいては、isnd_mbx を使用してください。

<<メッセージの形式例>>

```
typedef struct user_msg{
        T_MSG  t_msg;         /* T_MSG 構造体 */
        B      data[16];      /* ユーザーメッセージデータ */
} USER_MSG;
```

<<優先度付きメッセージの形式例>>

```
typedef struct user_msg{
        T_MSG_PRI     t_msg;         /* T_MSG_PRI 構造体 */
        B             data[16];      /* ユーザーメッセージデータ */
} USER_MSG;
```

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
typedef struct pri_message
{
       T_MSG_PRI      msgheader;
       char   body[12];
} PRI_MSG;

void task(void)
{
  PRI_MSG     msg;
              :
  msg.msgpri = 5;
    if( snd_mbx( ID_msg,(T_MSG)&msg) != E_OK ){
      error("nonexistent¥n");
  }
      :
}
```

rcv_mbx　メールボックスからの受信
prcv_mbx　メールボックスからの受信(ポーリング)
iprcv_mbx　メールボックスからの受信(ハンドラ専用)
trcv_mbx　メールボックスからの受信(タイムアウト)

■ C 言語 API

```
ER ercd = rcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = prcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = iprcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = trcv_mbx( ID mbxid, T_MSG **ppk_msg, TMO tmout );
```

■ パラメータ

ID	mbxid	送信対象のメールボックス ID 番号
TMO	tmout	タイムアウト値(trcv_mbx の場合)
T_MSG	**ppk_msg	受信メッセージを格納する領域先頭へのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_MSG	**ppk_msg	受信メッセージを格納する領域先頭へのポインタ

■ エラーコード

E_NOEXS	未登録状態(mbxid のメールボックスは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

mbxid で示されたメールボックスから、メッセージを受信し、ppk_msg の指す領域に受信したメッセージの先頭アドレスを格納します。対象メールボックスにデータが存在する場合は、その先頭の(最古の)データを受信します。

一方、メールボックスにメッセージがないメールボックスに対して、rcv_mbx,trcv_mbx を発行した場合、これらのサービスコールを発行したタスクは、実行状態からメッセージ受信待ち状態に移行し、メッセージ受信待ち行列につながれます。その際、mbxid のメールボックス属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。prcv_mbx,iprcv_mbx の場合は、直ちにリターンし、エラーE_TMOUT を返します。

trcv_mbx サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、prcv_mbx と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、rcv_mbx サービスコールと同じ動作をします。

rcv_mbx,trcv_mbx サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、rcv_mbx,trcv_mbx,prcv_mbx,iprcv_mbx サービスコールが発行され、待ち解除条件が満足された場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。

◆ **他のタスクから発行した del_mbx サービスコールによって待ち状態の対象となっているメールボックスが削除された場合**

この場合、エラーコードは、E_DLT を返します。

存在しないメールボックスに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

タスクコンテキストにおいては、rcv_mbx,trcv_mbx,prcv_mbx サービスコール、非タスクコンテキストにおいては、iprcv_mbx を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

typedef struct fifo_message
{
       T_MSG   head;
       char    body[12];
} FIFO_MSG;
void task()
{
   FIFO_MSG *msg;
              :
     if( rcv_mbx((T_MSG *)&msg, ID_vmbx) == E_RLWAI )
         error("forced wakeup¥n");
              :
              :
     if( prcv_mbx((T_MSG *)&msg, ID_vmbx) != E_TMOUT )
         error("Timeout¥n");
              :
              :
     if( trcv_mbx((T_MSG *)&msg, ID_vmbx,10) != E_DLT )
         error("Mailbox deleted¥n");
              :
}
```

ref_mbx　メールボックスの状態参照
iref_mbx　メールボックスの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_mbx( ID mbxid, T_RMBX *pk_rmbx );
ER ercd = iref_mbx( ID mbxid, T_RMBX *pk_rmbx );
```

■ パラメータ

ID	mbxid	対象メールボックスID番号
T_RMBX	*pk_rmbx	メールボックス状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RMBX	*pk_rmbx	メールボックス状態を返すパケットへのポインタ

pk_rmbxの内容

```
typedef   struct t_rmbx{
    ID      wtskid   +0   2    受信待ちタスクID
    T_MSG   *pk_msg  +4   4    次に受信されるメッセージパケット
} T_RMBX;
```

■エラーコード

E_NOEXS	未登録状態(mbxidのメールボックスは存在しない)

■ 機能説明

mbxidで示されたメールボックスの各種の状態を返します。

- **wtskid**
  wtskid には受信待ち行列の先頭タスクの ID 番号を返します。待ちタスクの無い場合はTSK_NONEを返します。
- ***pk_msg**
  次に受信されるメッセージの先頭アドレスを返します。次に受信されるメッセージがない場合は、NULLを返します。

存在しないメールボックスに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、ref_mbx、非タスクコンテキストからは、iref_mbxを使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
   T_RMBX pk_rmbx;
       :
   ref_mbx(ID_mbx,&pk_rmbx);
       :
}
```

1.9. 拡張同期・通信機能(メッセージバッファ)

cre_mbf　メッセージバッファの生成
acre_mbf　メッセージバッファの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_mbf( ID mbfid, T_CMBF *pk_cmbf );
ER_ID mbfid = acre_mbf( T_CMBF *pk_cmbf );
```

■ **パラメータ**

ID	mbfid	生成するデータキューID 番号(cre_mbf 使用時)
T_CMBF	*pk_cmbf	メッセージバッファ生成情報を格納した構造体へのポインタ

pk_cmbf の内容

```
typedef   struct t_cmbf {
    ATR     mbfatr   0     4     データキュー属性
    UINT    maxmsz   +4    4     メッセージの最大サイズ(バイト数)
    SIZE    mbfsz    +8    4     メッセージバッファ領域のサイズ(バイト数)
    VP      mbf      +12   4     メッセージバッファ領域の先頭アドレス
} T_CMBF;
```

■ **リターンパラメータ**

- cre_mbf の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_mbf の場合

ER_ID	mbfid>0	生成したメッセージバッファの ID 番号
	mbfid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_mbf 使用時)
E_OBJ	オブジェクト状態が不正 (mbfid のメッセージバッファが存在(cre_mbf 使用時))
E_NOMEM	空きメモリ無し

■ **機能説明**

cre_mbf サービスコールは、mbfid で示されたメッセージバッファを pk_cmbf の内容で生成します。acre_mbf サービスコールは、未使用のメッセージバッファ ID をメッセージバッファ ID として割り当て、pk_cmbf の内容でメッセージバッファを生成します。割り当てた ID 番号を、戻り値 mbfid として返します。生成するメッセージバッファの情報 pk_cmbf について以下に示します。

◆ **mbfatr(メッセージバッファ属性)**

mbfatr には、((TA_TFIFO||TA_TPRI) | [__MR_INT || __MR_EXT])を指定できます。
メッセージバッファのメッセージ受信待ち行列は、mbfatr の値にかかわらず常に FIFO 待ちとなります。また、メッセージバッファに送信されるメッセージは優先度を持たず、FIFO で管理されます。

属性	意味
TA_TFIFO	待ちタスクのキューイングは FIFO 順
TA_TPRI	待ちタスクのキューイングは優先度順
__MR_INT	メッセージバッファ領域を内蔵 RAM から確保する
__MR_EXT	メッセージバッファ領域を外部 RAM から確保する

◆ **maxmsz**

生成するメッセージバッファで扱うことのできるメッセージの最大長を指定してください。なお、maxmsz は、M3T-MR32R は参照しません。

◆ **mbfsz**

生成するメッセージバッファのサイズを指定してください。なお、bufsz=0 を指定することも可能です。この場合、メッセージバッファの送受信側が完全に同期した通信を行うことになります。

◆ **mbf**

ユーザが確保したメッセージバッファ領域の先頭アドレスを指定します。この値が NULL の場合は、mbfatr の値(__MR_INT,__MR_EXT)に応じてカーネルがメッセージバッファ領域を割り当てます。NULL でない場合は、mbfatr の__MR_INT、__MR_EXT の値は、無視されます。

cre_mbf サービスコールにおいて、すでに存在するメッセージバッファに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_mbf サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。
cre_mbf サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大メッセージバッファ数までです。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mpl1 1
void task1(void)
{
   T_CMBF pk_cmbf;
   ER ercd;
   pk_cmbf.mbfatr = __MR_INT;
   pk_cmbf.maxmsz = 30;
   pk_cmbf.mbfsz = 200;
   pk_cmbf.mbf = NULL;
               :
   ercd = cre_mbf( ID_mpl1, &pk_cmbf );
               :
}
```

del_mbf　　メッセージバッファの削除

■ C言語API

```
ER ercd = del_mbf( ID mbfid );
```

■ パラメータ

ID	mbfid	削除するメッセージバッファ ID 番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mbfid のメッセージバッファは存在しない)

■ 機能説明

mbfid で示されたメッセージバッファを削除します。削除されたメッセージバッファは、メッセージバッファ領域が解放され、再度、同じ ID 番号で新しいメッセージバッファを生成することができます。削除するメッセージバッファに対して、メッセージ受信を待っているタスク、メッセージ送信を待っているタスクが存在しても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、メッセージ受信待ちもしくはメッセージ送信待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコード E_DLT が返されます。

本サービスコールは、cre_mbf,acre_mbf によって生成されたメッセージバッファに対して有効です。コンフィギュレーションファイルに静的に定義されたメッセージバッファに対して本サービスコールを発行した場合の動作は保証されません。

存在しないメッセージバッファに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mbf2 2
void task1()
{
      :
    del_mbf( ID_mbf2 );
      :
    ext_tsk();
}
```

snd_mbf　メッセージバッファへの送信
psnd_mbf　メッセージバッファへの送信(ポーリング)
tsnd_mbf　メッセージバッファへの送信(タイムアウト)

■ C言語API

```
ER ercd = snd_mbf( ID mbfid, VP msg, UINT msgsz );
ER ercd = psnd_mbf( ID mbfid, VP msg, UINT msgsz );
ER ercd = tsnd_mbf( ID mbfid, VP msg, UINT msgsz,TMO tmout );
```

■ パラメータ

ID	mbfid	送信対象のメッセージバッファ ID 番号
TMO	tmout	タイムアウト値(tsnd_mbf の場合)
VP	msg	送信するメッセージ
UINT	msgsz	送信するメッセージのサイズ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mbfid のメッセージバッファは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

mbfid で示されたメッセージバッファに対して、msg で示されたメッセージを送信します。送信するメッセージのサイズは、msgsz で示されたバイト数です。対象メッセージバッファにメッセージ受信待ちタスクが存在する場合は、メッセージバッファにデータを格納せず、メッセージ受信待ち行列の先頭タスクにメッセージを送信し、そのタスクのメッセージ受信待ち状態を解除します。
既にメッセージで一杯になったメッセージバッファに対して、snd_mbf,tsnd_mbf を発行した場合、これらのサービスコールを発行したタスクは、実行状態からメッセージ送信待ち状態に移行し、メッセージバッファの空きを待つ送信待ち行列につながれます。
また、対象メッセージバッファに既に送信待ち状態のタスクが存在する場合は、メッセージをメッセージバッファに格納せず、送信待ち行列につながれます。その際、mbfid のメッセージバッファ属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。psnd_mbf の場合は、直ちにリターンし、エラーE_TMOUT を返します。
tsnd_mbf サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、psnd_mbf と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、snd_mbf サービスコールと同じ動作をします。
受信待ちタスクがなく、メッセージバッファ領域も一杯でない場合、送信したメッセージはメッセージサイズとともにメッセージバッファに格納されます。
snd_mbf,tsnd_mbf サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、rcv_mbf,trcv_mbf,prcv_mbf サービスコールが発行され、待ち解除条件が満足された場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_mbf サービスコールによって待ち状態の対象となっているメッセージバッファが削除された場合**
  この場合、エラーコードは、E_DLT を返します。
- **他のタスクから発行した vrst_mbf サービスコールによって待ち状態の対象となっているメッセージバッファが初期化された場合**
  この場合、エラーコードは、EV_RST を返します。

メッセージバッファで送信を待っているタスクが rel_wai,ter_tsk,タイムアウトにより待ちが解除された場合や、chg_pri によって送信待ち行列の先頭のタスクが変化する場合、新たに先頭になったタスクから順にメッセージバッファにメッセージを送信可能であれば、メッセージを送信する処理を行います。
存在しないメッセージバッファに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    ER ercd;
    char *msg="abcdef";
            :
    ercd = snd_mbf( ID_mbf, (VP)msg, 6);
            :
            :
    ercd = psnd_mbf( ID_mbf, (VP)msg, 6);
            :
            :
    ercd = tsnd_mbf( ID_mbf, (VP)msg, 6, 10 );
            :
}
```

rcv_mbf　メッセージバッファからの受信
prcv_mbf　メッセージバッファからの受信(ポーリング)
trcv_mbf　メッセージバッファからの受信(タイムアウト)

■ C言語API

```
ER_UINT msgsz = rcv_mbf( ID mbfid, VP msg );
ER_UINT msgsz = prcv_mbf( ID mbfid, VP msg);
ER_UINT msgsz = trcv_mbf( ID mbfid, VP msg, TMO tmout );
```

■ パラメータ

ID	mbfid	受信対象のメッセージバッファID番号
TMO	tmout	タイムアウト値(trcv_mbfの場合)
VP	msg	受信メッセージを格納した領域へのポインタ

■ リターンパラメータ

ER_UINT	msgsz<0	エラーコード
	msgsz>0	受信したメッセージサイズ
VP	msg	受信メッセージを格納した領域へのポインタ

■ エラーコード

E_NOEXS	未登録状態(mbfidのメッセージバッファは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

mbfidで示されたメッセージバッファからメッセージを受信し、msgで指定した領域にメッセージを格納します。また、受信したメッセージサイズを戻り値として返します。対象となるメッセージバッファに対して、送信待ちタスクが存在する場合、受信したメッセージサイズと送信待ち行列の先頭にあるタスクの送信メッセージサイズとを比較します。

1. 受信メッセージサイズが送信メッセージサイズよりも大きい場合
   メッセージバッファにメッセージを送信し、送信待ちタスクを送信待ち状態から実行可能(READY)状態に移行します。
2. 受信メッセージサイズが送信メッセージサイズよりも小さい場合
   メッセージバッファにメッセージを送信せず、送信待ち状態のままで本サービスコールの処理を終了します。

1の処理が終了した後、まだ送信待ちタスクがある場合は、上記と同様の比較処理を行っていきます。

一方、メッセージバッファ領域にメッセージが格納されていないメッセージバッファに対して、rcv_mbf,trcv_mbfを発行した場合、これらのサービスコールを発行したタスクは、実行状態からデータ受信待ち状態に移行し、データ受信待ち行列につながれます。このとき、受信待ち行列へは、FIFO順でつながれます。prcv_mbfの場合は、直ちにリターンし、エラーE_TMOUTを返します。

trcv_mbfサービスコールの場合は、tmoutには、待ち時間をms単位で指定します。tmoutに指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義のtick_denoの値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmoutにTMO_POL=0を指定した場合は、タイムアウト値として0を指定したことを示し、prcv_mbfと同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、rcv_mbfサービスコールと同じ動作をします。

rcv_mbf,trcv_mbfサービスコール実行による待ち状態は、以下に示す場合に解除されます。

◆ **tmoutの時間が経過する前に、rcv_mbf,trcv_mbf,prcv_mbfサービスコールが発**

行され、待ち解除条件が満足された場合
この場合、エラーコードは、E_OK を返します。

- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_mbf サービスコールによって待ち状態の対象となっているメッセージバッファが削除された場合**
  この場合、エラーコードは、E_DLT を返します。

存在しないメッセージバッファに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
   char   msg[128];
     if( rcv_mbf(ID_mbf ,(VP)&msg ) == E_DLT )
       error("Buffer deleted"n");
               :
     if( prcv_mbf(ID_mbf ,(VP)&msg ) == E_OK )
       printf("get message¥n");
               :
     if( trcv_mbf(ID_mbf ,(VP)&msg, 10 ) == E_RLWAI )
       error("forced wakeup"n");
               :
}
```

ref_mbf　メッセージバッファの状態参照
iref_mbf　メッセージバッファの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_mbf( ID mbfid, T_RMBF *pk_rmbf );
ER ercd = iref_mbf( ID mbfid, T_RMBF *pk_rmbf );
```

■ パラメータ

ID	mbfid	対象メッセージバッファ ID 番号
T_RMBF	*pk_rmbf	メッセージバッファ状態を返すパケットへのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RMBF	*pk_rmbf	メッセージバッファ状態を返すパケットへのポインタ

pk_rmbf の内容

```
typedef   struct t_rmbf{
```

ID	stskid	+0	2	送信待ちタスク ID
ID	rtskid	+0	2	受信待ちタスク ID
UINT	smsgcnt	+4	4	メッセージバッファに入っているメッセージのカウント数
SIZE	fmbfsz	+8	4	空きバッファのサイズ(バイト数)

```
} T_RMBF;
```

■ エラーコード

E_NOEXS	未登録状態(mbfid のメッセージバッファは存在しない)

■ 機能説明

mbfid で示されたメッセージバッファの各種の状態を返します。

- **stskid**
  stskid には送信待ち行列の先頭タスクの ID 番号を返します。
  待ちタスクの無い場合は TSK_NONE を返します。
- **rtskid**
  rtskid には受信待ち行列の先頭タスクの ID 番号を返します。
  待ちタスクの無い場合は TSK_NONE を返します。
- **smsgcnt**
  smsgcnt には、メッセージバッファに入っているメッセージのカウント数を返します。
- **fmbfsz**
  空きバッファのサイズを返します。

存在しないメッセージバッファに対して、本サービスコールを発行した場合は、エラーE_NOEXSを返します。
本サービスコールは、タスクコンテキストからは、ref_mbf、非タスクコンテキストからは、iref_mbf を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  T_RMBF pk_rmbf;
      :
    ref_mbf(ID_mbf, &pk_rmbf,);
      :
}
```

1.10. 拡張同期･通信機能(ランデヴ)

cre_por　ランデヴポートの生成
acre_por　ランデヴポートの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_por( ID porid, T_CPOR *pk_cpor );
ER_ID porid = acre_por( T_CPOR *pk_cpor );
```

■ **パラメータ**

ID	porid	生成するランデヴポート ID 番号(cre_por 使用時)
T_CPOR	*pk_cpor	ランデヴポート生成情報を格納した構造体へのポインタ

pk_cpor の内容

```
typedef   struct t_cpor {
    ATR     poratr   0     4     ランデヴポート属性
    UINT    maxcmsz  +4    4     呼び出しメッセージの最大サイズ(バイト数)
    UINT    maxrmsz  +8    4     返答メッセージの最大サイズ(バイト数)
} T_CPOR;
```

