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。

RENESAS

ユーザーズ・マニュアル

RX850 Pro

リアルタイム・オペレーティング・システム

基礎編

対象デバイス

V850 シリーズ ™

対象リアルタイム OS

RX850 Pro Ver.3.15

資料番号 U13773JJ3V0UM00（第 3 版）
発行年月 December 2002 N CP(K)

［メ　モ］

保守／廃止

目 次 要 約

- 本資料に記載されている内容は2002年11月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品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，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，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，ソフトウエアおよびこれらに関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，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「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。

  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

  特別水準：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

  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，生命維持のための装置またはシステム等

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。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNEC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う。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E　02.11

保守／廃止

本版で改訂された主な箇所

箇　所	内　容
全般	V850 ファミリを V850 シリーズに変更
p.21	1.7　実行環境の記述を変更
p.21	1.8　開発環境の記述を変更
p.22	1.9　システム構築手順のニュークリアス・ライブラリに/librxpm.a を追加
p.67	6.3.4　メモリブロックの返却の注意 2 の記述を変更
p.78	7.6.5　周期起動ハンドラ中の割り込みの記述を変更
p.181	11.8.5　メモリ・プール管理機能システム・コールの rel_blk の注意の記述を変更
p.196	11.8.6　時間管理機能システム・コールの set_tim と get_tim 中のシステム・クロック SYSTIME の構造の記述を変更
p.237	付録 B　Q&A を追加

本文欄外の★印は，本版で改訂された主な箇所を示しています。

保守／廃止

はじめに

対象者　このマニュアルは，V850 シリーズの応用システムを設計，開発するユーザを対象としています。

目　的　このマニュアルは，次の構　成に示す RX850 Pro の機能をユーザに理解していただくことを目的としています。

構　成　このマニュアルは，大きく分けて次の内容で構成しています。

- 概　説
- ニュークリアス
- タスク管理機能
- 同期通信機能
- 割り込み管理機能
- メモリ・プール管理機能
- 時間管理機能
- スケジューラ
- システム初期化処理
- インタフェース・ライブラリ
- システム・コール

読み方　このマニュアルの読者には，電気，論理回路，マイクロコンピュータ，C 言語，アセンブリ言語に関する一般知識が必要です。

V850 シリーズのハードウエア機能，命令機能を知りたいとき
　→各製品のユーザーズ・マニュアルを参照してください。

凡　例　注　　：本文中につけた注の説明
注意　　：気をつけて読んでいただきたい内容
備考　　：本文の補足説明
数の表記　：2 進数 … XXXX または B’XXXX
10 進数 … XXXX
16 進数 … 0xXXXX または H’XXXX
2 のべき数を示す接頭語（アドレス空間，メモリ容量）：
K（キロ）　2^{10}＝1024
M（メガ）　2^{20}＝1024^2

保守／廃止

関連資料　**このマニュアルを使用する場合は，次の資料もあわせてご覧ください。**
関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。
あらかじめご了承ください。

開発ツールに関する資料（ユーザーズ・マニュアル）　**（1/2）**

資　料　名		資料番号	
		和文	英文
IE-703002-MC（V851™, V852™, V853™, V854™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SV1™, V850/SF1™, V850/SC1™, V850/SC2™, V850/SC3™用インサーキット・エミュレータ）		U11595J	U11595E
IE-703003-MC-EM1（V853用インサーキット・エミュレータ・オプション・ボード）		U11596J	U11596E
IE-703008-MC-EM1（V854用インサーキット・エミュレータ・オプション・ボード）		U12420J	U12420E
IE-703017-MC-EM1（V850/SA1用インサーキット・エミュレータ・オプション・ボード）		U12898J	U12898E
IE-703037-MC-EM1（V850/SB1, V850/SB2用インサーキット・エミュレータ・オプション・ボード）		U14151J	U14151E
IE-703040-MC-EM1（V850/SV1用インサーキット・エミュレータ・オプション・ボード）		U14337J	U14337E
IE-703079-MC-EM1（V850/SF1用インサーキット・エミュレータ・オプション・ボード）		U15447J	U15447E
IE-703089-MC-EM1（V850/SC1, V850/SC2, V850/SC3用インサーキット・エミュレータ・オプション・ボード）		U15776J	U15776E
IE-703102-MC（V850E/MS1™, V850E/MS2™用インサーキット・エミュレータ）		U13875J	U13875E
IE-703102-MC-EM1（V850E/MS1, V850E/MS2用インサーキット・エミュレータ・オプション・ボード）， IE-703102-MC-EM1-A（V850E/MS1用インサーキット・エミュレータ・オプション・ボード）		U13876J	U13876E
IE-V850E-MC（V850E/IA1™, V850E/IA2™用インサーキット・エミュレータ）， IE-V850E-MC-A（V850E1（NB85Eコア），V850E/MA1™, V850E/MA2™用インサーキット・エミュレータ）		U14487J	U14487E
IE-V850E-MC-EM1-A（V850E1（NB85Eコア）用インサーキット・エミュレータ・オプション・ボード）		作成予定	作成予定
IE-V850E-MC-EM1-B, IE-V850E-MC-MM2（V850E1（NB85Eコア）用インサーキット・エミュレータ・オプション・ボード）		U14482J	U14482E
IE-703107-MC-EM1（V850E/MA1, V850E/MA2用インサーキット・エミュレータ・オプション・ボード）		U14481J	U14481E
IE-703116-MC-EM1（V850E/IA1用インサーキット・エミュレータ・オプション・ボード）		U14700J	U14700E
IE-703114-MC-EM1（V850E/IA2用インサーキット・エミュレータ・オプション・ボード）		作成予定	作成予定
CA850 Ver.2.50　Cコンパイラ・パッケージ	操作編	U16053J	U16053E
	C言語編	U16054J	U16054E
	PM plus編	U16055J	作成予定
	アセンブリ言語編	U16042J	U16042E
ID850 Ver.2.40　統合ディバッガ	操作編　Windows®ベース	U15181J	U15181E
SM850 Ver.2.40　システム・シミュレータ	操作編　Windowsベース	U15182J	U15182E
SM850 Ver.2.00以上　システム・シミュレータ	外部部品ユーザ・オープン・インタフェース仕様編	U14873J	U14873E

資　料　名		資料番号	
		和文	英文
RX850 Ver.3.13以上　リアルタイムOS	基礎編	U13430J	U13430E
	インストレーション編	U13410J	U13410E
	テクニカル編	U13431J	U13431E
RX850 Pro Ver.3.15　リアルタイムOS	基礎編	このマニュアル	U13773E
	インストレーション編	U13774J	U13774E
	テクニカル編	U13772J	U13772E
RX-NET　ネットワーク・ライブラリ（TCP/IP）		U15083J	－
RX-NET　ネットワーク・ライブラリ（PPP）		U15303J	－
RX-NET　ネットワーク・ライブラリ（DNS）		U15304J	－
RX-NET　ネットワーク・ライブラリ（DHCP）		U15382J	－
RX-NET　ネットワーク・ライブラリ（SMTP）		U15505J	－
RX-NET　ネットワーク・ライブラリ（POP）		U15539J	－
RX-NET　ネットワーク・ライブラリ（FTP）		U15946J	－
RX-NET　ネットワーク・ライブラリ（WebServer）		U16294	－
RD850　Ver.3.01　タスク・ディバッガ		U13737J	U13737E
RD850 Pro Ver.3.01　タスク・ディバッガ		U13916J	U13916E
AZ850 Ver.3.10　システム・パフォーマンス・アナライザ		U14410J	U14410E
PG-FP4　フラッシュ・メモリ・プログラマ		U15260J	U15260E

保守／廃止

目　　次

保守／廃止

保守／廃止

図の目次（1/2）

保守／廃止

図の目次（2/2）

保守／廃止

表の目次

第1章　概　　説

マイクロプロセッサは半導体技術の進歩に従って急速に普及し，今日ではあらゆる分野で利用されるようになってきました。しかし，マイクロプロセッサを取り巻く処理プログラム量は増大し，各種ハードウエアにあわせた固有のプログラムをそのつど作成することが困難となりました。

そこで，高性能，多機能化へと進むマイクロプロセッサの能力を完全に引き出すために，オペレーティング・システム（Operating System：OS）の重要性が高まってきました。

厳密な分け方ではありませんが，OS にはプログラム開発用と制御用の 2 種類があります。プログラム開発用の OS は，開発に使用するハードウエア構成をある程度固定（パーソナル・コンピュータなど）することができるため，標準的な OS（MS-DOS™，Windows，UNIX™ OS など）が流通しやすい環境にあります。

これに対し，制御用の OS は，制御機器に組み込まれて使用します。つまり，おのおののシステムによってハードウエア構成が異なり，しかも，用途に応じた効率の良い動作が要求されるため，標準的な OS が流通しにくい環境にあります。

NEC エレクトロニクスでは，V850 シリーズを開発，発売し，より強力なマイクロプロセッサを提供する一方で，このような市場状況を考慮し，高機能なマイクロプロセッサが持つ機能を十分に引き出すため，また，将来にわたっての体系的なソフトウエアの構築を支援するために，RX850 Pro を開発，発売しました。

RX850 Pro は高性能，高機能なマイクロプロセッサの応用範囲を拡大し，いっそうの汎用性を持たせるために開発されたリアルタイム，マルチタスク処理を実現する制御用 OS です。

1.1　概　　要

RX850 Pro は，効率の良いリアルタイム，マルチタスク処理環境を提供するとともに，対象プロセッサの制御機器分野における応用範囲を拡大することを目的として開発された，組み込み型制御用リアルタイム・マルチタスク OS です。

また，ターゲット・システムに組み込んで使用することを前提として開発されているため，ROM 化を意識し，高速かつコンパクトな OS となっています。

1.2　リアルタイム OS

制御機器分野におけるシステムでは，内外の事象変化に対する即応性が要求されます。しかし，従来のシステムでは，このような要求を単純な割り込み処理で対処してきたため，制御機器が高性能化，多機能化するにつれ，単純な割り込み処理だけでの対処が難しくなってきています。

つまり，システムの複雑化，処理プログラム量の増大により，内外の事象変化に対する処理を，どのような順序で実行させるかを管理することが困難になってきたということです。

そこで，このような問題に対処するために考えられたのがリアルタイム OS です。

リアルタイム OS は，内外の事象変化に対して即応し，最適な処理プログラムを，最適な順序で実行させることを主な目的としています。

保守／廃止

1.3 マルチタスク OS

OS の管理下で実行する処理プログラムの最小単位を，「タスク」と呼びます。また，1 つのプロセッサ上で複数のタスクを時間的に同時実行させることを，「マルチタスキング」と呼びます。

実際には，プロセッサ自体は，1 度に 1 つの処理プログラム（命令）しか実行することができません。しかし，複数のタスクの実行を何らかの基準（きっかけ）を利用して切り替えることにより，あたかも複数のタスクが同時実行しているかのようになります。

このように，システム内で定めた何らかの基準を利用してタスクを切り替え，タスクの並列処理を可能にした OS がマルチタスク OS です。

マルチタスク OS は，タスクを並列に実行させることにより，システム全体の処理能力を向上させることを主な目的としています。

1.4 特　徴

次に，RX850 Pro の特徴を示します。

(1) μITRON3.0仕様に準拠

RX850 Pro は，組み込み型制御用 OS のアーキテクチャとして代表的なμITRON3.0 仕様に準拠した設計が行われており，レベル E までの機能を実装しています。

なお，μITRON3.0 仕様とは，組み込み型制御用リアルタイム・システムのオペレーティング・システム仕様です。

(2) 高い汎用性

RX850 Pro は，μITRON3.0 仕様で規定されているシステム・コール（67 種類）のほかに，RX850 Pro オリジナルのシステム・コール（7 種類）も提供し，アプリケーション・システムの汎用性を高めています。

なお，RX850 Pro では，アプリケーション・システムが使用する機能（システム・コール）のみをシステム構築時に選択することができるため，コンパクトでありながら，ユーザのニーズに最適なリアルタイム・マルチタスク OS を構築できます。

(3) リアルタイム処理，マルチタスク処理の実現

完全なリアルタイム処理，マルチタスク処理を実現するために豊富な機能を提供しています。

- ●タスク管理機能
- ●タスク付属同期機能
- ●同期通信管理機能
- ●割り込み処理管理機能
- ●メモリ・プール管理機能
- ●時間管理機能
- ●システム管理機能
- ●スケジューリング機能

(4) スケジューリングのロック機能

ディスパッチ処理（タスクのスケジューリング処理）を禁止／再開する機能を提供しています。

これによりユーザは，処理プログラム・レベルからのディスパッチ処理の禁止／再開が可能となります。

(5) ROM化の実現

ターゲット・システムに組み込んで使用することを想定したリアルタイム・マルチタスク OS であるため，ROM 化を意識し，コンパクトな設計が行われています。

(6) オリジナル命令の活用

V850 シリーズ マイクロプロセッサの高速な命令実行速度と，オリジナル命令の活用により，高速処理を実現しています。

(7) 内蔵ROM/RAMの活用

V850 シリーズの内蔵 ROM/RAM の活用により，高速な命令実行，高速なデータ・アクセスを実現しています。

(8) アプリケーション・ユーティリティの提供

アプリケーション・システムを構築するうえで有益な２つのユーティリティを提供しています。

- ●コンフィギュレータ　CF850 Pro
- ●高級言語インタフェース・ライブラリ

(9) Cコンパイラ

次に示す V850 シリーズ用 C コンパイラに対応しています。

- ●CA850　　NEC エレクトロニクス製
- ●CCV850　Green Hills Software™, Inc.製

1.5 構　　成

RX850 Pro は，ニュークリアス，システム初期化処理，インタフェース・ライブラリ，システム・コンフィギュレータといった，4 つのサブシステムから構成されています。

次に，これらの概要を示します。

(1) ニュークリアス

ニュークリアスとは，リアルタイム，マルチタスク制御を行う RX850 Pro の中心（核）となる部分であり，次に示す機能を提供しています。

- ●管理オブジェクトの生成，初期化処理
- ●処理プログラム（タスク，非タスク）から発行されたシステム・コールに対応した処理
- ●ターゲット・システムの内外で発生した事象に対し，次に実行すべき処理プログラム（タスク，非タスク）の選択処理

なお，管理オブジェクトの生成，初期化処理とシステム・コールに対応した処理は各管理モジュールで，処理プログラムの選択はスケジューラで行われます。

保守／廃止

(2) システム初期化処理

システム初期化処理は，RX850 Pro が動作するうえで必要となるハードウエアの初期化処理とソフトウエアの初期化処理から構成されます。

したがって RX850 Pro では，システムが起動した際，まず最初に行われる処理がシステム初期化処理となります。

なお，システム初期化処理のうち，実行環境のハードウエア構成に依存する部分（ブート処理，ハードウエア初期化部）とソフトウエア環境を快適なものとする部分（ソフトウエア初期化部）については，サンプル・ソース・ファイルを提供しています。

これにより，さまざまなターゲット・システムへの移植性を向上させるとともに，カスタマイズ化を容易なものとしています。

(3) インタフェース・ライブラリ

処理プログラム（タスク，非タスク）を C 言語で記述した場合，システム・コールの発行と拡張 SVC ハンドラの呼び出しは外部関数形式となります。しかし，ニュークリアスが理解できる発行形式（ニュークリアス発行形式）と外部関数形式には相違があります。

そこで，外部関数形式で発行されたシステム・コールや外部関数形式で呼び出された拡張 SVC ハンドラをニュークリアス発行形式に変換し，処理プログラムとニュークリアスの仲介役を行うのがインタフェース・ライブラリです。

なお，RX850 Pro が提供するインタフェース・ライブラリは，NEC エレクトロニクス製 V850 シリーズ用 C コンパイラ CA850 用と Green Hills Software 社製 C クロス V800 コンパイラ CCV850 用の 2 種類が用意されています。

(4) コンフィギュレータ　CF850 Pro

RX850 Pro を使ったシステムを構築する場合，RX850 Pro に提供する各種データを保持した情報ファイル（システム情報テーブル，ブランチ・テーブル，システム情報ヘッダ・ファイル）が必要となります。

基本的に，これら情報ファイルは，規定された形式のデータ羅列であるため，各種エディタを用いて記述することは可能ですが，記述性，可読性の面で劣ったものとなります。

そこで RX850Pro では，記述性，可読性に優れた独自の記述形式で作成されたファイル（コンフィギュレーション・ファイル）を情報ファイルへと変換するユーティリティを提供しています。

このユーティリティが“コンフィギュレータ CF850 Pro”であり，コンフィギュレータは独自の記述形式で作成されたコンフィギュレーション・ファイルを入力ファイルとして読み込んだのち，システム情報テーブル，ブランチ・テーブル，システム情報ヘッダ・ファイルといった情報ファイルを出力します。

1.6 適用分野

RX850 Pro は，次のような装置に適しています。

●モータ制御を利用したシステム

例　PPC，プリンタ，FAX

●低消費電力が必要なシステム

例　携帯電話，PHS，ディジタル・スチル・カメラ

保守／廃止

★ 1.7　実行環境

RX850 Pro は，組み込み型制御用の OS として開発されているため，次に示すハードウエアを備えたターゲット・システム上で動作します。

(1) 動作CPU

- V850 コア
- V850E1 コア
- V850E2 コア
- V850ES コア
- V850ES/Kx1 シリーズ
- V850ES/Kx1＋シリーズ

(2) 周辺コントローラ

RX850 Pro では，さまざまな実行環境に対応するために，ニュークリアス内のハードウエア依存部を切り出し，サンプル・ソース・ファイルで提供しています。このため，サンプル・ソース・ファイルを各ターゲット・システム用に書き換えることで対応が可能です。そのため特定の周辺コントローラを要求しません。

(3) 外部RAMへのデータ・アクセスについて

RX850 Pro の管理領域が配置される SPOL0 領域は 8 ビット（1 バイト）単位でデータ・アクセスできる必要があります。つまり外部 RAM 領域に 8 ビット・アクセスができないような場合，その領域に SPOL0 領域を配置することができません。配置されると動作中にデータが欠落し，正常動作しません。

スタックやメモリ・プールへのアクセスに関しては，8 ビット・アクセスがないので，SPOL1 領域に関しては，配置しても問題ありません。

ただし，SPOL1 領域内に作ったメモリ・ブロックなどにデータ・アクセスするコードに関しては，コンパイル・オプションなどで 8 ビット・アクセスを抑制する必要があります。

なお，この問題は SPOL0 領域を V850 シリーズの内蔵 RAM に配置することができれば解決できます。

★ 1.8　開発環境

システムを開発するうえで必要となるハードウエア環境とソフトウエア環境を，次に示します。

1.8.1　ハードウエア環境

(1) ホスト・マシン

- Windows が動作する PC
- SPARCstation™

(2) インサーキット・エミュレータ

使用する CPU に合わせて選択してください。詳細については，パンフレットなどを参照してください。

(3) インサーキット・エミュレータ用I/Oボード

使用する CPU に合わせて選択してください。詳細については，パンフレットなどを参照してください。

(4) PCインタフェース・ボード

使用するインサーキット・エミュレータ，ホスト・マシンに合わせて選択してください。

1.8.2 ソフトウエア環境

(1) OS（かっこ内はホスト・マシン）

- Windows98, Me, 2000, WindowsNT™ 4.0（PC-9800 シリーズ，IBM PC/AT 互換機）
- Solaris™2.x（SPARCstation）

(2) クロス・ツール

- CA850（NEC エレクトロニクス製）
- CCV850（GHS 社製）

(3) ディバッガ

- ID850（NEC エレクトロニクス製）
- SM850（NEC エレクトロニクス製）
- MULTI™, MULTI2000（GHS 社製）
- PARTNER（KMC 社製）

(4) タスク・ディバッガ

- RD850 Pro（NEC エレクトロニクス製）

備考　RX850 Pro パッケージに付属。

(5) システム・パフォーマンス・アナライザ

- AZ850（NEC エレクトロニクス製）

★ 1.9 システム構築手順

システム構築とは，RX850 Pro の提供媒体（CGMT，または 3.5 インチ FD）からユーザの開発環境（ホスト・マシン）上に転送したファイル群を用いて，ロード・モジュールを作成したあと，ターゲット・システムへ組み込むことです。

次に，RX850 Pro システムを構築する際の手順を示します。

ただし，詳細な説明は，**RX850 Pro　ユーザーズ・マニュアル　インストレーション編（U13774J）**を参照してください。

保守／廃止

(1) コンフィギュレーション・ファイルの作成

(2) 情報定義ファイルの作成

- ●システム情報テーブル（SIT）
- ●システム・コール・テーブル（SCT）
- ●システム情報ヘッダ・ファイル

なお，これらの情報テーブルは，コンフギュレータを利用して作成します。

(3) システム初期化処理の作成

- ●ブート処理
- ●ハードウエア初期化部
- ●ソフトウエア初期化部

(4) 処理プログラムの作成

- ●タスク
- ●割り込みハンドラ
- ●周期起動ハンドラ
- ●拡張 SVC ハンドラ
- ●拡張 SVC ハンドラ用インタフェース・ライブラリ

これらのプログラムは，C 言語やアセンブリ言語を用いて作成します。

(5) 初期化データの退避領域作成（CA850使用時のみ）

(6) リンク・ディレクティブ・ファイル（セクション・マップ・ファイル）の作成

(7) ロード・モジュールの作成

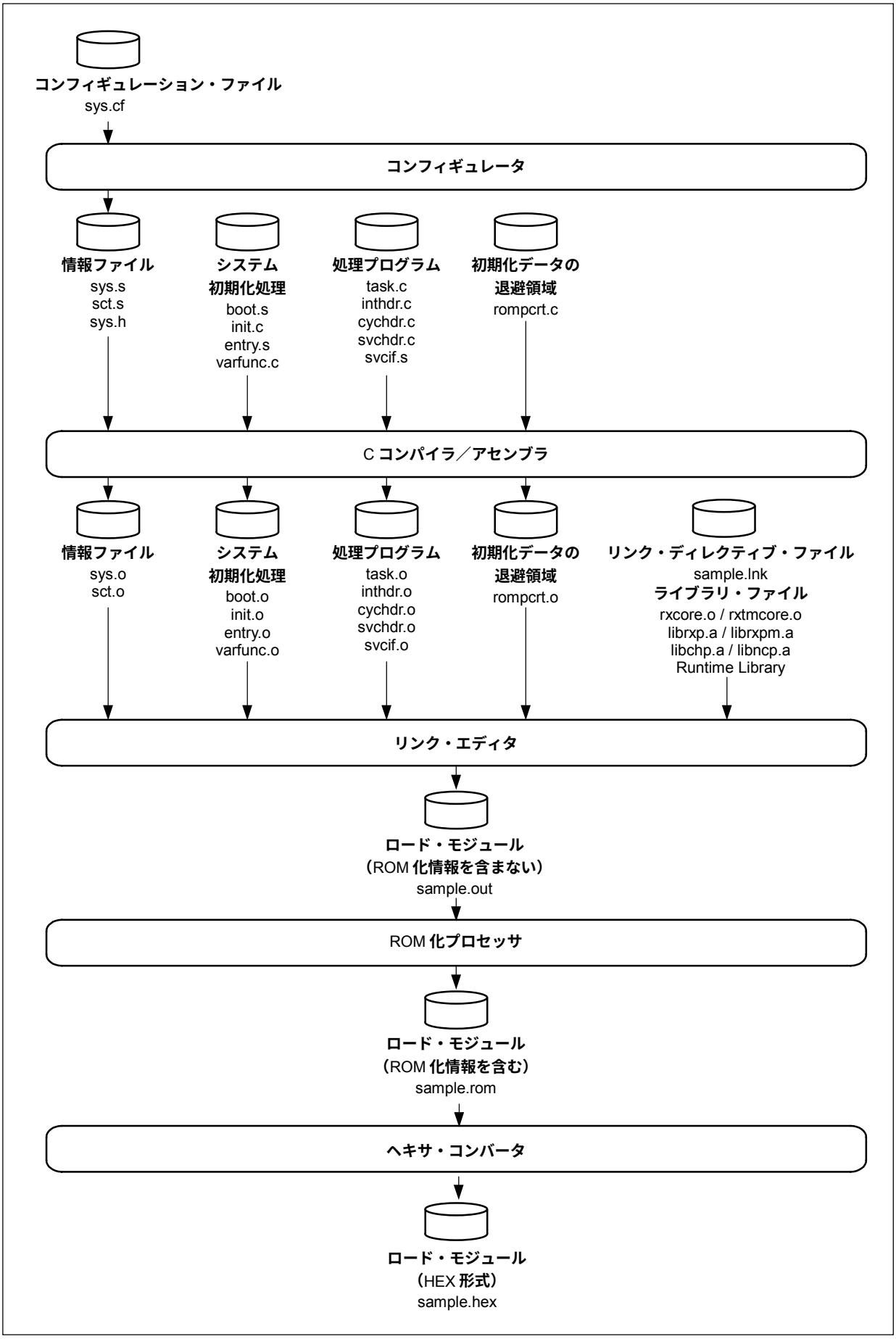
(8) システムへの組み込み

備考 Green Hills Software 社製 C クロス V800 コンパイラ CCV850 を使用した場合，“ 初期化データの退避領域 ”を作成する必要はありません。なお，初期データの退避領域の作成についての詳細は，**CA850　C コンパイラ・パッケージ　ユーザーズ・マニュアル　操作編（U16053J）**を参照してください。

図 1－1 に NEC エレクトロニクス製 V850 シリーズ用 C コンパイラ CA850 使用時のシステム構築手順の例を，図 1－2 に Green Hills Software 社製 C クロス V800 コンパイラ CCV850 使用時のシステム構築手順の例をそれぞれ示します。

保守／廃止

図1－1　システム構築手順例（CA850使用時）



保守／廃止

次に，図 1－1 に示した各種ファイルの概要を示します。これらのファイルは，サンプルで提出されています。

● コンフィギュレーション・ファイル

sys.cf　：コンフィギュレーション・ファイル

● 情報ファイル

sys.s　：システム情報テーブル
sct.s　：ブランチ・テーブル
sys.h　：システム情報ヘッダ・ファイル

● システム初期化処理

boot.s　：ブート処理
init.c　：ハードウエア初期化部（割り込みコントローラの初期化）
entry.s　：ハードウエア初期化部（割り込み／例外エントリ）
varfunc.c　：ソフトウエア初期化部