■ **リターンパラメータ**

- cre_por の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_por の場合

ER_ID	porid>0	生成したランデヴポートの ID 番号
	porid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_mbx 使用時)
E_OBJ	オブジェクト状態が不正 (porid のデータキューが存在(cre_por 使用時))

■ **機能説明**

cre_por サービスコールは、porid で示されたランデヴポートを pk_cpor の内容で生成します。acre_por サービスコールは、未使用のランデヴポート ID をランデヴポート ID として割り当て、pk_cpor の内容でランデヴポートを生成します。割り当てた ID 番号を、戻り値 porid として返します。生成するデータキューの情報 pk_cpor について以下に示します。
生成するランデヴポートの情報 pk_cpor について以下に示します。

◆ **poratr(ポート属性)**

poratr には、(TA_TFIFO||TA_TPRI)を指定できます。
ランデヴポートの受付待ち行列は、poratr の値にかかわらず常に FIFO 待ちとなります。

属性	意味
TA_TFIFO	呼び出し待ちタスクのキューイングは FIFO 順
TA_TPRI	呼び出し待ちタスクのキューイングは優先度順

◆ **maxcmsz**

呼出時に使用するメッセージの最大長を指定します。なお、maxcmsz は、M3T-MR32R は参照しません。

◆ **maxrmsz**

返答時に使用するメッセージの最大長を指定します。なお、maxrmsz は、M3T-MR32R は参照しません。

すでに存在するランデヴポートに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。
本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシ

ステムの最大ランデヴポート数までです。acre_por サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1()
{
   T_CPOR setpor;
      :
   setpor.maxcmsz = 300;
   setpor.maxrmsz = 200;
   cre_por( 1, &setpor );
      :
}
```

del_por　　ランデヴポートの削除

■ C言語API

```
ER ercd = del_por( ID porid );
```

■ パラメータ

ID	porid	削除するランデヴポートID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(porid のランデヴポートは存在しない)

■ 機能説明

porid で示されたランデヴポートを削除します。削除されたランデヴポートは、再度、同じ ID 番号で新しいランデヴポートを生成することができます。削除するランデヴポートに対して、ランデヴ呼び出し待ちタスクが存在する場合、ランデヴ受付待ちタスクが存在する場合であっても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコード E_DLT が返されます。

本サービスコールは、cre_por,acre_por によって生成されたランデヴポートに対して有効です。コンフィギュレーションファイルに静的に定義されたランデヴポートに対して本サービスコールを発行した場合の動作は保証されません。

存在しないランデヴポートに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_por2 2
void task1()
{
      :
    del_por( ID_por2 );
      :
    ext_tsk();
}
```

cal_por　ランデヴの呼び出し
tcal_por　ランデヴの呼び出し(タイムアウト)

■ C言語API

```
ER_UINT rmsgsz = cal_por( ID porid, RDVPTN calptn, VP msg, UINT cmsgsz );
ER_UINT rmsgsz = tcal_por( ID porid, RDVPTN calptn, VP msg, UINT cmsgsz,
                           TMO tmout );
```

■ パラメータ

ID	porid	呼び出し対象のランデヴポートID番号
RDVPTN	calptn	呼び出し条件ビットパターン
VP	msg	呼び出しメッセージの先頭アドレス (返答メッセージを格納する先頭番地)
UINT	cmsgsz	呼び出しメッセージのサイズ(バイト数)
TMO	tmout	タイムアウト値(tcal_porの場合)

■ リターンパラメータ

ER_UINT	rmsgsz<0	エラーコード
	rmsgsz>=0	返答メッセージのサイズ

■ エラーコード

E_NOEXS	未登録状態(poridのランデヴポートは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

ポートに対してランデヴ呼出を行います。porid で指定したポートにおいてランデヴ受付待ち状態(acp_por あるいは tacp_por による待ち)のタスクがあり、その受付待ち状態のタスクと本サービスコールを発行するタスクとの間でランデヴ成立条件が満たされた場合は、ランデヴ成立となります。ランデヴ成立とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどうかによって決まります。0 以外の場合はランデヴ成立となります。

ランデヴ成立時、ランデヴ受付待ちタスクの状態は、待ち状態から実行可能(READY)状態へ移行し、本サービスコールを発行したタスクは、ランデヴ終了待ち状態となります。ランデヴ終了待ち状態となったタスクは、ランデヴ受付タスクが rpl_rdv を実行することにより待ち状態が解除されます。この時点で cal_por,tcal_por サービスコールが終了します。ランデヴが成立した時、呼出側タスクは受付側タスクにメッセージを送信します。msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクが acp_por,tacp_por,pacp_por で指定したメッセージ格納領域にコピーします。また、ランデヴ終了時には、受付側タスクから呼出側タスクに返答メッセージを送信することができます。受付側タスクが、rpl_rdv で指定した返答メッセージの内容が呼出側タスクの cal_por で指定した msg が示すメッセージ格納領域にコピーされます。

ランデヴ呼出とランデヴ受付を行う 2 つのタスク間でのメッセージの交換は、双方のタスクが持っているメッセージを互いにコピーしあうことになります。そのため、以前に持っていたメッセージは破壊されます。

porid で指定したポートに受付待ちタスクがなかった場合や、受付待ちタスクがあってもランデヴ成立条件が満たされなかった場合(論理積の結果が 0 の場合)、本サービスコールを発行したタスクは、このポートの呼出側待ち状態となり、呼出待ち行列につながれます。その際、porid のランデヴポート属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。

tcal_por サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく

動作しません。タイムアウトの対象となるのは、呼び出し待ち状態が解除された時点ではなく、ランデヴ終了待ち状態が解除された時点までです。
tmout に TMO_POL=0 を指定できません。指定した場合の動作は保証されません。また、tmout=TMO_FEVR(-1)を指定した場合は、永久待ちの指定で、cal_por と同じ動作をします。
cal_por,tcal_por サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、rpl_rdv によってランデヴが終了した場合**
  この場合、エラーコードは、E_OK を返します。
- **ランデヴが成立しないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **ランデヴは成立したが、rpl_rdv によるランデヴ応答がないまま tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_por サービスコールによって待ち状態の対象となっているランデヴポートが削除された場合**
  この場合、エラーコードは、E_DLT を返します。

存在しないポートに対して、本サービスコールを発行した場合、エラーE_NOEXS を返します。本サービスコールの引数 calptn には 0 を指定することはできませんが、0 を指定してもエラーにはなりません。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
    char msg[128];
     INT   cmsgsz;
      :
    cal_por(ID_por1, 0x3, (VP)msg, cmsgsz);
      :
    tcal_por(ID_por1, 0x3, (VP)msg, cmsgsz, 10);
      :
}
```

acp_por　ランデヴの受付
pacp_por　ランデヴの受付(ポーリング)
tacp_por　ランデヴの受付(タイムアウト)

■ C言語API

```
ER_UINT cmsgsz = acp_por( ID porid, RDVPTN acpptn, RDVNO *p_rdvno, VP
msg );
ER_UINT cmsgsz = pacp_por( ID porid, RDVPTN acpptn, RDVNO *p_rdvno,
VP msg );
ER_UINT cmsgsz = tacp_por( ID porid, RDVPTN acpptn, RDVNO *p_rdvno,
VP msg,
                           TMO tmout );
```

■ パラメータ

ID	porid	受付対象のランデヴポートID番号
RDVPTN	acpptn	受付条件ビットパターン
VP	msg	呼び出しメッセージを格納する先頭アドレス
RDVNO	*p_rdvno	成立したランデヴ番号へのポインタ
TMO	tmout	タイムアウト値(tcal_porの場合)

■ リターンパラメータ

ER_UINT	cmsgsz<0	エラーコード
	cmsgsz>=0	呼び出しメッセージのサイズ
RDVNO	*p_rdvno	成立したランデヴ番号へのポインタ

■ エラーコード

E_NOEXS	未登録状態(poridのランデヴポートは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

ポートに対してランデヴ受付を行います。porid で指定したポートにおいてランデヴ呼出待ち状態(cal_por あるいは tcal_por による待ち)のタスクがあり、その呼出待ち行列の中のタスクと本サービスコールを発行するタスクとの間でランデヴ成立条件が満たされた場合は、ランデヴ成立となります。ランデヴ成立とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどうかによって決まります。0 以外の場合はランデヴ成立となります。ランデヴ成立時、ランデヴ呼出待ち行列中のタスクは、待ち行列から外れ、ランデヴ呼出待ちの状態からランデヴ終了待ち状態に変わります。この時、呼出側タスクは受付側タスクに呼出メッセージを送信します。

cal_por、tcal_por で指定した msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクが acp_por,tacp_por,pacp_por で指定したメッセージ格納領域にコピーします。この時、呼出メッセージのサイズ cmsgsz は、acp_por の戻り値となります。

acp_por,tacp_por 使用時は、porid で指定したポートに呼出待ちタスクがなかった場合や、呼出待ちタスクがあってもランデヴ成立条件が満たされなかった場合、本サービスコールを発行したタスクは、このポートの受付待ち状態となり受付待ち行列につながれます。pacp_por の場合は、直ちにリターンし、エラーE_TMOUT を返します。

tcal_acp サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pacp_por と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、acp_por サービスコールと同じ動作をします。

acp_por,tacp_por サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、ランデヴが成立した場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**
  この場合、エラーコードは、E_RLWAI を返します。
- **他のタスクから発行した del_por サービスコールによって待ち状態の対象となっているランデヴポートが削除された場合**
  この場合、エラーコードは、E_DLT を返します。

ランデヴ受付側タスクは、ランデヴ返答(成立したランデヴに関する rpl_rdv を発行)を行う前に、再度、本サービスコールを発行することができます。こうすることで、同時に複数のランデヴを行うことができます。このときのポートは、前と同じであっても異なっていても構いません。
rdvno は、同時に成立しているランデヴを区別するための情報です。ランデヴ終了時に、rpl_rdv のパラメータやランデヴ回送時の fwd_por のパラメータに使用されます。この情報には、ランデヴ成立相手の呼出側タスクを指定するための情報が含まれています。
本サービスコールの引数 acpptn には、0 を指定することはできませんが、acpptn に 0 を指定してもエラーにはなりません。
存在しないポートに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    char msg[128];
    RDVNO rdvno;
              :
    acp_por(ID_por1, 0x3, &rdvno, &msg);
              :
    pacp_por(ID_por1, 0x3, &rdvno, &msg);
              :
    tacp_por(ID_por1, 0x3, &rdvno, &msg, 10);
              :
}
```

fwd_por　　ランデヴの回送

■ C言語API

```
ER ercd = fwd_por( ID porid, RDVPTN calptn, RDVNO rdvno, VP msg, UINT
cmsgsz );
```

■ パラメータ

ID	porid	回送先のランデヴポートID番号
RDVPTN	calptn	呼び出し条件ビットパターン
RDVNO	rdvno	回送するランデヴ番号
VP	msg	呼び出しメッセージの先頭アドレス (返答メッセージを格納する先頭番地)
UINT	cmsgsz	呼び出しメッセージのサイズ(バイト数)

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(porid のランデヴポートは存在しない)
E_OBJ	オブジェクト状態が不正(rdvno が不正)

■ 機能説明

ランデヴ番号 rdvno で、受け付けたランデヴを porid で指定した別のポートへ回送します。本サービスコールは、現在、ランデヴ中の状態(acp_por,tacp_por,pacp_por を実行した後)にあるタスクからのみ発行することができます。ランデヴ番号 rdvno で指定した呼出タスクのランデヴ成立を解除し、porid のポートに対して、呼出タスクをランデヴ呼出待ち状態に戻します。つまり、呼出タスクの状態は、ランデヴ終了待ち状態からランデヴ呼出状態に戻ることになります。受付待ちタスクがある場合は、本サービスコールで指定した calptn と受付側タスクの acpptn との論理積がとられ 0 でなければランデヴが成立となります。ランデヴが成立したタスクは、本サービスコールで指定した msg のアドレスに格納されているメッセージを cmsgsz バイト分、受付側タスクにコピーし、ランデヴ終了待ち状態になります。

受付待ちタスクがなかったり、回送先のポートでランデヴが成立しなかった場合は、呼出側タスクは、呼出待ち状態になります。その際、porid のランデヴポート属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。

本サービスコールを発行したタスクは、回送したポートでの呼出タスクのランデヴ状態に関係なく待ち状態にはならずそのまま実行を継続します。

本サービスコールの引数に以下に示す場合を指定しても、エラーにはなりません。以下に示すチェックは、ユーザ側で行ってください。

- **cmsgsz が、回送後のポートの送信最大メッセージサイズ maxcmsz を超えた場合**
- **cmsgsz が、回送前のポートの受信最大メッセージサイズ maxrmsz を超えた場合**

rdvno が不正な場合は、エラーE_OBJ を返します。

存在しないポートに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task()
{
    char *msg=”abcdef”;
    RDVNO rdvno;
            :
    fwd_por(ID_porid, 0x3, rdvno, (VP)msg, 6);
            :
}
```

rpl_rdv　　ランデヴの終了

■ C言語API

```
ER ercd = rpl_rdv( RDVNO rdvno, VP msg, UINT rmsgsz );
```

■ パラメータ

RDVNO	rdvno	終了させるランデヴ番号
VP	msg	返答メッセージの先頭アドレス
UINT	rmsgsz	返答メッセージのサイズ(バイト数)

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_OBJ	オブジェクト状態エラー(rdvnoが不正)

■ 機能説明

呼出側タスクに対して返答を返し、ランデヴを終了します。本サービスコールは、現在、ランデヴ中の状態(acp_por,tacp_por,pacp_porを実行した後)にあるタスクからのみ発行することができます。
呼出側タスクは、ランデヴ終了待ち状態が解除され、msg のアドレスに格納されているメッセージを rmsgsz バイトだけ呼出側タスクの cal_por,tcal_por で指定した msg アドレス以下にコピーされます。呼出側タスクの状態は、ランデヴ終了待ち状態から実行可能(READY)状態に移行します。
本サービスコールの引数に以下に示す場合を指定しても、エラーにはなりません。以下に示すチェックは、ユーザ側で行ってください。

- **rmsgsz に負の値を指定した場合**
- **rmsgsz が、返答最大メッセージサイズ maxrmsz を超えた場合**

ランデヴ成立後に呼出側タスクが何らかの理由で、ランデヴ終了前(rpl_rdv 実行前)に異常終了したような場合、ランデヴ受付側のタスクは直接それを知ることができません。この場合、ランデヴ受付側のタスクには、本サービスコールからエラーE_OBJ が返されます。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include “kernel_id.h”
void task()
{
    char *msg=”abcdef”;
    RDVNO rdvno;
    INT rmsgsz;
      :
    rpl_rdv(rdvno, (VP)msg, 6);
      :
}
```

ref_por　ランデヴポートの状態参照
iref_por　ランデヴポートの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_por( ID porid, T_RPOR *pk_rpor );
ER ercd = iref_por( ID porid, T_RPOR *pk_rpor );
```

■ パラメータ

ID	porid	対象ランデヴポートID番号
T_RPOR	*pk_rpor	ランデヴポート状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RPOR	*pk_rpor	ランデヴポート状態を返すパケットへのポインタ

pk_rporの内容

```
typedef   struct t_rpor{
    ID      ctskid   +0   2    呼び出し待ちタスクID
    ID      atskid   +2   2    受付待ちタスクID
} T_RPOR;
```

■エラーコード

E_NOEXS	未登録状態(poridのランデヴポートは存在しない)

■ 機能説明

poridで示されたランデヴポートの各種の状態を返します。

- **ctskid**
  ctskidには対象ポートの呼出待ち行列の先頭タスクID番号を返します。待ちタスクの無い場合はTSK_NONEを返します。
- **atskid**
  atskidには、対象ポートの受付待ち行列の先頭タスクID番号を返します。待ちタスクの無い場合はTSK_NONEを返します。

存在しないメールボックスに対して、本サービスコールを発行した場合は、エラーE_NOEXSを返します。

本サービスコールは、タスクコンテキストからは、ref_por、非タスクコンテキストからは、iref_porを使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
  T_RPOR rpor;
      :
    ref_por(ID_por ,&rpor);
      :
}
```

ref_rdv　ランデヴの状態参照
iref_rdv　ランデヴの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_rdv( RDVNO rdvno, T_RRDV *pk_rrdv );
ER ercd = iref_rdv( RDVNO rdvno, T_RRDV *pk_rrdv );
```

■ パラメータ

RDVNO	rdvno	状態参照対象のランデヴ番号
T_RRDV	*pk_rrdv	ランデヴ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RRDV	*pk_rrdv	ランデヴ状態を返すパケットへのポインタ

pk_rrdv の内容

```
typedef   struct t_rrdv{
    ID      wtskid    +0    2     ランデヴ終了待ちタスク ID
} T_RRDV;
```

■エラーコード

なし

■ 機能説明

rdvno で示されたランデヴの状態を返します。

◆ **wtskid**

rdvnoで指定されたランデヴを呼び出したタスクが、指定されたランデヴの終了待ち状態である場合、そのタスク ID を wtskid に返します。そのタスクがランデヴ終了待ちでない場合、ランデヴ番号として解釈できない場合は、TSK_NONE を返します。

本サービスコールは、タスクコンテキストからは、ref_rdv、非タスクコンテキストからは、iref_rdv を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
   T_RRDV rrdv;
   RDVNO rdvno;
       :
     ref_rdv( rdvno,&rrdv);
       :
}
```

1.11. メモリプール管理機能(固定長メモリプール)

cre_mpf 固定長メモリプールの生成
acre_mpf 固定長メモリプールの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_mpf( ID mpfid, T_CMPF *pk_cmpf );
ER_ID mpfid = acre_mpf( T_CMPF *pk_cmpf );
```

■ **パラメータ**

ID	mpfid	生成する固定長メモリプール ID 番号(cre_mpf 使用時)
T_CMPF	*pk_cmpf	固定長メモリプール生成情報を格納した構造体へのポインタ

pk_cmpf の内容