● 処理プログラム

task.c　：タスク
inthdr.c　：割り込みハンドラ
cychdr.c　：周期起動ハンドラ
svchdr.c　：拡張 SVC ハンドラ
svcif.s　：拡張 SVC ハンドラ用インタフェース・ライブラリ

● 初期化データの退避領域

rompcrt.s　：初期化データの退避領域

● リンク・ディレクティブ・ファイル

sample.lnk　：リンク・ディレクティブ・ファイル

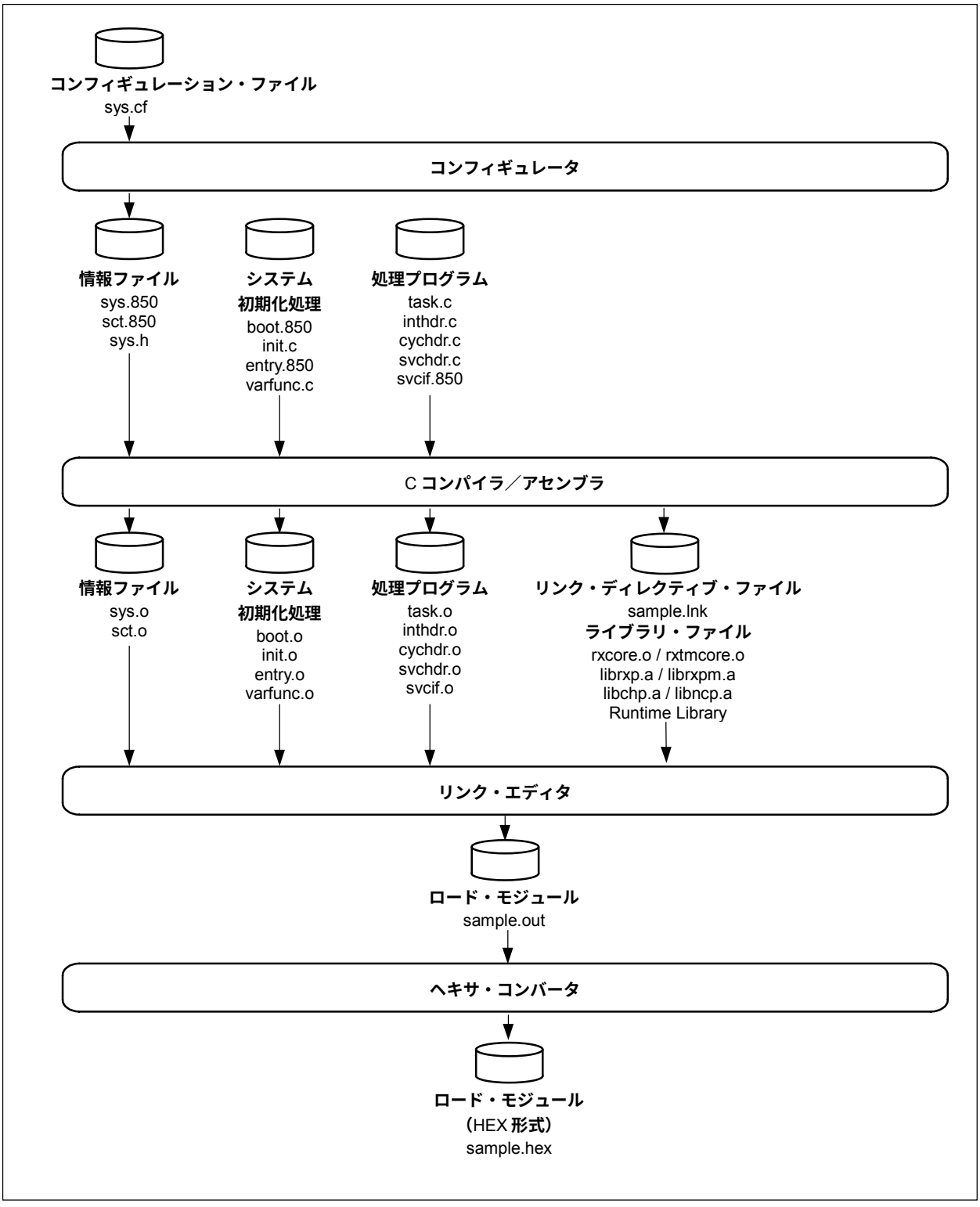
● ニュークリアス・オブジェクト

rxcore.o / rxtmcore.o　：ニュークリアス共通部
librxp.a / librxpm.a　：ニュークリアス・ライブラリ
libchp.a / libncp.a　：システム・コール用インタフェース・ライブラリ

● ロード・モジュール

sample.out　：ROM 化情報を含まない
sample.rom　：ROM 化情報を含む
sample.hex　：HEX 形式

図1－2　システム構築手順例（CCV850使用時）



保守／廃止

次に，図 1－2 に示した各種ファイルの概要を示します。これらのファイルは，サンプルで提出されています。

● コンフィギュレーション・ファイル

sys.cf　：コンフィギュレーション・ファイル

● 情報ファイル

sys.850　：システム情報テーブル
sct.850　：ブランチ・テーブル
sys.h　：システム情報ヘッダ・ファイル

● システム初期化処理

boot.850　：ブート処理
init.c　：ハードウエア初期化部（割り込みコントローラの初期化）
entry.850　：ハードウエア初期化部（割り込み／例外エントリ）
varfunc.c　：ソフトウエア初期化部

● 処理プログラム

task.c　：タスク
inthdr.c　：割り込みハンドラ
cychdr.c　：周期起動ハンドラ
svchdr.c　：拡張 SVC ハンドラ
svcif.850　：拡張 SVC ハンドラ用インタフェース・ライブラリ

● リンク・ディレクティブ・ファイル

sample.lnk　：リンク・ディレクティブ・ファイル

● ニュークリアス・オブジェクト

rxcore.o / rxtmcore.o　：ニュークリアス共通部
librxp.a / librxpm.a　：ニュークリアス・ライブラリ
libchp.a / libncp.a　：システム・コール用インタフェース・ライブラリ

● ロード・モジュール

sample.out　：ROM 化情報を含む／含まない
sample.hex　：HEX 形式

保守／廃止

第2章　ニュークリアス

この章では，RX850 Pro の核であるニュークリアスについて説明します。

2.1 概　　要

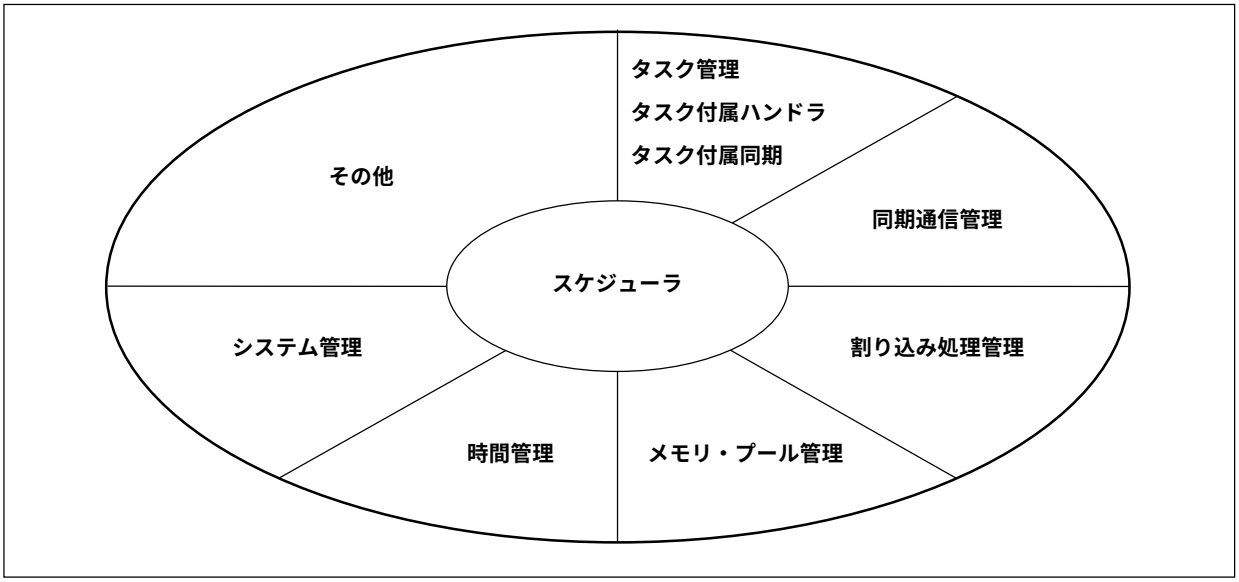
ニュークリアスとは，リアルタイム，マルチタスク制御を行う RX850 Pro の核となる部分であり，次に示す機能を提供しています。

- 管理オブジェクトの生成，初期化処理
- 処理プログラム（タスク，非タスク）から発行されたシステム・コールに対応した処理
- ターゲット・システムの内外で発生した事象に対応した，次に実行すべき処理プログラム（タスク，非タスク）の選択処理

なお，管理オブジェクトの生成，初期化処理とシステム・コールに対応した処理は各種管理モジュールで，処理プログラムの選択処理はスケジューラで行われます。

次に，RX850 Pro のニュークリアスの構成を示します。

図2－1　ニュークリアスの構成



保守／廃止

2.2 機　　能

ニュークリアスは，各種管理モジュールとスケジューラから構成されています。
次に，各種管理モジュールとスケジューラの機能概要を示します。
なお，これらの機能に関する詳細は，**第3章　タスク管理機能 ～ 第8章　スケジューラ**を参照してください。

(1) タスク管理機能

RX850 Pro の処理単位である，「タスクの生成，起動，実行，停止，終了，削除」などといったタスクの状態操作とタスクの状態管理を行います。

(2) 同期通信機能

「排他制御，待ち合わせ，通信」といったタスク間の同期通信機能として，次の 3 種類の機能を提供します。

排他制御機能　：セマフォ
待ち合わせ機能：イベント・フラグ
通信機能　　：メールボックス

(3) 割り込み管理機能

「間接起動割り込みハンドラの登録，直接起動割り込みハンドラからの復帰，割り込み許可レベルの変更／獲得」などといったマスカブル割り込みに関連した処理を行います。

(4) メモリ・プール管理機能

コンフィギュレーション時に指定されたメモリ領域を，次の 2 種類の領域に分けて管理します。

●RX850 Pro 用の領域
　各種管理オブジェクト
　メモリ・プール

●処理プログラム（タスク，非タスク）用の領域
　処理プログラムのテキスト領域
　処理プログラムのデータ領域
　処理プログラムのスタック領域

なお，RX850 Pro では，ダイナミックなメモリ・プール管理も行っており，作業用のメモリ領域が必要になった際に獲得し，不要になった際には返却できるといった機能が用意されています。
ユーザは，このようなダイナミックなメモリ操作を行うことにより，限りあるメモリ領域を効率良く使用することができます。

(5) 時間管理機能

ハードウエア（クロック・コントローラなど）により一定周期で発生するクロック割り込みを利用した，タイマ・オペレーション機能（タスクの遅延起床，周期起動ハンドラの起動）などを提供します。

(6) スケジューラ

ダイナミックに変化するタスクの状態を観察することにより，タスクの実行順序を管理，決定し，最適なタスクにプロセッサの使用権を与えます。

なお，RX850 Pro では，タスクの実行順序を決定する方法として，優先度方式と FCFS 方式を採用しています。このため，スケジューラは，駆動された時点で，各タスクに付けられている優先度を参照し，実行可能な状態（run 状態または ready 状態）にあるタスクの中から最適なものを選び出し，プロセッサの使用権を与えます。

備考　RX850 Pro におけるタスクの優先度は，その値が小さいほど高い優先度であることを示します。

第 3 章　タスク管理機能

この章では，RX850 Pro が行うタスク管理機能について説明します。

3.1　概　　要

タスクは実行実体であり，サイズなどが一様ではなく，直接管理することが困難です。そこで，RX850 Pro では，タスクと 1 対 1 に対応した管理オブジェクトを用いることにより，タスクが取り得る状態の管理，タスク自体の管理を行っています。

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

3.2　タスクの状態

タスクは，処理を実行するのに必要な資源の獲得状況や事象発生の有無などにより，さまざまな状態へ遷移します。

そこで，RX850 Pro では，タスクの遷移する状態を，次の 7 種類に分けて管理しています。

(1) 未登録（non_existent）状態

タスクとして生成されていない状態，または，削除された際に遷移する状態です。

なお，non_existent 状態のタスクは，その実行実体がメモリ上に配置されていても，RX850 Pro が管理していない状態です。

(2) 休止（dormant）状態

タスクとして生成されたときの状態，または，タスクとしての処理を終了したときに遷移する状態です。

なお，dormant 状態のタスクは，RX850 Pro のスケジューリング対象から除外されています。

また，wait 状態とは次のような相違点があります。

- すべての資源を解放している
- タスク・コンテキストが処理再開時に初期化される
- 状態操作を伴うシステム・コール（ter_tsk，chg_pri など）がエラーとなる

(3) 実行可能（ready）状態

タスクとして処理を実行するうえで必要な準備は整っているが，より高い優先度（同じ優先度の場合もある）を持つほかのタスクが実行されているため，プロセッサの実行権が割り当てられるのを待っている状態です。

なお，ready 状態のタスクは，RX850 Pro のスケジューリング対象となります。

保守／廃止

(4) 実行（run）状態

プロセッサの実行権が割り当てられ，現在タスクとして処理を実行中の状態です。

なお，run 状態のタスクは，システム全体で 1 タスクしか存在しません。

(5) 待ち（wait）状態

処理を実行するために必要な条件が整わないため，処理の実行が中断した状態です。

なお，wait 状態からの処理再開は，処理の実行が中断した箇所からとなります。したがって，処理を再開するうえで必要となるタスク・コンテキストは，中断直前の値が復元されます。

また，RX850 Pro では，wait 状態へと遷移する原因となった条件の種類により，次の 6 つの状態に分けて管理しています。

起床待ち状態	：slp_tsk，tslp_tsk システム・コールを発行した際，自タスクのカウンタ（起床要求の発行回数を保持）が 0x0 の場合に遷移する状態です。
資源待ち状態	：wai_sem，twai_sem システム・コールを発行した際，対象セマフォから資源を獲得できなかった場合に遷移する状態です。
イベント・フラグ待ち状態	：wai_flg，twai_flg システム・コールを発行した際，対象イベント・フラグが要求した条件を満たしていなかった場合に遷移する状態です。
メッセージ待ち状態	：rcv_msg，trcv_msg システム・コールを発行した際，対象メールボックスからメッセージを受信できなかった場合に遷移する状態です。
メモリ・ブロック待ち状態	：get_blk，tget_blk システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった場合に遷移する状態です。
時間経過待ち状態	：dly_tsk システム・コールを発行した際，遷移する状態です。

(6) 強制待ち（suspend）状態

他タスクから強制的に実行を中断させられた状態です。

なお，suspend 状態からの処理再開は，処理の実行が中断した箇所からとなります。したがって，処理を再開するうえで必要となるタスク・コンテキストは，中断直前の値が復元されます。

備考　RX850 Pro では，suspend 状態をネストさせることも可能です（127 回まで）。

(7) 二重待ち（wait_suspend）状態

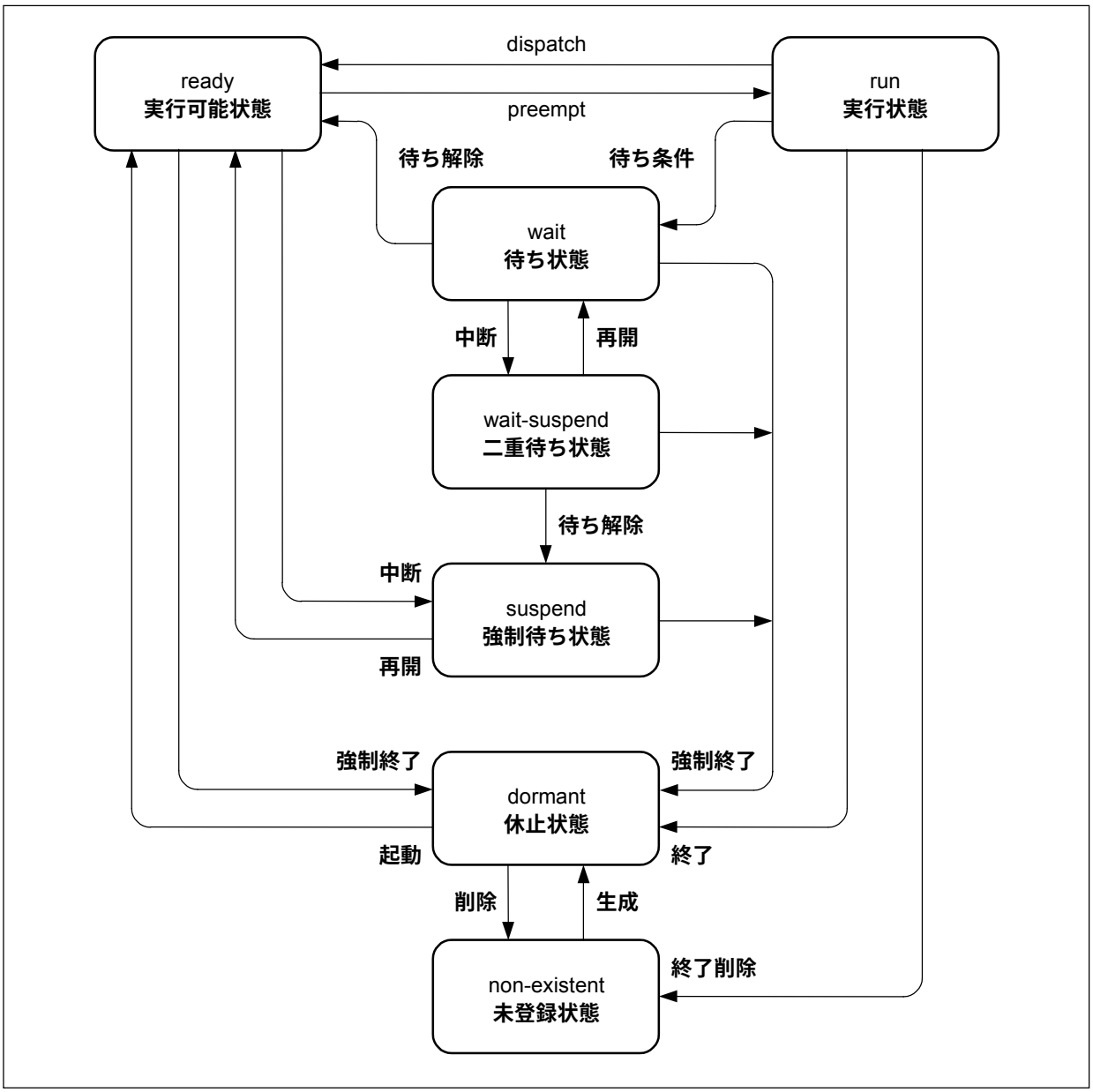
wait 状態と suspend 状態が複合した状態です。

wait 状態が解除されると suspend 状態へ，suspend 状態が解除されると wait 状態へと遷移します。

図 3－1 に，タスクの状態遷移を示します。

保守／廃止

図3－1　タスクの状態遷移



3.3　タスクの生成

RX850 Pro では，タスクの生成において，「システム初期化処理（ニュークリアス初期化部）において，スタティックに生成する」，「処理プログラム内からシステム・コールを発行し，ダイナミックに生成する」の 2 種類のインタフェースを用意しています。

なお，RX850 Pro におけるタスクの生成とは，タスクを管理するための領域（管理オブジェクト）をシステム・メモリ領域に確保したあと，初期化し，タスクの状態を non_existent 状態から dormant 状態へと遷移させることです。

保守／廃止

(1) スタティックに登録する場合

タスクをスタティックに登録する場合，コンフィギュレーション時に指定します。

RX850 Pro では，システム初期化処理時，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に定義された情報を基にタスクの生成処理を行い，管理対象とします。

(2) ダイナミックに登録する場合

タスクをダイナミックに登録する場合，処理プログラム（タスク）内から cre_tsk システム・コールを発行します。

RX850 Pro では，cre_tsk システム・コール発行時，パラメータで指定された情報を基にタスクの生成処理を行い，管理対象とします。

3.4　タスクの起動

RX850 Pro におけるタスクの起動は，タスクを dormant 状態から ready 状態へ遷移させ，スケジューリング対象とすることです。

なお，タスクの起動は，パラメータでタスクを指定して sta_tsk システム・コールを発行することにより行います。

●sta_tsk システム・コール

パラメータで指定されたタスクを dormant 状態から ready 状態へと遷移させます。

3.5　タスクの終了

RX850 Pro におけるタスクの終了は，各種状態（ready 状態，run 状態，wait 状態，suspend 状態，wait_suspend 状態）のタスクを dormant 状態に遷移させ，RX850 Pro のスケジューリング対象から除外することです。

RX850 Pro では，タスクの終了形態として次に示す 2 種類を用意しています。

正常終了：すべての処理が順調に完了し，スケジューリング対象である必要がなくなったとき，タスク自身で終了する形態です。

強制終了：処理途中で何らかの異常が発生し，緊急に処理を終了しなければならないとき，他タスクから終了させられる形態です。

タスクの終了は，次に示すシステム・コールの発行により行います。

●ext_tsk システム・コール

自タスクを run 状態から dormant 状態へと遷移させます。

●exd_tsk システム・コール

自タスクを run 状態から non_existent 状態へと遷移させます。

●ter_tsk システム・コール

パラメータで指定されたタスクを強制的に dormant 状態へと遷移させます。

保守／廃止

3.6　タスクの削除

RX850 Pro におけるタスクの削除は，run 状態または dormant 状態のタスクを non_existent 状態へと遷移させ，RX850 Pro の管理下から除外することです。

タスクの削除は，次に示すシステム・コールの発行により行います。

●exd_tsk システム・コール

自タスクを run 状態から non_existent 状態へと遷移させます。

●del_tsk システム・コール

パラメータで指定されたタスクを dormant 状態から non_existent 状態へと遷移させます。

3.7　タスク内での処理

RX850 Pro では，タスクを切り替える際，独自のスケジューリング処理を行っています。
したがって，タスクの処理を記述する際には，次の点に注意してください。

(1) レジスタの退避と復帰

RX850 Pro では，タスクを切り替える際，C コンパイラ（CA850，CCV850）の関数呼び出し規約に従った，作業用レジスタの退避処理，復帰処理を行っています。したがって，タスクの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要はありません。

ただし，タスクをアセンブリ言語で記述し，レジスタ変数用レジスタを使用する場合は，タスクの開始部分でレジスタ変数用レジスタの退避処理を，終了部分でレジスタ変数用レジスタの復帰処理を記述する必要があります。

(2) スタックの切り替え

RX850 Pro では，タスクを切り替える際，各タスク専用のタスク用スタックへの切り替え処理を行っています。したがって，タスクの開始部分と終了部分でスタックの切り替え処理を記述する必要はありません。

(3) システム・コールの発行制限

RX850 Pro のシステム・コールのなかには，タスク内で発行できないものもあります。
次に，タスク内で発行可能なシステム・コールの一覧を示します。

●タスク管理機能システム・コール

cre_tsk	del_tsk	sta_tsk	ext_tsk	exd_tsk
ter_tsk	dis_dsp	ena_dsp	chg_pri	rot_rdq
rel_wai	get_tid	ref_tsk	vget_tid	

●タスク付属同期機能システム・コール

sus_tsk	rsm_tsk	frsm_tsk	slp_tsk	tslp_tsk
wup_tsk	can_wup			

●同期通信機能システム・コール

cre_sem	del_sem	sig_sem	wai_sem	preq_sem
twai_sem	ref_sem	vget_sid	cre_flg	del_flg
set_flg	clr_flg	wai_flg	pol_flg	twai_flg
ref_flg	vget_fid	cre_mbx	del_mbx	snd_msg
rcv_msg	prcv_msg	trcv_msg	ref_mbx	vget_mid

●割り込み管理機能システム・コール

def_int	ena_int	dis_int	loc_cpu	unl_cpu
chg_icr	ref_icr			

●メモリ・プール管理機能システム・コール

cre_mpl	del_mpl	get_blk	pget_blk	tget_blk
rel_blk	ref_mpl	vget_pid		

●時間管理機能システム・コール

set_tim	get_tim	dly_tsk	def_cyc	act_cyc
ref_cyc				

●システム管理機能システム・コール

get_ver	ref_sys	def_svc	viss_svc

3.7.1　タスク情報の獲得

タスク情報の獲得は，ref_tsk システム・コールを発行することにより行われます。

●ref_tsk システム・コール

パラメータで指定されたタスクのタスク情報（拡張情報，現在の優先度など）を獲得します。

次に，タスク情報の詳細を示します。

- 拡張情報
- 現在の優先度
- タスクの状態
- wait 状態の種類
- 待ち対象オブジェクト（セマフォ，イベント・フラグなど）の ID 番号
- 起床要求数
- サスペンド要求数
- キーID 番号

保守／廃止

3.7.2　ID番号の獲得

タスクのID番号の獲得は，vget_tidシステム・コールを発行することにより行われます。

●vget_tidシステム・コール

パラメータで指定されたタスクのID番号を獲得します。

システム・コールでタスクに対する操作をするとき，タスクのID番号が必要です。タスクを生成する際，ID番号をユーザが一意に決定するか，自動的に振り当てるかを指定できます。しかし，自動的に振り当てると指定した場合，ユーザはタスクのID番号を知ることができません。そのために必要となるものが「キーID番号」です。キーID番号はタスク生成の際に一意に指定します。

このキーID番号をパラメータとしてvget_tidシステム・コールを発行すると，そのキーID番号を持つタスクのID番号を取得できます。

保守／廃止

第 4 章　同期通信機能

この章では，RX850 Pro が行う同期通信機能について説明します。

4.1 概　　要

複数のタスクが並行に実行されている環境（マルチタスク処理）では，あるタスクの処理結果により，次に実行されるタスクが選択されたり，タスクの処理内容に違いが出るなど，タスク間で処理の実行条件を制限しあったり，処理内容が相互に関係しているという場合があります。