```
typedef   struct t_cmpf {
    ATR     mpfatr   0     4    固定長メモリプール属性
    UINT    blkcnt   +4    4    獲得可能なメモリブロック数(個数)
    UINT    blksz    +8    4    メモリブロックのサイズ(バイト数)
    VP      mpf      +12   4    固定長メモリプール領域の先頭アドレス
} T_CMPF;
```

■ **リターンパラメータ**

- cre_mpf の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_mpf の場合

ER_ID	mpfid>0	生成したメッセージバッファの ID 番号
	mpfid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_mpf 使用時)
E_OBJ	オブジェクト状態が不正 (mpfid の固定長メモリプールが存在(cre_mpf 使用時))
E_NOMEM	空きメモリ無し

■ **機能説明**

cre_mpf サービスコールは、mpfid で示された固定長メモリプールを pk_cmpf の内容で生成します。acre_mpf サービスコールは、未使用の固定長メモリプール ID を固定長メモリプール ID として割り当て、pk_cmpf の内容で固定長メモリプールを生成します。割り当てた ID 番号を、戻り値 mpfid として返します。メモリプールとして利用するメモリ領域を確保し、生成したメモリプールの管理ブロックデータの初期化を行います。
生成するメモリプールの情報 pk_cmpf について以下に示します。

◆ **mpfatr(固定長メモリプール属性)**

mpfatr には、((TA_TFIFO||TA_TPRI) | [__MR_INT || __MR_EXT])を指定できます。

属性	意味
TA_TFIFO	待ちタスクのキューイングは FIFO 順
TA_TPRI	待ちタスクのキューイングは優先度順
__MR_INT	固定長メモリプール領域を内蔵 RAM から確保する
__MR_EXT	固定長メモリプール領域を外部 RAM から確保する

◆ **blkcnt**

生成するメモリプールのブロック数を指定してください。ここでは、1 から 1024 までの値が指定できます。

◆ **blksz**

生成するメモリプールの 1 ブロックのブロックサイズを指定してください。

◆ **mpf**

ユーザが確保した固定長メモリプール領域の先頭アドレスを指定します。この値が NULL の場合は、mpfatr の値(__MR_INT,__MR_EXT)に応じてカーネルが固定長メモリプール領域を割り当てます。NULL でない場合は、mpfatr の__MR_INT、__MR_EXT の値は、無視されます。

cre_mpf サービスコールにおいて、すでに存在する固定長メモリプールに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_mpf サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。
cre_mpf サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大固定長メモリプール数までです。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mpf1 1
void task(void)
{
   T_CMPF   setmpf;
      :
   setmpf.mpfatr = __MR_INT;
   setmpf.blkcnt = 20;
   setmpf.blfsz = 400;
   setmpf.mpf = NULL;
   cre_mpf(ID_mpf1,&setmpf);
      :
}
```

del_mpf　　固定長メモリプールの削除

■ C言語API

```
ER ercd = del_mpf( ID mpfid );
```

■ パラメータ

ID	mpfid	削除する固定長メモリプールID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mpfidの固定長メモリプールは存在しない)

■ 機能説明

mpfid で示された固定長メモリプールを削除します。削除された固定長メモリプールは、固定長メモリプール領域が解放され、再度、同じ ID 番号で新しい固定長メモリプールを生成することができます。削除する固定長メモリプールに対して、メモリ獲得を待っているタスクが存在しても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、メモリ獲得待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコード E_DLT が返されます。

本サービスコールは、cre_mpf,acre_mpf によって生成された固定長メモリプールに対して有効です。コンフィギュレーションファイルに静的に定義された固定長メモリプールに対して本サービスコールを発行した場合の動作は保証されません。

存在しない固定長メモリプールに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
      :
   del_mpf(ID_mpf1);
      :
}
```

get_mpf 固定長メモリブロックの獲得
pget_mpf 固定長メモリブロックの獲得(ポーリング)
ipget_mpf 固定長メモリブロックの獲得(ハンドラ専用)
tget_mpf 固定長メモリブロックの獲得(タイムアウト)

■ C言語API

```
ER ercd = get_mpf( ID mpfid, VP *p_blk );
ER ercd = pget_mpf( ID mpfid, VP *p_blk );
ER ercd = ipget_mpf( ID mpfid, VP *p_blk );
ER ercd = tget_mpf( ID mpfid, VP *p_blk,TMO tmout );
```

■ パラメータ

ID	mpfid	獲得対象の固定長メモリプール ID 番号
TMO	tmout	タイムアウト値(tget_mpf の場合)
VP	*p_blk	獲得したメモリブロック先頭アドレスへのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
VP	*p_blk	獲得したメモリブロック先頭アドレスへのポインタ

■ エラーコード

E_NOEXS	未登録状態(mpfid の固定長メモリプールは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
EV_RST	メモリプール領域クリアによって待ち状態が解除された

■ 機能説明

mpfid で示される固定長メモリプールからメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blf に格納します。獲得したメモリブロックの内容は不定です。
指定した固定長メモリプールにメモリブロックがない場合は、tget_mpf,get_mpf サービスコール使用時には、本サービスコールを発行したタスクは、メモリブロック獲得待ち状態に移行し、メモリブロック獲得待ち行列につながれます。その際、mpfid の固定長メモリプール属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。pget_mpf,ipget_mpf サービスコール使用時は、直ちにリターンし、エラーE_TMOUT を返します。
tget_mpf サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pget_mpf と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、get_mpf サービスコールと同じ動作をします。
get_mpf,tget_mpf サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、rel_mpf,irel_mpf サービスコールが発行され、待ち解除条件が満足された場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち状態が強制解除された場合**

この場合、エラーコードは、E_RLWAI を返します。

- **他のタスクから発行した del_mpf サービスコールによって待ち状態の対象となっているメモリプールが削除された場合**
  この場合、エラーコードは、E_DLT を返します。
- **他のタスクから発行した vrst_mpf サービスコールによって待ち状態の対象となっているメモリプールが初期化された場合**
  この場合、エラーコードは、EV_RST を返します。

本サービスコールによって獲得されたメモリブロックの値は、初期化しないため不定となります。
存在しないメモリプールに対して、本サービスコールを発行した場合、エラーコード E_NOEXS を返します。
本サービスコールは、タスクコンテキストからは、get_mpf,pget_mpf,tget_mpf、非タスクコンテキストからは、ipget_mpf を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <mr32.h>
#include "kernel_id.h"
VP      p_blk;
void task()
{
    if( get_mpf(ID_mpf ,&p_blk) != E_OK ){
        error("Not enough memory¥n");
  }
              :
              :
    if( pget_mpf(ID_mpf ,&p_blk) != E_OK ){
        error("Not enough memory¥n");
  }
              :
              :
    if( tget_mpf(ID_mpf ,&p_blk, 10) != E_OK ){
        error("Not enough memory¥n");
  }

}
```

rel_mpf　固定長メモリブロックの解放
irel_mpf　固定長メモリブロックの解放(ハンドラ専用)

■ C言語API

```
ER ercd = rel_mpf( ID mpfid, VP blk );
ER ercd = irel_mpf( ID mpfid, VP blk);
```

■ パラメータ

ID	mpfid	解放するメモリブロックの固定長メモリプール ID 番号
VP	blk	返却するメモリブロックの先頭アドレス

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mpfid の固定長メモリブロックは存在しない)

■ 機能説明

blk に示される先頭アドレスをもつメモリブロックを解放します。解放するメモリブロックの先頭アドレスは必ず、get_mpf,tget_mpf,pget_mpf,ipget_mpf で獲得した先頭アドレスを指定してください。本サービスコールは、blk の内容をチェックしません。従って、blk に正しいアドレスが格納されていない場合は正しく動作しません。

対象メモリプールの待ち行列にタスクがつながれている場合は、待ち行列の先頭タスクを待ち行列からはずし、レディキューにつなぎかえこのタスクにメモリブロックを割り当てます。この時のタスクの状態は、メモリブロック待ち状態から実行(RUN)状態あるいは実行可能(READY)状態へ移行します。

タスクコンテキストにおいては、rel_mpf サービスコール、非タスクコンテキストにおいては、irel_mpf を使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mpf1 1
void task()
{
    VP  p_blf;
    if( get_mpf(ID_mpf1,&p_blf) != E_OK )
        error("Not enough memory ¥n");
              :
    rel_mpf(ID_mpf1,p_blf);
}
```

ref_mpf　固定長メモリプールの状態参照
iref_mpf　固定長メモリプールの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_mpf( ID mpfid, T_RMPF *pk_rmpf );
ER ercd = iref_mpf( ID mpfid, T_RMPF *pk_rmpf );
```

■ パラメータ

ID	mpfid	対象固定長メモリプール ID 番号
T_RMPF	*pk_rmpf	固定長メモリプール状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RMPF	*pk_rmpf	固定長メモリプール状態を返すパケットへのポインタ

pk_rmpf の内容

```
typedef   struct t_rmpf{
    ID      wtskid   +0   2    メモリブロック獲得待ちタスク ID
    UINT    fblkcnt  +4   4    空きメモリブロック数(個数)
} T_RMPF;
```

■エラーコード

E_NOEXS	未登録状態(mpfid の固定長メモリプールは存在しない)

■ 機能説明

mpfid で示されたメッセージバッファの各種の状態を返します。

- **wtskid**
  wtskid にはメモリブロック獲得待ち行列の先頭タスクの ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- **fblkcnt**
  指定したメモリプールの空きブロック数を返します。

存在しない固定長メモリプールに対して、本サービスコールを発行した場合は、エラーE_NOEXSを返します。
本サービスコールは、タスクコンテキストからは、ref_mpf、非タスクコンテキストからは、iref_mpf を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  T_RMPF rmpf;
      :
  ref_mpf(ID_mpf ,&rmpf);
      :
    ext_tsk();
}
```

1.12. メモリプール管理機能(可変長メモリプール)

cre_mpl　　可変長メモリプールの生成
acre_mpl　　可変長メモリプールの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_mpl( ID mplid, T_CMPL *pk_cmpl );
ER_ID mplid = acre_mpl( T_CMPL *pk_cmpl );
```

■ **パラメータ**

ID	mplid	生成する可変長メモリプール ID 番号(cre_mpl 使用時)
T_CMPL	*pk_cmpl	可変長メモリプール生成情報を格納した構造体へのポインタ

pk_cmpl の内容

```
typedef   struct t_cmpl {
    ATR     mplatr    0     4    可変長メモリプール属性
    SIZE    mplsz     +4    4    確保するメモリプールサイズ
    UINT    maxblksz  +8    4    獲得するメモリサイズのうち最大サイズ
    VP      mpl       +12   4    可変長メモリプール領域の先頭アドレス
} T_CMPL;
```

■ **リターンパラメータ**

- cre_mpl の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_mpl の場合

ER_ID	mplid>0	生成したメッセージバッファの ID 番号
	mplid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_mpl 使用時)
E_OBJ	オブジェクト状態が不正 (mplid の可変長メモリプールが存在(cre_mpl 使用時))
E_NOMEM	空きメモリ無し

■ **機能説明**

cre_mpl サービスコールは、mplid で示された可変長メモリプールを pk_cmpl の内容で生成します。acre_mpl サービスコールは、未使用の可変長メモリプール ID を可変長メモリプール ID として割り当て、pk_cmpl の内容で可変長メモリプールを生成します。割り当てた ID 番号を、戻り値 mplid として返します。メモリプールとして利用するメモリ領域を確保し、生成したメモリプールの管理ブロックデータの初期化を行います。
生成するメモリプールの情報 pk_cmpl について以下に示します。

◆ **mplatr(可変長メモリプール属性)**

mplatr には、((TA_TFIFO) | [__MR_INT || __MR_EXT])を指定できます。TA_TPRI は指定できません。

属性	意味
TA_TFIFO	待ちタスクのキューイングは FIFO 順
__MR_INT	可変長メモリプール領域を内蔵 RAM から確保する
__MR_EXT	可変長メモリプール領域を外部 RAM から確保する

◆ **mplsz**

ここで設定した値のメモリ領域を確保し、可変長メモリプール領域として利用します。

◆ **maxblksz**

M3T-MR32R の可変長メモリプールは、4 つの固定長メモリブロックのサイズに分けて、ユー

ザが指定したサイズに最も最適なメモリブロックを 4 つのサイズの中から選択し、メモリの割り当てを行います。各固定長メモリブロックのサイズは、maxblksz をユーザが指定することで決定します。

◆ **mpl**

ユーザが確保した可変長メモリプール領域の先頭アドレスを指定します。この値が NULL の場合は、mplatr の値(__MR_INT,__MR_EXT)に応じてカーネルが可変長メモリプール領域を割り当てます。NULL でない場合は、mplatr の__MR_INT、__MR_EXT の値は、無視されます。

cre_mpl サービスコールにおいて、すでに存在する可変長メモリプールに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_mpl サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。

cre_mpl サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大可変長メモリプール数までです。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
  T_CMPL   setmpl;
     :
    setmpl.mplatr = __MR_INT;
    setmpl.mplsz = 5000;
    setmpl.maxblksz = 400;
    setmpl.mpl = NULL;
    cre_mpl(ID_mpl1,&setmpl);
      :
}
```

del_mpl　　可変長メモリプールの削除

■ C言語API

```
ER ercd = del_mpl( ID mplid );
```

■ パラメータ

ID	mplid	削除する可変長メモリプールID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mplidの可変長メモリプールは存在しない)

■ 機能説明

mplidで示された可変長メモリプールを削除します。削除された可変長メモリプールは、可変長メモリプール領域が解放され、再度、同じID番号で新しい可変長メモリプールを生成することができます。削除する可変長メモリプールに対して、メモリ獲得を待っているタスクが存在しても、本サービスコールは正常終了します。その時、待ち状態にあったタスクは、メモリ獲得待ち状態が解除され、実行可能(READY)状態に移行します。この場合待ちが解除されたタスクに対してエラーコードE_DLTが返されます。

本サービスコールは、cre_mpl,acre_mplによって生成された可変長メモリプールに対して有効です。コンフィギュレーションファイルに静的に定義された可変長メモリプールに対して本サービスコールを発行した場合の動作は保証されません。

存在しない可変長メモリプールに対して、本サービスコールを発行した場合は、エラーE_NOEXSを返します。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task(void)
{
      :
    del_mpl(ID_mpl1);
      :
}
```

get_mpl 可変長メモリブロックの獲得
pget_mpl 可変長メモリブロックの獲得(ポーリング)
tget_mpl 可変長メモリブロックの獲得(タイムアウト)

■ C 言語 API

```
ER ercd = get_mpl( ID mplid, UINT blksz, VP *p_blk );
ER ercd = pget_mpl( ID mplid, UINT blksz, VP *p_blk );
ER ercd = tget_mpl( ID mplid, UINT blksz, VP *p_blk,TMO tmout );
```

■ パラメータ

ID	mplid	獲得対象の可変長メモリプール ID 番号
UINT	blksz	獲得するメモリのサイズ(バイト数)
TMO	tmout	タイムアウト値(tget_mpl の場合)
VP	*p_blk	獲得したメモリの先頭アドレスへのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
VP	*p_blk	獲得したメモリブロック先頭アドレスへのポインタ

■ エラーコード

E_NOEXS	未登録状態(mplid の可変長メモリプールは存在しない)
E_DLT	待ちオブジェクト削除
E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
EV_RST	メモリプール領域クリアによって待ち状態が解除された

■ 機能説明

mplid で示される可変長メモリプールからメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blk に格納します。獲得したメモリブロックの内容は、不定です。

tget_mpl,get_mpl サービスコール使用時、既に可変長メモリ獲得待ち行列にタスクが存在する場合や指定した可変長メモリプールに指定したサイズの連続した空き領域がない場合は、本サービスコールを発行したタスクは、可変長メモリ獲得待ち状態に移行し、可変長メモリ獲得待ち行列につながれます。その際、タスクは FIFO 順で待ち行列につながれます。[4]

pget_mpl サービスコール使用時は、直ちにリターンし、エラーE_TMOUT を返します。

tget_mpl サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pget_mpl と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、get_mpl サービスコールと同じ動作をします。

get_mpl,tget_mpl サービスコール実行による待ち状態は、以下に示す場合に解除されます。

- **tmout の時間が経過する前に、rel_mpl サービスコールが発行され、待ち解除条件が満足された場合**
  この場合、エラーコードは、E_OK を返します。
- **待ち解除条件が満足されないまま、tmout 経過し、最初のタイムティックが発生した場合**
  この場合、エラーコードは、E_TMOUT を返します。
- **他のタスクおよびハンドラから発行した rel_wai、irel_wai サービスコールによって待ち**

[4] M3T-MR32R では、可変長メモリプールの TA_TPRI 属性をサポートしないため、常に FIFO 順でタスクを待ち行列につなぎます。

状態が強制解除された場合
この場合、エラーコードは、E_RLWAI を返します。

- **他のタスクから発行した del_mpl サービスコールによって待ち状態の対象となっているメモリプールが削除された場合**
  この場合、エラーコードは、E_DLT を返します。
- **他のタスクから発行した vrst_mpl サービスコールによって待ち状態の対象となっているメモリプールが削除された場合**
  この場合、エラーコードは、EV_RST を返します。

メッセージバッファで送信を待っているタスクが rel_wai,ter_tsk,タイムアウトにより待ちが解除され、可変長メモリ獲得待ち行列の先頭のタスクが変化する場合、新たに先頭になったタスクについてメモリ獲得可能であればメモリ獲得処理を行います。
存在しないメモリプールに対して、本サービスコールを発行した場合、エラーコード E_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mpl1 1
VP      p_blk;
void task(void)
{
    if( get_mpl( ID_mpl1, 70, &p_blk ) != E_OK ){
        error("Can't get memory¥n");
  }
            :
    if( pget_mpl( ID_mpl1, 70, &p_blk ) != E_OK ){
        error("Can't get memory ¥n");
  }
            :
    if( tget_mpl( ID_mpl1, 70, &p_blk, 10 ) != E_OK ){
        error("Can't get memory ¥n");
  }
}
```

rel_mpl　　可変長メモリプールブロックの解放

■ C言語API

```
ER ercd = rel_mpl( ID mplid, VP blk );
```

■ パラメータ

ID	mplid	解放するメモリブロックの可変長メモリプール ID 番号
VP	blk	返却するメモリブロックの先頭アドレス

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mplid の可変長メモリブロックは存在しない)

■ 機能説明

blk に示される先頭アドレスをもつメモリブロックを解放します。解放するメモリブロックの先頭アドレスは必ず、get_mpl,tget_mpl,pget_mpl で獲得した先頭アドレスを指定してください。
本サービスコールは、blk の内容をチェックしません。従って、blk に正しいアドレスが格納されていない場合は正しく動作しません。
メモリブロックを解放した結果、対象メモリプールの可変長メモリ獲得待ち行列の先頭タスクが要求するだけの連続した空き領域が存在すれば、そのタスクを可変長メモリ獲得待ち行列からはずし、レディキューにつなぎかえ、このたすくにメモリブロックを割り当てます。この時のタスクの状態は、メモリブロック待ち状態から実行(RUN)状態あるいは実行可能(READY)状態へ移行します。
移行の待ち行列につながれているタスクに対してもメモリブロックを割り付け可能である限り同様の処理を行います。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
#define ID_mpl1 1
void task()
{
    VP   p_blk;
    /* 60バイトのメモリブロックを獲得 */
    if( pget_mpl(ID_mpl1,60,&p_blk) != E_OK )
        error("Not enough memory ¥n");
            :
    rel_mpl(ID_mpl1,p_blk);     /* メモリブロックを解放 */
}
```

ref_mpl　可変長メモリプールの状態参照
iref_mpl　可変長メモリプールの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_mpl( ID mplid, T_RMPL *pk_rmpl );
ER ercd = iref_mpl( ID mplid, T_RMPL *pk_rmpl );
```

■ パラメータ

ID	mplid	対象可変長メモリプール ID 番号
T_RMPL	*pk_rmpl	可変長メモリプール状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RMPL	*pk_rmpl	可変長メモリプール状態を返すパケットへのポインタ

pk_rmpl の内容

```
typedef   struct t_rmpl{
```

ID	wtskid	+0	2	メモリ獲得待ちタスク ID
SIZE	fmplsz	+4	4	空きメモリサイズ(バイト数)
UINT	fblksz	+8	4	すぐに獲得可能なメモリの最大サイズ(バイト数)

```
} T_RMPL;
```

■エラーコード

E_NOEXS	未登録状態(mplid の可変長メモリプールは存在しない)

■ 機能説明

mplid で示されたメッセージバッファの各種の状態を返します。

- **wtskid**
  wtskid にはメモリ獲得待ち行列の先頭タスク(最も早く待ちに入ったタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- **fmplsz**
  空きメモリサイズを返します。(このサイズにはカーネルが管理する領域も含まれています。)
- **fblksz**
  すぐに獲得できるメモリの最大サイズを返します。

存在しない可変長メモリプールに対して、本サービスコールを発行した場合は、エラーE_NOEXSを返します。
本サービスコールは、タスクコンテキストからは、ref_mpl、非タスクコンテキストからは、iref_mpl を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  T_RMPL rmpl;
    ref_mpl(ID_mpl1,&rmpl);
}
```

1.13. 時間管理機能

set_tim　システム時刻の設定
iset_tim　システム時刻の設定(ハンドラ専用)

■ **C 言語 API**

```
ER ercd = set_tim( SYSTIM *p_systim );
ER ercd = iset_tim( SYSTIM *p_systim );
```

■ **パラメータ**

SYSTIM	*p_systim	設定するシステム時刻を示すパケットへのポインタ

p_systim の内容

```
typedef   struct t_systim {
    UH      utime     0    2    上位 16bit
    UW      ltime     +4   4    下位 32bit
} SYSTIM;
```

■ **リターンパラメータ**

ER	ercd	正常終了(E_OK)またはエラーコード

■ **エラーコード**

なし

■ **機能説明**

システム時刻の現在値を p_systim で示される値に更新します。p_systim に指定する時刻の単位は、タイムティック数ではなく"ms"となります。

p_systim に指定可能な値は、0x7FFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。

本サービスコールは、タスクコンテキストからは、set_tim、非タスクコンテキストからは、iset_tim を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
   SYSTIME time;        /* 時刻データ格納変数 */
   time.utime = 0;        /* 上位時刻データの設定 */
   time.ltime = 0;        /* 下位時刻データの設定 */
   set_tim( &time );      /* システム時刻の変更 */
}
```

get_tim　システム時刻の取得
iget_tim　システム時刻の取得(ハンドラ専用)

■ C言語API

```
ER ercd = get_tim( SYSTIM *p_systim );
ER ercd = iget_tim( SYSTIM *p_systim );
```

■ パラメータ

SYSTIM	*p_systim	現在のシステム時刻を返すパケットへのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
SYSTIM	*p_systim	現在のシステム時刻を返すパケットへのポインタ

p_systimの内容

```
typedef   struct t_systim {
    UH      utime     0     2     上位16bit
    UW      ltime     +4    4     下位32bit
} SYSTIM;
```

■ エラーコード

なし

■ 機能説明

システム時刻の現在値をp_systimに格納します。取得する時刻の単位は、"ms"です。
本サービスコールは、タスクコンテキストからは、get_tim、非タスクコンテキストからは、iget_timを使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
   SYSTIME     time;            /* 時刻データ格納変数 */
   get_tim( &time );   /* 時刻データの読み出し */
     printf("system_clock.utime = %X¥n",time.utime);
     printf("system_clock.ltime = %X¥n",time.ltime);
}
```

isig_tim　　タイムティックの供給

■ 機能説明

システム時刻を更新します。
コンフィギュレーションファイルにてシステムクロック割込ハンドラ(__sys_timer)を登録すると、定義された tick_deno / tick_nume(ms)の間隔で isig_tim が自動的に起動されるようになっています。本機能は、サービスコールとして実装されていませんのでアプリケーションから呼び出すことは出来ません。
タイムティック供給時には、カーネル(__sys_timer)は、以下の処理を行います。

(1) システム時刻の更新
(2) アラームハンドラの起動
(3) 周期ハンドラの起動
(4) tslp_tsk などタイムアウト付きサービスコールで待ち状態になっているタスクのタイムアウト処理

タイムアウト処理によって待ち行列の先頭タスクがタイムアウトし、待ち行列からはずされた場合、以降につながれているタスクの待ち状態を解除することがあります。(tget_mpl, snd_mbf,tsnd_mbf による待ちの場合)

1.14. 時間管理機能(周期ハンドラ)

cre_cyc　周期ハンドラの生成
acre_cyc　周期ハンドラの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_cyc( ID cycid, T_CCYC *pk_ccyc );
ER_ID cycid = acre_cyc( T_CCYC *pk_ccyc );
```

■ **パラメータ**

ID	cycid	生成する周期ハンドラ ID 番号(cre_cyc 使用時)
T_CCYC	*pk_ccyc	周期ハンドラ生成情報を格納した構造体へのポインタ

pk_ccyc の内容

```
typedef   struct t_ccyc {
    ATR      cycatr   0     4     周期ハンドラ属性
    VP_INT   exinf    +4    4     拡張情報
    FP       cychdr   +8    4     周期ハンドラエントリアドレス
    RELTIM   cyctim   +12   4     周期ハンドラの起動周期
    RELTIM   cycphs   +16   4     周期ハンドラの起動位相
} T_CCYC;
```

■ **リターンパラメータ**

- cre_cyc の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_cyc の場合

ER_ID	cycid>0	生成した周期ハンドラの ID 番号
	cycid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_cyc 使用時)
E_OBJ	オブジェクト状態が不正 (cycid の周期ハンドラが存在(cre_cyc 使用時))

■ **機能説明**

cre_cyc サービスコールは、cycid で示された周期ハンドラを pk_ccyc の内容で生成します。acre_cyc サービスコールは、未使用の周期ハンドラ ID を周期ハンドラ ID として割り当て、pk_ccyc の内容で周期ハンドラを生成します。割り当てた ID 番号を、戻り値 cycid として返します。生成する周期ハンドラの情報 pk_ccyc について以下に示します。

◆ **cycatr(周期ハンドラ属性)**

cycatr には、[TA_HLNG || TA_ASM] |[TA_STA] | [TA_PHS] を指定できます。カーネルは、TA_HLNG,TA_ASM の値を無視し、C 言語記述、アセンブリ言語の記述を区別しません。

属性	意味
TA_STA	周期ハンドラは動作開始
TA_PHS	起動位相の保存
TA_HLNG	周期ハンドラを高級言語で記述した際に指定します。
TA_ASM	周期ハンドラをアセンブリ言語で記述した際に指定します。

◆ **exinf**

周期ハンドラの拡張情報を指定します。個々で指定された拡張情報は周期ハンドラを起動するときの引数として渡されます。

◆ **cychdr**

周期ハンドラのエントリアドレスを指定します。

- **cyctim**
  周期ハンドラの起動間隔を ms 単位で指定します。ここで指定した起動間隔をもとに次に起動すべき時刻[5]を計算し、その時刻経過後の最初のタイムティックで周期ハンドラは起動します。cyctim には、0は指定できません。また、指定してもエラーは返しません。
- **cycphs**
  周期ハンドラの起動位相をms単位で指定します。属性にTA_PHSが指定されている場合、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。指定されていない場合は、sta_cyc,ista_cyc サービスコールが呼び出された時刻を基準として、周期ハンドラを次に起動する時刻を決定します。cycphs に 0 が指定された場合は、最初のタイムティックで周期ハンドラを起動します。

cycphs>cyctimとなるcycphs,cyctimの値を指定することは出来ません。しかし、指定してもエラーは返しません。cyctim,cycphs に指定可能な値は、0xFFFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。
周期ハンドラを生成するサービスコールが呼び出されてから、指定した cycphs(起動位相)以上の時間が経過した時が、周期ハンドラを一回目に起動すべき時刻となります。その後はcyctim(起動周期)が経過する毎に、周期ハンドラが起動されます。

cre_cyc サービスコールにおいて、すでに存在する周期ハンドラに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_cyc サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。
cre_cyc サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大周期ハンドラ数までです。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

[5] ここで計算された起動時刻は、set_tim,iset_tim による時刻変更の影響を受けません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
   T_CCYC pk_ccyc1;
       :
   pk_ccyc1.cycatr= (ATR)TA_HLNG | TA_STA;
   pk_ccyc1.cychdr= (FP)cyc1;
   pk_ccyc1.cycact = TCY_ON;
   pk_ccyc1.cyctim = 200;
   cre_cyc( ID_cyc1, & pk_ccyc1 );
       :
}
void cyc1(void)
{
   :
}
```

del_cyc　　周期ハンドラの削除

■ C 言語 API

```
ER ercd = del_cyc( ID cycid );
```

■ パラメータ

ID	cycid	削除する周期ハンドラ ID 番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(cycid の周期ハンドラは存在しない)

■ 機能説明

cycid で示された周期ハンドラを削除します。削除された周期ハンドラは、再度、同じ ID 番号で新しい周期ハンドラを生成することができます。

存在しない周期ハンドラに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、cre_cyc,acre_cyc によって生成された周期ハンドラに対して有効です。コンフィギュレーションファイルに静的に定義された周期ハンドラに対して本サービスコールを発行した場合の動作は保証されません。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
  del_cyc( ID_cyc1 );
      :
}
```

sta_cyc　周期ハンドラの動作開始
ista_cyc　周期ハンドラの動作開始(ハンドラ専用)

■ C言語API

```
ER ercd = sta_cyc( ID cycid );
ER ercd = ista_cyc( ID cycid );
```

■ パラメータ

ID	cycid	動作させる周期ハンドラID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(cycidの周期ハンドラは存在しない)

■ 機能説明

cycid で示された周期ハンドラを動作している状態に移行させます。周期ハンドラ属性にTA_PHS が指定されていない場合は、このサービスコールを呼び出した時刻を基準として、その時刻から起動周期が経過する毎に周期ハンドラが起動されます。

TA_PHSが指定されておらず、既に動作状態の周期ハンドラに対して本サービスコールを発行した場合、周期ハンドラが次に起動する時刻を再設定します。

TA_PHS が指定されており、既に動作状態の周期ハンドラに対して本サービスコールを発行した場合、本サービスコールは起動時刻の再設定はしません。

存在しない周期ハンドラに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、sta_cyc、非タスクコンテキストからは、ista_cyc を使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
  sta_cyc ( ID_cyc1 );
      :
}
```

stp_cyc 周期ハンドラの動作停止
istp_cyc 周期ハンドラの動作停止(ハンドラ専用)

■ C言語API

```
ER ercd = stp_cyc( ID cycid );
ER ercd = istp_cyc( ID cycid );
```

■ パラメータ

ID	cycid	停止する周期ハンドラID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(cycidの周期ハンドラは存在しない)

■ 機能説明

cycidで示された周期ハンドラを動作していない状態に移行させます。
存在しない周期ハンドラに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。
本サービスコールは、タスクコンテキストからは、stp_cyc、非タスクコンテキストからは、istp_cycを使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
  stp_cyc ( ID_cyc1 );
      :
}
```

ref_cyc　周期ハンドラの状態参照
iref_cyc　周期ハンドラの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_cyc( ID cycid, T_RCYC *pk_rcyc );
ER ercd = iref_cyc( ID cycid, T_RCYC *pk_rcyc );
```

■ パラメータ

ID	cycid	対象周期ハンドラ ID 番号
T_RCYC	*pk_rcyc	周期ハンドラ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RCYC	*pk_rcyc	周期ハンドラ状態を返すパケットへのポインタ

pk_rcyc の内容

```
typedef   struct t_rcyc{
    STAT    cycstat  +0   4    周期ハンドラの動作状態
    RELTIM  lefttim  +4   4    周期ハンドラ起動までの時間
} T_RCYC;
```

■エラーコード

E_NOEXS	未登録状態(cycid の周期ハンドラは存在しない)

■ 機能説明

cycid で示された周期ハンドラの各種の状態を返します。

◆ **cycstat**

対象周期ハンドラの状態を返します。

- ･TCYC_STA　周期ハンドラは動作している
- ･TCYC_STP　周期ハンドラは動作していない

◆ **lefttim**

対象周期ハンドラの次に起動するまでの残り時間を返します。単位は、"ms"です。対象周期ハンドラが動作していない場合は、不定値となります。

存在しない周期ハンドラに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、ref_cyc、非タスクコンテキストからは、iref_cyc を使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    T_RCYC rcyc;
    ref_cyc( ID_cyc1 ,&rcyc );
}
```

1.15. 時間管理機能(アラームハンドラ)

cre_alm　アラームハンドラの生成
acre_alm　アラームハンドラの生成(ID 自動割り当て)

■ **C 言語 API**

```
ER ercd = cre_alm( ID almid, T_CALM *pk_calm );
ER_ID almid = acre_alm( T_CALM *pk_calm );
```

■ **パラメータ**

ID	almid	生成するアラームハンドラ ID 番号(cre_alm 使用時)
T_CALM	*pk_calm	アラームハンドラ生成情報を格納した構造体へのポインタ

pk_calm の内容

```
typedef   struct t_calm {
    ATR      almatr   0     4     アラームハンドラ属性
    VP_INT   exinf    +4    4     拡張情報
    FP       almhdr   +8    4     アラームハンドラエントリアドレス
} T_CALM;
```

■ **リターンパラメータ**

- cre_alm の場合

ER	ercd	正常終了(E_OK)またはエラーコード

- acre_alm の場合

ER_ID	almid>0	生成したアラームハンドラの ID 番号
	almid<0	エラーコード

■ **エラーコード**

E_NOID	空き ID なし(acre_alm 使用時)
E_OBJ	オブジェクト状態が不正 (almid のアラームハンドラが存在(cre_alm 使用時))

■ **機能説明**

cre_alm サービスコールは、almid で示されたアラームハンドラを pk_calm の内容で生成します。acre_alm サービスコールは、未使用のアラームハンドラ ID をアラームハンドラ ID として割り当て、pk_calm の内容でアラームハンドラを生成します。割り当てた ID 番号を、戻り値 almid として返します。生成するアラームハンドラの情報 pk_calm について以下に示します。

◆ **almatr(アラームハンドラ属性)**

almatr には、[TA_HLNG || TA_ASM]を指定できます。
カーネルは、TA_HLNG,TA_ASM の値を無視し、C 言語記述、アセンブリ言語の記述を区別しません。

属性	意味
TA_HLNG	アラームハンドラを高級言語で記述した際に指定します。
TA_ASM	アラームハンドラをアセンブリ言語で記述した際に指定します。

◆ **exinf**

アラームハンドラの拡張情報を指定します。個々で指定された拡張情報はアラームハンドラを起動するときの引数として渡されます。

◆ **almhdr**

アラームハンドラのエントリアドレスを指定します。

アラームハンドラの生成直後は、アラームハンドラの起動時刻は設定されておらず、動作も停止状態となっています。起動時刻の設定、動作の開始は、sta_alm,ista_alm サービスコールを使用する必要があります。

cre_alm サービスコールにおいて、すでに存在するアラームハンドラに対して、本サービスコールを発行した場合は、エラーE_OBJ を返します。acre_alm サービスコールにおいて空きIDが存在しない場合は、エラーE_NOID を返します。
cre_alm サービスコールにおいて本サービスコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大アラームハンドラ数までです。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
   T_CALM pk_calm1;
       :
   pk_calm1.almhdr= (FP)alm1;
   cre_alm( ID_alm1, &pk_calm1 );
       :
}
void alm1(void)
{
   :
}
```

del_alm　　アラームハンドラの削除

■ C言語API

```
ER ercd = del_alm( ID almid );
```

■ パラメータ

ID	almid	削除するアラームハンドラID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(almidのアラームハンドラは存在しない)

■ 機能説明

almidで示されたアラームハンドラを削除します。削除されたアラームハンドラは、再度、同じID番号で新しいアラームハンドラを生成することができます。
存在しないアラームハンドラに対して、本サービスコールを発行した場合は、エラー E_NOEXSを返します。
本サービスコールは、cre_alm,acre_alm によって生成されたアラームハンドラに対して有効です。コンフィギュレーションファイルに静的に定義されたアラームハンドラに対して本サービスコールを発行した場合の動作は保証されません。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
  del_alm( ID_alm1 );
      :
}
```

sta_alm　アラームハンドラの動作開始
ista_alm　アラームハンドラの動作開始(ハンドラ専用)

■ C言語API

```
ER ercd = sta_alm( ID almid, RELTIM almtim );
ER ercd = ista_alm( ID almid, RELTIM almtim );
```

■ パラメータ

ID	almid	動作させるアラームハンドラID番号
RELTIM	almtim	アラームハンドラの起動時刻

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(almidのアラームハンドラは存在しない)

■ 機能説明

almid で示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 で指定された時間後の時刻と設定し、アラームハンドラを動作している状態に移行させます。almtimには、ms単位で時間を設定します。almtimに指定可能な値は、0xFFFFFFFF/tick_deno(コンフィギュレーションファイルのシステム定義の tick_deno の値)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。

既に動作しているアラームハンドラが指定された場合は、以前の起動時刻の設定を解除し、新しい起動時刻に更新します。almtimに0が指定された場合は、次のタイムティックでアラームハンドラが起動します。

存在しないアラームハンドラに対して、本サービスコールを発行した場合は、エラー E_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、sta_alm、非タスクコンテキストからは、ista_alm を使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
   sta_alm( ID_alm1, 100 );
      :
}
```

stp_alm　アラームハンドラの動作停止
istp_alm　アラームハンドラの動作停止(ハンドラ専用)

■ C言語API

```
ER ercd = stp_alm( ID almid );
ER ercd = istp_alm( ID almid );
```

■ パラメータ

ID	almid	停止するアラームハンドラID番号

■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(almidのアラームハンドラは存在しない)