このため，あるタスクが他タスクの実行結果が出るまで実行を中断したり，処理を継続するうえで必要な条件が整うまで待つといった，タスク間の連絡機能が必要となります。

RX850 Pro では，このような機能を，「同期機能」と呼びます。なお，同期機能には，「排他制御機能」と「待ち合わせ機能」があり，RX850 Pro では，排他制御機能としてセマフォを，待ち合わせ機能としてイベント・フラグを提供しています。

また，マルチタスク処理では，他タスクから処理結果を通知してもらうといった，タスク間の通信機能も必要となります。

RX850 Pro では，このような機能を，「通信機能」と呼びます。なお，RX850 Pro では，通信機能としてメールボックスを提供しています。

4.2 セマフォ

マルチタスク処理では，並行に動作する複数のタスクが，限られた数の資源（A/D コンバータ，コプロセッサ，ファイル，プログラムなど）を同時に使用するといった，資源の競合を防ぐ機能が必要となります。RX850 Pro では，このような機能を実現するために，非負数の計数型セマフォを提供しています。

なお，セマフォに対するダイナミックな操作は，次に示すセマフォ関連のシステム・コールを用いて行います。

cre_sem　：セマフォを生成する
del_sem　：セマフォを削除する
sig_sem　：資源を返却する
wai_sem　：資源を獲得する
preq_sem ：資源を獲得する（ポーリング）
twai_sem ：資源を獲得する（タイムアウトあり）
ref_sem　：セマフォ情報を獲得する
vget_sid　：セマフォの ID 番号を獲得する

備考　タスクの実行に必要となる各種要素を「資源」と呼びます。つまり，資源とは，A/D コンバータ，コプロセッサなどといったハードウエアや，ファイル，プログラムなどといったソフトウエアのすべてを指します。

保守／廃止

4.2.1　セマフォの生成

RX850 Pro では，セマフォの生成において，「システム初期化処理（ニュークリアス初期化部）において，スタティックに生成する」，「処理プログラム内からシステム・コールを発行し，ダイナミックに生成する」の 2 種類のインタフェースを用意しています。

RX850 Pro におけるセマフォの生成とは，セマフォを管理するための領域（管理オブジェクト）をシステム・メモリ領域から確保したあと，初期化することです。

(1) スタティックに登録する場合

セマフォをスタティックに登録する場合，コンフィギュレーション時に指定します。

RX850 Pro では，システム初期化処理時，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に定義された情報を基にセマフォの生成処理を行い，管理対象とします。

(2) ダイナミックに登録する場合

セマフォをダイナミックに登録する場合，処理プログラム（タスク）内から cre_sem システム・コールを発行します。

RX850 Pro では，cre_sem システム・コール発行時，パラメータで指定された情報を基にセマフォの生成処理を行い，管理対象とします。

4.2.2　セマフォの削除

セマフォの削除は，del_sem システム・コールを発行することにより行います。

●del_sem システム・コール

パラメータで指定されたセマフォを削除します。

これにより，対象セマフォは RX850 Pro の管理対象から除外されます。

ただし，このシステム・コールを発行した際，対象セマフォの待ちキューにタスクがキューイングされていた場合には，該当タスクを待ちキューから外すとともに，wait 状態（資源待ち状態）から ready 状態へと遷移させます。

なお，wait 状態を解除されたタスクには，wait 状態へと遷移するきっかけとなったシステム・コール（wai_sem，twai_sem）の戻り値として E_DLT が返されます。

4.2.3　資源の返却

資源の返却は，sig_sem システム・コールを発行することにより行います。

●sig_sem システム・コール

パラメータで指定されたセマフォに資源を返却（セマフォ・カウンタに 0x1 を加算）します。

ただし，このシステム・コールを発行した際，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却処理（セマフォ・カウンタの加算）は行わず，該当タスク（待ちキューの先頭タスク）に資源を渡します。

これにより，該当タスクは待ちキューから外れ，wait 状態（資源待ち状態）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

保守／廃止

4.2.4　資源の獲得

資源の獲得は，wai_sem，preq_sem，または twai_sem システム・コールを発行することにより行います。

●wai_sem システム・コール

パラメータで指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，対象セマフォから資源を獲得できなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューにキューイングしたあと，run 状態から wait 状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態は次の場合に解除され，ready 状態へ遷移します。

- sig_sem システム・コールが発行された
- del_sem システム・コールが発行され，対象セマフォが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

備考　自タスクを対象セマフォの待ちキューにキューイングする際は，対象セマフォ生成時（コンフィギュレーション時，または cre_sem システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

●preq_sem システム・コール

パラメータで指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，対象セマフォから資源を獲得できなかった（空き資源が存在しなかった）場合には，戻り値として E_TMOUT が返されます。

●twai_sem システム・コール

パラメータで指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，対象セマフォから資源を獲得できなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューにキューイングしたあと，run 状態から wait 状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された待ち時間が経過した
- sig_sem システム・コールが発行された
- del_sem システム・コールが発行され，対象セマフォが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

備考　自タスクを対象セマフォの待ちキューにキューイングする際は，対象セマフォ生成時（コンフィギュレーション時または cre_sem システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

保守／廃止

4.2.5　セマフォ情報の獲得

セマフォ情報の獲得は，ref_sem システム・コールを発行することにより行われます。

●ref_sem システム・コール

パラメータで指定されたセマフォのセマフォ情報（拡張情報，待ちタスクの有無など）を獲得します。

次に，セマフォ情報の詳細を示します。

- 拡張情報
- 待ちタスクの有無
- 現在の資源数
- 生成時に指定した最大資源数
- キーID 番号

4.2.6　ID 番号の獲得

セマフォの ID 番号の獲得は，vget_sid システム・コールを発行することにより行われます。

●vget_sid システム・コール

パラメータで指定されたセマフォの ID 番号を獲得します。

システム・コールでセマフォに対する操作をするとき，セマフォの ID 番号が必要です。セマフォを生成する際，ID 番号をユーザが一意に決定するか，自動的に振り当てるかを指定できます。しかし，自動的に振り当てると指定した場合，ユーザはセマフォの ID 番号を知ることができません。そのために必要となるものが「キー ID 番号」です。キーID 番号はセマフォ生成の際に一意に指定します。

このキーID 番号をパラメータとして vget_sid システム・コールを発行すると，そのキーID 番号を持つセマフォの ID 番号を取得できます。

4.2.7　セマフォによる排他制御

次に，セマフォを使用してタスク間の排他制御を行った場合の動作例を示します。

条　　件

●タスクの優先度

タスク A ＞ タスク B

●タスクの状態

タスク A：run 状態

タスク B：ready 状態

●セマフォの属性

セマフォの初期資源数　　：0x1

セマフォの最大資源数　　：0x5

タスクのキューイング順序 ：FIFO 順

保守／廃止

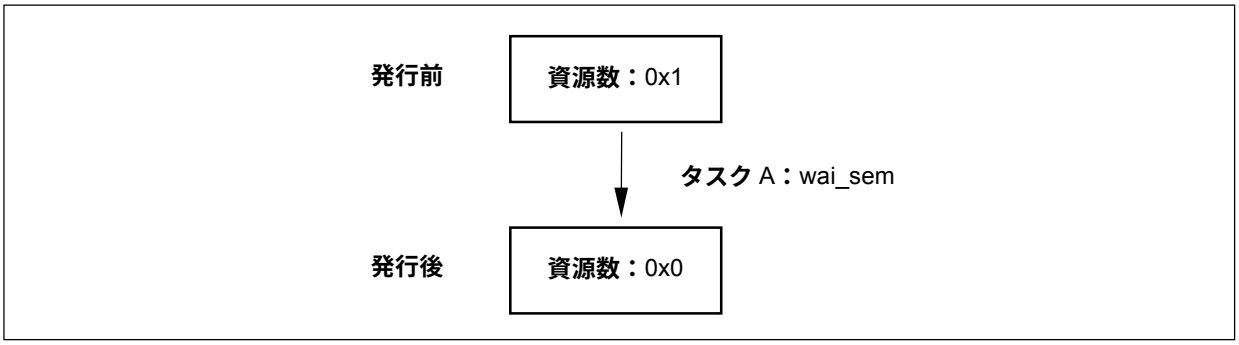
(1)　タスクAがwai_semシステム・コールを発行します。

現在 RX850 Pro が管理している対象セマフォの資源数は「0x1」です。したがって，RX850 Pro は対象セマフォのカウンタから 0x1 を減算します。

このとき，タスク A は wait 状態（資源待ち状態）へと遷移することはなく，run 状態のままとなります。

なお，対象セマフォのカウンタは図 4－1 のようになります。

図4－1　セマフォ・カウンタの状態

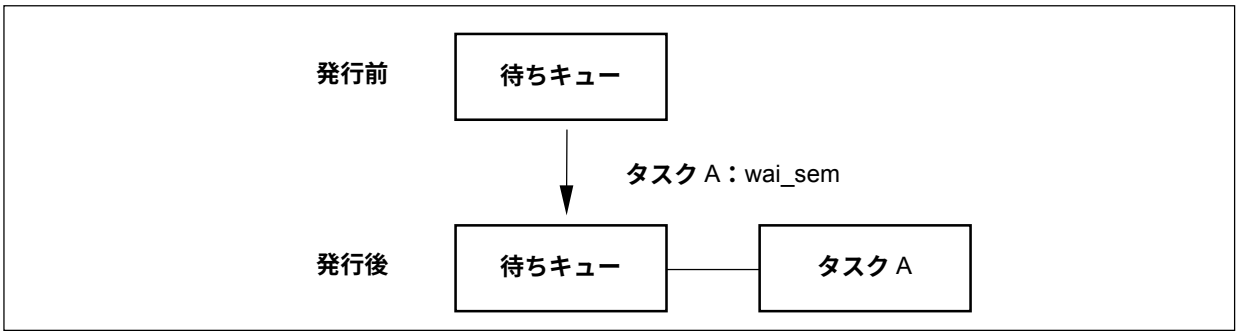


(2)　タスクAがwai_semシステム・コールを発行します。

現在 RX850 Pro が管理している対象セマフォの資源数は「0x0」です。したがって，RX850 Pro はタスク A を run 状態から wait 状態（資源待ち状態）へと遷移させ，対象セマフォの待ちキューの最後尾にキューイングします。

このとき，対象セマフォの待ちキューは図 4－2 のようになります。

図4－2　待ちキューの状態（wai_sem発行時）



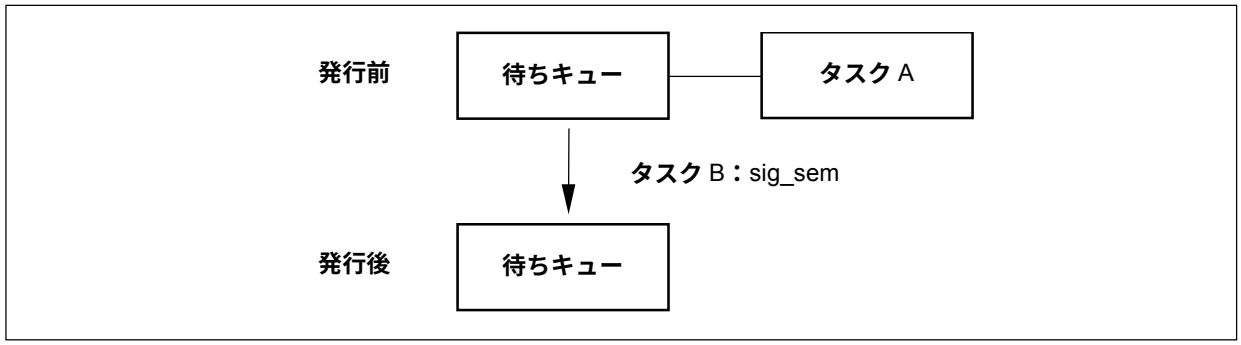
(3)　タスクAの資源待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移します。

(4)　タスクBがsig_semシステム・コールを発行します。

これにより，対象セマフォの待ちキューにキューイングされているタスク A が資源待ち状態から ready 状態へと遷移します。

このとき，対象セマフォの待ちキューは，図 4－3 のようになります。

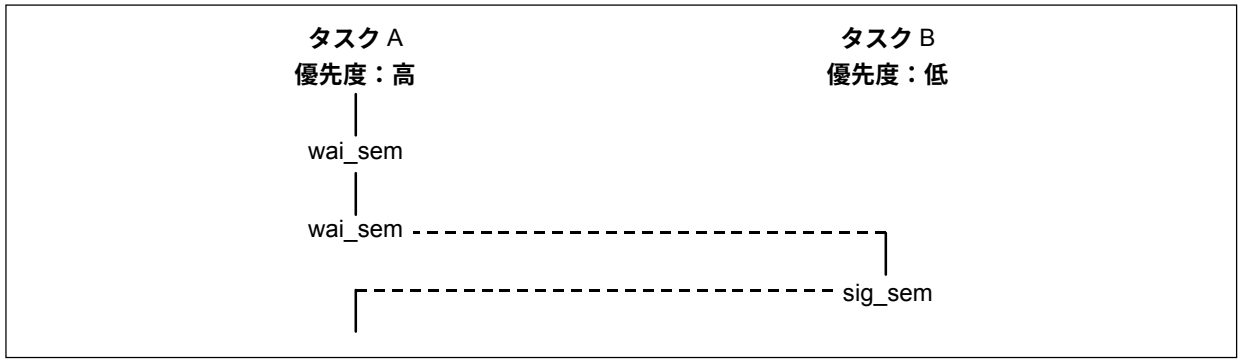
図4－3　待ちキューの状態（sig_sem発行時）



（5） 優先度の高いタスクAがready状態からrun状態へと遷移します。
また，タスク B は run 状態から ready 状態へと遷移します。

図 4－4 に，（1）から（5）で説明した排他制御の流れを示します。

図4－4　セマフォによるタスクの排他制御



4.3　イベント・フラグ

マルチタスク処理では，あるタスクの処理結果が出るまで，他タスクが処理の実行再開を待つといった，タスク間の待ち合わせ機能が必要となります。このような場合，「処理結果が出た」という事象（イベント）が発生したか否かを，他タスクで判断できるような機能があればよく，RX850 Pro ではこのような機能を実現するために，イベント・フラグを提供しています。

イベント・フラグは，事象の有無を表す 1 ビットのフラグから構成されるデータの集合体です。RX850 Pro のイベント・フラグは，32 ビットを一塊の情報として扱っており，各ビットに意味を持たせたり，数ビットの組み合わせで意味を持たせたりできます。

なお，イベント・フラグに対するダイナミックな操作には，次に示すイベント・フラグ関連のシステム・コールを使用します。

cre_flg　：イベント・フラグを生成する
del_flg　：イベント・フラグを削除する
set_flg　：ビット・パターンをセットする
clr_flg　：ビット・パターンをクリアする
wai_flg　：ビット・パターンをチェックする
pol_flg　：ビット・パターンをチェックする（ポーリング）
twai_flg　：ビット・パターンをチェックする（タイムアウトあり）
ref_flg　：イベント・フラグ情報を獲得する
vget_fld　：イベント・フラグの ID 番号を獲得する

4.3.1　イベント・フラグの生成

RX850 Pro では，イベント・フラグの生成において，「システム初期化処理（ニュークリアス初期化部）において，スタティックに生成する」，「処理プログラム内からシステム・コールを発行し，ダイナミックに生成する」の 2 種類のインタフェースを用意しています。

なお，RX850 Pro におけるイベント・フラグの生成とは，イベント・フラグを管理するための領域（管理オブジェクト）をシステム・メモリ領域に確保したあと，初期化することです。

(1) スタティックに登録する場合

イベント・フラグをスタティックに登録する場合，コンフィギュレーション時に指定します。

RX850 Pro では，システム初期化処理時，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に定義された情報を基にイベント・フラグの生成処理を行い，管理対象とします。

(2) ダイナミックに登録する場合

イベント・フラグをダイナミックに登録する場合，処理プログラム（タスク）内から cre_flg システム・コールを発行します。

RX850 Pro では，cre_flg システム・コール発行時，パラメータで指定された情報を基にイベント・フラグの生成処理を行い，管理対象とします。

4.3.2　イベント・フラグの削除

イベント・フラグの削除は，del_flg システム・コールを発行することにより行います。

●del_flg システム・コール

パラメータで指定されたイベント・フラグを削除します。

これにより，対象イベント・フラグは RX850 Pro の管理対象から除外されます。

ただし，このシステム・コールを発行した際，対象イベント・フラグの待ちキューにタスクがキューイングされていた場合には，該当タスクを待ちキューから外すとともに，wait 状態（イベント・フラグ待ち状態）から ready 状態へと遷移させます。

なお，wait 状態を解除されたタスクには，wait 状態へと遷移するきっかけとなったシステム・コール（wait_flg，twait_flg）の戻り値として E_DLT が返されます。

保守／廃止

4.3.3　ビット・パターンのセット

イベント・フラグのビット・パターンのセットは，set_flg システム・コールの発行により行います。

●set_flg システム・コール

パラメータで指定されたイベント・フラグにビット・パターンをセットします。

ただし，このシステム・コールを発行した際，対象イベント・フラグの待ちキューにキューイングされているタスクの待ち条件を満足した場合には，該当タスクを待ちキューから外します。これにより，該当タスクは wait 状態（イベント・フラグ待ち状態）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

4.3.4　ビット・パターンのクリア

イベント・フラグのビット・パターンのクリアは，clr_flg システム・コールの発行により行います。

●clr_flg システム・コール

パラメータで指定されたイベント・フラグのビット・パターンをクリアします。

ただし，このシステム・コールを発行した際，すでに対象イベント・フラグのビット・パターンが 0 にクリアされていても，エラーとして扱われないので注意が必要です。

4.3.5　ビット・パターンのチェック

イベント・フラグのビット・パターンのチェックは，wai_flg，pol_flg，または twai_flg システム・コールの発行により行われます。

●wai_flg システム・コール

パラメータで指定されたイベント・フラグに要求する待ち条件を満足するビット・パターンがセットされているかどうかをチェックします。

このシステム・コールを発行した際，対象イベント・フラグのビット・パターンが要求する待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューの最後尾にキューイングしたあと，run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態は次の場合に解除され，ready 状態へ遷移します。

- set_flg システム・コールが発行され，要求する待ち条件がセットされた
- del_mbx システム・コールが発行され，対象イベント・フラグが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

●pol_flg システム・コール

パラメータで指定されたイベント・フラグに要求する待ち条件を満足するビット・パターンがセットされているかどうかをチェックします。

このシステム・コールを発行した際，対象イベント・フラグのビット・パターンが要求する待ち条件を満足していなかった場合は，戻り値として E_TMOUT が返されます。

●twai_flg システム・コール

パラメータで指定されたイベント・フラグに要求する待ち条件を満足するビット・パターンがセットされているかどうかをチェックします。

保守／廃止

このシステム・コールを発行した際，対象イベント・フラグのビット・パターンが要求する待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューの最後尾にキューイングしたあと，run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された待ち時間が経過した
- set_flg システム・コールが発行され，要求する待ち条件がセットされた
- del_mbx システム・コールが発行され，対象イベント・フラグが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

また，RX850 Pro では，イベント・フラグの待ち条件や条件成立時の処理を次のように指定できます。

(1) 待ち条件

●AND 待ち

要求ビット・パターンで「1」が設定されている全ビットが，対象イベント・フラグに設定されるまで待ちます。

●OR 待ち

要求ビット・パターンで「1」が設定されているいずれかのビットが，対象イベント・フラグに設定されるまで待ちます。

(2) 条件成立時

●ビット・パターンのクリア

待ち条件を満足した際，対象イベント・フラグのビット・パターンをクリアします。

4.3.6　イベント・フラグ情報の獲得

イベント・フラグ情報の獲得は，ref_flg システム・コールを発行することにより行われます。

●ref_flg システム・コール

パラメータで指定されたイベント・フラグのイベント・フラグ情報（拡張情報，待ちタスクの有無など）を獲得します。

次に，イベント・フラグ情報の詳細を示します。

- 拡張情報
- 待ちタスクの有無
- 現在のビット・パターン
- キーID 番号

4.3.7　ID 番号の獲得

イベント・フラグの ID 番号の獲得は，vget_fid システム・コールを発行することにより行われます。

●vget_fid システム・コール

パラメータで指定されたイベント・フラグの ID 番号を獲得します。

システム・コールでイベント・フラグに対する操作をするとき，イベント・フラグの ID 番号が必要です。

イベント・フラグを生成する際，ID 番号をユーザが一意に決定するか，自動的に振り当てるかを指定できます。しかし，自動的に振り当てると指定した場合，ユーザはイベント・フラグの ID 番号を知ることができません。そのために必要となるものが「キーID 番号」です。キーID 番号はイベント・フラグ生成の際に一意に指定します。

このキーID 番号をパラメータとして vget_fid システム・コールを発行すると，そのキーID 番号を持つイベント・フラグの ID 番号を取得できます。

4.3.8　イベント・フラグによる待ち合わせ

次に，イベント・フラグを使用してタスク間の待ち合わせを行った場合の動作例を示します。

条　　件

●タスクの優先度

タスク A ＞ タスク B

●タスクの状態

タスク A：run 状態

タスク B：ready 状態

●イベント・フラグの属性

初期ビット・パターン　　：0x0

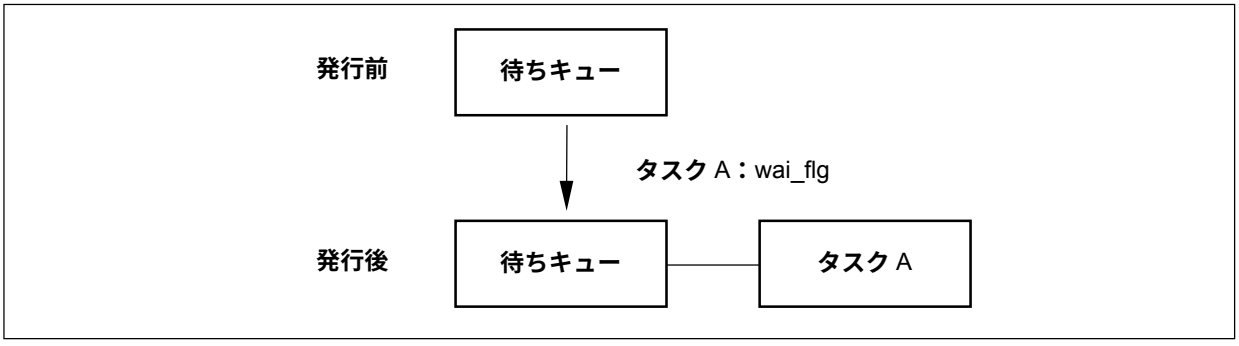
待ちキューにキューイング可能なタスク数：1 タスクのみ

(1) タスクAがwai_flgシステム・コールを発行します。要求ビット・パターンは「0x1」，待ち条件は「TWF_ANDW | TWF_CLR」とします。

現在，RX850 Pro が管理している対象イベント・フラグのビット・パターンは「0x0」です。したがって，RX850 Pro はタスク A を run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させ，対象イベント・フラグの待ちキューの最後尾にキューイングします。

このとき，対象イベント・フラグの待ちキューは，図 4－5 のようになります。

図4－5　待ちキューの状態（wai_flg発行時）



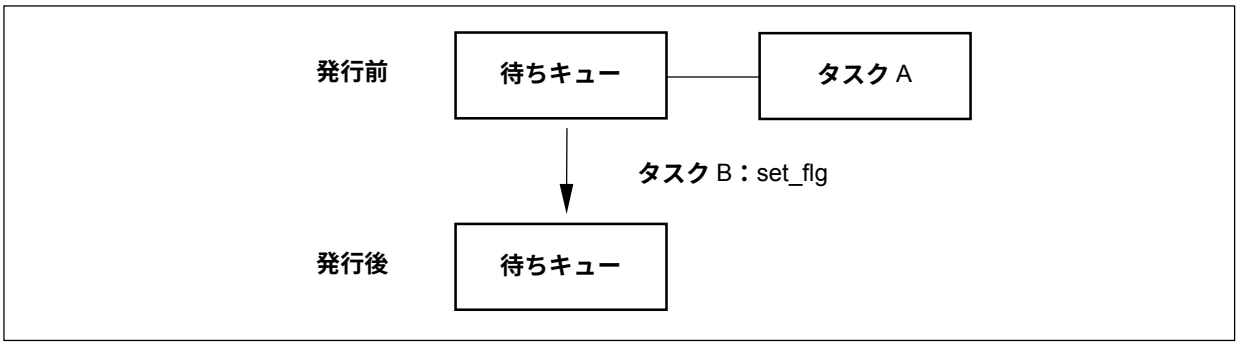
(2) タスクAのイベント・フラグ待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移します。

(3)　タスクBがset_flgシステム・コールを発行します。設定ビット・パターンは「0x1」とします。

これにより，対象イベント・フラグの待ちキューにキューイングされているタスク A の待ち条件を満足したため，タスク A がイベント・フラグ待ち状態から ready 状態へと遷移します。また，タスク A は，wai_flg システム・コールを発行した際「TWF_CLR」を指定していたため，対象イベント・フラグのビット・パターンはクリアされます。

このとき，対象イベント・フラグの待ちキューは，図 4－6 のようになります。

図4－6　待ちキューの状態（set_flg発行時）

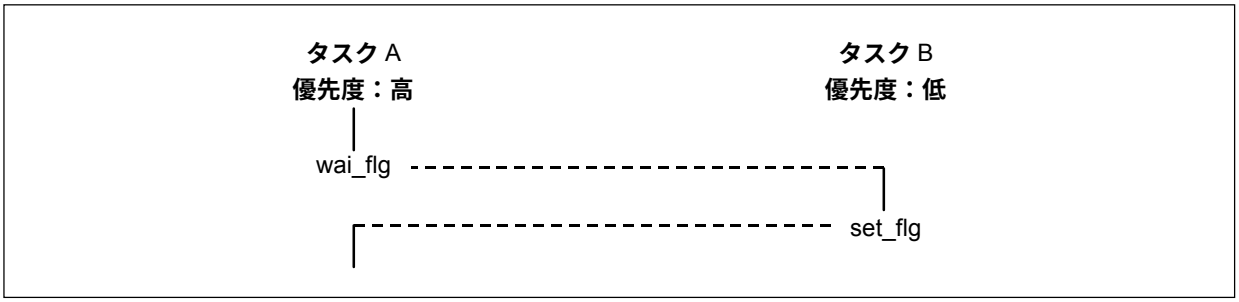


(4)　優先度の高いタスクAがready状態からrun状態へと遷移します。

なお，タスク B は run 状態から ready 状態へと遷移します。

図 4－7 に，(1) から (4) で説明した待ち合わせの流れを示します。

図4－7　イベント・フラグによるタスクの待ち合わせ



4.4　メールボックス

マルチタスク処理では，他タスクから処理結果を通知してもらうといった，タスク間の通信機能が必要となります。RX850 Pro では，このような機能を実現するためにメールボックスを提供しています。

RX850 Pro におけるメールボックスは，タスク専用の待ちキューとメッセージ専用の待ちキューを持ち，タスク間のメッセージ通信機能として使用するほかに，タスク間の待ち合わせ機能としても使用できます。

なお，メールボックスに対するダイナミックな操作は，次に示すメールボックス関連のシステム・コールを用いて行います。

cre_mbx　：メールボックスを生成する
del_mbx　：メールボックスを削除する
snd_msg　：メッセージを送信する
rcv_msg　：メッセージを受信する
prcv_msg ：メッセージを受信する（ポーリング）
trcv_msg　：メッセージを受信する（タイムアウトあり）
ref_mbx　：メールボックス情報を獲得する
vget_mid　：メールボックスの ID 番号を獲得する

4.4.1　メールボックスの生成

RX850 Pro では，メールボックスの生成において，「システム初期化処理（ニュークリアス初期化部）において，スタティックに生成する」，「処理プログラム内からシステム・コールを発行し，ダイナミックに生成する」の 2 種類のインタフェースを用意しています。

RX850 Pro におけるメールボックスの生成とは，メールボックスを管理するための領域（管理オブジェクト）をシステム・メモリ領域から確保したあと，初期化することです。

(1) スタティックに登録する場合

メールボックスをスタティックに登録する場合，コンフィギュレーション時に指定します。

RX850 Pro では，システム初期化処理時，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に定義された情報を基にメールボックスの生成処理を行い，管理対象とします。

(2) ダイナミックに登録する場合

メールボックスをダイナミックに登録する場合，処理プログラム（タスク）内から cre_mbx システム・コールを発行します。

RX850 Pro では，cre_mbx システム・コール発行時，パラメータで指定された情報を基にメールボックスの生成処理を行い，管理対象とします。

4.4.2　メールボックスの削除

メールボックスの削除は，del_mbx システム・コールを発行することにより行われます。

●del_mbx システム・コール

パラメータで指定されたメールボックスを削除します。

これにより，対象メールボックスは RX850 Pro の管理対象から除外されます。

ただし，このシステム・コールを発行した際，対象メールボックスのタスク用待ちキューにタスクがキューイングされていた場合には，該当タスクをタスク用待ちキューから外すとともに，wait 状態（メッセージ待ち状態）から ready 状態へと遷移させます。

なお，wait 状態を解除されたタスクには，wait 状態へと遷移するきっかけとなったシステム・コール（rcv_msg，trcv_msg）の戻り値として E_DLT が返されます。

保守／廃止

また，このシステム・コールを発行した際，対象メールボックスのメッセージ用待ちキューにメッセージがキューイングされていた場合は，該当メッセージをメッセージ用待ちキューから外すとともに，メッセージ用領域を獲得したメモリ・プールに返却します。このため，メモリ・プールから獲得したメモリ・ブロック以外の領域をメッセージ用領域として使用していた場合に，メールボックスを削除するような動作については，動作保証の対象となりません。このシステム・コールによって削除される可能性のあるメールボックスについては，必ずメモリ・プールから獲得したメモリ・ブロックをメッセージ用領域として使用してください。

4.4.3　メッセージの送信

メッセージの送信は，snd_msg システム・コールを発行することにより行われます。

●snd_msg システム・コール

パラメータで指定されたメールボックスにメッセージを送信します。

ただし，このシステム・コールを発行した際，対象メールボックスのタスク用待ちキューにタスクがキューイングされていた場合には，メッセージのキューイング操作は行わず，該当タスク（タスク用待ちキューの先頭タスク）にメッセージを渡します。

これにより，該当タスクは待ちキューから外れ，wait 状態（メッセージ待ち状態）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

対象メールボックスのタスク用待ちキューにタスクがキューイングされていない場合には，メッセージは対象メールボックスのメッセージ用待ちキューにキューイングされます。

備考　メッセージを対象メールボックスのメッセージ用待ちキューにキューイングする際は，対象メールボックス生成時（コンフィギュレーション時または cre_mbx システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

4.4.4　メッセージの受信

メッセージの受信は，rcv_msg，prcv_msg，または trcv_msg システム・コールを発行することにより行われます。

●rcv_msg システム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行した際，対象メールボックスからメッセージを受信できなかった（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューの最後尾にキューイングしたあと，run 状態から wait 状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態は次の場合に解除され，ready 状態へ遷移します。

- snd_msg システム・コールが発行された
- del_mbx システム・コールが発行され，対象メールボックスが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

備考　自タスクを対象メールボックスのタスク用待ちキューにキューイングする際は，対象メールボックス生成時（コンフィギュレーション時または cre_mbx システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

●prcv_msg システム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行した際，対象メールボックスからメッセージを受信できなかった（メッセージ用待ちキューにメッセージが存在しなかった）場合には，戻り値として E_TMOUT が返されます。

●trcv_msg システム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行した際，対象メールボックスからメッセージを受信できなかった（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューの最後尾にキューイングしたあと，run 状態から wait 状態（メッセージ待ち状態）へと遷移します。

なお，メッセージ待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された待ち時間が経過した
- snd_msg システム・コールが発行された
- del_mbx システム・コールが発行され，対象メールボックスが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

備考　自タスクを対象メールボックスのタスク用待ちキューにキューイングする際は，対象メールボックス生成時（コンフィギュレーション時または cre_mbx システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

4.4.5　メッセージ

RX850 Pro では，メールボックスを介してタスク間でやり取りされる情報を「メッセージ」と呼びます。

なお，メッセージは，メールボックスを介して任意のタスクに送ることができます。ただし，RX850 Pro におけるタスク間通信では，メッセージの先頭アドレスを受信側タスクに渡すだけであり，メッセージの内容がほかの領域にコピーされるわけではないので注意が必要です。

(1) メッセージ領域の確保

メッセージ用の領域は，get_blk，pget_blk，または tget_blk システム・コールを発行し，RX850 Pro が管理しているメモリ・プールから確保することを推奨しています。

また，メッセージの先頭 4 バイトは，メッセージ用待ちキューにキューイングする際のリンク領域として使用されます。

したがって，メッセージはメッセージ用領域の先頭から 4 バイト以降に格納してください。

(2) メッセージ内容の作成

RX850 Pro では，メールボックスに対して送信するメッセージの長さ，内容についての規定はありません。先頭 4 バイト以降のメッセージの長さ，内容については，メールボックスを使用して通信を行うタスク間で規定します。

注意　RX850 Proでは，メッセージの先頭４バイトをメッセージ用待ちキューにキューイングする際のリンク領域として使用します。したがって，メッセージを対象メールボックスに送信する場合は，snd_msgシステム・コールを発行する前にメッセージの先頭4バイトに「0x0」を設定してください。

なお，snd_msgシステム・コールを発行した際，メッセージの先頭4バイトに「0x0」以外の値が設定されていた場合には，RX850 Proは「対象メッセージはすでにメッセージ用待ちキューにキューイングされている」と判断し，メッセージの送信処理は行わず，戻り値としてE_OBJを返します。

(3) メッセージの優先度

RX850 Pro では，メッセージのキューイング方法として優先度順を指定できます。メッセージの優先度を指定する場合は，メッセージ用待ちキューにキューイングする際のリンク領域の 4 バイトとは別に 2 バイト使用します。したがって，メッセージはメッセージ用領域の先頭から 6 バイト以降に格納してください。なお，メッセージの優先度は 1～0x7fff の正の整数値で，値が小さいほど優先度が高くなります。

4.4.6　メールボックス情報の獲得

メールボックス情報の獲得は，ref_mbx システム・コールを発行することにより行います。

●ref_mbx システム・コール

パラメータで指定されたメールボックスのメールボックス情報（拡張情報，待ちタスクの有無など）を獲得します。

次に，メールボックス情報の詳細を示します。

- 拡張情報
- 待ちタスクの有無
- 待ちメッセージの有無
- キーID 番号

4.4.7　ID 番号の獲得

メールボックスの ID 番号の獲得は，vget_mid システム・コールを発行することにより行われます。

●vget_mid システム・コール

パラメータで指定されたメールボックスの ID 番号を獲得します。

システム・コールでメールボックスに対する操作をするとき，メールボックスの ID 番号が必要です。メールボックスを生成する際，ID 番号をユーザが一意に決定するか，自動的に振り当てるかを指定できます。

しかし，自動的に振り当てると指定した場合，ユーザはメールボックスの ID 番号を知ることができません。そのために必要となるものが「キーID 番号」です。キーID 番号はメールボックス生成の際に一意に指定します。

このキーID 番号をパラメータとして vget_mid システム・コールを発行すると，そのキーID 番号を持つメールボックスの ID 番号を取得できます。

保守／廃止

4.4.8　メールボックスによるタスク間通信

次に，メールボックスを使用してタスク間通信を行った場合の動作例を示します。

条　　件

●タスクの優先度

タスク A ＞ タスク B

●タスクの状態

タスク A：run 状態

タスク B：ready 状態

●メールボックスの属性

タスクのキューイング順序　　：FIFO 順

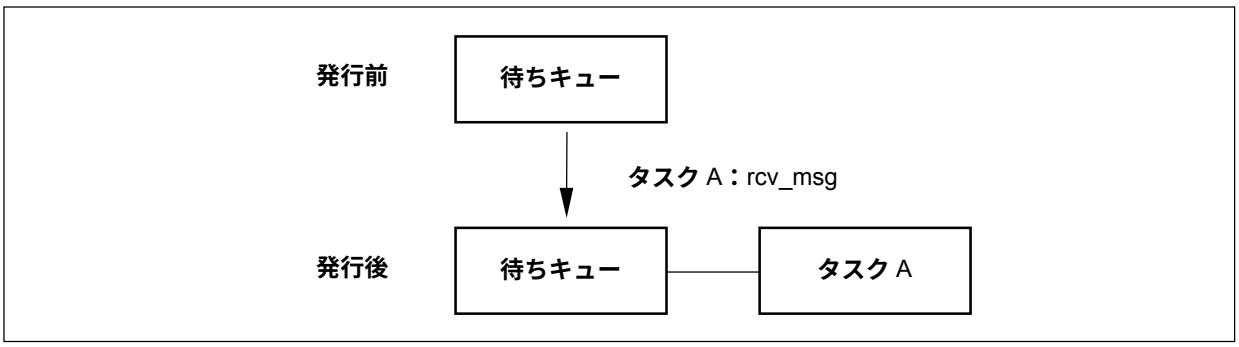
メッセージのキューイング順序　：FIFO 順

(1)　タスクAがrcv_msgシステム・コールを発行します。

現在，RX850 Pro が管理している対象メールボックスのメッセージ用待ちキューにはメッセージがキューイングされていません。したがって，RX850 Pro はタスク A を run 状態から wait 状態（メッセージ待ち状態）へと遷移させ，対象メールボックスのタスク用待ちキューの最後尾にキューイングします。

このとき，対象メールボックスのタスク用待ちキューは，図 4－8 のようになります。

図4－8　タスク用待ちキューの状態（rcv_msg発行時）



(2)　タスクAのメッセージ待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移します。

(3)　タスクBがget_blkシステム・コールを発行します。

これにより，メモリ・プールからメッセージ用の領域（メモリ・ブロック）が確保されます。

(4)　タスクBがメモリ・ブロックにメッセージを書き込みます。

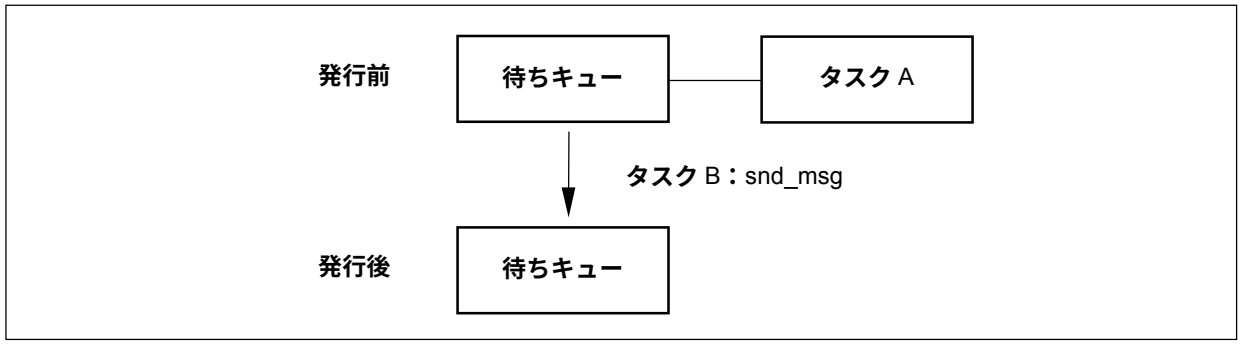
(5)　タスクBがsnd_msgシステム・コールを発行します。

これにより，対象メールボックスのタスク用待ちキューにキューイングされていたタスク A が，メッセージ待ち状態から ready 状態へと遷移します。

このとき，対象メールボックスのタスク用待ちキューは，図 4－9 のようになります。

保守／廃止

図4－9　タスク用待ちキューの状態（snd_msg発行時）



(6)　**優先度の高いタスク**A**が**ready**状態から**run**状態へと遷移します。**

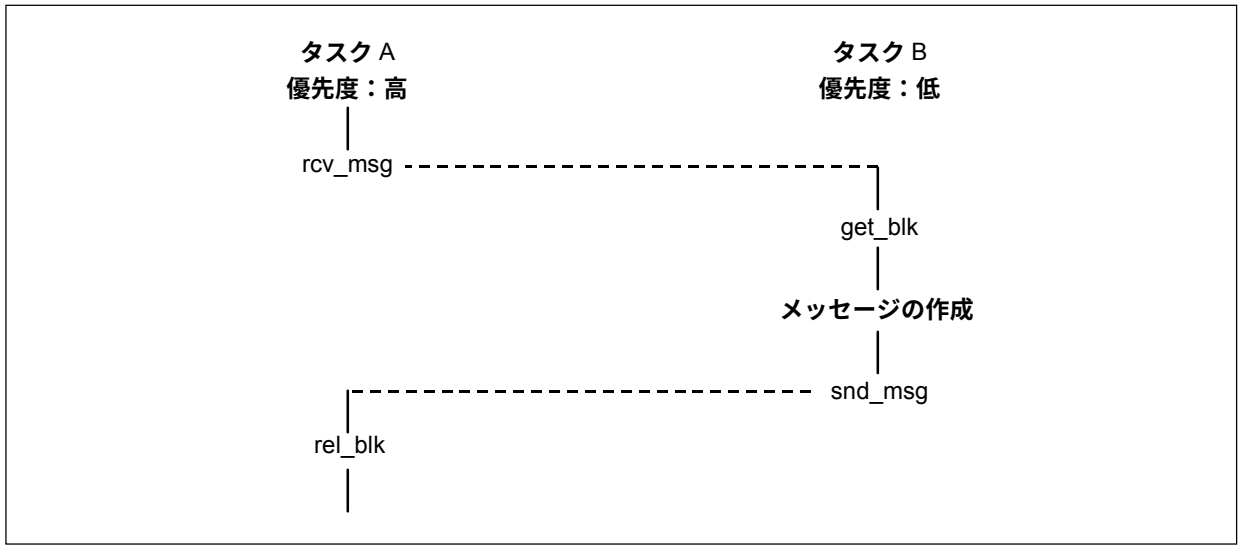
なお，タスク B **は** run **状態から** ready **状態へと遷移します。**

(7)　**タスク**A**が**rel_blk**システム・コールを発行します。**

これにより，メッセージ用の領域として確保されていたメモリ・ブロックがメモリ・プールへ解放されます。

図 4－10 **に，**（1）**から**（7）**で説明したタスク間通信の流れを示します。**

図4－10　メールボックスによるタスク間通信



第 5 章　割り込み管理機能

この章では，RX850 Pro が行う割り込み管理機能について説明します。

5.1　概　　要

RX850 Pro が行う割り込み管理では，次のような処理が行われます。

- ●割り込みハンドラの登録
- ●割り込みハンドラの起動
- ●割り込みハンドラからの復帰
- ●割り込み許可レベルの変更／獲得

5.2　割り込みハンドラ

割り込みハンドラは，割り込みが発生した際ただちに起動される割り込み処理専用ルーチンで，タスクとは独立したものとして扱われます。したがって，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，割り込みハンドラに制御が移ります。

なお，RX850 Pro では，その割り込みの発生から割り込みハンドラを起動するまでの応答性を考慮し，2 種類の割り込みハンドラ用インタフェースを提供しています。

●直接起動割り込みハンドラ

RX850 Pro を介在させることなく起動される割り込み処理専用ルーチンです。

●間接起動割り込みハンドラ

RX850 Pro による割り込み前処理（レジスタの退避処理，スタックの切り替え処理など）を行わせたのちに起動される割り込み処理専用ルーチンです。

また，RX850 Pro では，割り込みハンドラ内でシステム・コールが発行された際のスケジューリング処理に特異性を持たせています。

つまり，RX850 Pro では，割り込みハンドラ内でタスクのスケジューリング処理が必要となるシステム・コール（chg_pri，sig_sem など）が発行された際には，待ちキューの操作などといった処理を行うだけであり，実際のスケジューリング処理は，割り込みハンドラからの復帰処理（ret_int システム・コール，return 命令の発行など）まで遅延され，一括して行われます。

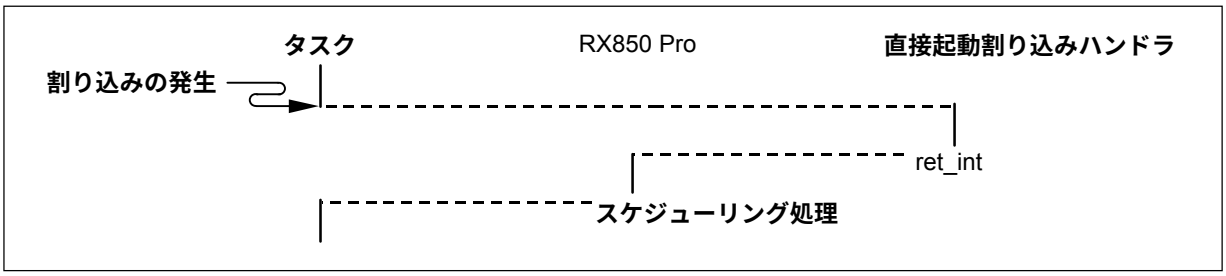
保守／廃止

5.3　直接起動割り込みハンドラ

直接起動割り込みハンドラは，割り込みが発生した際，RX850 Pro を介在させることなく起動される割り込み処理専用ルーチンです。このため，ハードウエアの限界に近い高速な応答性が期待されます。

図 5－1 に，割り込みハンドラの動作の流れを示します。

図5－1　直接起動割り込みハンドラの動作の流れ



5.3.1　直接起動割り込みハンドラの登録

直接起動割り込みハンドラの登録は，割り込みが発生した際にプロセッサが制御を移すハンドラ・アドレスに直接起動割り込みハンドラを割り付ける，または，直接起動割り込みハンドラへの分岐命令を設定することにより行われます。具体的な設定方法については， **A.5 直接起動割り込みハンドラ**の節を参照してください。

5.3.2　直接起動割り込みハンドラ内での処理

直接起動割り込みハンドラの処理を記述する際には，次の点に注意してください。

(1) レジスタの退避と復帰

直接起動割り込みハンドラに制御が移った際の作業用レジスタの内容は，割り込み発生時のものとなります。したがって，直接起動割り込みハンドラ内で作業用レジスタを使用する場合は，直接起動割り込みハンドラの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要があります。RX850 Pro では，アセンブリ言語での記述におけるユーザの負担を軽減するため，直接起動割り込みハンドラ用マクロを用意しています。記述の際はこのマクロを利用すると便利です。

(2) スタックの切り替え

直接起動割り込みハンドラに制御が移った際のスタックは，割り込み発生時のスタックとなります。したがって，割り込みハンドラ用スタックを使用する場合は，直接起動割り込みハンドラの開始部分で割り込みハンドラ用スタックへの切り替え処理を，終了部分で割り込み発生時のスタックへの切り替え処理を記述する必要があります。なお，直接起動割り込みハンドラ用マクロを利用して記述した場合，マクロの中でスタックを割り込みハンドラ用スタックへ切り替えています。

(3) システム・コールの発行制限

次に，直接起動割り込みハンドラ内で発行可能なシステム・コールの一覧を示します。

●タスク管理機能システム・コール

sta_tsk	chg_pri	rot_rdq	rel_wai	get_tid
ref_tsk	vget_tid			

保守／廃止

●タスク付属同期機能システム・コール

sus_tsk	rsm_tsk	frsm_tsk	wup_tsk	can_wup

●同期通信機能システム・コール

sig_sem	preq_sem	ref_sem	vget_sid	set_flg
clr_flg	pol_flg	ref_flg	vget_fid	snd_msg
prcv_msg	ref_mbx	vget_mid		

●割り込み管理機能システム・コール

def_int	ret_int	ret_wup	ena_int	dis_int
chg_icr	ref_icr			

●メモリ・プール管理機能システム・コール

pget_blk	rel_blk	ref_mpl	vget_pid

●時間管理機能システム・コール

set_tim	get_tim	def_cyc	act_cyc	ref_cyc

●システム管理機能システム・コール

get_ver	ref_sys	def_svc	viss_svc

(4) 直接起動割り込みハンドラからの復帰処理

直接起動割り込みハンドラからの復帰処理は，ハンドラの終了部分で ret_int，ret_wup システム・コールを発行することにより行われます。

●ret_int システム・コール

直接起動割り込みハンドラから復帰します。

●ret_wup システム・コール

パラメータで指定されたタスクに起床要求を発行したあと，直接起動割り込みハンドラから復帰します。

RX850 Pro では，直接起動割り込みハンドラ内でタスクのスケジューリング処理が必要となるシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューの操作などの処理が行われるだけであり，実際のスケジューリング処理は，直接起動割り込みハンドラからの復帰処理（ret_int，ret_wup システム・コールの発行）まで遅延され，一括して行われます。

注意 1. ret_int，ret_wup システム・コールでは，外部割り込みコントローラに対する処理終了通知（EOI コマンドの発行）は行いません。したがって，外部割り込み要求によって起動した直接起動割り込みハンドラから復帰する場合は，これらのシステム・コールを発行する前に，外部割り込みコントローラに対し処理終了を通知してください。

2. RX850 Pro では，直接起動割り込みハンドラを記述する際のマクロを用意しています。ハンドラ復帰時もマクロを使用した記述を行うことを推奨していますが，そのマクロの中では，必要なレジスタの復帰と ret_int，ret_wup システム・コールを発行しています。そのため，マクロを使用して直接起動割り込みハンドラを復帰する場合は，これらのシステム・コールを改めて使用する必要がありません。
   詳しい記述方法は，巻末の「付録 A プログラミングのために」を参照してください。
3. 直接起動割り込みハンドラが使用する GP（グローバル・ポインタ）と TP（テキスト・ポイ

保守／廃止

ンタ）の値が不定になります。そのため，直接起動割り込みハンドラの先頭（マクロ記述の後）で直接起動割り込みハンドラが使用する GP と TP の値をあらためてセットする必要があります。復帰処理は RX850 Pro で行っていますので，その記述は必要ありません。具体的な記述方法は，A.5 直接起動割り込みハンドラを参照してください。

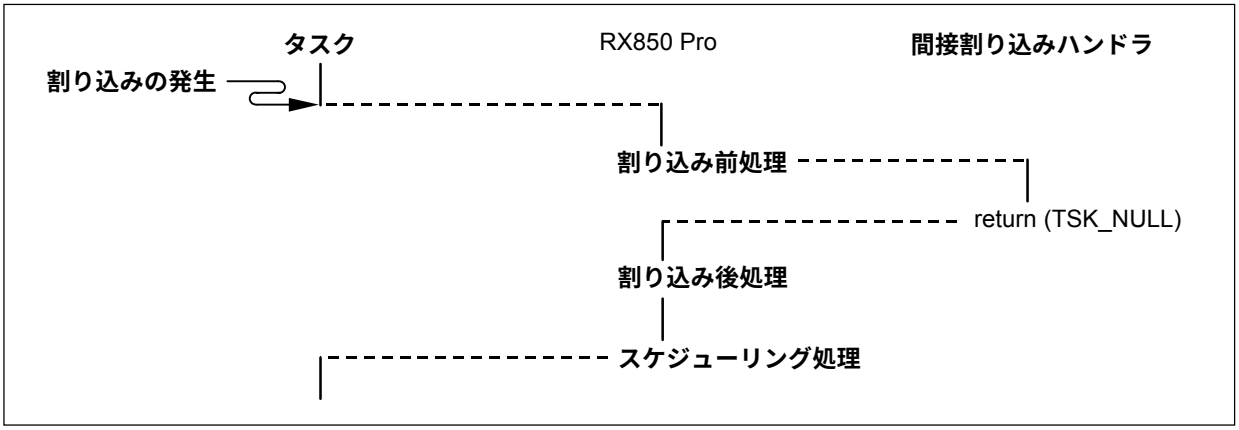
5.4　間接起動割り込みハンドラ

間接起動割り込みハンドラは，割り込みが発生した際，RX850 Pro による割り込み前処理（レジスタの退避処理，スタックの切り替え処理など）を行わせたあと起動される割り込み処理専用ルーチンです。

このため，直接起動割り込みハンドラに比べて応答性の面では多少劣りますが，RX850 Pro による割り込み前処理が行われているため，ハンドラ内での処理が簡素化されるといった利点を有しています。