■ 機能説明

almidで示されたアラームハンドラを動作していない状態に移行させます。
存在しないアラームハンドラに対して、本サービスコールを発行した場合は、エラー E_NOEXSを返します。
本サービスコールは、タスクコンテキストからは、stp_alm、非タスクコンテキストからは、istp_almを使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
  stp_alm( ID_alm1 );
      :
}
```

ref_alm　アラームハンドラの状態参照
iref_alm　アラームハンドラの状態参照(ハンドラ専用)

■ C言語API

```
ER ercd = ref_alm( ID almid, T_RALM *pk_ralm );
ER ercd = iref_alm( ID almid, T_RALM *pk_ralm );
```

■ パラメータ

ID	almid	対象アラームハンドラID番号
T_RALM	*pk_ralm	アラームハンドラ状態を返すパケットへのポインタ

■リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RALM	*pk_ralm	アラームハンドラ状態を返すパケットへのポインタ

pk_ralmの内容

```
typedef   struct t_ralm{
    STAT    almstat  +0   4    アラームハンドラの動作状態
    RELTIM  lefttim  +4   4    アラームハンドラ起動までの時間
} T_RALM;
```

■エラーコード

E_NOEXS	未登録状態(almidのアラームハンドラは存在しない)

■ 機能説明

almidで示されたアラームハンドラの各種の状態を返します。

◆ **almstat**

対象アラームハンドラの状態を返します。

- ･TALM_STA　アラームハンドラは動作している
- ･TALM_STP　アラームハンドラは動作していない

◆ **lefttim**

対象アラームハンドラの次に起動するまでの残り時間を返します。単位は、"ms"です。対象アラームハンドラが動作していない場合は、不定値となります。

存在しないアラームハンドラに対して、本サービスコールを発行した場合は、エラーE_NOEXS を返します。

本サービスコールは、タスクコンテキストからは、ref_alm、非タスクコンテキストからは、iref_almを使用してください。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void func()
{
    T_RALM   ralm;
    ref_alm( ID_alarm, &ralm );
              :
}
```

1.16. システム状態機能

rot_rdq　タスク優先順位の回転
irot_rdq　タスク優先順位の回転(ハンドラ専用)

■ C言語API

```
ER ercd = rot_rdq( PRI tskpri );
ER ercd = irot_rdq( PRI tskpri );
```

■ パラメータ

PRI	tskpri	回転するタスク優先度

■ リターンパラメータ

ER	ercd	正常終了(E_OK)

■ エラーコード

なし

■ 機能説明

tskpri で示された優先度のレディキューを回転します。すなわち、その優先度のレディキューの先頭につながれているタスクをレディキューの最後尾につなぎかえ、同一優先度のタスクの実行を切り替えます。この様子を表 1に示します。

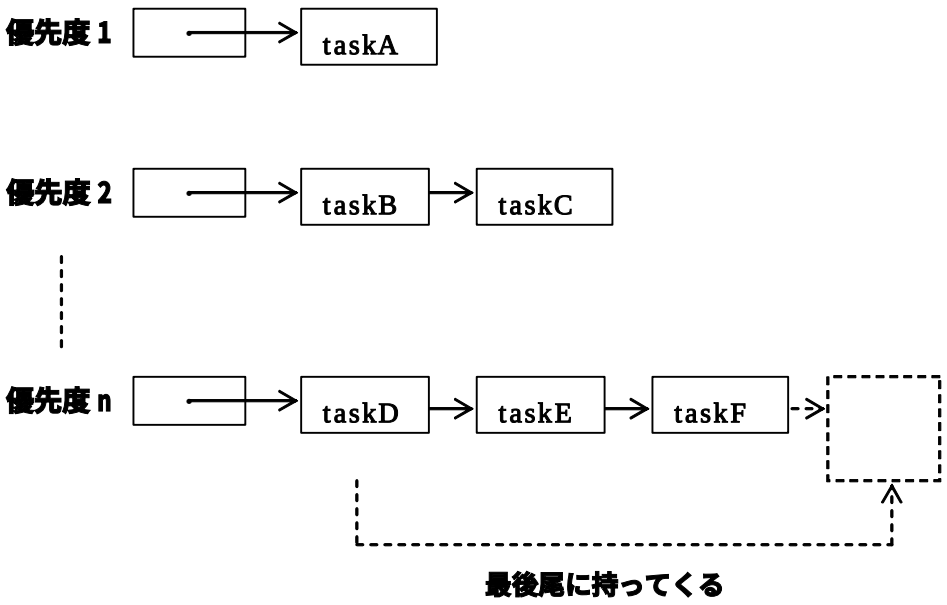

表 1　rot_rdq サービスコールによるレディキューの操作

このサービスコールを一定時間間隔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おこなうことができます。rot_rdq サービスコール使用時は、tskpri=TPRI_SELF の指定により、自タスクの持つ優先度のレディキューを回転させます。irot_rdq サービスコールで TPRI_SELF を指定することは出来ません。指定してもエラーとなりません。
また、本サービスコールで自タスクの優先度を指定した場合には、自タスクがそのレディキューの最後尾にまわることになります。なお、指定した優先度のレディキューにタスクがない場合は何も行いません。
本サービスコールは、タスクコンテキストからは、rot_rdq、非タスクコンテキストからは、irot_rdq を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
        :
    rot_rdq( 2 );
        :
}
```

get_tid　実行中タスク ID の参照
iget_tid　実行中タスク ID の参照(ハンドラ専用)

■ C 言語 API

```
ER ercd = get_tid( ID *p_tskid );
ER ercd = iget_tid( ID *p_tskid );
```

■ パラメータ

ID	*p_tskid	タスク ID へのポインタ

■ リターンパラメータ

ER	ercd	正常終了(E_OK)
ID	*p_tskid	タスク ID へのポインタ

■ エラーコード

なし

■ 機能説明

実行状態のタスク ID を p_tskid の指す領域に返します。タスクから本サービスコールを発行した場合、自タスクの ID 番号を返します。また、非タスクコンテキストから本サービスコールを発行した場合は、そのとき実行していたタスク ID を返します。実行状態のタスクがない場合は、TSK_NONE を返します。

本サービスコールは、タスクコンテキストからは、get_tid、非タスクコンテキストからは、iget_tid を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
  ID tskid;
    :
   get_tid(&tskid);
    :
}
```

loc_cpu　CPU ロック状態への移行
iloc_cpu　CPU ロック状態への移行(ハンドラ専用)

■ C 言語 API

```
ER ercd = loc_cpu();
ER ercd = iloc_cpu();
```

■ パラメータ

なし

■ リターンパラメータ

ER	ercd	正常終了(E_OK)

■ エラーコード

なし

■ 機能説明

システム状態を CPU ロック状態とし、割込みとタスクのディスパッチを禁止します。CPU ロック状態の特長を以下に示します。

(1) CPU ロック状態の間は、タスクのスケジューリングは行われません。
(2) すべての外部割り込みが禁止されます。
(3) CPU ロック状態から呼び出し可能なサービスコールは、以下のサービスコールのみです。その他のサービスコールが呼び出された場合の動作は保証されません。

- ext_tsk
- exd_tsk
- sns_tex
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn

CPU ロック状態は、以下の操作で解除されます。

(a) unl_cpu, iunl_cpu サービスコールの呼び出し
(b) ext_tsk, exd_tsk サービスコールの呼び出し

CPU ロック状態と CPU ロック解除状態の間の遷移は、loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, ext_tsk,exd_tsk サービスコールによってのみ発生します。割込みハンドラ、アラームハンドラ、周期ハンドラ、タスク例外処理ルーチンの終了時には、必ず CPU ロック解除状態でなければなりません。[6]CPU ロック状態の場合、動作は保証されません。なお、これらのハンドラ開始時は、常に CPU ロック解除状態です。

すでに CPU ロック状態のときに、再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

本サービスコールは、タスクコンテキストからは、loc_cpu、非タスクコンテキストからは、iloc_cpu を使用してください。

[6] ただし、割込ハンドラ終了時は、CPU ロック解除状態でなければいけませんが、割込は禁止状態でなければいけません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
   loc_cpu();
      :
}
```

unl_cpu　CPU ロック状態の解除
iunl_cpu　CPU ロック状態の解除(ハンドラ専用)

■ C 言語 API

```
ER ercd = unl_cpu();
ER ercd = iunl_cpu();
```

■ パラメータ

なし

■ リターンパラメータ

ER	ercd	正常終了(E_OK)

■ エラーコード

なし

■ 機能説明

loc_cpu, iloc_cpu サービスコールによって設定されていた CPU ロック状態を解除します。ディスパッチ許可状態から unl_cpu サービスコールを発行した場合、タスクのスケジューリングが行われます。割込みハンドラ内で iloc_cpu を呼び出し、CPU ロック状態に移行した場合は、割込みハンドラからリターンする前に必ず iunl_cpu を呼び出し、CPU ロック状態を解除してください。

CPU ロック状態とディスパッチ禁止状態は、独立して管理されます。そのため、unl_cpu, iunl_cpu サービスコールでは、dis_dsp サービスコールによるディスパッチ禁止状態は解除されません。

本サービスコールは、タスクコンテキストからは、unl_cpu、非タスクコンテキストからは、iunl_cpu を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
     :
    unl_cpu();
     :
}
```

dis_dsp　　ディスパッチの禁止

■ C 言語 API

```
ER ercd = dis_dsp();
```

■ パラメータ

なし

■ リターンパラメータ

ER	ercd	正常終了(E_OK)

■ エラーコード

なし

■ 機能説明

システム状態をディスパッチ禁止状態にします。ディスパッチ禁止状態の特長を、以下に示します。

(1) タスクのスケジューリングが行われなくなるため、自タスク以外のタスクが実行状態に移行することはなくなります。
(2) 割込みは受け付けられます。
(3) 待ち状態になるサービスコールを呼び出せません。

ディスパッチ禁止状態の間に以下の操作を行うと、システム状態はタスク実行状態に戻ります。

(a) ena_dsp サービスコールの呼び出し
(b) ext_tsk, exd_tsk サービスコールの呼び出し

ディスパッチ禁止状態とディスパッチ許可状態の間の遷移は、dis_dsp, ena_dsp, ext_tsk, exd_tsk サービスコールによってのみ発生します。
すでにディスパッチ禁止状態のときに再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
        :
  dis_dsp();
        :
}
```

ena_dsp　　ディスパッチの許可

■ C言語API

```
ER ercd = ena_dsp();
```

■ パラメータ

なし

■ リターンパラメータ

ER	ercd	正常終了(E_OK)

■ エラーコード

なし

■ 機能説明

dis_dspサービスコールによって設定されていたディスパッチ禁止状態を解除します。それにより、システムがタスク実行状態になった場合は、タスクのスケジューリングが行われます。
タスク実行状態から本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
    ena_dsp();
      :
}
```

sns_ctx　　コンテキストの参照

■ C 言語 API

```
ER ercd = sns_ctx();
```

■ パラメータ

なし

■ リターンパラメータ

BOOL	state	TRUE:非タスクコンテキスト FALSE:タスクコンテキスト

■ エラーコード

なし

■ 機能説明

非タスクコンテキストから呼び出された場合に TRUE、タスクコンテキストから呼び出された場合に FALSE を返します。本サービスコールは、CPU ロック状態からも呼び出せます。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
    sns_ctx();
      :
}
```

sns_loc　　CPU ロック状態の参照

■ C 言語 API

```
ER ercd = sns_loc();
```

■ パラメータ

なし

■ リターンパラメータ

BOOL	state	TRUE:CPU ロック状態 FALSE:CPU ロック解除状態

■ エラーコード

なし

■ 機能説明

システムが CPU ロック状態の場合に TRUE、CPU ロック解除状態の場合に FALSE を返します。
本サービスコールは、CPU ロック状態からも呼び出せます。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
    sns_loc();
      :
}
```

sns_dsp　　ディスパッチ禁止状態の参照

■ C 言語 API

```
ER ercd = sns_dsp();
```

■ パラメータ

なし

■ リターンパラメータ

	BOOL	state	TRUE：ディスパッチ禁止状態 FALSE：ディスパッチ許可状態

■ エラーコード

なし

■ 機能説明

システムがディスパッチ禁止状態の場合に TRUE、ディスパッチ許可状態の場合に FALSE を返します。

本サービスコールは、CPU ロック状態からも呼び出せます。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
    sns_dsp();
      :
}
```

sns_dpn　　ディスパッチ保留状態の参照

■ C 言語 API

```
ER ercd = sns_dpn();
```

■ パラメータ

なし

■ リターンパラメータ

BOOL	state	TRUE：ディスパッチ保留状態 FALSE：ディスパッチ保留状態ではない

■ エラーコード

なし

■ 機能説明

システムがディスパッチ保留状態の場合に TRUE、そうでない場合に FALSE を返します。具体的には、以下の全ての条件が満足される場合に FALSE を返し、その他の場合には TRUE を返します。

(1) ディスパッチ禁止状態でない
(2) CPU ロック状態でない
(3) タスクまたはタスク例外処理ルーチンである

本サービスコールは、CPU ロック状態からも呼び出せます。システムがディスパッチ禁止状態の場合に TRUE、ディスパッチ許可状態の場合に FALSE を返します。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
      :
    sns_dpn();
      :
}
```

1.17. 割込管理機能

def_inh　割込ハンドラの定義
idef_inh　割込ハンドラの定義(ハンドラ専用)

■ **C 言語 API**

```
ER ercd = def_inh( INHNO inhno, T_DINH *pk_dinh );
ER ercd = idef_inh( INHNO inhno, T_DINH *pk_dinh );
```

■ **パラメータ**

INHNO	inhno	定義する割込ハンドラ番号
T_DINH	*pk_dinh	割込ハンドラ定義情報を格納したパケットへのポインタ

pk_dinh の内容

```
typedef   struct t_dinh {
    ATR     inhatr   0     4     割込ハンドラ属性
    FP      inthdr   +4    4     割込ハンドラエントリアドレス
} T_DINH;
```

■ **リターンパラメータ**

ER	ercd	正常終了(E_OK)

■ **エラーコード**

なし

■ **機能説明**

inhno で示される割り込み番号に割り込みハンドラを定義し、割り込みハンドラを使用可能にします。すなわち、inhno で示される割り込みベクタの領域に割り込みエントリアドレスを設定します。

◆ **intatr**

カーネルは、TA_HLNG,TA_ASM の値を無視し、C 言語記述、アセンブリ言語の記述を区別しません。

属性	意味
TA_HLNG	割込ハンドラを高級言語で記述した際に指定します。
TA_ASM	割込ハンドラをアセンブリ言語で記述した際に指定します。

◆ **inthdr**

定義する割込ハンドラのエントリアドレスを指定します。

pk_dinh=NULL とした場合には、前に定義した割り込みハンドラの定義解除を行います。
すでに定義済みの割り込み番号に対して割り込みハンドラを再定義することも可能です。再定義の場合、あらかじめ定義の解除を行う必要はありません。既に定義されている割り込み番号に対して新しいハンドラを再定義してもエラーにはなりません。
本サービスコールは、カーネルが管理する割り込みベクタテーブル(INTERRUPT_VECTOR セクション)を RAM 上に割り当てないと正常に動作しません。
本サービスコールは、タスクコンテキストからは、def_inh、非タスクコンテキストからは、idef_inh を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
   T_DINH dinh1;
       :
   dinh1.inthdr = inh1;
   def_inh( ID_inh1, &dinh1 );
       :
}
void inh1(void)
{
   :
}
```

1.18. システム構成理機能

ref_ver　　バージョン情報の参照
iref_ver　　バージョン情報の参照(ハンドラ専用)

■ C 言語 API

```
ER ercd = ref_ver( T_RVER *pk_rver );
ER ercd = iref_ver( T_RVER *pk_rver );
```

■ パラメータ

T_RVER	*pk_rver	バージョン情報を返すパケットへのポインタ

pk_rver の内容

```
typedef   struct t_rver {
    UH      maker    0     2    カーネルのメーカーコード
    UH      prid     +2    2    カーネルの識別番号
    UH      spver    +4    2    ITRON 仕様のバージョン番号
    UH      prver    +6    2    カーネルのバージョン番号
    UH      prno[4]  +8    2    カーネル製品の管理情報
} T_RVER;
```

■ リターンパラメータ

ER	ercd	正常終了(E_OK)

■ エラーコード

なし

■ 機能説明

現在実行中のカーネルのバージョンに関する情報を読み出し、その結果を pk_rver の指す領域に返します。

pk_rver の指すパケットには、次の情報を返します。

- **maker**
  株式会社ルネサステクノロジを示す H'115 が返されます。
- **prid**
  M3T-MR32R の内部識別 IDH'221 が返されます。
- **spver**
  μITRON 仕様書 Ver4.02.00 に準拠していることを示す H'5402 が返されます。
- **prver**
  M3T-MR32R のバージョンを示す H'400 が返されます。
- **prno**
  - **prno[0]**
    製品のﾘﾘｰｽ番号'01'が得られます。
  - **prno[1]**
    製品のﾘﾘｰｽ年の下 2 桁(西暦)と月 H'0411 が得られます。
  - **prno[2]**
    拡張のための予約。
  - **prno[3]**
    拡張のための予約。

本サービスコールは、タスクコンテキストからは、ref_ver、非タスクコンテキストからは、iref_ver を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    T_RVER   pk_rver;
    ref_ver( &pk_rver );
}
```

1.19. 拡張機能

vrst_dtq　　データキュー領域の初期化

■ C言語API

```
ER ercd = vrst_dtq( ID dtqid );
```

■ パラメータ

ID	dtqid	クリアするデータキューID

■ リターンパラメータ

ER	ercd	正常終了(E_OK) またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(dtqid のデータキューは存在しない)

■ 機能説明

dtqid で示されたデータキューに格納されているデータをクリアします。データキュー領域にデータがなく、データ送信待ち行列にタスクがつながれている場合、データ送信待ち行列につながれているすべてのタスクの待ち状態を解除します。さらに解除されたタスクに対してエラーEV_RST が返されます。
対象データキューが存在しない場合は、E_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1(void)
{
      :
    vrst_dtq( ID_dtq1 );
      :
}
```

vrst_mbx　　メールボックス領域の初期化

■ C 言語 API

```
ER ercd = vrst_mbx( ID mbxid );
```

■ パラメータ

ID	mbxid	クリアするメールボックス ID

■ リターンパラメータ

ER	ercd	正常終了(E_OK) またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mbxid のメールボックスは存在しない)

■ 機能説明

mbxid で示されたメールボックスに格納されているメッセージをクリアします。
対象メールボックスが存在しない場合は、E_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1(void)
{
      :
    vrst_mbx( ID_mbx1 );
      :
}
```

vrst_mbf　　メッセージバッファ領域の初期化

■ C 言語 API

```
ER ercd = vrst_mbf( ID mbfid );
```

■ パラメータ

ID	mbfid	クリアするメッセージバッファ ID

■ リターンパラメータ

ER	ercd	正常終了(E_OK) またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mbfid のメッセージバッファは存在しない)