図 5－2 に，間接起動割り込みハンドラの動作の流れを示します。

図5－2　間接起動割り込みハンドラの動作の流れ



5.4.1　間接起動割り込みハンドラの登録

RX850 Pro では，間接起動割り込みハンドラの登録において，「システム初期化処理（ニュークリアス初期化部）において，スタティックに登録する」，「処理プログラム内からシステム・コールを発行して，ダイナミックに登録する」の 2 種類のインタフェースを用意しています。

RX850 Pro における間接起動割り込みハンドラの登録とは，間接起動割り込みハンドラを管理するための領域（管理オブジェクト）をシステム・メモリから確保したあと，初期化することです。

(1) スタティックに登録する場合

間接起動割り込みハンドラをスタティックに登録する場合，コンフィギュレーション時に指定します。

RX850 Pro では，システム初期化処理時，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に定義された情報を基に間接起動割り込みハンドラの登録処理を行い，管理対象とします。

(2) ダイナミックに登録する場合

間接起動割り込みハンドラをダイナミックに登録する場合，処理プログラム（タスク，非タスク）内から def_int システム・コールを発行します。

RX850 Pro では，def_int システム・コール発行時，パラメータで指定された情報を基に間接起動割り込みハンドラの登録処理を行い，管理対象とします。

保守／廃止

5.4.2　間接起動割り込みハンドラ内での処理

間接起動割り込みハンドラの処理を記述する際には，次の点に注意してください。

(1) レジスタの退避と復帰

RX850 Pro では，間接起動割り込みハンドラに制御を移すときと，間接起動割り込みハンドラから復帰するときに C コンパイラ（CA850，または CCV850）の関数呼び出し規約に従った，作業レジスタの退避処理，復帰処理を行っています。したがって，間接起動割り込みハンドラの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要がありません。

(2) スタックの切り替え

RX850 Pro では，間接起動割り込みハンドラに制御を移すときと割り込みハンドラから復帰するときに，スタックの切り替え処理を行います。したがって，間接起動割り込みハンドラの開始部分で割り込みハンドラ用スタックへの切り替え処理を，終了部分で割り込み発生時のスタックへの切り替え処理を記述する必要はありません。

ただし，コンフィギュレーション時に割り込みハンドラ用スタックを定義していない場合は，スタックの切り替え処理は行われず，割り込み発生時のスタックが使用されます。

(3) システム・コールの発行制限

次に，間接起動割り込みハンドラ内で発行可能なシステム・コールの一覧を示します。

●タスク管理機能システム・コール

sta_tsk	chg_pri	rot_rdq	rel_wai	get_tid
rer_tsk	vget_tid			

●タスク付属同期機能システム・コール

sus_tsk	rsm_tsk	frsm_tsk	wup_tsk	can_wup

●同期通信機能システム・コール

sig_sem	preq_sem	ref_sem	vget_sid	set_flg
clr_flg	pol_flg	ref_flg	vget_fid	snd_msg
prcv_msg	ref_mbx	vget_mid		

●割り込み管理機能システム・コール

def_int	ena_int	dis_int	ref_icr	chg_icr

●メモリ・プール管理機能システム・コール

pget_blk	rel_blk	ref_mpl	vget_pid

●時間管理機能システム・コール

set_tim	get_tim	def_cyc	act_cyc	ref_cyc

●システム管理機能システム・コール

get_ver	ref_sys	def_svc	viss_svc

保守／廃止

(4) 間接起動割り込みハンドラからの復帰処理

間接起動割り込みハンドラからの復帰処理は，ハンドラの終了部分で return 命令を発行することにより行われます。

●return (TSK_NULL) 命令

間接起動割り込みハンドラから復帰します。

●return (ID *tskid*) 命令

パラメータで指定されたタスクに起床要求を発行したあと，間接起動割り込みハンドラから復帰します。

RX850 Pro では，間接起動割り込みハンドラ内でタスクのスケジューリング処理が必要となるシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューの操作などの処理が行われるだけであり，実際のスケジューリング処理は，間接起動割り込みハンドラからの復帰処理（return 命令の発行）まで遅延され，一括して行われます。

注意　return 命令では，外部割り込みコントローラに対する処理終了通知（EOI コマンドの発行）は行いません。したがって，外部割り込み要求によって起動した間接起動割り込みハンドラから復帰する場合は，これらのシステム・コールを発行する前に，外部割り込みコントローラに対し処理終了を通知してください。

5.5　マスカブル割り込みの受け付け禁止／再開

RX850 Pro では，ユーザの処理プログラムからマスカブル割り込みの受け付け状態を操作し，マスカブル割り込みの受け付けを禁止／再開する機能が提供されています。

この機能は，次に示すシステム・コールをタスクまたはハンドラ内から発行することにより使用できます。

●loc_cpu システム・コール

マスカブル割り込みの受け付けを禁止したあと，ディスパッチ処理（タスクのスケジューリング処理）を禁止します。

これにより，このシステム・コールが発行されてから unl_cpu システム・コールが発行されるまでの間は，ほかのタスクやハンドラに制御が移ることはありません。

●unl_cpu システム・コール

マスカブル割り込みの受け付けを許可したあと，ディスパッチ処理（タスクのスケジューリング処理）を再開します。

これにより，loc_cpu システム・コールの発行により禁止されていたマスカブル割り込みの受け付けが許可されるとともに，ディスパッチ処理も再開されます。

図 5－3 に割り込みをマスクしない場合（通常時）の制御の流れを，図 5－4 に loc_cpu システム・コールを発行した場合の制御の流れを示します。

保守／廃止

図5－3　割り込みのマスク処理をしない場合（通常時）の制御の流れ

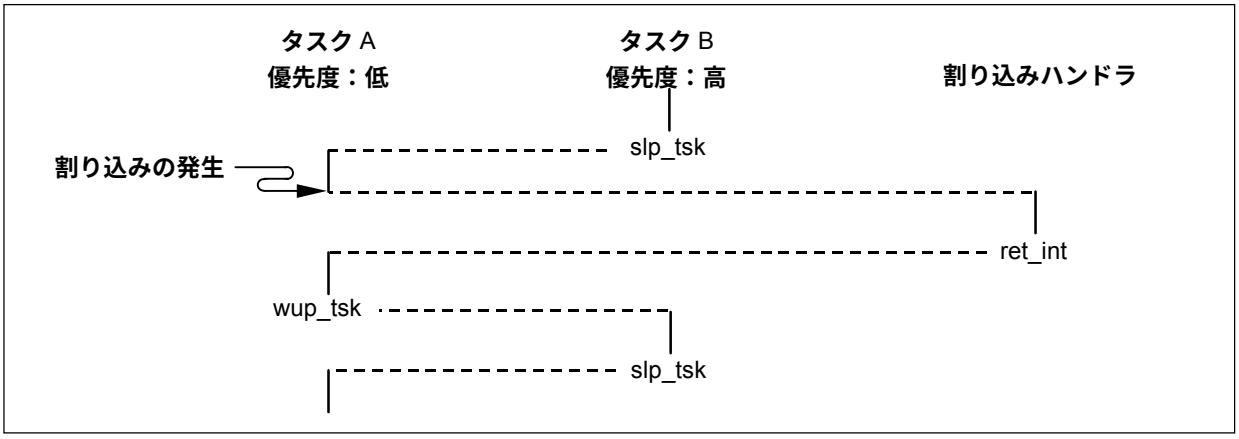
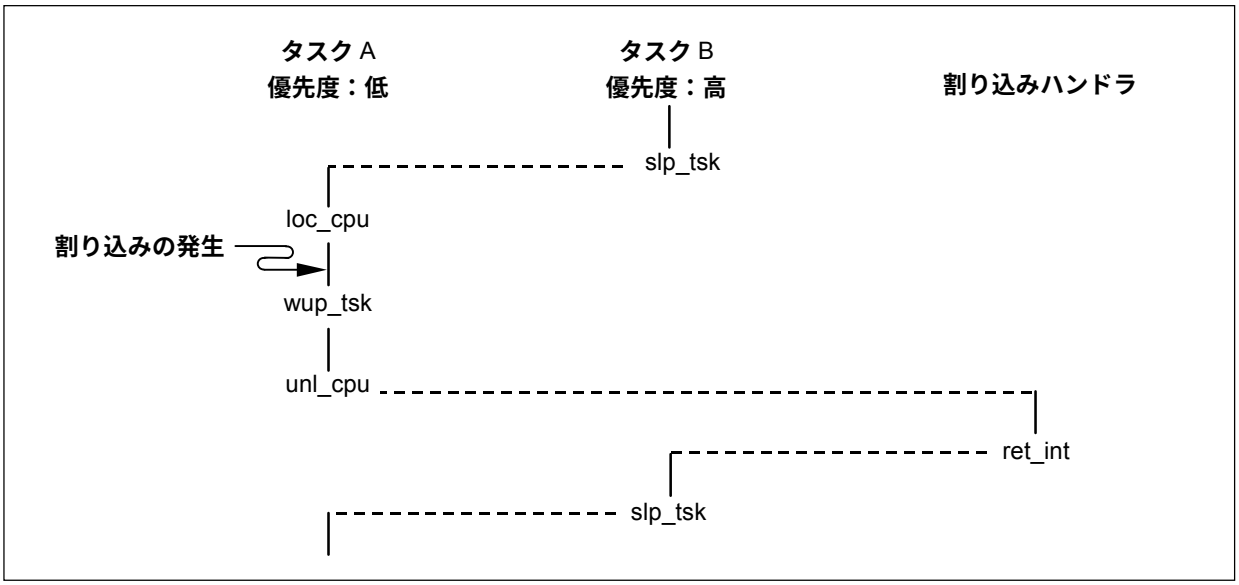


図5－4　loc_cpuシステム・コールを発行した場合の制御の流れ



5.6　割り込み制御レジスタの変更／獲得

割り込み制御レジスタの変更／獲得は，chg_icr または ref_icr システム・コールを発行することにより行われます。

●chg_icr システム・コール

パラメータで指定された割り込み制御レジスタの内容を変更します。

●ref_icr システム・コール

パラメータで指定された割り込み制御レジスタの内容を獲得します。

注意　V850E コア上で動作させている場合，割り込み制御レジスタ関連のシステム・コールである chg_icr および ref_icr を発行すると，期待した割り込み制御レジスタが操作されない場合があります。RX850 Pro では，割り込み要因番号から割り込み制御レジスタのアドレスを算出しています。しかし，V850E コアの場合，割り込み要因番号と割り込み制御レジスタの並びが，他の V850 シリーズと異なってい

保守／廃止

るため，正しいレジスタ・アドレスが得られなくなっています。そのため，V850E コアにおける chg_icr および ref_icr システム・コールの使用を制限します。割り込み制御レジスタをアプリケーション上から操作したい場合は，これらのシステム・コールは使わずに，直接レジスタを操作してください。

5.7　ノンマスカブル割り込み

ノンマスカブル割り込みは，割り込み優先順位の対象外となっており，すべての割り込みに優先して受け付けられます。また，プロセッサを割り込み禁止（PSW の ID フラグをセット）状態にしても受け付けられます。したがって，RX850 Pro の処理中や割り込みハンドラの処理中であっても，ノンマスカブル割り込みは受け付けられます。

RX850 Pro では，ノンマスカブル割り込みに対応した割り込みハンドラ内でシステム・コールを発行した場合，その動作は保証していません。

5.8　クロック割り込み

RX850 Pro では，ハードウエア（クロック・コントローラなど）により一定周期で発生するクロック割り込みを利用して時間管理を行っています。

つまり，クロック割り込みが発生した場合には，RX850 Pro のシステム・クロック処理が呼び出され，タスクの時間経過待ち，周期起動ハンドラの起動などといった時間に関連した処理が行われます。

なお，時間管理についての詳細は，**第 7 章 時間管理機能**を参照してください。

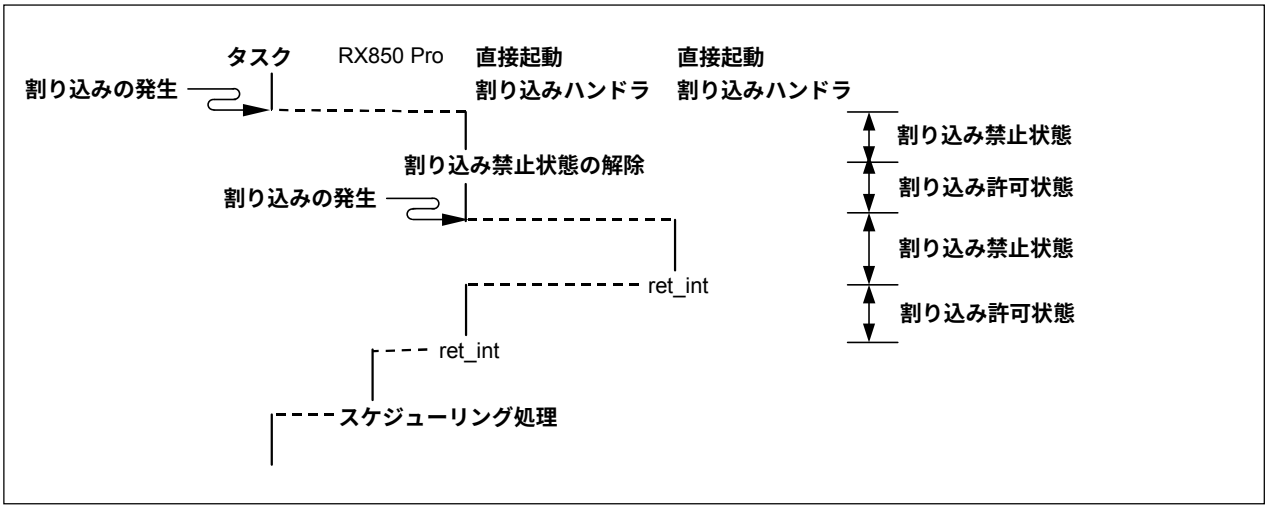
5.9　多重割り込み

割り込みハンドラ内でさらに割り込みが発生することを「多重割り込み」と呼んでいます。RX850 Pro は多重割り込みにも対応します。

ただし，割り込みハンドラは，割り込み禁止（PSW の ID フラグをセット）状態で処理を開始するため，多重割り込みを受け付けるには，割り込みハンドラ内で割り込み禁止状態の解除処理を記述する必要があります。

図 5－5 に，多重割り込み発生時の動作の流れを示します。

図5－5　多重割り込み発生時の動作の流れ



保守／廃止

第 6 章　メモリ・プール管理機能

この章では，RX850 Pro が行うメモリ・プール管理機能について説明します。

6.1　概　　要

RX850 Pro では，システムを管理するための情報テーブル，各種機能を実現するための管理ブロックを配置するためのメモリ領域，そしてメモリ・プールを使用するためのメモリ領域が必要となります。

これらを配置するメモリ領域には，次にあげる 4 種類があります。

- System Memory Pool 0　　（キーワード：SPOL0）
- System Memory Pool 1　　（キーワード：SPOL1）
- User Memory Pool 0　　（キーワード：UPOL0）
- User Memory Pool 1　　（キーワード：UPOL1）

これらのメモリ領域には「各種資源管理テーブル」，「タスク・スタック」，「割り込みハンドラ・スタック」，「メモリ・プール用のメモリ」の 4 種類が配置されます。また，配置できる領域の組み合わせは次のとおりです。

表 6－1　メモリ情報の配置組み合わせ

各種資源管理テーブル	タスク・スタック	割り込みスタック	メモリ・プール
SPOL0	SPOL0 または SPOL1	SPOL0 または SPOL1	UPOL0 または UPOL1

それぞれのメモリ領域の先頭アドレスとサイズは，コンフィギュレーション・ファイルにて設定します。SPOL0 は必ず作成する必要があります。SPOL1 はタスク・スタック，割り込みスタックを SPOL0 以外に配置したい場合に作成します。UPOL0 と UPOL1 はメモリ・プール管理機能を使用したい場合に作成します。なお，UPOL1 は UPOL0 を作成している場合に作成可能です。

6.2　管理オブジェクト

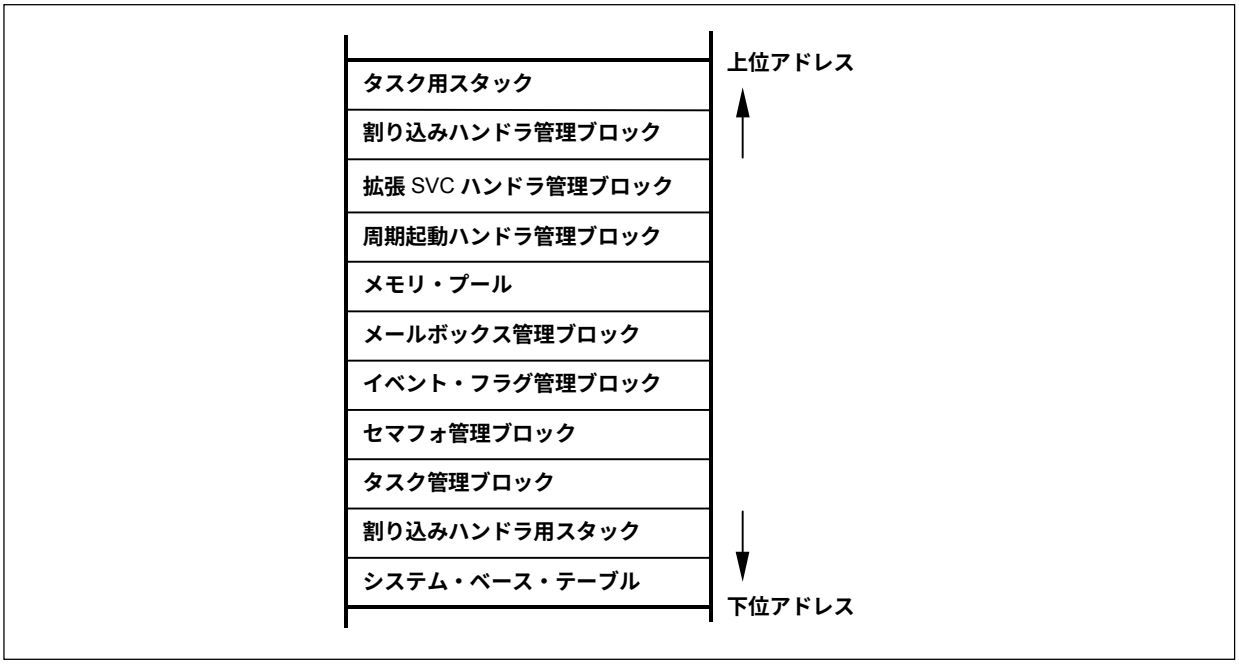
RX850 Pro が提供する機能を実現するうえで必要となる管理オブジェクトを示します。これらの管理オブジェクトは，コンフィギュレーション時に指定した情報をもとに，システム初期化処理時に生成／初期化されます。また，これらの管理オブジェクトは SPOL0（タスク用スタックと割り込みハンドラ用スタックは SPOL1 でも可）に配置されます。

●システム・ベース・テーブル
●タスク管理ブロック
●セマフォ管理ブロック
●イベント・フラグ管理ブロック

●メールボックス管理ブロック
●メモリ・プール管理ブロック
●メモリ・ブロック管理ブロック
●周期起動ハンドラ管理ブロック
●拡張 SVC ハンドラ管理ブロック
●メモリ・プール
●タスク用スタック
●割り込みハンドラ用スタック
●割り込みハンドラ管理ブロック

図 6－1 に，管理オブジェクトの配置例を示します。

図6－1　管理オブジェクトの配置例



6.3　メモリ・プールとメモリ・ブロック

RX850 Pro では，アプリケーション中でメモリ領域を獲得，解放するダイナミックなメモリ・プール管理機能を行っています。作業用のメモリ領域が必要となったときに獲得し，不要になったときに解放できるといった機能が用意されています。この機能により，限りあるメモリ領域を効率良く使用できます。

メモリ・プールとして使用できるメモリ領域は，UPOL0 または UPOL1 です。コンフィギュレーション時にメモリ・プールを定義するとき，またはシステム・コール（cre_mpl）を発行してメモリ・プールを作成するときに，UPOL0，UPOL1 のどちらの領域を使うものかを指定します。

RX850 Pro が提供しているメモリ・プールは可変長メモリ・プールです。固定長メモリ・プールは搭載していません。

メモリ・プールは複数のメモリ・ブロックから構成されており，メモリ・プールに対する操作はメモリ・ブロックを単位として行います。

メモリ・プールに対するダイナミックな操作は，次に示すメモリ・プール関連のシステム・コールを使用しま

す。

cre_mpl　：メモリ・プールを生成する
del_mpl　：メモリ・プールを削除する
get_blk　：メモリ・ブロックを獲得する
pget_blk　：メモリ・ブロックを獲得する（ポーリング）
tget_blk　：メモリ・ブロックを獲得する（タイムアウトあり）
rel_blk　：メモリ・ブロックを解放する
ref_mpl　：メモリ・プール情報を獲得する
vget_pid　：メモリ・プールの ID 情報を獲得する

6.3.1　メモリ・プールの生成

RX850 Pro では，メモリ・プールの生成において，「システム初期化処理（ニュークリアス初期化部）において，スタティックに生成する」，「処理プログラム内からシステム・コールを発行し，ダイナミックに生成する」の 2 種類のインタフェースを用意しています。

RX850 Pro におけるメモリ・プールの生成とは，メモリ・プールを管理するための領域（管理オブジェクト）とメモリ・プールの実体を，システム・メモリ領域から確保したあと，初期化することです。

(1) スタティックに登録する場合

メモリ・プールをスタティックに登録する場合，コンフィギュレーション時に指定します。

RX850 Pro では，システム初期化処理時，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に定義された情報を基にメモリ・プールの生成処理を行い，管理対象とします。

(2) ダイナミックに登録する場合

メモリ・プールをダイナミックに登録する場合，処理プログラム（タスク）内から cre_mpl システム・コールを発行します。

RX850 Pro では，cre_mpl システム・コール発行時，パラメータで指定された情報を基にメモリ・プールの生成処理を行い，管理対象とします。

備考　メモリ・プールを生成すると，指定サイズ以外にRX850 Proがメモリ・プール管理領域として，メモリ・プールの先頭から8バイトを使用します。したがって，生成したメモリ・プールのサイズは「指定サイズ＋ 8バイト」となります。

6.3.2　メモリ・プールの削除

メモリ・プールの削除は，del_mpl システム・コールを発行することにより行われます。

●del_mpl システム・コール

パラメータで指定されたメモリ・プールを削除します。

これにより，対象メモリ・プールは，RX850 Pro の管理対象から除外されます。

ただし，このシステム・コールを発行した際，対象メモリ・プールの待ちキューにタスクがキューイングされていた場合には，該当タスクを待ちキューから外すとともに，wait 状態（メモリ・ブロック待ち状態）から ready 状態へと遷移させます。

なお，wait 状態を解除されたタスクには，wait 状態へと遷移するきっかけとなったシステム・コール

保守／廃止

（get_blk，tget_blk）の戻り値として E_DLT が返されます。

また，このシステム・コールを発行すると，対象メモリ・プールが管理しているメモリ・ブロックも RX850 Pro の管理対象から除外されます。このシステム・コールを発行する前にすでにタスクが対象メモリ・プールからメモリ・ブロックを獲得していた場合は，そのメモリ・ブロックは動作保証外となるので，メモリ・プールの削除については注意が必要です。

6.3.3　メモリ・ブロックの獲得

メモリ・ブロックの獲得は，get_blk，pget_blk，または tget_blk システム・コールを発行することにより行います。

注意　RX850 Proでは，メモリ・ブロックを獲得するときにメモリ・クリアを行いません。したがって，獲得したメモリ・ブロックの内容は不定です。

メモリ・ブロックを獲得すると，要求サイズ以外に RX850 Pro がメモリ管理領域としてメモリ・プールから 8 バイト使用します。また，RX850 Pro は要求サイズを自動的に 4 バイトでアラインするので，残りの空きメモリ・ブロック・サイズには注意してください。

次に，獲得したメモリ・ブロックのサイズを求める式を示します。

メモリ・ブロックのサイズ（blk_siz）= align 4（ユーザ要求サイズ）＋ 8

●get_blk システム・コール

パラメータで指定されたメモリ・プールからメモリ・ブロックを獲得します。

ただし，このシステム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった（要求した空き領域が存在しなかった）場合には，自タスクを対象メモリ・プールの待ちキューにキューイングしたあと，run 状態から wait 状態（メモリ・ブロック待ち状態）へと遷移させます。

なお，メモリ・ブロック待ち状態は次の場合に解除され，ready 状態へと遷移します。

- rel_blk システム・コールが発行され，要求サイズのメモリ・ブロックが返却された
- del_mpl システム・コールが発行され，対象メモリ・プールが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

備考　自タスクを対象メモリ・プールの待ちキューにキューイングする際は，対象メモリ・プール生成時（コンフィギュレーション時またはcre_mplシステム・コール発行時）に指定された順（FIFO順または優先度順）に行われます。

●pget_blk システム・コール

パラメータで指定されたメモリ・プールからメモリ・ブロックを獲得します。

ただし，このシステム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった（要求したサイズの空き領域が存在しなかった）場合には，戻り値として E_TMOUT が返されます。

●tget_blk システム・コール

パラメータで指定されたメモリ・プールからメモリ・ブロックを獲得します。

ただし，このシステム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった（要求したサイズの空き領域が存在しなかった）場合には，自タスクを対象メモリ・プールの待ち

保守／廃止

キューにキューイングしたあと，run 状態から wait 状態（メモリ・ブロック待ち状態）へと遷移させます。
なお，メモリ・ブロック待ち状態は次の場合に解除され，ready 状態へと遷移します。

- パラメータで指定された待ち時間が経過した
- rel_blk システム・コールが発行され，要求サイズのメモリ・ブロックが返却された
- del_mpl システム・コールが発行され，対象メモリ・プールが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

備考　自タスクを対象メモリ・プールの待ちキューにキューイングする際は，対象メモリ・プール生成時（コンフィギュレーション時またはcre_mplシステム・コール発行時）に指定された順（FIFO順または優先度順）に行われます。

6.3.4　メモリ・ブロックの返却

メモリ・ブロックの返却は，rel_blk システム・コールを発行することにより行われます。

●rel_blk システム・コール

パラメータで指定されたメモリ・プールにメモリ・ブロックを返却します。

ただし，このシステム・コールを発行した際，対象メモリ・プールの待ちキューにキューイングされているタスク（待ちキューの先頭タスク）が要求したサイズを満足するようなメモリ・ブロックであった場合には，該当タスクにメモリ・ブロックを渡します。

これにより，該当タスクは待ちキューから外れ，wait 状態（メモリ・ブロック待ち状態）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

注意 1. RX850 Proでは，メモリ・ブロックを返却するときにメモリ・クリアを行いません。したがって，返却したメモリ・ブロックの内容は不定です。

★ **2. RX850 Proには2つの仕様のrel_blkシステム・コールがあります。**

（1）rel_blkシステム・コールでメモリ・ブロックを返却する際，メモリ・ブロックの先頭4バイトが0でなかった場合に，メモリ・ブロックを返却せずに戻り値E_OBJで終了する。

（2）rel_blkシステム・コールを発行すると，メモリ・ブロックの先頭4バイトが0でなくてもメモリ・ブロックを返却する（戻り値E_OK）。

（1）はメモリ・ブロックがメールボックスのメッセージ領域として使用されていた場合を考慮したもので，これまでのRX850 Proに搭載されていたrel_blkと同じ仕様のものです。
メモリ・ブロックがメールボックスのメッセージ領域として使用されていた場合，先頭4バイトがメッセージの待ちキューのリンク領域になります。つまりメッセージが，メールボックスにキューイングされているときにrel_blkシステム・コールを発行し，必ずメモリ・ブロックが返却する仕様にすると，キューにつながれていたメッセージ領域が返却されてしまうことになります。これを防ぐため，リンク領域である先頭4バイトが0でなければ，メッセージ領域として使用されているメモリ・ブロックと判断し，メモリ・ブロックを返却せずに戻り値E_OBJで終了します。この仕様のrel_blkを使用してメモリ・ブロックを返却するときは，必ず先頭4バイトを0クリアする必要があります。

保守／廃止

これらの仕様のrel_blkは，別々のライブラリに搭載されていますので，どちらか一方のrel_blkを使用することになります。使用する方のライブラリをリンクしてください。

（1）メモリ・ブロックの先頭4バイトを0でクリアする必要のあるrel_blkが搭載されたライブラリ　→　librxp.a

（2）メモリ・ブロックの先頭4バイトを0でクリアしなくてもよいrel_blkが搭載されたライブラリ　→　librxpm.a

3. メモリ・ブロックを返却するメモリ・プールは，get_blk，pget_blk，tget_blkシステム・コールを発行した際に指定したメモリ・プールと同じにしてください。

6.3.5　メモリ・プール情報の獲得

メモリ・プール情報の獲得は，ref_mpl システム・コールを発行することにより行われます。

●ref_mpl システム・コール

パラメータで指定されたメモリ・プールのメモリ・プール情報（拡張情報，待ちタスクの有無など）を獲得します。

次に，メモリ・プール情報の詳細を示します。

・拡張情報
・待ちタスクの有無
・空き領域の合計サイズ
・獲得可能な最大メモリ・ブロック・サイズ
・キーID 番号

6.3.6　ID 番号の獲得

メモリ・プールの ID 番号の獲得は，vget_pid システム・コールを発行することにより行われます。

●vget_pid システム・コール

パラメータで指定されたメモリ・プールの ID 番号を獲得します。

システム・コールでメモリ・プールに対する操作をするとき，メモリ・プールの ID 番号が必要です。メモリ・プールを生成する際，ID 番号をユーザが一意に決定するか，自動的に振り当てるかを指定できます。

しかし，自動的に振り当てると指定した場合，ユーザはメモリ・プールの ID 番号を知ることができません。そのために必要となるものが「キーID 番号」です。キーID 番号はメモリ・プール生成の際に一意に指定します。

このキーID 番号をパラメータとして vget_pid システム・コールを発行すると，そのキーID 番号を持つメモリ・プールの ID 番号を取得できます。

保守／廃止

6.3.7　メモリ・プールによるメモリ・ブロックの動的管理

次に，メモリ・プールを使用してタスクでのメモリの動的使用を行った場合の動作例を示します。

条　　件

●タスクの優先度

タスク A ＞ タスク B

●タスクの状態

タスク A：run 状態

タスク B：ready 状態

●メモリ・プールの属性

空きメモリ・ブロック・サイズ ：0x20

タスクのキューイング順序　　：FIFO 順

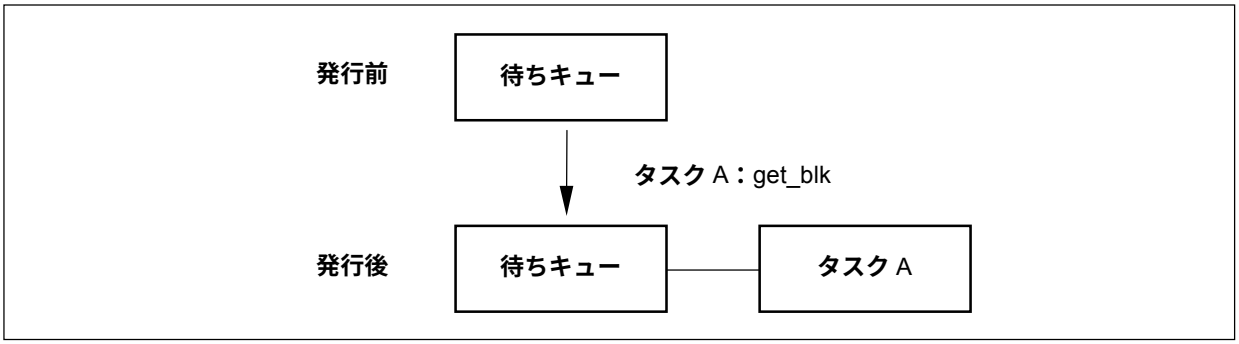
(1)　タスクAがget_blkシステム・コールを発行します。

要求メモリ・ブロック・サイズは「0x30」とします。

現在，RX850 Pro が管理している対象メモリ・プールの空きメモリ・ブロック・サイズは「0x20」です。したがって，RX850 Pro はタスク A を run 状態から wait 状態（メモリ・ブロック待ち状態）へと遷移させ，対象メモリ・プールの待ちキューの最後尾にキューイングします。

このとき，対象メモリ・プールの待ちキューは，図 6－2 のようになります。

図6－2　待ちキューの状態（get_blk発行時）



(2)　タスクAのメモリ・プール待ち状態への遷移にともない，タスクBがready状態からrun状態へと遷移します。

保守／廃止

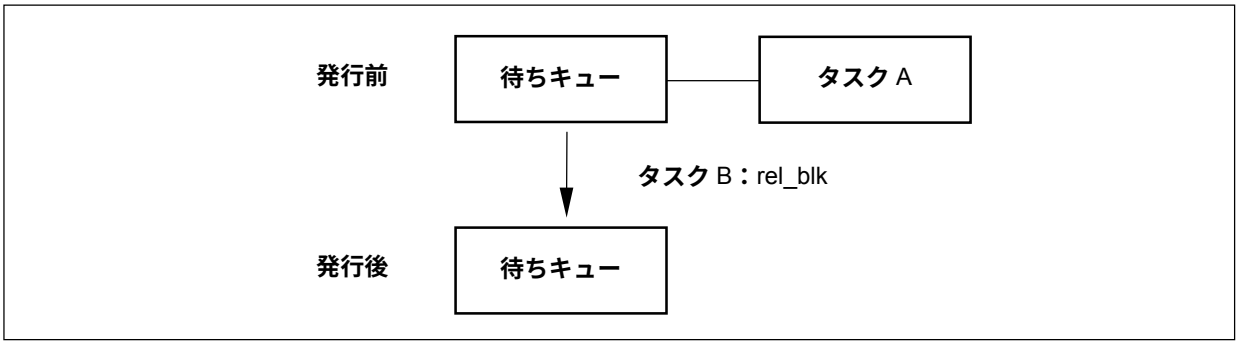
(3)　タスクBがrel_blkシステム・コールを発行します。

返却メモリ・ブロック・サイズは，「0x16」とします。

これにより，対象メモリ・プールの待ちキューにキューイングされているタスク A の要求メモリ・ブロック・サイズを満たしたため，タスク A がメモリ・ブロック待ちから ready 状態へと遷移します。

このとき，対象メモリ・プールの待ちキューは，図 6－3 のようになります。

図6－3　待ちキューの状態（rel_blk発行時）

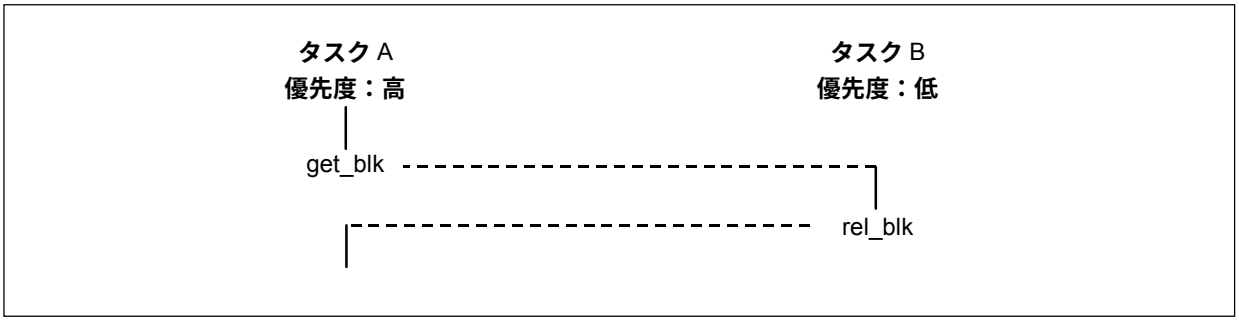


(4)　優先度の高いタスクAがready状態からrun状態へと遷移します。

なお，タスク B は run 状態から ready 状態へと遷移します。

図 6－4 に，(1) から (4) で説明したメモリ・プールによるメモリの動的使用の流れを示します。

図6－4　メモリ・プールによるメモリの動的使用



第 7 章　時間管理機能

この章では，RX850 Pro が行う時間管理機能について説明します。

7.1　概　　要

RX850 Pro における時間管理は，ハードウエア（クロック・コントローラなど）により一定周期で発生するクロック割り込みを利用して行います。

クロック割り込みが発生すると，RX850 Pro のシステム・クロック処理が呼び出され，システム・クロックの更新やタスクの遅延起床，周期起動ハンドラの起動などといった，時間に関連した処理が行われます。

7.2　システム・クロック

システム・クロックは，RX850 Pro が時間管理の際に使用する時刻（48 ビット幅，単位：ミリ秒）を保持したソフトウエア・タイマです。

システム・クロックは，システム初期化処理で「0x0」に設定されたあと，システム・クロック処理で基本クロック周期（コンフィギュレーション時に指定）を単位として更新されます。

注意　RX850 Proが管理するシステム・クロックは，48ビット幅で構成されています。このため，RX850 Proでは，桁あふれした数値（48ビットでは表せない数値）は無視されます。

7.2.1　システム・クロックの設定と読み出し

システム・クロックの設定は set_tim システム・コールを，読み出しは get_tim システム・コールを発行することにより行います。

●set_tim システム・コール

システム・クロックにパラメータで指定された時刻を設定します。

●get_tim システム・コール

システム・クロックの現時刻をパラメータで指定されるパケットに格納します。

7.3　タイマ・オペレーション

リアルタイム処理では，あるタスクの処理を一定時間だけ中断させたり，あるハンドラの処理を一定周期で実行させたりといった，時間と同期した機能（タイマ・オペレーション機能）が必要となります。そこで，RX850 Proでは，タイマ・オペレーション機能として，タスクの遅延起床，タイムアウト，周期起動ハンドラの起動といった機能を提供しています。

保守／廃止

7.4　タスクの遅延起床

タスクの遅延起床とは，一定の時間が経過するまでの間，タスクを run 状態から wait 状態（時間経過待ち状態）へと遷移させ，時間が経過した時点で wait 状態を解除し，ready 状態へ遷移させるものです。

タスクの遅延起床は，dly_tsk システム・コールを発行することにより行います。

●dly_tsk システム・コール

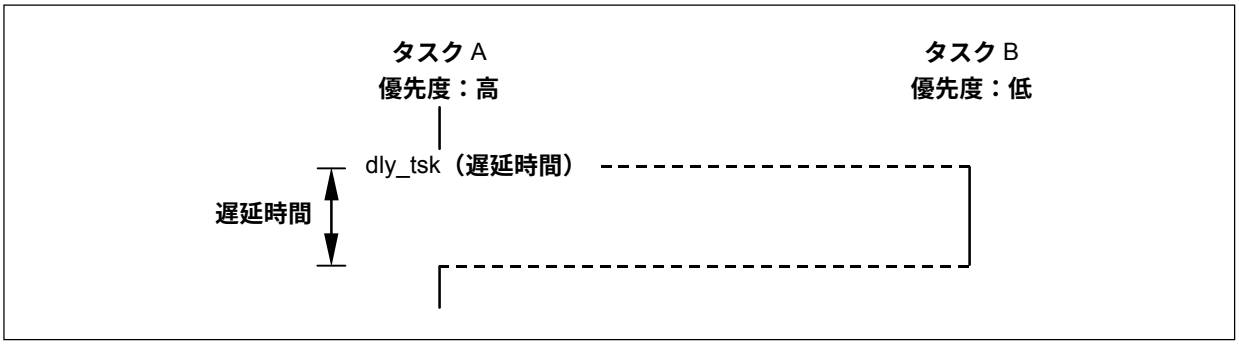
パラメータで指定された遅延時間だけ，自タスクを run 状態から wait 状態（時間経過待ち状態）へと遷移させます。

なお，時間経過待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された遅延時間が経過した
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

図 7－1 に，dly_tsk システム・コールを発行した際の処理の流れを示します。

図7－1　dly_tskシステム・コール発行時の処理の流れ



7.5　タイムアウト

タイムアウトは，要求する条件が即時に成立しなかった場合，一定の時間が経過するまでの間，タスクを run 状態から wait 状態（起床待ち状態，資源待ち状態など）へと遷移させ，時間が経過したときには wait 状態を解除し，ready 状態へと遷移させるものです。

タイムアウトは，tslp_tsk，twai_sem，twai_flg，trcv_msg，tget_blk システム・コールを発行することにより行われます。

保守／廃止

●tslp_tsk システム・コール

自タスクに発行されている起床要求を 1 回分だけ解除（起床要求カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，自タスクの起床要求カウンタが 0x0 であった場合には，起床要求の解除（起床要求カウンタの減算）は行わず，自タスクを run 状態から wait 状態（起床待ち状態）へと遷移させます。

なお，起床待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された時間が経過した
- wup_tsk システム・コールが発行された
- ret_wup システム・コールが発行された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

●twai_sem システム・コール

パラメータで指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，対象セマフォから資源を獲得できなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューにキューイングしたあと，run 状態から wait 状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された待ち時間が経過した
- sig_sem システム・コールが発行された
- del_sem システム・コールが発行され，対象セマフォが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

●twai_flg システム・コール

パラメータで指定されたイベント・フラグに，要求する待ち条件を満足するビット・パターンがセットされているかどうかをチェックします。

ただし，このシステム・コールを発行した際，対象イベント・フラグのビット・パターンが待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューの最後尾にキューイングしたあと，run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された待ち時間が経過した
- set_flg システム・コールが発行され，要求する待ち条件がセットされた
- del_flg システム・コールが発行され，対象イベント・フラグが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

●trcv_msg システム・コール

パラメータで指定されたメールボックスからメッセージを受信します。

ただし，このシステム・コールを発行した際，対象メールボックスからメッセージを受信できなかった（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューにキューイングしたあと，run 状態から wait 状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された待ち時間が経過した
- snd_msg システム・コールが発行された
- del_mbx システム・コールが発行され，対象メールボックスが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

●tget_blk システム・コール

パラメータで指定されたメモリ・プールからメモリ・ブロックを獲得します。

ただし，このシステム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった（要求したサイズの空き領域が存在しなかった）場合には，自タスクを対象メモリ・プールの待ちキューにキューイングしたあと，run 状態から wait 状態（メモリ・ブロック待ち状態）へと遷移させます。

なお，メモリ・ブロック待ち状態は次の場合に解除され，ready 状態へ遷移します。

- パラメータで指定された待ち時間が経過した
- rel_blk システム・コールが発行され，要求するサイズのメモリ・ブロックが返却された
- del_mpl システム・コールが発行され，対象メモリ・プールが削除された
- rel_wai システム・コールが発行され，強制的に待ち状態が解除された

7.6　周期起動ハンドラ

周期起動ハンドラは，一定の起動時間に達した際，ただちに起動される周期処理専用ルーチンで，ユーザが記述する周期的な処理プログラムの中で，実行開始までのオーバヘッドが最も小さな処理プログラムです。

周期起動ハンドラはタスクとは独立したものとして扱われます。このため，ハンドラの起動時間に達したときには，システム内で最高優先度を持つタスクが実行中であっても，その処理が中断されて周期起動ハンドラに制御が移ります。

周期起動ハンドラに対するダイナミックな操作には，次に示す周期起動ハンドラ関連のシステム・コールや命令を使用します。

def_cyc：周期起動ハンドラを登録する
act_cyc：周期起動ハンドラの活性状態を制御する
ref_cyc：周期起動ハンドラ情報を獲得する
return　：周期起動ハンドラから復帰する

保守／廃止

7.6.1　周期起動ハンドラの登録

RX850 Pro では，周期起動ハンドラの登録において，「システム初期化処理（ニュークリアス初期化部）において，スタティックに登録する」，「処理プログラム内からシステム・コールを発行し，ダイナミックに登録する」の 2 種類のインタフェースを用意しています。

RX850 Pro における周期起動ハンドラの登録とは，周期起動ハンドラを管理するための領域（管理オブジェクト）をシステム・メモリ領域から確保したあと，初期化することです。

（1）スタティックに登録する場合

周期起動ハンドラをスタティックに登録する場合，コンフィギュレーション時に指定します。

RX850 Pro では，システム初期化処理時，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に定義された情報を基に周期起動ハンドラの登録処理を行い，管理対象とします。

（2）ダイナミックに登録する場合

周期起動ハンドラをダイナミックに登録する場合，処理プログラム（タスク，非タスク）内から def_cyc システム・コールを発行します。

RX850 Pro では，def_cyc システム・コール発行時，パラメータで指定された情報を基に周期起動ハンドラの登録処理を行い，管理対象とします。

7.6.2　周期起動ハンドラの活性状態

周期起動ハンドラの活性状態は，RX850 Pro が周期起動ハンドラを起動するかどうかを判定する際の基準の 1 つです。

活性状態は，周期起動ハンドラ登録時（コンフィギュレーション時または def_cyc システム・コール発行時）に設定されますが，RX850 Pro では，この活性状態をユーザの処理プログラムからダイナミックに変更する機能を提供しています。

●act_cyc システム・コール

周期起動ハンドラの活性状態をパラメータで指定された状態に変更します。

TCY_OFF：周期起動ハンドラの活性状態を OFF 状態に変更する
TCY_ON　：周期起動ハンドラの活性状態を ON 状態に変更する
TCY_INI　：周期起動ハンドラの周期カウンタを初期化する

RX850 Pro では，周期起床ハンドラの活性状態が OFF 状態であっても，周期カウンタのカウント処理を行っています。このため，act_cyc システム・コールを発行し，活性状態を OFF 状態から ON 状態に変更した場合，1 回目の起動要求が発行されるまでの間隔は，周期起動ハンドラ登録時（コンフィギュレーション時または def_cyc システム・コール発行時）に指定した起動時間間隔よりも短くなる可能性があります。したがって，1 回目の起動要求を周期起動ハンドラ登録時に指定した時間間隔で発行するには，act_cyc システム・コールを発行する際，周期起動ハンドラの起動再開（TCY_ON）のほかに，周期カウンタの初期化（TCY_INI）をあわせて指定してください。

図 7－2，図 7－3 に，処理プログラムから act_cyc システム・コールを発行して周期起動ハンドラの活性状態を OFF から ON に変更した場合の処理の流れを示します。

なお，図 7－2，図 7－3 におけるΔT は，周期起動ハンドラ登録時に指定したハンドラの起動時間間隔を表します。また，図 7－2 におけるΔt とΔT の関係は，$\Delta t \leqq \Delta T$ であるものとします。

保守／廃止

図7－2　act_cyc（TCY_ON）発行時の処理の流れ

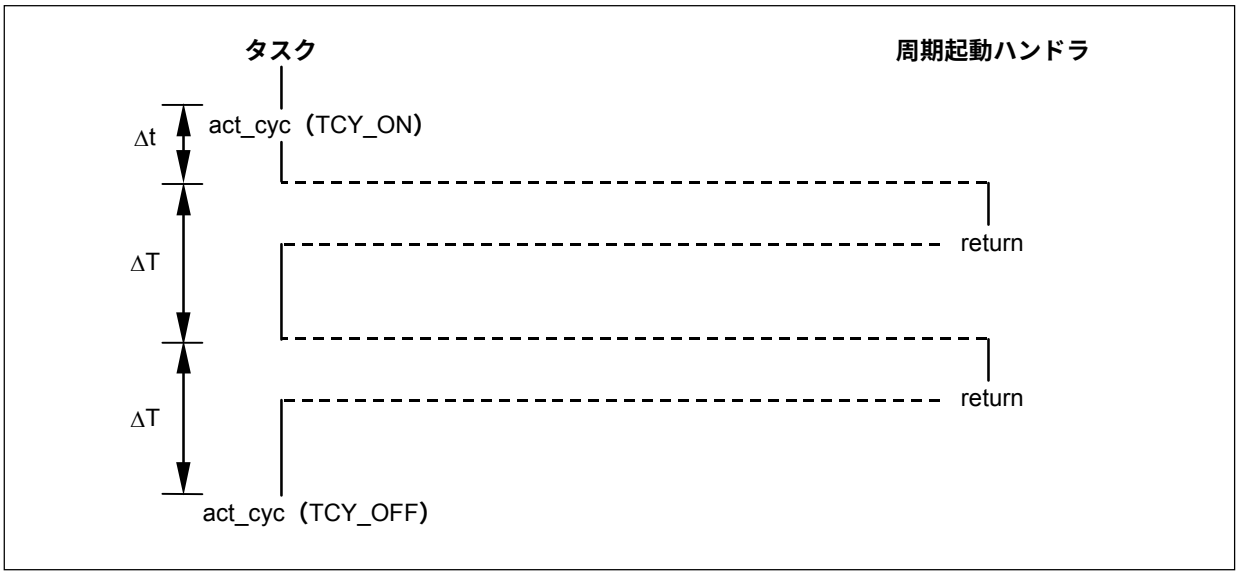
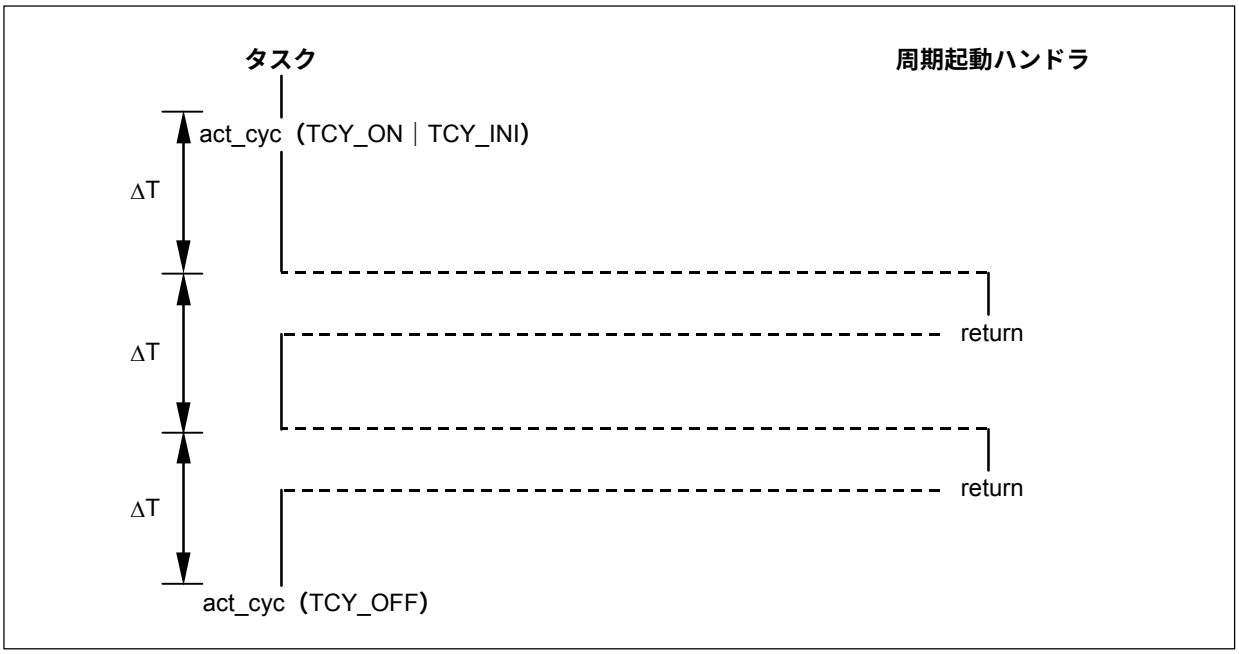


図7－3　act_cyc（TCY_ON｜TCY_INI）発行時の処理の流れ



7.6.3　周期起動ハンドラ内での処理

RX850 Pro では，タイマ割り込みが発生してから周期起動ハンドラに制御を移す際，独自の割り込み前処理を行っています。また，周期起動ハンドラから制御を戻す際にも，独自の割り込み後処理を行っています。

したがって，周期起動ハンドラの処理を記述する際には，次の点に注意してください。

(1) レジスタの退避と復帰

RX850 Pro では，周期起動ハンドラに制御を移すときと周期起動ハンドラから復帰するときに，C コンパイラ（CA850，または CCV850）の関数呼び出し規約に従って作業用レジスタの退避処理，復帰処理を行っています。したがって，周期起動ハンドラの開始部分で作業用レジスタの退避処理を，終了部分で作業用レジスタの復帰処理を記述する必要はありません。