■ 機能説明

mbfid で示されたメッセージバッファに格納されているメッセージをクリアします。メッセージバッファ領域にメッセージがなく、メッセージ送信待ち行列にタスクがつながれている場合、メッセージ送信待ち行列につながれているすべてのタスクの待ち状態を解除します。さらに解除されたタスクに対してエラーEV_RST が返されます。
対象メッセージバッファが存在しない場合は、E_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1()
{
      :
    vrst_mbf(ID_mbf1)
      :
}
```

vrst_mpf　　固定長メモリプール領域の初期化

■ C 言語 API

```
ER ercd = vrst_mpf( ID mpfid );
```

■ パラメータ

ID	mpfid	クリアする固定長メモリプール ID

■ リターンパラメータ

ER	ercd	正常終了(E_OK) またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mpfid の固定長メモリプールは存在しない)

■ 機能説明

mpfid で示された固定長メモリプール初期状態にします。メモリブロック獲得待ち行列にタスクがつながれている場合、メモリブロック獲得待ち行列につながれているすべてのタスクの待ち状態を解除します。さらに解除されたタスクに対してエラーEV_RST が返されます。
対象固定長メモリプールが存在しない場合は、E_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1()
{
      :
    vrst_mpf(ID_mpf1))
      :
}
```

vrst_mpl　　可変長メモリプール領域の初期化

■ C言語API

```
ER ercd = vrst_mpl( ID mplid );
```

■ パラメータ

ID	mplid	クリアする可変長メモリプール ID

■ リターンパラメータ

ER	ercd	正常終了(E_OK) またはエラーコード

■ エラーコード

E_NOEXS	未登録状態(mplid の可変長メモリプールは存在しない)

■ 機能説明

mplidで示された可変長メモリプール初期状態にします。メモリブロック獲得待ち行列にタスクがつながれている場合、メモリブロック獲得待ち行列につながれているすべてのタスクの待ち状態を解除します。さらに解除されたタスクに対してエラーEV_RST が返されます。
対象可変長メモリプールが存在しない場合は、E_NOEXS を返します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場合は正常に動作しません。

■ 使用例

《 c言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1()
{
      :
    vrst_mpl(ID_mpl1))
      :
}
```

第2章　スタック使用量の計算

2.1. 各サービスコールのスタック使用量

MR32R は、全てのサービスコールにおいて、サービスコール発行コンテキストのスタック領域で 76 バイト、システムスタック領域において 4 バイト消費します。さらに表 2 サービスコールのスタック使用量に示すサービスコールについては、表で示したサイズ分だけ余分にシステムスタック領域を消費します。

表 2 サービスコールのスタック使用量

サービスコール	使用システムスタック
cre_tsk	60
del_tsk	28
exd_tsk	28
cre_dtq	60
del_dtq	28
cre_mbf	60
del_mbf	28
cre_mpf	60
del_mpf	28
cre_mpl	68
del_mpl	28
get_mpl	68
tget_mpl	68
pget_mpl	68
rel_mpl	20
vrst_mpl	40

以下にサービスコールのスタック使用量計算例を示します。

(1) タスクから sta_tsk を発行した場合
発行コンテキストがタスクであるため、発行タスクのスタック領域は、76 バイト消費します。システムスタックは、4 バイト消費します。
(2) 割り込みハンドラから isnd_mbx を発行した場合
発行コンテキストは、非タスクコンテキストであるため、システムスタックを 76 バイト消費します。さらに、4 バイト分消費し、合計 80 バイト消費することになります。この場合、タスクのスタック領域を消費することはありません。
(3) タスクから cre_mpl を発行した場合
発行コンテキストが発行タスクのスタック領域は、76 バイト消費します。システムスタックは、4 バイト消費します。さらに、表 2 サービスコールのスタック使用量に示すサイズ分余分にシステムスタックを消費するため、システムスタックは、合計 64 バイト消費します。

2.2. スタックサイズの算出方法

MR32R のスタックには、システムスタックとユーザスタックの 2 種類があります。スタックサイズの計算方法は、ユーザー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

タスクが使用するスタックです。従って、MR32R を使ってアプリケーションプログラムを記述する場合には、各タスクごとにスタック領域を確保する必要があります。

●システムスタック

MR32R 内部もしくはハンドラ実行中に使用するスタックサイズです。MR32R では、サービスコールをタスクが発行するとユーザースタックからシステムスタックに切り替えます。システムスタックは、マイコンの割り込みスタックを使用します。

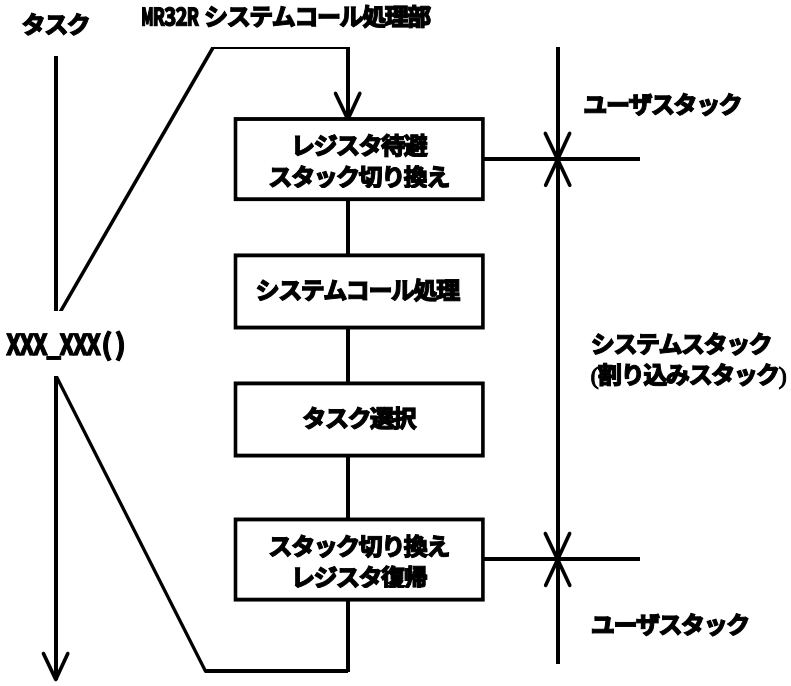

表 3　　システムスタックとユーザスタック

システムスタックとユーザスタックの各セクションの配置は以下のようになります。

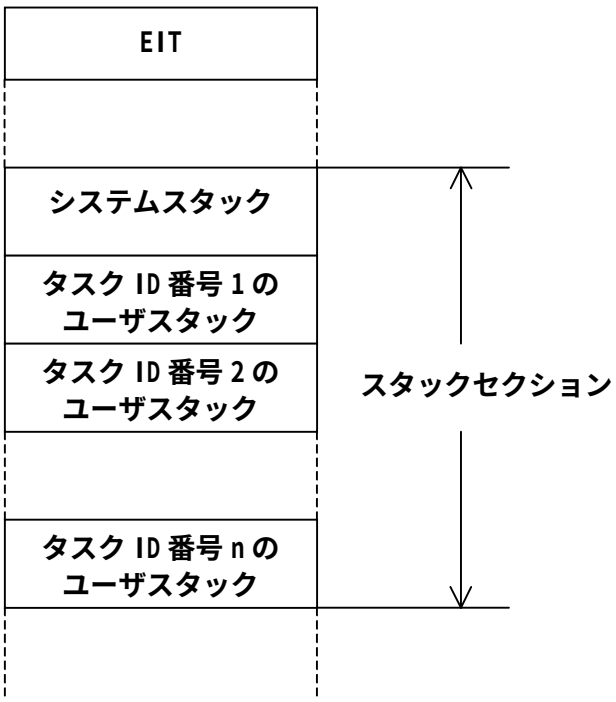

表 4　　スタックの配置

2.2.1. ユーザスタックの算出方法

ユーザスタックは、各タスクごとに算出する必要があります。以下にアプリケーションを C 言語で記述した場合とアセンブリ言語で記述した場合のスタックの算出方法を以下に示します。

- C 言語でアプリケーションを記述した場合

C 言語でアプリケーションを記述した場合、下記に示す方法でユーザスタックサイズを求めることができます。

1. タスクで使用しているスタックサイズを算出します。
2. **MR32R** が消費するスタックサイズ
   タスクからサービスコールを発行することによりそのサービスコールが消費するスタックサイズは、76 バイトになります。

上記 1、2 の合計サイズが、ユーザースタックサイズとなります。

●アセンブリ言語でアプリケーションを記述した場合

1. ユーザプログラムで使用する部分
   タスクでレジスタをスタックに保存する場合に使用するサイズです。
2. サービスコール発行により消費するスタックサイズ
   タスクからサービスコールを発行することによりそのサービスコールが消費するスタックサイズは、76 バイトになります。

上記 1、2 の合計サイズが、ユーザースタックサイズとなります。
表 5にユーザスタックの算出例を示します。

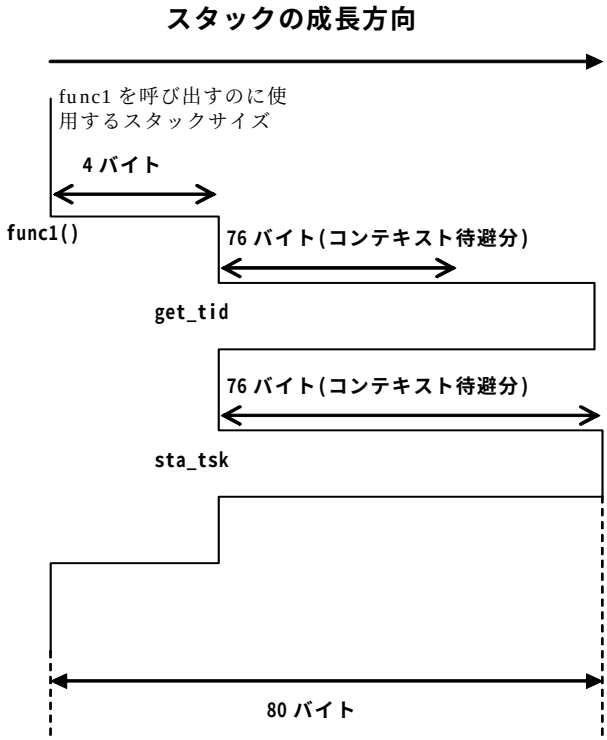

表 5　　ユーザースタックサイズの算出例

2.2.2. システムスタックの算出方法

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[7]に割り込みが発生し、その上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。

$$\text{システムスタックの必要量} = \alpha + \sum \beta i (+\gamma)$$

- α

  使用するサービスコールの中で最大のシステムスタックサイズ。
  例えば、sta_tsk、ext_tsk、cre_tsk を使用する場合、

サービスコール名	システムスタックサイズ
cre_tsk	60 バイト
sta_tsk	4 バイト
ext_tsk	4 バイト
slp_tsk	4 バイト

  となるのでこの場合、使用するサービスコールの中で最大のシステムスタックサイズは cre_tsk の場合で 60 バイトです。

- βi

  割り込みハンドラの使用するスタックサイズ。詳細は後述します。

- γ

  システムクロック割り込みハンドラの使用するスタックサイズ。詳細は後述します。

[7] ユーザスタックからシステムスタックに切り替えた後

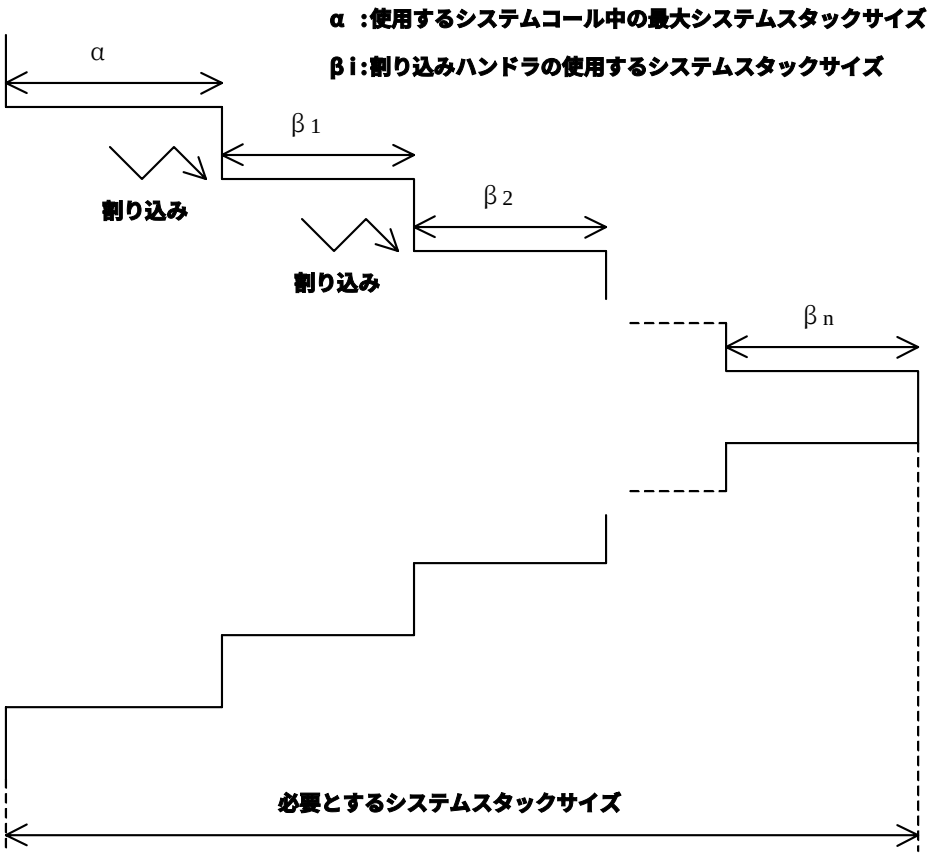

表 6　　システムスタックサイズの算出方法

【 割り込みハンドラの使用するスタックサイズβi 】

サービスコール中に発生した割り込みハンドラの使用するスタックサイズは以下の 3 つのサイズの合計となります。

- コンテキスト待避領域
- 割り込み処理から呼び出されるサブルーチンのスタック使用量の最大値
- ユーザの記述する部分で使用するスタック使用量[8]

C 言語で記述したときも、アセンブリ言語で記述したときもスタック使用量は上記の方法で算出することができます。

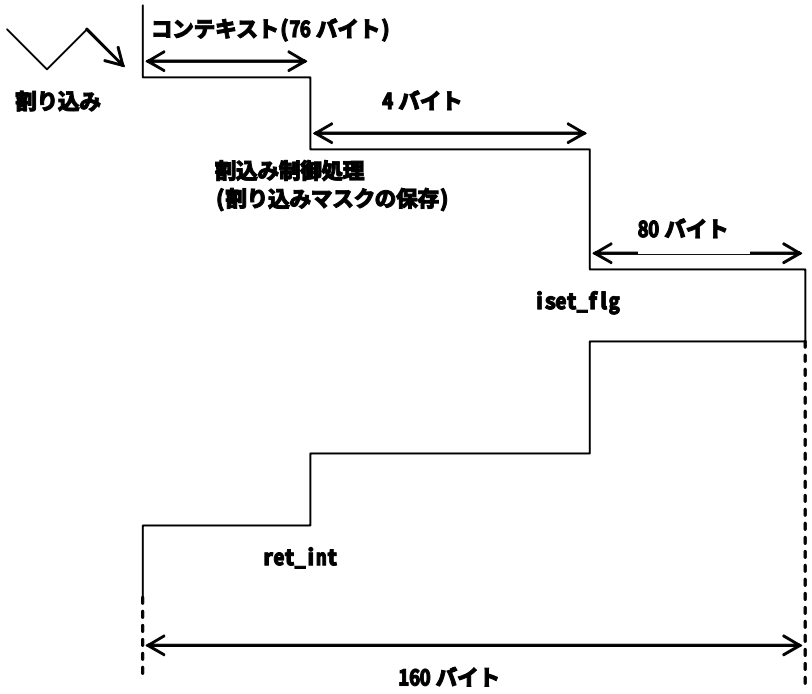

表 7　　割り込みハンドラの使用するスタック量

[8] __RESTORE_IPL_from_STACK、__SAVE_IPL_to_STACK でレジスタを退避するのに使用するスタックサイズです。これらの割り込み制御プログラムについての詳細は MR32R ユーザーズマニュアル 割り込み制御プログラムの作成方法をご覧ください。

【 システムクロック割り込みハンドラが使用するシステムスタックサイズ γ 】

システムクロックを使用しないときは、システムクロック割り込みハンドラが使用するシステムスタックを加算する必要はありません。

システムクロック割り込みハンドラが使用するシステムスタック量 γ は以下に示すサイズです。

88+周期起動ハンドラ、アラームハンドラでスタック使用量の最も大きいサイズ

になります。(上記数値には、コンテキスト待避分を含みます。)

割り込みハンドラとシステムクロック割り込みハンドラを併用して使用する場合は、双方の使用するスタックサイズを加算してください。

第3章 付録

3.1. サービスコール一覧

タスク管理機能サービスコール

サービスコール名		機能
cre_tsk		タスクを生成します
acre_tsk		タスクを生成します(自動 ID 割り当て)
del_tsk		タスクを削除します
act_tsk	[S]	タスクを起動します
iact_tsk	[S]	タスクを起動します
can_act	[S]	タスク起動要求をクリアします
ican_act	[S]	タスク起動要求をクリアします(ハンドラ専用)
sta_tsk		タスクを起動します(起動コード指定)
ista_tsk		タスクを起動します(起動コード指定,ハンドラ専用)
ext_tsk	[S]	自タスクを正常終了します
exd_tsk		自タスクを終了後削除します
ter_tsk	[S]	他タスクを強制的に異常終了します
chg_pri	[S]	タスクの優先度を変更します
ichg_pri		タスクの優先度を変更します(ハンドラ専用)
get_pri	[S]	優先度を取得します
iget_pri		優先度を取得します(ハンドラ専用)
ref_tsk		タスクの状態を参照します
iref_tsk		タスクの状態を参照します(ハンドラ専用)
ref_tst		タスク状態を参照します(簡易版)
iref_tst		タスク状態を参照します(簡易版,ハンドラ専用)

タスク付属同期機能サービスコール

サービスコール名		機能
slp_tsk	[S]	タスクを待ち状態へ移行する
tslp_tsk	[S]	タスクを一定時間待ち状態へ移行する
wup_tsk	[S]	待ち状態のタスクを起床します
iwup_tsk	[S]	待ち状態のタスクを起床します(ハンドラ専用)
can_wup	[S]	タスクの起床要求を無効にします
ican_wup		タスクの起床要求を無効にします(ハンドラ専用)
rel_wai	[S]	待ち状態を強制解除します
irel_wai	[S]	待ち状態を強制解除します(ハンドラ専用)
sus_tsk	[S]	タスクを強制待ち状態を移行します
isus_tsk		タスクを強制待ち状態を移行します(ハンドラ専用)
rsm_tsk	[S]	強制待ち状態のタスクを再開します
irsm_tsk		強制待ち状態のタスクを再開します(ハンドラ専用)
frsm_tsk	[S]	強制待ち状態のタスクを強制再開します
ifrsm_tsk		強制待ち状態のタスクを強制再開します(ハンドラ専用)
dly_tsk	[S]	タスクの実行を一定時間遅延します。

タスク例外処理ルーチン

サービスコール名		機能
def_tex		タスク例外を登録します。
ras_tex	[S]	タスク例外の起動を要求します
iras_tex	[S]	タスク例外の起動を要求します(ハンドラ専用)
dis_tex	[S]	タスク例外の起動を禁止します
ena_tex	[S]	タスク例外の起動を許可します
sns_tex	[S]	タスク例外禁止状態を参照します
ref_tex		タスク例外の状態を参照します
iref_tex		タスク例外の状態を参照します(ハンドラ専用)

セマフォ

サービスコール名		機能
cre_sem		セマフォを生成します
acre_sem		セマフォを生成します(自動 ID 割り当て)
del_sem		セマフォを削除します
sig_sem	[S]	セマフォ資源を返却します
isig_sem	[S]	セマフォ資源を返却します(ハンドラ専用)
wai_sem	[S]	セマフォ資源を獲得します
twai_sem	[S]	セマフォ資源を獲得します(タイムアウトあり)
pol_sem	[S]	セマフォ資源を獲得します(ポーリング)
ipol_sem		セマフォ資源を獲得します(ポーリング,ハンドラ専用)
ref_sem		セマフォの状態を参照します
iref_sem		セマフォの状態を参照します(ハンドラ専用)

イベントフラグ

サービスコール名		機能
cre_flg		イベントフラグを生成します
acre_flg		イベントフラグを生成します(自動 ID 割り当て)
del_flg		イベントフラグを削除します
set_flg	[S]	イベントフラグをセットします
iset_flg	[S]	イベントフラグをセットします (ハンドラ専用)
clr_flg	[S]	イベントフラグをクリアします
iclr_flg		イベントフラグをクリアします(ハンドラ専用)
wai_flg	[S]	イベントフラグを待ちます
twai_flg	[S]	イベントフラグを待ちます(タイムアウトあり)
pol_flg	[S]	イベントフラグを得ます(ポーリング)
ipol_flg		イベントフラグを得ます(ポーリング, ハンドラ専用)
ref_flg		イベントフラグの状態を参照します
iref_flg		イベントフラグの状態を参照します(ハンドラ専用)

データキュー

サービスコール名		機能
cre_dtq		データキューを生成します
acre_dtq		データキューを生成します(自動 ID 割り当て)
del_dtq		データキューを削除します
snd_dtq	[S]	データキューにデータを送信します
psnd_dtq	[S]	データキューにデータを送信します(ポーリング)
ipsnd_dtq	[S]	データキューにデータを送信します(ポーリング, ハンドラ専用)
tsnd_dtq	[S]	データキューにデータを送信します(タイムアウトあり)
fsnd_dtq	[S]	データキューにデータを強制送信します
ifsnd_dtq	[S]	データキューにデータを強制送信します(ハンドラ専用)
rcv_dtq	[S]	データキューからデータを受信します
prcv_dtq	[S]	データキューからデータを受信します(ポーリング)
iprcv_dtq		データキューからデータを受信します(ポーリング, ハンドラ専用)
trcv_dtq	[S]	データキューからデータを受信します(タイムアウトあり)
ref_dtq		データキューの状態を参照します
iref_dtq		データキューの状態を参照します(ハンドラ専用)

メールボックス

サービスコール名	機能
cre_mbx	メールボックスを生成します
acre_mbx	メールボックスを生成します(自動 ID 割り当て)
del_mbx	メールボックスを削除します
snd_mbx [S]	メッセージを送信します
isnd_mbx	メッセージを送信します(ハンドラ専用)
rcv_ mbx [S]	メッセージを受信します
trcv_ mbx [S]	メッセージを受信します(タイムアウトあり)
prcv_ mbx [S]	メッセージを受信します(ポーリング)
iprcv_mbx	メッセージを受信します(ポーリング, ハンドラ専用)
ref_mbx	メールボックスの状態を参照します
iref_mbx	メールボックスの状態を参照します(ハンドラ専用)

メッセージバッファ

サービスコール名	機能
cre_mbf	メッセージバッファを生成します
acre_mbf	メッセージバッファを生成します(自動 ID 割り当て)
del_mbf	メッセージバッファを削除します
snd_mbf	メッセージバッファへ送信します
tsnd_mbf	メッセージバッファへ送信します(タイムアウトあり)
psnd_mbf	メッセージバッファへ送信します(ポーリング)
rcv_mbf	メッセージバッファから受信します
trcv_mbf	メッセージバッファから受信します(タイムアウトあり)
prcv_mbf	メッセージバッファから受信します(ポーリング)
ref_mbf	メッセージバッファの状態を参照します
ref_mbf	メッセージバッファの状態を参照します(ハンドラ専用)

ランデヴポート

サービスコール名	機能
cre_por	ランデヴ用ポートを生成します
acre_por	ランデヴ用ポートを生成します(自動 ID 割り当て)
del_por	ランデヴ用ポートを削除します
cal_por	ポートに対するランデヴ呼出を行います
tcal_por	ポートに対するランデヴ呼出を行います(タイムアウトあり)
acp_por	ポートに対するランデヴ受付を行います
tacp_por	ポートに対するランデヴ受付を行います(タイムアウトあり)
pacp_por	ポートに対するランデヴ受付を行います(ポーリング)
fwd_por	ランデヴを別のポートに回送します
rpl_rdv	ランデヴ返答を行います
ref_rdv	ランデヴ状態を参照します
iref_drv	ランデヴ状態を参照します(ハンドラ専用)
ref_por	ランデヴポート状態を参照します
iref_por	ランデヴポート状態を参照します(ハンドラ専用)

固定長メモリプール

サービスコール名		機能
cre_mpf		固定長メモリプールを生成します
acre_mpf		固定長メモリプールを生成します(自動 ID 割り当て)
del_mpf		固定長メモリプールを削除します
get_mpf	[S]	メモリブロックを獲得します
tget_mpf	[S]	メモリブロックを獲得します(タイムアウトあり)
pget_mpf	[S]	メモリブロックを獲得します(ポーリング)
iget_mpf		メモリブロックを獲得します(ポーリング, ハンドラ専用)
rel_mpf	[S]	メモリブロックを解放します
irel_mpf		メモリブロックを解放します(ハンドラ専用)
ref_mpf		固定長メモリプールの状態を参照します
ief_mpf		固定長メモリプールの状態を参照します(ハンドラ専用)

可変長メモリプール

サービスコール名	機能
cre_mpl	可変長メモリプールを生成します
acre_mpl	可変長メモリプールを生成します(自動 ID 割り当て)
del_mpl	可変長メモリプールを削除します
get_mpl	メモリブロックを獲得します
tget_mpl	メモリブロックを獲得します(タイムアウトあり)
pget_npl	メモリブロックを獲得します(ポーリング)
rel_mpl	メモリブロックを解放します
ref_mpl	可変長メモリプールの状態を参照します
iref_mpl	可変長メモリプールの状態を参照します(ハンドラ専用)

時間管理機能

サービスコール名		機能
set_tim	[S]	システム時刻を設定します
iset_tim		システム時刻を設定します(ハンドラ専用)
get_tim	[S]	システム時刻を参照します
iget_tim		システム時刻を参照します(ハンドラ専用)
isig_tim	[S]	タイムティックを供給します(ハンドラ専用)
		(本機能は、カーネルに組み込まれています)

周期ハンドラ

サービスコール名		機能
cre_cyc		周期ハンドラを生成します
acre_cyc		周期ハンドラを生成します(自動 ID 割り当て)
sta_cyc	[S]	周期ハンドラの動作を開始します
ista_cyc		周期ハンドラの動作を開始します(ハンドラ専用)
stp_cyc	[S]	周期ハンドラの動作を停止します
stp_cyc		周期ハンドラの動作を停止します(ハンドラ専用)
ref_cyc		周期ハンドラの状態を参照します
iref_cyc		周期ハンドラの状態を参照します(ハンドラ専用)

アラームハンドラ

サービスコール名	機能
cre_alm	アラームハンドラを生成します
acre_alm	アラームハンドラを生成します(自動 ID 割り当て)
sta_ alm	アラームハンドラの動作を開始します
ista_ alm	アラームハンドラの動作を開始します(ハンドラ専用)
stp_ alm	アラームハンドラの動作を停止します
stp_ alm	アラームハンドラの動作を停止します(ハンドラ専用)
ref_ alm	アラームハンドラの状態を参照します
iref_ alm	アラームハンドラの状態を参照します(ハンドラ専用)

システム状態管理機能

サービスコール名		機能
rot_rdq	[S]	タスク優先順位を回転させます
irot_rdq	[S]	タスク優先順位を回転させます(ハンドラ専用)
get_tid	[S]	タスク ID 番号を参照します
iget_tid	[S]	タスク ID 番号を参照します(ハンドラ専用)
loc_cpu	[S]	CPU ロック状態に移行させます
iloc_cpu	[S]	CPU ロック状態に移行させます(ハンドラ専用)
unl_cpu	[S]	CPU ロック状態を解除します
iunl_cpu	[S]	CPU ロック状態を解除します(ハンドラ専用)
dis_dsp	[S]	ディスパッチ禁止状態に移行させます
ena_dsp	[S]	ディスパッチ禁止状態を解除します
sns_ctx	[S]	コンテキスト状態を参照します
sns_loc	[S]	CPU ロック状態を参照します
sns_dsp	[S]	ディスパッチ禁止状態を参照します
sns_dpn	[S]	ディスパッチ保留状態を参照します

割り込み管理機能

サービスコール名	機能
def_int	割り込みハンドラを定義します
idef_int	割り込みハンドラを定義します(ハンドラ専用)

システム構成管理機能

サービスコール名	機能
ref_ver	バージョン情報を参照します
iref_ver	バージョン情報を参照します(ハンドラ専用)

拡張機能

サービスコール名	機能
vrst_dtq	データキューを初期化します
vrst_mbx	メールボックスを初期化します
vrst_mbf	メッセージバッファを初期化します
vrst_mpf	固定長メモリプールを初期化します
vrst_mpl	可変長メモリプールを初期化します

3.2. エラーコード一覧

エラーコード	値	説明
E_OK	0	正常終了
E_ILUSE	-28	サービスコール不正使用
E_OBJ	-41	オブジェクトの状態が不正
E_QOVR	-43	キューイングまたはネストのオーバーフロー
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_RLWAI	-49	待ち状態強制解除
E_NOEXS	-42	オブジェクトが存在していない
E_DLT	-51	待ちオブジェクトが削除された
E_NOMEM	-33	メモリ不足
E_NOID	-34	割り当て可能な ID 番号がない
EV_RST	-127	リセットのため待ちが解除された

3.3. アセンブリ言語インタフェース

アセンブリ言語でサービスコールを発行する場合、サービスコールの呼び出し用マクロを使用します。サービスコールの呼び出し用マクロ内の処理は、各パラメータをレジスタに設定してから、ソフトウエア割り込みによりサービスコールのルーチンの実行を開始します。また、サービスコールの呼び出し用マクロを使用せず直接サービスコールを呼び出した場合、将来のバージョンにおいて互換性が保証できなくなります。以下にアセンブリ言語インタフェースの一覧表を記載します。機能コードについては、μITRON 仕様で規定された値は使用しておりません。

サービスコール発行後、以下の表に示す戻り値以外のレジスタの値はサービスコール発行直前の値となります。なお、PSW の SM、IE、C は、サービスコール発行前の値が保存されてますが、BSM、BIC、BC は不定となります。

タスク管理機能

システムコール名	TRAP番号	引数			戻り値	
		R0 機能コード	R1	R2	R0	R2
acre_tsk	#7	H'00		pk_ctsk	tskid	
cre_tsk	#7	H'04	tskid	pk_ctsk	ercd	
del_tsk	#7	H'08	tskid		ercd	
act_tsk	#7	H'0C	tskid		ercd	
iact_tsk	#7	H'0C	tskid		ercd	
can_act	#7	H'10	tskid		actcnt	
ican_act	#7	H'10	tskid		actcnt	
sta_tsk	#7	H'14	tskid	stacd	ercd	
ista_tsk	#7	H'14	tskid	stacd	ercd	
ext_tsk	#7	H'18				
exd_tsk	#7	H'1c				
ter_tsk	#7	H'20	tskid		ercd	
chg_pri	#7	H'24	tskid	tskpri	ercd	
ichg_pri	#7	H'24	tskid	tskpri	ercd	
get_pri	#7	H'28	tskid		ercd	tskpri
iget_pri	#7	H'28	tskid		ercd	tskpri
ref_tsk	#7	H'2c	tskid	pk_rtsk	ercd	
iref_tsk	#7	H'2c	tskid	pk_rtsk	ercd	
ref_tst	#7	H'30	tskid	pk_rtst	ercd	
iref_tst	#7	H'30	tskid	pk_rtst	ercd	

タスク付属同期機能

システム コール名	TRAP 番号	引数					戻り値	
		R0 機能コード	R1	R2	R3	R4	R0	R2
slp_tsk	#7	H'34				-1	ercd	
tslp_tsk	#7	H'34				tmout	ercd	
wup_tsk	#7	H'38	tskid				ercd	
iwup_tsk	#7	H'38	tskid				ercd	
can_wup	#7	H'3c	tskid				wupcnt	
ican_wup	#7	H'3c	tskid				wupcnt	
rel_wai	#7	H'40	tskid				ercd	
irel_wai	#7	H'40	tskid				ercd	
sus_tsk	#7	H'44	tskid				ercd	
isus_tsk	#7	H'44	tskid				ercd	
rsm_tsk	#7	H'48	tskid				ercd	
irsm_tsk	#7	H'48	tskid				ercd	
frsm_tsk	#7	H'4c	tskid				ercd	
ifrsm_tsk	#7	H'4c	tskid				ercd	
dly_tsk	#7	H'50				dlytim	ercd	

タスク例外機能

システム コール名	TRAP 番号	引数					戻り値	
		R0 機能コード	R1	R2	R3	R4	R0	R2
def_tex	#7	H'54	tskid	pk_dtex			ercd	
ras_tex	#7	H'58	tskid	rasptn			ercd	
iras_tex	#7	H'58	tskid	rasptn			ercd	
dis_tex	#7	H'5c					ercd	
ena_tex	#7	H'60					ercd	
sns_tex	#7	H'64					state	
ref_tex	#7	H'68	tskid	pk_rtex			ercd	
iref_tex	#7	H'68	tskid	pk_rtex			ercd	

同期・通信機能(セマフォ)

システム コール名	TRAP 番号	引数					戻り値		
		R0 機能コード	R1	R2	R3	R4	R0	R2	R3
acre_sem	#7	H'6c		pk_csem			semid		
cre_sem	#7	H'70	semid	pk_csem			ercd		
del_sem	#7	H'74	semid				ercd		
sig_sem	#7	H'78	semid				ercd		
isig_sem	#7	H'78	semid				ercd		
wai_sem	#7	H'7c	semid			-1	ercd		
twai_sem	#7	H'7c	semid			tmout	ercd		
pol_sem	#7	H'7c	semid			0	ercd		
ipol_sem	#7	H'7c	semid			0	ercd		
ref_sem	#7	H'80	semid	pk_rsem			ercd		
iref_sem	#7	H'80	semid	pk_rsem			ercd		

同期・通信機能(イベントフラグ)

システム コール名	TRAP 番号	引数					戻り値		
		R0 機能コード	R1	R2	R3	R4	R0	R2	R3
acre_flg	#7	H'84		pk_cflg			flgid		
cre_flg	#7	H'88	flgid	pk_cflg			ercd		
del_flg	#7	H'8c	flgid				ercd		
set_flg	#7	H'90	flgid	setptn			ercd		
iset_flg	#7	H'90	flgid	setptn			ercd		
clr_flg	#7	H'94	flgid	clrptn			ercd		
iclr_flg	#7	H'94	flgid	clrptn			ercd		
wai_flg	#7	H'98	flgid	waiptn	wfmode	-1	ercd	flgptn	
twai_flg	#7	H'98	flgid	waiptn	wfmode	tmout	ercd	flgptn	
pol_flg	#7	H'98	flgid	waiptn	wfmode	0	ercd	flgptn	
ipol_flg	#7	H'98	flgid	waiptn	wfmode	0	ercd	flgptn	
ref_flg	#7	H'9c	flgid	pk_rflg			ercd		
iref_flg	#7	H'9c	flgid	pk_rflg			ercd		

同期・通信機能(データキュー)

システム コール名	TRAP 番号	引数					戻り値		
		R0 機能コード	R1	R2	R3	R4	R0	R2	R3
acre_dtq	#7	H'a0		pk_cdtq			dtqid		
cre_dtq	#7	H'a4	dtqid	pk_cdtq			ercd		
del_dtq	#7	H'a8	dtqid				ercd		
snd_dtq	#7	H'ac	dtqid	data		-1	ercd		
tsnd_dtq	#7	H'ac	dtqid	data		tmout	ercd		
psnd_dtq	#7	H'ac	dtqid	data		0	ercd		
ipsnd_dtq	#7	H'ac	dtqid	data		0	ercd		
fsnd_dtq	#7	H'b0	dtqid	data			ercd		
ifsnd_dtq	#7	H'b0	dtqid	data			ercd		
rcv_dtq	#7	H'b4	dtqid			-1	ercd	data	
trcv_dtq	#7	H'b4	dtqid			tmout	ercd	data	
prcv_dtq	#7	H'b4	dtqid			0	ercd	data	
iprcv_dtq	#7	H'b4	dtqid			0	ercd	data	
ref_dtq	#7	H'b8	dtqid	pk_rdtq			ercd		
iref_dtq	#7	H'b8	dtqid	pk_rdtq			ercd		

同期・通信機能(メールボックス)

システム コール名	TRAP 番号	引数					戻り値		
		R0 機能コード	R1	R2	R3	R4	R0	R2	R3
acre_mbx	#7	H'bc		pk_cmbx			mbxid		
cre_mbx	#7	H'c0	mbxid	pk_cmbx			ercd		
del_mbx	#7	H'c4	mbxid				ercd		
snd_mbx	#7	H'c8	mbxid	pk_msg			ercd		
isnd_mbx	#7	H'c8	mbxid	pk_msg			ercd		
rcv_mbx	#7	H'cc	mbxid			-1	ercd	pk_msg	
trcv_mbx	#7	H'cc	mbxid			tmout	ercd	pk_msg	
prcv_mbx	#7	H'cc	mbxid			0	ercd	pk_msg	
iprcv_mbx	#7	H'cc	mbxid			0	ercd	pk_msg	
ref_mbx	#7	H'd0	mbxid	pk_rmbx			ercd		
iref_mbx	#7	H'd0	mbxid	pk_rmbx			ercd		

拡張同期・通信機能(メッセージバッファ)

システム コール名	TRAP 番号	引数					戻り値
		R0 機能コード	R1	R2	R3	R4	R0
acre_mbf	#7	H'd4		pk_cmbf			mbfid
cre_mbf	#7	H'd8	mbfid	pk_cmbf			ercd
del_mbf	#7	H'dc	mbfid				ercd
snd_mbf	#7	H'e0	mbfid	msg	msgsz	-1	ercd
tsnd_mbf	#7	H'e0	mbfid	msg	msgsz	tmout	ercd
psnd_mbf	#7	H'e0	mbfid	msg	msgsz	0	ercd
rcv_mbf	#7	H'e4	mbfid	msg		-1	msgsz
trcv_mbf	#7	H'e4	mbfid	msg		tmout	msgsz
prcv_mbf	#7	H'e4	mbfid	msg		0	msgsz
ref_mbf	#7	H'e8	mbfid	pk_rmbf			ercd
iref_mbf	#7	H'e8	mbfid	pk_rmbf			ercd

拡張同期・通信機能(ランデヴ)

システムコール名	TRAP番号	引数						戻り値		
		R0 機能コード	R1	R2	R3	R4	R5	R0	R2	R3
acre_por	#7	H'ec		pk_cpor				porid		
cre_por	#7	H'f0	porid	pk_cpor				ercd		
del_por	#7	H'f4	porid					ercd		
cal_por	#7	H'f8	porid	msg	cmsgsz	-1	calptn	rmsgsz		
tcal_por	#7	H'f8	porid	msg	cmsgsz	tmout	calptn	rmsgsz		
acp_por	#7	H'fc	porid	msg		-1	acpptn	cmsgsz		rdvno
tacp_por	#7	H'fc	porid	msg		tmout	acpptn	cmsgsz		rdvno
pacp_por	#7	H'fc	porid	msg		0	acpptn	cmsgsz		rdvno
fwd_por	#7	H'100	porid	rdvno	cmsgsz	msg	calptn	ercd		
rpl_rdv	#7	H'104	rdvno	msg	rmsgsz			ercd		
ref_por	#7	H'108	porid	pk_rpor				ercd		
iref_por	#7	H'108	porid	pk_rpor				ercd		
ref_rdv	#7	H'10c	porid	pk_rrdv				ercd		
iref_rdv	#7	H'10c	porid	pk_rrdv				ercd		

メモリプール管理機能(固定長メモリプール)

システムコール名	TRAP番号	引数					戻り値	
		R0 機能コード	R1	R2	R3	R4	R0	R2
acre_mpf	#7	H'110		pk_cmpf			mpfid	
cre_mpf	#7	H'114	mpfid	pk_cmpf			ercd	
del_mpf	#7	H'118	mpfid				ercd	
get_mpf	#7	H'11c	mpfid			-1	ercd	blk
tget_mpf	#7	H'11c	mpfid			tmout	ercd	blk
pget_mpf	#7	H'11c	mpfid			0	ercd	blk
ipget_mpf	#7	H'11c	mpfid			0	ercd	blk
rel_mpf	#7	H'120	mpfid	mpf			ercd	
irel_mpf	#7	H'120	mpfid	mpf			ercd	
ref_mpf	#7	H'124	mpfid	pk_rmpf			ercd	
iref_mpf	#7	H'124	mpfid	pk_rmpf			ercd	

メモリプール管理機能(可変長メモリプール)

システム コール名	TRAP 番号	引数					戻り値	
		R0 機能コード	R1	R2	R3	R4	R0	R2
acre_mpl	#7	H'128		pk_cmpl			mplid	
cre_mpl	#7	H'12c	mplid	pk_cmpl			ercd	
del_mpl	#7	H'130	mplid				ercd	
get_mpl	#7	H'134	mplid	blksz		-1	ercd	blk
tget_mpl	#7	H'134	mplid	blksz		tmout	ercd	blk
pget_mpl	#7	H'134	mplid	blksz		0	ercd	blk
rel_mpl	#7	H'138	mplid	blk			ercd	
ref_mpl	#7	H'13c	mplid	pk_rmpl			ercd	
iref_mpl	#7	H'13c	mplid	pk_rmpl			ercd	

時間管理機能

システム コール名	TRAP 番号	引数			戻り値
		R0 機能コード	R1	R2	R0
set_tim	#7	H'140		pk_tim	ercd
iset_tim	#7	H'140		pk_tim	ercd
get_tim	#7	H'144		pk_tim	ercd
iget_tim	#7	H'144		pk_tim	ercd

周期ハンドラ

システム コール名	TRAP 番号	引数					戻り値	
		R0 機能コード	R1	R2	R3	R4	R0	R2
acre_cyc	#7	H'148		pk_ccyc			cycid	
cre_cyc	#7	H'14c	cycid	pk_ccyc			ercd	
del_cyc	#7	H'150	cycid				ercd	
sta_cyc	#7	H'154	cycid				ercd	
ista_cyc	#7	H'154	cycid				ercd	
stp_cyc	#7	H'158	cycid				ercd	
istp_cyc	#7	H'158	cycid				ercd	
ref_cyc	#7	H'15c	cycid	pk_rcyc			ercd	
iref_cyc	#7	H'15c	cycid	pk_rcyc			ercd	

アラームハンドラ

システム コール名	TRAP 番号	引数				戻り値
		R0 機能コード	R1	R2	R5	R0
acre_alm	#7	H'160		pk_calm		almid
cre_alm	#7	H'164	almid	pk_calm		ercd
del_alm	#7	H'168	almid			ercd
sta_alm	#7	H'16c	almid		almtim	ercd
ista_alm	#7	H'16c	almid		almtim	ercd
stp_alm	#7	H'170	almid			ercd
istp_alm	#7	H'170	almid			ercd
ref_alm	#7	H'174	almid	pk_ralm		ercd
iref_alm	#7	H'174	almid	pk_ralm		ercd

システム状態管理機能

システム コール名	TRAP 番号	引数			戻り値	
		R0 機能コード	R1	R2	R0	R2
rot_rdq	#7	H'178		tskpri	ercd	
irot_rdq	#7	H'178		tskpri	ercd	
get_tid	#7	H'17c		p_tskid	ercd	
iget_tid	#7	H'17c		p_tskid	ercd	
loc_cpu	#7	H'180			ercd	
iloc_cpu	#7	H'180			ercd	
unl_cpu	#7	H'184			ercd	
iunl_cpu	#7	H'184			ercd	
dis_dsp	#7	H'188			ercd	
ena_dsp	#7	H'18c			ercd	
sns_ctx	#7	H'190			state	
sns_loc	#7	H'194			state	
sns_dsp	#7	H'198			state	
sns_dpn	#7	H'19c			state	

割り込み管理機能

システム コール名	TRAP 番号	引数			戻り値	
		R0 機能コード	R1	R2	R0	R2
def_inh	#7	H'1a0	inhno	pk_dinh	ercd	
idef_inh	#7	H'1a0	inhno	pk_dinh	ercd	

システム構成管理機能

システム コール名	TRAP 番号	引数			戻り値	
		R0 機能コード	R1	R2	R0	R2
ref_ver	#7	H'1a4		pk_rver	ercd	
iref_ver	#7	H'1a4		pk_rver	ercd	

拡張機能

システム コール名	TRAP 番号	引数			戻り値
		R0 機能コード	R1	R2	R0
vrst_dtq	#7	H'1a8	dtqid		ercd
vrst_mbx	#7	H'1ac	mbxid		ercd
vrst_mbf	#7	H'1b0	mbfid		ercd
vrst_mpf	#7	H'1b4	mpfid		ercd
vrst_mpl	#7	H'1b8	mplid		ercd

3.4. データタイプ