保守／廃止

(2) スタックの切り替え

RX850 Pro では，周期起動ハンドラに制御を移すときと周期起動ハンドラから復帰するときに，スタックの切り替えを行います。したがって，周期起動ハンドラの開始部分で割り込みハンドラ用スタックへの切り替え処理を，終了部分で割り込み発生時のスタックへの切り替え処理を記述する必要はありません。

ただし，コンフィギュレーション時に割り込みハンドラ用スタックを定義していない場合は，スタックの切り替え処理は行われず，割り込み発生時のスタックが使用されます。

(3) システム・コールの発行制限

次に，周期起動ハンドラ内で発行可能なシステム・コールの一覧を示します。

●タスク管理機能システム・コール

sta_tsk	chg_pri	rot_rdq	rel_wai	get_tid
ref_tsk	vget_tid			

●タスク付属同期機能システム・コール

sus_tsk	rsm_tsk	frsm_tsk	wup_tsk	can_wup

●同期通信機能システム・コール

sig_sem	preq_sem	ref_sem	vget_sid	set_flg
clr_flg	pol_flg	ref_flg	vget_fid	snd_msg
prcv_msg	ref_mbx	vget_mid		

●割り込み管理機能システム・コール

def_int	ena_int	dis_int	chg_icr	ref_icr

●メモリ・プール管理機能システム・コール

pget_blk	rel_blk	ref_mpl	vget_pid

●時間管理機能システム・コール

set_tim	get_tim	def_cyc	act_cyc	ref_cyc

●システム管理機能システム・コール

get_ver	ref_sys	def_svc	viss_svc

(4) 周期起動ハンドラからの復帰処理

周期起動ハンドラからの復帰処理は，ハンドラの終了部分で return 命令を発行することにより行われます。

RX850 Pro では，周期起動ハンドラ内でタスクのスケジューリング処理が必要となるシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューの操作などの処理が行われるだけであり，実際のスケジューリング処理は，周期起動ハンドラからの復帰処理（return 命令の発行）まで遅延され，一括して行われます。

保守／廃止

7.6.4　周期起動ハンドラ情報の獲得

周期起動ハンドラ情報の獲得は，ref_cyc システム・コールを発行することにより行われます。

●ref_cyc システム・コール

パラメータで指定された周期起動ハンドラの周期起動ハンドラ情報（拡張情報，残り時間など）を獲得します。

次に，周期起動ハンドラ情報の詳細を示します。

- 拡張情報
- 次に周期起動ハンドラを起動するまでの残り時間
- 現在の活性状態

★　7.6.5　周期起動ハンドラ中の割り込み

周期起動ハンドラは，割り込み許可状態で開始されます。周期起動ハンドラ中では割り込みを禁止したい場合，ハンドラの先頭で割り込みを禁止してください。

RX850 Pro では，2 種類のニュークリアス共通部（rxcore.o, rxtmcore.o）を提供しており，使用するニュークリアス共通部によって，周期起動ハンドラ内で受け付け可能な割り込みが異なります。

●rxcore.o 使用時

周期起動ハンドラは，タイマ・ハンドラから呼び出されますが，タイマ・ハンドラ実行中に割り込み終了処理を行っていますので，すべての割り込みレベルの割り込みを受け付けることができます。

●rxtmcore.o 使用時

周期起動ハンドラは，タイマ・ハンドラから呼び出されますが，タイマ・ハンドラ実行中に割り込み終了処理を行っていませんので，タイマ割り込みよりも優先順位の高い割り込みのみ受け付けることが可能です。なお，割り込みを許可してもタイマ割り込みは保留されるため，周期起動ハンドラ内で時間のかかる処理を行う場合は，実際に経過した時間と RX850 Pro が管理している時間とにずれが生じる可能性があるため，注意が必要です。

また，周期起動ハンドラは間接起動割り込みハンドラとして実現されているので，実行時はハンドラ・スタック上で動作します。

7.6.6　周期起動ハンドラの起動順序

同時に起動間隔時間が経過した周期起動ハンドラが複数ある場合，指定した起動間隔時間が短いハンドラから順に起動されます。また，他の周期起動ハンドラ実行中に起動間隔時間が経過した場合，その周期起動ハンドラはただちには起動されず，現在実行中の周期起動ハンドラが終了したあとに起動されます。

第 8 章　スケジューラ

この章では，RX850 Pro が行うスケジューリング処理について説明します。

8.1　概　　要

RX850 Pro のスケジューラでは，ダイナミックに変化するタスクの状態を観察することによりタスクの実行順序を管理，決定し，最適なタスクにプロセッサの使用権を与えます。

8.2　駆動方式

RX850 Pro のスケジューラは，何らかの事象が発生した際に駆動される，事象駆動方式を採用しています。

なお，RX850 Pro における「何らかの事象」とは，タスクの状態遷移を引き起こす可能性のあるシステム・コールの発行，ハンドラからの復帰命令の発行，およびクロック割り込みの発生を意味します。これらの事象が発生するとスケジューラが駆動し，タスクのスケジューリング処理を行います。

次に，スケジューラを駆動するシステム・コールの一覧を示します。

●タスク管理機能システム・コール

sta_tsk　ext_tsk　exd_tsk　ena_dsp　chg_pri
rot_rdq　rel_wai

●タスク付属同期機能システム・コール

rsm_tsk　frsm_tsk　slp_tsk　tslp_tsk　wup_tsk

●同期通信機能システム・コール

del_sem　sig_sem　wai_sem　twai_sem　del_flg
set_flg　wai_flg　twai_flg　del_mbx　snd_msg
rcv_msg　trcv_msg

●割り込み管理機能システム・コール

ret_int　ret_wup　vret_clk　unl_cpu

●メモリ・プール管理機能システム・コール

del_mpl　get_blk　tget_blk　rel_blk

●時間管理機能システム・コール

dly_tsk

8.3　スケジューリング方式

RX850 Pro のスケジューリング方式は，優先度方式と FCFS（First Come First Service）方式を採用しています。

このため，スケジューラは，実行可能な状態（run 状態，ready 状態）にあるタスクの優先度を参照し，その中から最適なタスクを選び出し，プロセッサの使用権を与えます。

保守／廃止

8.3.1　優先度方式

各タスクには，処理を実行するうえでの優先順位を決定する優先度が付けられています。

したがって，スケジューラは，実行可能な状態（run 状態，ready 状態）にあるタスクの優先度を参照し，その中から最も高い優先度（最高優先度）を持つタスクを選び出し，プロセッサの使用権を与えます。

備考　RX850 Proにおけるタスクの優先度は，その値が小さいほど高い優先度であることを示します。

8.3.2　FCFS 方式

RX850 Pro では，複数のタスクに同一の優先度を割り付けることが可能です。このため，優先度方式におけるタスクを選び出す基準である，「最も高い優先度（最高優先度）を持つタスク」が複数存在する場合があります。

そこで，スケジューラは，最高優先度を持つタスクが複数存在する場合には，先に実行可能状態になったタスク（ready 状態になってから最も時間が経過しているタスク）を選び出し，プロセッサの使用権を与えます。

8.4　ラウンドロビン方式の実現

RX850 Pro では，優先度方式または FCFS 方式のスケジューリングを行っていますが，これらの方式では，タスクが複数存在した場合，最初にプロセッサの使用権を与えられたタスクがほかの状態へ遷移するか，またはプロセッサの使用権を放棄しないかぎり，他タスクが「同一の優先度でありながら処理を実行することができない」といった事態が生じてきます。

そこで，RX850 Pro では，このような事態を回避するスケジューリング方式（ラウンドロビン方式）を実現するために，rot_rdq などのようなシステム・コールを提供しています。

次に，ラウンドロビン方式の実現例を示します。

条　　件

●タスクの優先度

タスク A=タスク B=タスク C

●タスクの状態

タスク A　：run 状態
タスク B　：ready 状態
タスク C　：ready 状態

●周期起動ハンドラの属性

活性状態　　：ON 状態
起動時間間隔　：ΔT（単位：基本クロック周期）
処理内容　　：レディ・キューの回転（rot_rdq システム・コールの発行）

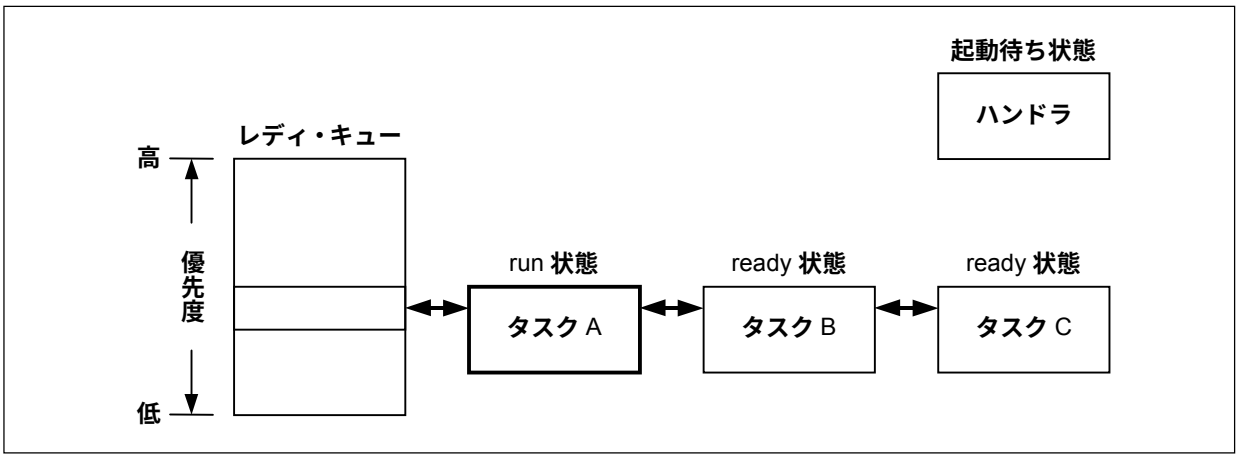
保守／廃止

(1)　**現在，タスク**A**が処理を実行しています。**

他タスク（タスク B**，タスク** C**）は，タスク** A **と同一の優先度を持ちますが，タスク** A **がほかの状態へ遷移するか，またはプロセッサの使用権を放棄しないかぎり，処理を実行できません。**

図 8－1 **に，このときのレディ・キューの状態を示します。**

図8－1　レディ・キューの状態（1）

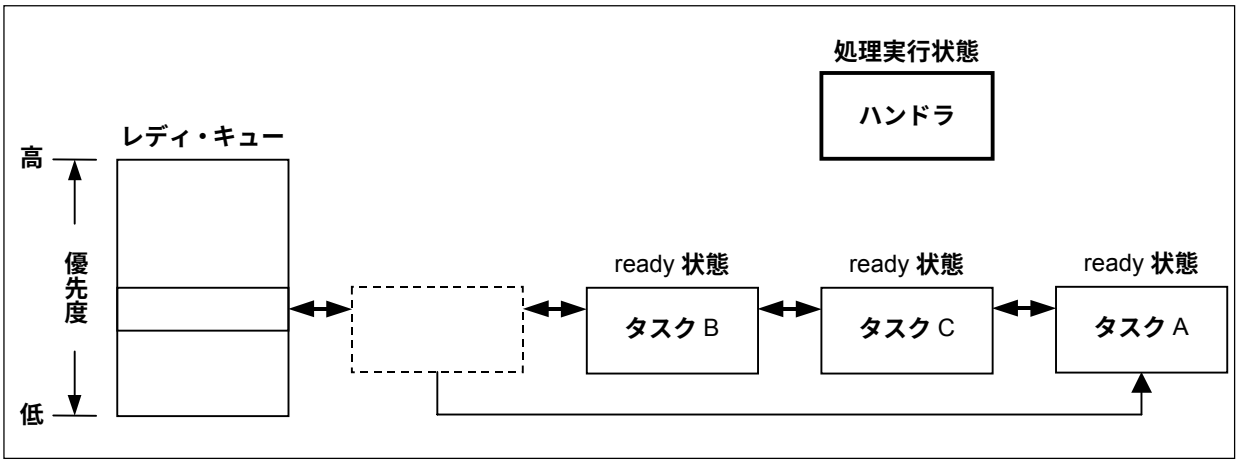


(2)　**時間の経過により周期起動ハンドラが起動し，**rot_rdq**システム・コールを発行します。**

これにより，タスク A **は優先度に応じたレディ・キューの最後尾にキューイングされるとともに，**run **状態から** ready **状態へと遷移します。**

図 8－2 **に，このときのレディ・キューの状態を示します。**

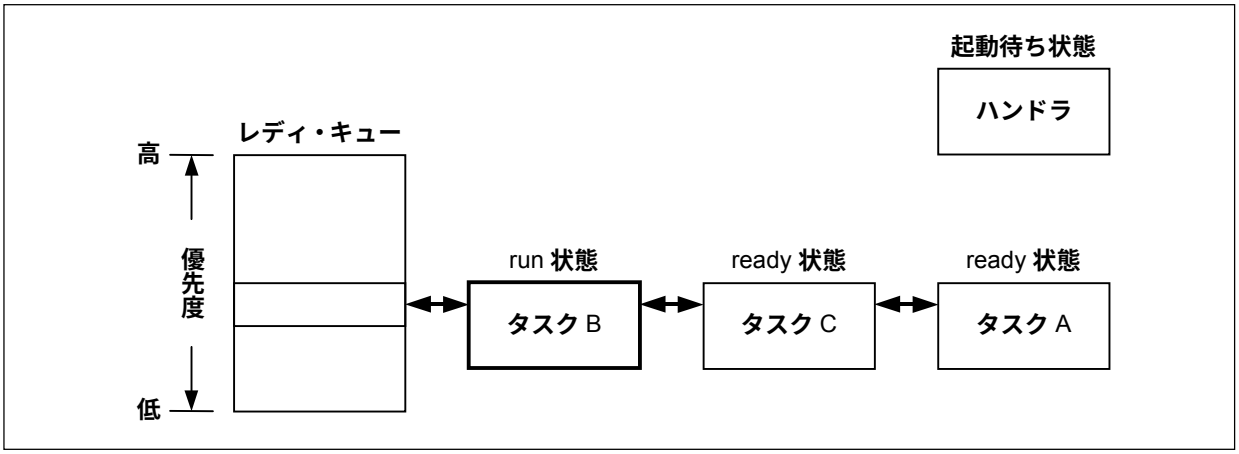
図8－2　レディ・キューの状態（2）



(3)　**タスクAがrun状態からready状態へと遷移し，タスクBがready状態からrun状態へと遷移します。**

図 8－3 **に，このときのレディ・キューの状態を示します。**

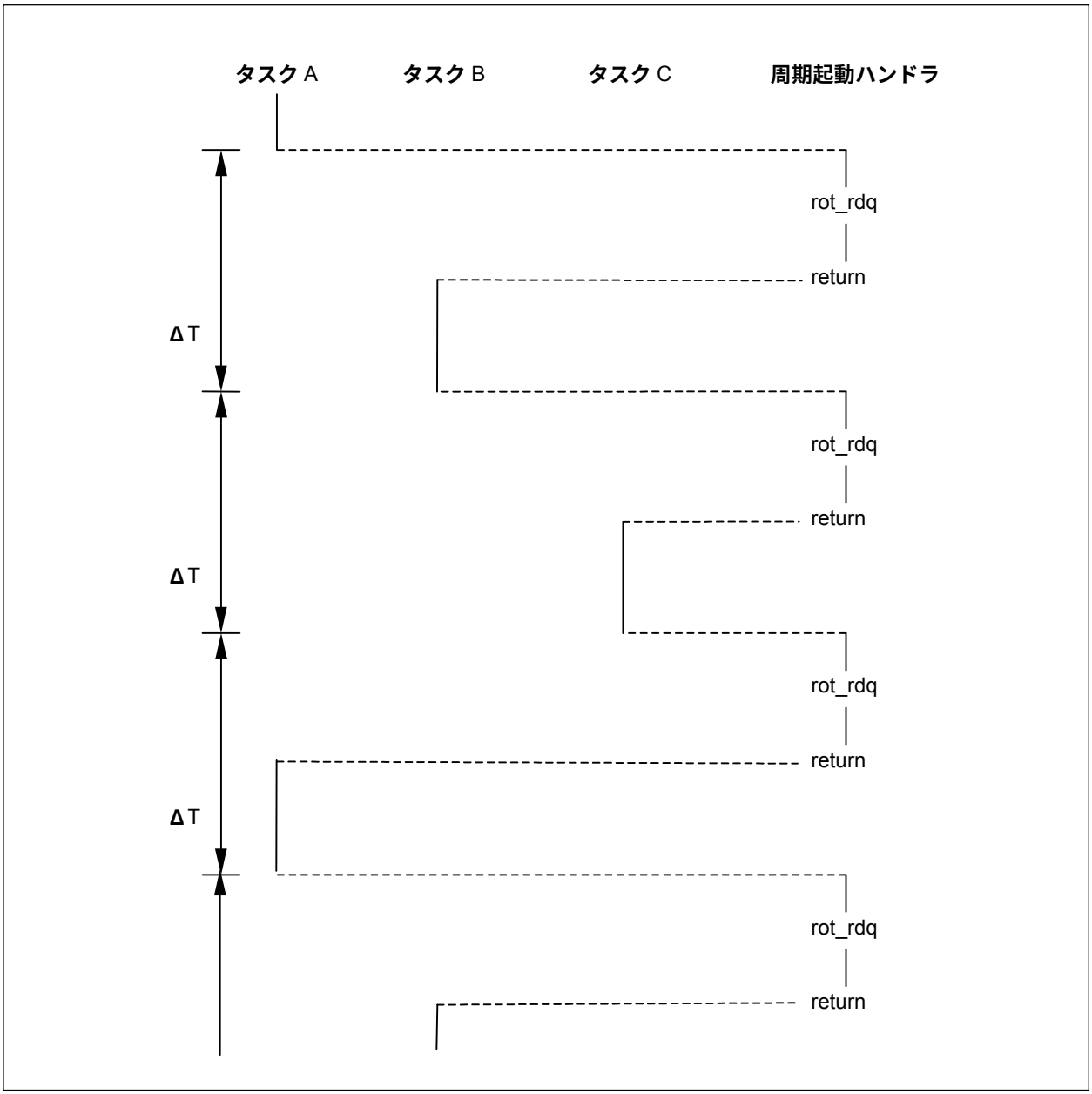
図8－3　レディ・キューの状態（3）



(4)　**このようにして，一定周期で起動される周期起動ハンドラから**rot_rdq**システム・コールを発行することにより，一定の時間（**∆T**）が経過したときに必ずタスクが切り替わるスケジューリング方式（ラウンドロビン方式）を実現できます。**

図 8－4 **に，ラウンドロビン方式による処理の流れを示します。**

図8－4　ラウンドロビン方式による処理の流れ



8.5　スケジューリングのロック機能

RX850 Pro では，ユーザの処理プログラム（タスク）からスケジューラの駆動を操作し，ディスパッチ処理（タスクのスケジューリング処理）を禁止／再開する機能が提供されています。

なお，これらの機能は，次に示すシステム・コールをタスク内から発行することにより実現されます。

●dis_dsp システム・コール

ディスパッチ処理（タスクのスケジューリング処理）を禁止します。

これにより，このシステム・コールの発行から ena_dsp システム・コールが発行されるまでの間は，他タスクに制御が移ることはありません。

保守／廃止

●ena_dsp システム・コール

ディスパッチ処理（タスクのスケジューリング処理）を再開します。

なお，RX850 Pro では，dis_dsp システム・コールの発行から ena_dsp システム・コールの発行までの間に，タスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューの操作などの処理が行われるだけであり，実際のスケジューリング処理は，ena_dsp システム・コールが発行されるまで遅延され，一括して行われます。

●loc_cpu システム・コール

マスカブル割り込みの受け付けを禁止したあと，ディスパッチ処理（タスクのスケジューリング処理）を禁止します。

これにより，このシステム・コールが発行されてから unl_cpu システム・コールが発行されるまでの間は，ほかのタスクやハンドラに制御が移ることはありません。

●unl_cpu システム・コール

マスカブル割り込みの受け付けを許可したあと，ディスパッチ処理（タスクのスケジューリング処理）を再開します。

なお，RX850 Pro では，loc_cpu システム・コールの発行から unl_cpu システム・コールの発行までの間にマスカブル割り込みが発生した場合，該当する割り込み処理（割り込みハンドラ）への移行は unl_cpu システム・コールが発行されるまで遅延されます。また，loc_cpu システム・コールの発行から unl_cpu システム・コールの発行までの間にタスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューの操作などの処理が行われるだけであり，実際のスケジューリング処理は unl_cpu システム・コールが発行されるまで遅延され，一括して行われます。

図 8－5 にスケジューリング処理が遅延されない場合（通常時）の制御の流れを，図 8－6，図 8－7 に，dis_dsp，loc_cpu システム・コールを発行した場合の制御の流れを示します。

図8－5　スケジューリング処理が遅延されない場合（通常時）の制御の流れ

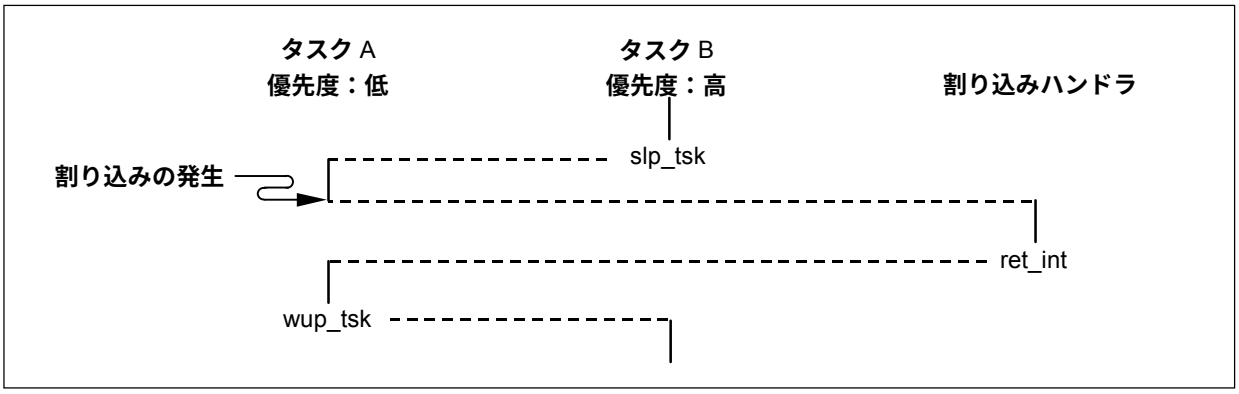


図8－6　dis_dspシステム・コールを発行した場合の制御の流れ

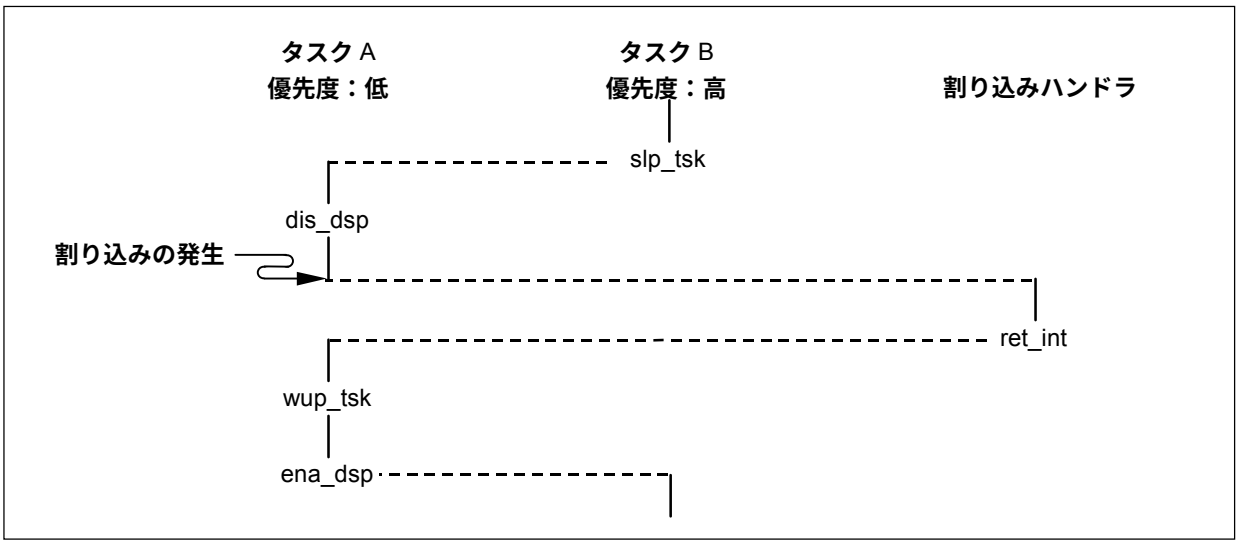
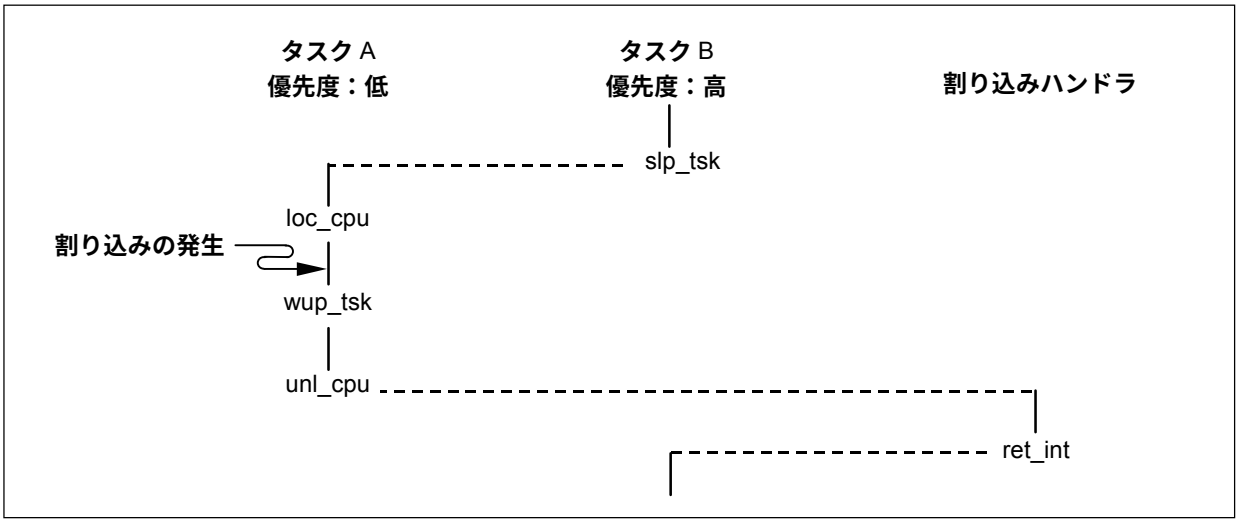