```
typedef    signed char      B;          /* 符号付き 8 ビット整数 */
typedef    signed short     H;          /* 符号付き 16 ビット整数 */
typedef    signed long      W;          /* 符号付き 32 ビット整数 */
typedef    unsigned char    UB;         /* 符号なし 8 ビット整数 */
typedef    unsigned short   UH;         /* 符号なし 16 ビット整数 */
typedef    unsigned long    UW;         /* 符号なし 32 ビット整数 */
typedef    char             VB          /* データタイプが一致しないもの符号付き
                                           (8 ビットサイズ) */
typedef    short            VH;         /* データタイプが一致しないもの符号付き
                                           (16 ビットサイズ) */
typedef    long             VW;         /* データタイプが一致しないもの符号付き
                                           (32 ビットサイズ) */
typedef    void             *VP;        /* データタイプが一致しないものへのポインタ */
typedef    void             (*FP)();    /* プログラムのスタートアドレス一般 */
typedef    W                INT         /* 符号付き 32 ビット整数 */
typedef    UW               UINT;       /* 符号なし 32 ビット整数 */
typedef    W                FN          /* 機能コード */
typedef    H                ID;         /* オブジェクト ID 番号 */
typedef    H                PRI;        /* タスク優先度 */
typedef    W                TMO;        /* タイムアウト */
typedef    W                ER;         /* エラーコード(符号付き整数) */
typedef    UW               ATR;        /* オブジェクト属性(符号なし整数) */
typedef    UW               STAT;       /* タスクの状態 */
typedef    UW               MODE;       /* サービスコールの動作モード */
typedef    UW               SIZE;       /* メモリ領域のサイズ */
typedef    UW               RELTIM      /* 相対時間 */
typedef    W                VP_INT;     /* データタイプが定まらないものへのポインタまた
                                           はプロセッサに自然なサイズの符号付き整数 */
typedef    struct           systim{     /* システム時刻 */
           UH               utime;      /* 時刻の上位 16bit */
           UW               ltimer;     /* 時刻の下位 32bit */
} SYSTIM;
typedef    W                ER_ID;      /* エラーコードまたは ID */
typedef    W                ER_UINT;    /* エラーコードまたは符号無し整数 */
```

3.5. 共通定数と構造体のパケット形式

```
----共通----
TRUE          1               /* 真 */
FALSE         0               /* 偽 */
----タスク管理関係----
typedef struct t_ctsk {
          ATR       tskatr;   /* タスク属性 */
          VP_INT    exinf;    /* 拡張情報 */
          FP        task;     /* タスク起動アドレス */
          PRI       itskpri;  /* タスク起動時優先度 */
          SIZE      stksz;    /* スタックサイズ */
          VP        stk;      /* タスクのスタック領域の先頭アドレス */
} T_CTSK;
TSK_SELF     0              /* 自タスク指定 */
TPRI_RUN     0              /* その時実行中の優先度を指定 */
typedef struct t_rtsk {
          STAT      tskstat;  /* タスクの状態 */
          PRI       tskpri;   /* タスクの現在優先度 */
          PRI       tskbpri;  /* タスクのベース優先度 */
          STAT      tskwait;  /* タスクの待ち要因 */
          ID        wid;      /* タスクの待ちオブジェクト ID */
          TMO       tskatr;   /* タイムアウトするまでの時間 */
          UINT      actcnt;   /* 起動要求キューイング数 */
          UINT      wupcnt;   /* 起床要求キューイング数 */
          UINT      suscnt;   /* 強制待ち要求ネスト数 */
} T_RTSK;
typedef struct t_rtst {
          STAT      tskstat;          /* タスクの状態 */
          STAT      tskwait;          /* タスクの待ち要因 */
} T_RTST;
----タスク例外関係----
typedef struct t_dtex {
          ATR       texatr;           /* タスク例外処理ルーチンの属性 */
          FP        texrtn;           /* タスク例外処理ルーチンの開始アドレス */
} T_DTEX;
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;

----セマフォ関係----
typedef struct t_csem {
          ATR       sematr;   /* セマフォ属性 */
          UINT      isemcnt;  /* セマフォの初期値 */
          UINT      maxsem;   /* セマフォの最大値 */
} T_CSEM;
typedef struct t_rsem {
          ID        wtskid;   /* 待ち行列先頭タスクの ID 番号 */
          INT       semcnt;   /* 現在のセマフォカウントの値 */
} T_RSEM;
```

```
----イベントフラグ関係----
typedef struct t_cflg {
        ATR     flgatr;  /* イベントフラグ属性 */
        FLGPTN  iflgptn;  /* イベントフラグの初期値 */
} T_CFLG;
wfmod:
        TWF_ANDW  H'0000    /* AND 待ち */
        TWF_ORW   H'0002    /* OR 待ち */
typedef struct t_rflg {
        ID      wtskid;  /* 待ち行列先頭タスクの ID 番号 */
        UINT    flgptn;  /* イベントフラグの現在のビットパターン */
} T_RFLG;
----データキュー関係----
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;
typedef struct t_rdtq {
        ID      stskid;          /* 送信待ち行列先頭タスクの ID 番号 */
        ID      rtskid;          /* 受信待ち行列先頭タスクの ID 番号 */
        UINT    sdtqcnt;         /* データキューに入っているデータの個数 */
} T_RDTQ;
----メールボックス関係----
typedef struct t_cmbx {
        ATR     mbxatr;          /* メールボックス属性 */
        PRI     maxpri;          /* メッセージの最大優先度 */
        VP      mprihd;          /* メッセージ優先度のキューヘッダ先頭番地 */
} T_CMBX;
typedef struct t_mbx {
        ID      wtskid;          /* 待ち行列先頭タスクの ID 番号*/
        T_MSG   *pk_msg;         /* 次に受信されるメッセージ */
} T_RMBX;
----メッセージバッファ関係----
typedef struct t_cmbf {
        ATR     mbfatr;          /* メッセージバッファ属性 */
        UINT    maxmsz;          /* メッセージの最大長 */
        SIZE    bufsz;           /* メッセージバッファのサイズ */
        VP      mbf;             /* メッセージバッファ領域の先頭アドレス */
} T_CMBF;
typedef struct t_rmbf {
        ID      stskid;  /* 送信待ち行列先頭タスクの ID 番号*/
        ID      rtskid;  /* 受信待ち行列先頭タスクの ID 番号*/
        UINT    smsgcnt; /* バッファに入っているメッセージ数 */
        SIZE    frbufsz; /* 空きバッファのサイズ */
} T_RMBF;
----ランデヴポート関係----
typedef struct t_cpor {
        ATR     poratr;          /* ポート属性 */
        UINT    maxcmsz;         /* 呼出時のメッセージの最大サイズ */
        UINT    maxrmsz;         /* 返答時のメッセージの最大サイズ */
} T_CPOR;
```

```
typedef struct t_rpor {
        ID      ctskid;   /* 呼出待ち行列先頭タスクの ID 番号*/
        ID      atskid;   /* 受付待ち行列先頭タスクの ID 番号*/
} T_RPOR;
----固定長メモリプール関係----
typedef struct t_cmpf {
        ATR     mpfatr;   /* メモリプール属性 */
        UINT    mpfcnt;   /* メモリブロック数 */
        UINT    blfsz;    /* メモリブロックサイズ */
        VP      mpf;      /* 固定長メモリプール領域の先頭アドレス */
} T_CMPF;
typedef struct t_rmpf {
        ID      wtskid;   /* メモリ獲得待ち行列先頭タスクの ID 番号*/
        UINT    frbcnt;   /* メモリブロック数 */
} T_RMPF;
----可変長メモリプール関係----
typedef struct t_cmpl {
        ATR     mplatr;   /* メモリプール属性 */
        SIZE    mplsz;    /* メモリプールサイズ */
        VP      mpl;      /* 可変長メモリプール領域の先頭アドレス */
        SIZE    maxblksz  /* メモリブロック最大サイズ */
} T_CMPL;
typedef struct t_rmpl {
        ID      wtskid;   /* メモリ獲得待ち行列先頭タスクの ID 番号*/
        SIZE    fmplsz;   /* 空き領域の合計サイズ */
        UINT    fblksz;   /* すぐに獲得可能な最大メモリブロックサイズ */
} T_RMPL;
----周期ハンドラ関係----
typedef struct t_ccyc {
        ATR     cycatr;   /* 周期ハンドラ属性 */
        VP_INT  exinf;    /* 拡張情報 */
        FP      cychdr;   /* 周期ハンドラ先頭アドレス */
        RELTIM  cyctim;   /* 起動周期 */
        RELTIM  cycphs;   /* 起動位相 */
} T_CCYC;
typedef struct   t_rcyc {
        STAT    cycstat;  /* 周期ハンドラ動作状態 */
        RELTIM  lefttim;  /* 周期ハンドラの起動までの残り時間 */
} T_RCYC;
----アラームハンドラ関係----
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;   /* アラームハンドラ先頭アドレス */
} T_CALM;
typedef struct t_ralm {
        STAT    almstat; /* アラームハンドラ動作状態 */
        RELTIM  lefttim;  /* アラームハンドラの起動までの残り時間 */
} T_RALM;
```

```
----割り込み関係----
typedef struct t_dinh {
        ATR     inhatr;   /* 割り込みハンドラ属性 */
        FP      inthdr;   /* 割り込みアドレス */
} T_DINTH;
/* システム管理関係 */
typedef struct t_rver {
        UH      maker;    /* メーカー */
        UH      prid;     /* 形式番号 */
        UH      spver;    /* 仕様書バージョン */
        UH      prver;    /* 製品バージョン */
        UH      prno[4];  /* 製品管理情報 */
} T_RVER;
```

Everywhere you imagine. RENESAS

M32R ファミリ用リアルタイム OS
リファレンスマニュアル
M3T-MR32R/4

発行年月日　2004 年 12 月 01 日 Rev.1.00

発行　　株式会社 ルネサス テクノロジ　営業企画統括部
〒100-0004　東京都千代田区大手町 2-6-2

編集　　株式会社 ルネサス ソリューションズ　第一応用技術部

© 2004. Renesas Technology Corp. and Renesas Solutions Corp., All rights reserved. Printed in Japan.

ルネサス テクノロジ
www.renesas.com

M3T-MR32R/4 V.4.00
リファレンスマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J0761-0100Z