図8－7　loc_cpuシステム・コールを発行した場合の制御の流れ



8.6　ハンドラ内でのスケジューリング

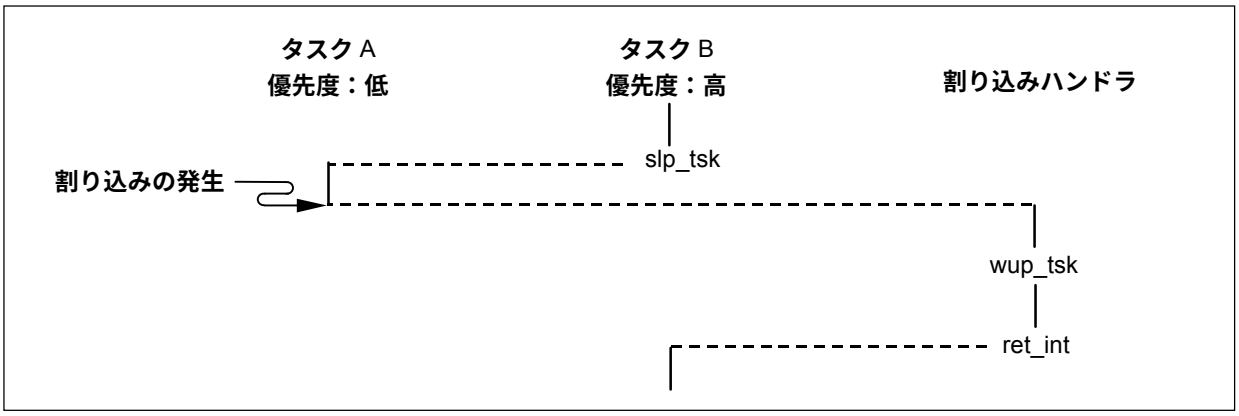
RX850 Pro では，各種ハンドラ（割り込みハンドラ，周期起動ハンドラ）を高速に終了させる目的で，ハンドラ内の処理が終了するまでの間，スケジューラの駆動を遅延します。

したがって，ハンドラ内でタスクのスケジューリング処理が必要になるシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューの操作などの処理が行われるだけであり，実際のスケジューリング処理はハンドラからの復帰処理（ret_int システム・コール，return 命令の発行）まで遅延され，一括して行われます。

図 8－8 に，ハンドラ内でスケジューリング処理が必要なシステム・コールが発行された場合の制御の流れを示します。

保守／廃止

図8－8　wup_tskシステム・コールを発行した場合の制御の流れ



8.7　アイドル・ハンドラ

8.7.1　アイドル・ハンドラ

アイドル・ハンドラは，すべてのタスク（ユーザの定義したタスク）が run 状態，および ready 状態でなくなったとき，すなわち RX850 Pro のスケジューリング対象となるタスクがシステム内に 1 つも存在しなくなったときに処理を実行します。

アイドル・ハンドラの処理とは，CPU を HALT 状態にすることです。したがって，タスクがシステム内に 1 つも存在しなくなると，RX850 Pro は CPU を HALT 状態にします。

ただし，このアイドル・ハンドラは，CPU を IDLE 状態，STOP 状態にすることはできません。IDLE 状態，STOP 状態にしたい場合，またはアイドル処理を自分で記述したい場合は，一番優先度の低いタスクを作成し，アイドル・タスクとしてください。これにより，アイドル・ハンドラと同様の処理を実現できます。ただし，HALT や IDLE，STOP 状態は割り込みによって解除されるため，アイドル・タスク内では割り込みを禁止状態のままにしないでください。

第 9 章　システム初期化処理

この章では，RX850 Pro が行うシステム初期化処理について説明します。

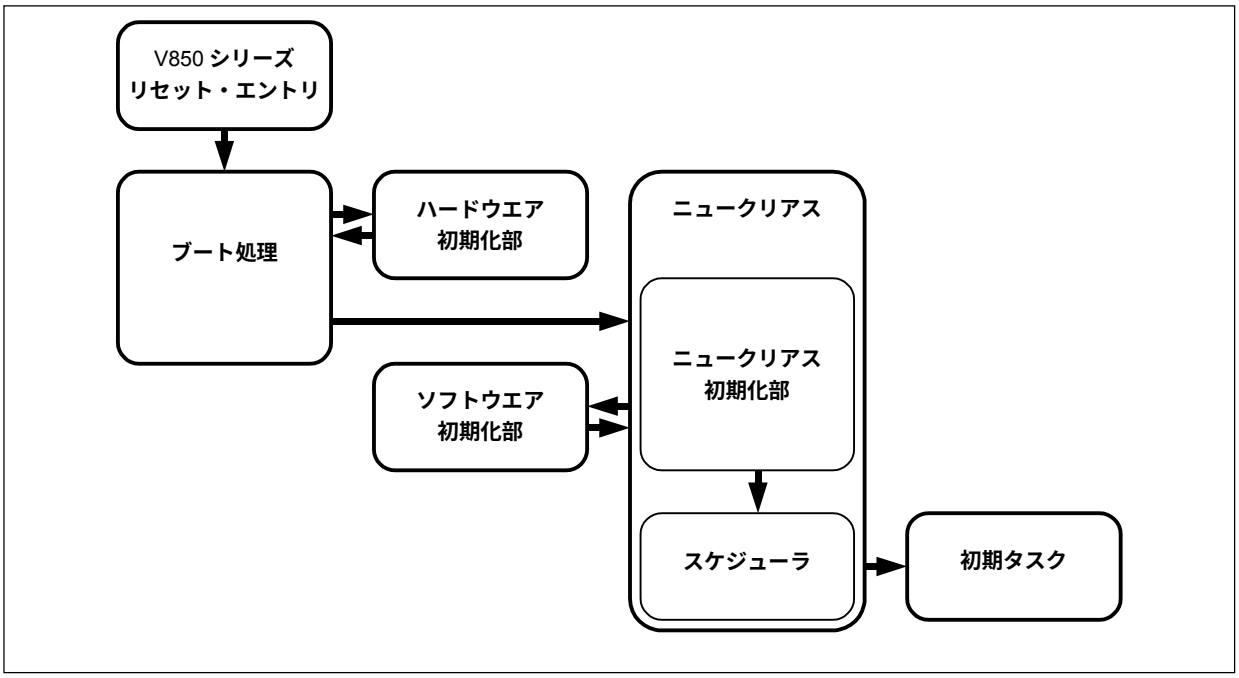
なお，システム初期化処理についての詳細は，**RX850 Pro　ユーザーズ・マニュアル　インストレーション編（U13774J）**を参照してください。

9.1 概　　要

システム初期化処理は，RX850 Pro が動作するうえで必要となるハードウエアの初期化処理とソフトウエアの初期化処理から構成されています。つまり，システムが起動したとき，RX850 Pro で最初に行われる処理が，システム初期化処理となります。

図 9－1 に，システム初期化処理の流れを示します。

図9－1　システム初期化処理の流れ



保守／廃止

9.2　ブート処理

ブート処理は，V850 シリーズのリセット・エントリ（ハンドラ・アドレス：0x0）に割り付けられた機能であり，システム初期化処理の中でも最初に実行されます。サンプルでは，boot.s（NEC エレクトロニクス版），boot.850（GHS 版）というファイルで行われています（関数名：_ _boot）。

ブート処理では，次のような処理を行います。

- gp, tp, ep レジスタの設定
- 初期値なしメモリ領域の初期化
- ハードウエア初期化部の呼び出し
- ニュークリアス初期化部に制御を移す

サンプルのブート処理は，その処理内容を，ユーザのニーズにあわせて書き換えます。

9.3　ハードウエア初期化部

ハードウエア初期化部は，ブート処理から呼び出される関数であり，実行環境（ターゲット・システム）上のハードウエアを初期化するために用意されたものです。サンプルでは，init.c というファイルで行われています（関数名：reset）。

ハードウエア初期化部では，次のような処理を行います。

- 内部ユニットの初期化
  - 割り込みコントローラの初期化
  - クロック・コントローラの初期化
- 周辺コントローラの初期化
- ブート処理に制御を戻す

ハードウエア初期部は，実行環境のハードウエア構成に依存します。

この部分を切り出すことにより，さまざまなターゲット・システムへの移植性を向上させるとともに，カスタマイズ化を容易なものにしています。ユーザの実行環境にあわせて書き換えてください。

保守／廃止

9.4　ニュークリアス初期化部

ニュークリアス初期化部は，ブート処理終了後に呼び出されます。情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に記述された情報（タスク情報，セマフォ情報など）をもとに，各種管理オブジェクトを生成／初期化します。この処理の終了後に，RX850 Pro が起動されます。なお，この処理部はニュークリアス・ライブラリに含まれています。

ニュークリアス初期化部では，次のような処理を行います。

- ●管理オブジェクトの生成／初期化
  - ・タスクの生成
  - ・セマフォの生成／初期化
  - ・イベント・フラグの生成／初期化
  - ・メモリ・プールの生成／初期化
  - ・間接起動割り込みハンドラの登録
  - ・周期起動ハンドラの登録
  - ・拡張 SVC ハンドラの登録
- ●初期タスクの起動
- ●システム・タスク（アイドル・タスク）の起動
- ●ソフトウエア初期化部の呼び出し
- ●スケジューラに制御を移す

9.5　ソフトウエア初期化部

ソフトウエア初期化部は，ニュークリアス初期化部から呼び出される関数です。RX850 Pro 起動前にしておきたい処理がある場合などに利用できます。サンプルでは，varfunc.c というファイルで行われています（関数名：varfunc）。

ソフトウエア初期化部では，次のような処理を行います。

- ●初期化データのコピー
- ●ニュークリアス初期化部に制御を戻す

保守／廃止

第 10 章　インタフェース・ライブラリ

この章では，インタフェース・ライブラリについて説明します。

なお，インタフェース・ライブラリについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　インストレーション編（U13774J）**を参照してください。

10.1　概　　要

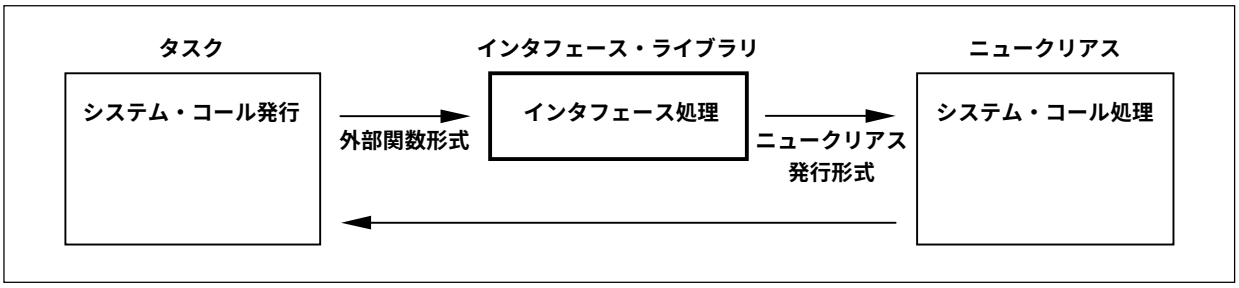
RX850 Pro では，ユーザの処理プログラムと RX850 Pro のニュークリアスの中間に位置するインタフェース・ライブラリを提供しています。インタフェース・ライブラリは，ニュークリアスが処理を行ううえで必要な各種情報の設定などを行ったあと制御を移す機能を持っています。

処理プログラム（タスク，非タスク）を C 言語で記述した場合，システム・コールの発行形式や拡張 SVC ハンドラの呼び出し形式は外部関数形式となります。しかし，ニュークリアスが理解可能な発行形式（ニュークリアス発行形式）と外部関数形式には相違があります。

そこで，システム・コールの発行形式や拡張 SVC ハンドラの呼び出し形式を外部関数形式からニュークリアス発行形式に変換する手続き（インタフェース）が必要になります。このような，処理プログラムとニュークリアスの仲介役を行うインタフェースは各システム・コールごとにあり，これを集めたものがインタフェース・ライブラリです。

図 10－1 に，インタフェース・ライブラリの位置付けを示します。

図10－1　インタフェース・ライブラリの位置付け



保守／廃止

10.2　インタフェース・ライブラリ内での処理

インタフェース・ライブラリでは，次のような処理を行っています。

●ニュークリアスが管理しているテーブルに必要な情報を設定する
●必要になるデータをレジスタに設定する
●システム・コールのエラー値を設定（ただし，ニュークリアス内で設定されるエラーは除く）したあと，処理プログラムへ処理を戻す

インタフェース・ライブラリを用意することにより，ニュークリアスとユーザの処理プログラムの分離が容易になります。たとえば，ニュークリアス本体を ROM 化したあとでユーザの処理プログラムを変更する必要が生じた場合でも，ニュークリアス本体が格納されている ROM を変更する必要がなくなります。また，ロード・モジュールを分割して作成するといったことも可能になります。

10.3　インタフェース・ライブラリの種類

RX850 Pro が提供するインタフェース・ライブラリには，システム・コールのパラメータをチェックする機能を持つものと持たないものの 2 種類があります。システムにどちらを組み込むかは，リンク時に指定します。

パラメータ・チェック機能のあるライブラリを使用すると，システム・コール発行時のパラメータの指定が不正だった場合，必ず戻り値が返されます。一方，パラメータ・チェック機能のないライブラリを使用すると，システム・コール発行時のパラメータの指定が不正だった場合などのとき，戻り値が返されない場合があります。

これらの 2 種類のライブラリは，用途に応じて使い分けることができます。たとえば，ディバグ時はパラメータ・チェック機能のあるものを使用し，実際の組み込み時はパラメータ・チェック機能のないものを使用することで，プログラムのパフォーマンス向上や容量の節減などを実現します。

備考　パラメータ・チェック機能のないライブラリで戻り値が返されるエラーは，**第 11 章　システム・コール**の各システム・コールの**戻り値**の欄で，「*」を付けて示しています。

注意　**パラメータ・チェック機能のないライブラリを使用したとき，戻り値が返されないエラーが発生した場合は，アプリケーション・システムの動作は保証されません。**

10.4　提供されているインタフェース・ライブラリ

RX850 Pro が提供しているインタフェース・ライブラリは，次の 2 種類です。

●NEC エレクトロニクス製 V850 シリーズ用 C コンパイラ「CA850」用
●米国 Green Hills Software, Inc.製 C クロス V800 コンパイラ「CCV850」用

備考　他のコンパイラを使用する場合は，そのコンパイラで使用するレジスタの状況によって，インタフェース・ライブラリを書き換える必要があります。 インタフェース・ライブラリの格納場所などについては，**RX850 Pro　ユーザーズ・マニュアル　インストレーション編（U13774J）**を参照してください。

保守／廃止

第 11 章　システム・コール

この章では，RX850 Pro が提供するシステム・コールについて説明します。

11.1　概　　要

システム・コールとは，ユーザの処理プログラム（タスク，非タスク）から RX850 Pro のサービス・ルーチンを呼び出すための手続き／機能です。システム・コールを利用すると，RX850 Pro が直接管理している資源（カウンタ，待ちキューなど）を間接的に操作できます。

RX850 Pro では，μITRON3.0 仕様で規定されているシステム・コール（65 種類）のほかに，RX850 Pro オリジナルのシステム・コール（7 種類）も提供し，アプリケーション・システムの汎用性を高めています。

システム・コールは，その機能により次に示す 7 グループに分類できます。

（1）タスク管理機能システム・コール（14種類）

タスクの状態操作を行うシステム・コールのグループです。

また，このグループには，タスクを生成，起動，終了，または削除する機能，ディスパッチ処理を禁止または再開する機能，タスクの優先度を変更する機能，タスクのレディ・キューを回転する機能，タスクの wait 状態を強制的に解除する機能，タスクの状態を参照する機能のシステム・コールも含まれます。

cre_tsk	del_tsk	sta_tsk	ext_tsk	exd_tsk
ter_tsk	dis_dsp	ena_dsp	chg_pri	rot_rdq
rel_wai	get_tid	ref_tsk	vget_tid	

（2）タスク付属同期機能システム・コール（7種類）

タスクに従属した同期操作を行うシステム・コールのグループです。

また，このグループには，タスクを suspend 状態に移行する，または suspend 状態のタスクを再開する機能，タスクを起床待ち状態に移行する，または起床待ち状態のタスクを起床させる機能，タスクの起床要求を無効にする機能などのシステム・コールも含まれます。

sus_tsk	rsm_tsk	frsm_tsk	slp_tsk	tslp_tsk
wup_tsk	can_wup			

（3）同期通信機能システム・コール（25種類）

タスク間の同期（排他制御，待ち合わせ）と通信を行うシステム・コールのグループです。

また，このグループには，セマフォを操作する機能，イベント・フラグを操作する機能，メールボックスを操作する機能のシステム・コールも含まれます。

保守／廃止

cre_sem	del_sem	sig_sem	wai_sem	preq_sem
twai_sem	ref_sem	vget_sid	cre_flg	del_flg
set_flg	clr_flg	wai_flg	pol_flg	twai_flg
ref_flg	vget_flg	cre_mbx	del_mbx	snd_msg
rcv_msg	prcv_msg	trcv_msg	ref_mbx	vget_mid

(4) 割り込み管理機能システム・コール（9種類）

マスカブル割り込みに依存した処理を行うシステム・コールのグループです。

また，このグループには，間接起動割り込みハンドラを登録または登録解除する機能，直接起動割り込みハンドラから復帰する機能，割り込み許可レベルを変更または参照する機能のシステム・コールも含まれます。

def_int	ret_int	ret_wup	ena_int	dis_int
loc_cpu	unl_cpu	chg_icr	ref_icr	

(5) メモリ・プール管理機能システム・コール（8種類）

メモリの割り当てを行うシステム・コールのグループです。

また，このグループには，メモリ・プールを生成または削除する機能，メモリ・ブロックを獲得または返却する機能，メモリ・プールの状態を参照する機能のシステム・コールも含まれます。

cre_mpl	del_mpl	get_blk	pget_blk	tget_blk
rel_blk	ref_mpl	vget_pid		

(6) 時間管理機能システム・コール（6種類）

時間に依存した処理を行うシステム・コールのグループです。

また，このグループには，システム・クロックの時刻を設定または参照する機能，タスクを時間経過待ち状態に遷移させる機能，周期起動ハンドラを登録または登録解除する機能，周期起動ハンドラの活性状態を制御または参照する機能のシステム・コールも含まれます。

set_tim	get_tim	dly_tsk	def_cyc	act_cyc
ref_cyc				

(7) システム管理機能システム・コール（4種類）

システムに依存した処理を行うシステム・コールのグループです。

また，このグループには，バージョン情報を獲得する機能，システムの状態を参照する機能，拡張 SVC **ハンドラを登録または登録解除する機能，拡張** SVC **ハンドラを呼び出す機能のシステム・コールも含まれます。**

get_ver	ref_sys	def_svc	viss_svc

保守／廃止

11.2　システム・コールの呼び出し

C 言語で記述された処理プログラム（タスク，非タスク）からシステム・コールを発行する場合，C 言語の関数として呼び出し，そのパラメータを引き数として渡します。

また，アセンブリ言語で記述された処理プログラムからシステム・コールを発行する場合は，使用する C コンパイラの関数呼び出し規約に従ってパラメータと戻り番地の設定を行ったあと，jarl 命令により呼び出します。

注意　RX850 Pro では，システム・コールのプロトタイプ宣言を stdrx85p.h ファイルで行っています。したがって，処理プログラムからシステム・コールを発行する際には，次のようにヘッダ・ファイルのインクルード処理を記述してください。

```
#include   <stdrx85p.h>
```

11.3　システム・コールの機能コード

RX850 Pro が提供するシステム・コールには，μITRON3.0 仕様に準拠した機能コードが割り当てられています。

表 11－1 に，システム・コールに割り当てられている機能コード一覧を示します。

なお，RX850 Pro では，1 以上の値については，ユーザが記述した拡張 SVC ハンドラを登録する際に指定する拡張機能コードとしています。

表11－1　システム・コールの機能コード一覧

機能コード	分　類
－256　～　－225	RX850 Pro オリジナルのシステム・コール
－224　～　－5	μITRON3.0 仕様に準拠したシステム・コール
－4　～　0	システム予約
1　～	拡張 SVC ハンドラ

保守／廃止

11.4　パラメータのデータ・タイプ

RX850 Pro が提供するシステム・コールのパラメータは，μITRON3.0 仕様に準拠したデータ・タイプを基本に定義されています。

表 11－2 に，システム・コールを発行する際に指定する各種パラメータのデータ・タイプ一覧を示します。

表11－2　パラメータのデータ・タイプ一覧

マクロ	データ・タイプ	意　　味
B	char	符号付き 8 ビット整数
H	short	符号付き 16 ビット整数
INT	int	符号付き 32 ビット整数
W	long	符号付き 32 ビット整数
UB	unsigned char	符号なし 8 ビット整数
UH	unsigned short	符号なし 16 ビット整数
UINT	unsigned int	符号なし 32 ビット整数
UW	unsigned long	符号なし 32 ビット整数
VB	char	データ・タイプが一定しない値（8 ビット）
VH	short	データ・タイプが一定しない値（16 ビット）
VW	long	データ・タイプが一定しない値（32 ビット）
＊VP	void	データ・タイプが一定しない値（ポインタ）
（＊FP）()	void	処理プログラムの起動アドレス
BOOL	short	ブール値
FN	short	機能コード
ID	short	オブジェクト ID 番号
BOOL_ID	short	待ちタスクの有無
HNO	short	周期起動ハンドラの指定番号
ATR	unsigned short	オブジェクトの属性
ER	long	エラー・コード
PRI	short	タスクの優先度
TMO	long	待ち時間
CYCTIME	long	周期起動時間間隔（残り時間）
DLYTIME	long	遅延時間

保守／廃止

11.5　パラメータ値の範囲

RX850 Pro が提供するシステム・コールのパラメータには，指定可能な値に範囲のあるもの，特定の値をシステムで予約しているものなどがあります。

表 11－3 に，システム・コールを発行する際に指定する各種パラメータの値域一覧を示します。

表11－3　パラメータの値域一覧

パラメータの種類	値　域
オブジェクトの ID 番号	0x0 ～ max_cnt 注1
オブジェクトのキーID 番号	－0x8000 ～ 0x7FFF 注2
割り込みハンドラの割り込みレベル	0x0 ～ 0xF
周期起動ハンドラの指定番号	0x1 ～ max_cnt
拡張 SVC ハンドラの拡張機能コード	0x1 ～ max_cnt
オブジェクトの優先度	0x1 ～ max_cnt
セマフォの最大資源数	0x1 ～ max_cnt
マスカブル割り込みの割り込み許可レベル	0x0 ～ 0xF
システム・クロックの時刻	0x0 ～ 0x7FFF FFFF FFFF
待ち時間	－0x1 ～ 0x7FFF FFFF
遅延時間	0x0 ～ 0x7FFF FFFF
周期起動ハンドラの起動時間間隔	0x1 ～ 0x7FFF FFFF
タスクのスタック・サイズ	0x0 ～ 0x7FFF FFFF
メモリ・プール・サイズ	0x1 ～ 0x7FFF FFFF
メモリ・ブロック・サイズ	0x1 ～ 0x7FFF FFFF
メッセージの優先度	0x1 ～ 0x7FFF

注1. max_cnt：システム・コンフィギュレーション時に指定した最大オブジェクト数

2. オブジェクトのキーID番号に“ 0x0 ”を指定することはできません。

保守／廃止

11.6　システム・コールからの戻り値

RX850 Pro が提供するシステム・コールの戻り値は，μITRON3.0 仕様に準拠した戻り値を基本に定義されています。

表 11－4 に，システム・コールからの戻り値一覧を示します。

表11－4　システム・コールからの戻り値一覧

マクロ	数値	意　味
E_OK	0	正常終了
E_NOMEN	－10	オブジェクト用の領域が確保できない
E_NOSPT	－17	CF 定義されていないシステム・コールまたは登録されていない拡張 SVC ハンドラを呼び出した
E_RSATR	－24	オブジェクト属性の指定が不正である
E_PAR	－33	パラメータの指定が不正である
E_ID	－35	ID 番号の指定が不正である
E_NOEXS	－52	対象オブジェクトが存在していない
E_OBJ	－63	指定したオブジェクトの状態が不適当である
E_OACV	－66	アクセス権のない ID 番号を指定した
E_CTX	－69	システム・コールを発行する状態が不適当である
E_QOVR	－73	カウント値が 127 を越えた
E_DLT	－81	対象オブジェクトが削除された
E_TMOUT	－85	タイムアウト
E_RLWAI	－86	rel_wai システム・コールにより，wait 状態を強制的に解除された

11.7　システム・コールの拡張

RX850 Pro では，システム・コールを拡張（ユーザが記述した関数を拡張システム・コールとしてニュークリアスに登録）することができます。

なお，拡張システム・コールとして登録する関数に制限はなく，標準システム・コール（RX850 Pro が提供するシステム・コール）を含ませることも可能です。ただし，タスク状態からのみ発行可能な標準システム・コールを含ませた場合には，拡張システム・コールの発行状態が「タスク状態からのみ発行可能」に制限されます。

また，拡張システム・コールは，ユーザ定義のシステム・コールとして位置づけられる一方で，タスクに準じた部分を持ちます。つまり，標準システム・コールと同様に，処理が終了した際にはスケジューラが起動され，最適なタスクへの選択処理が行われます。

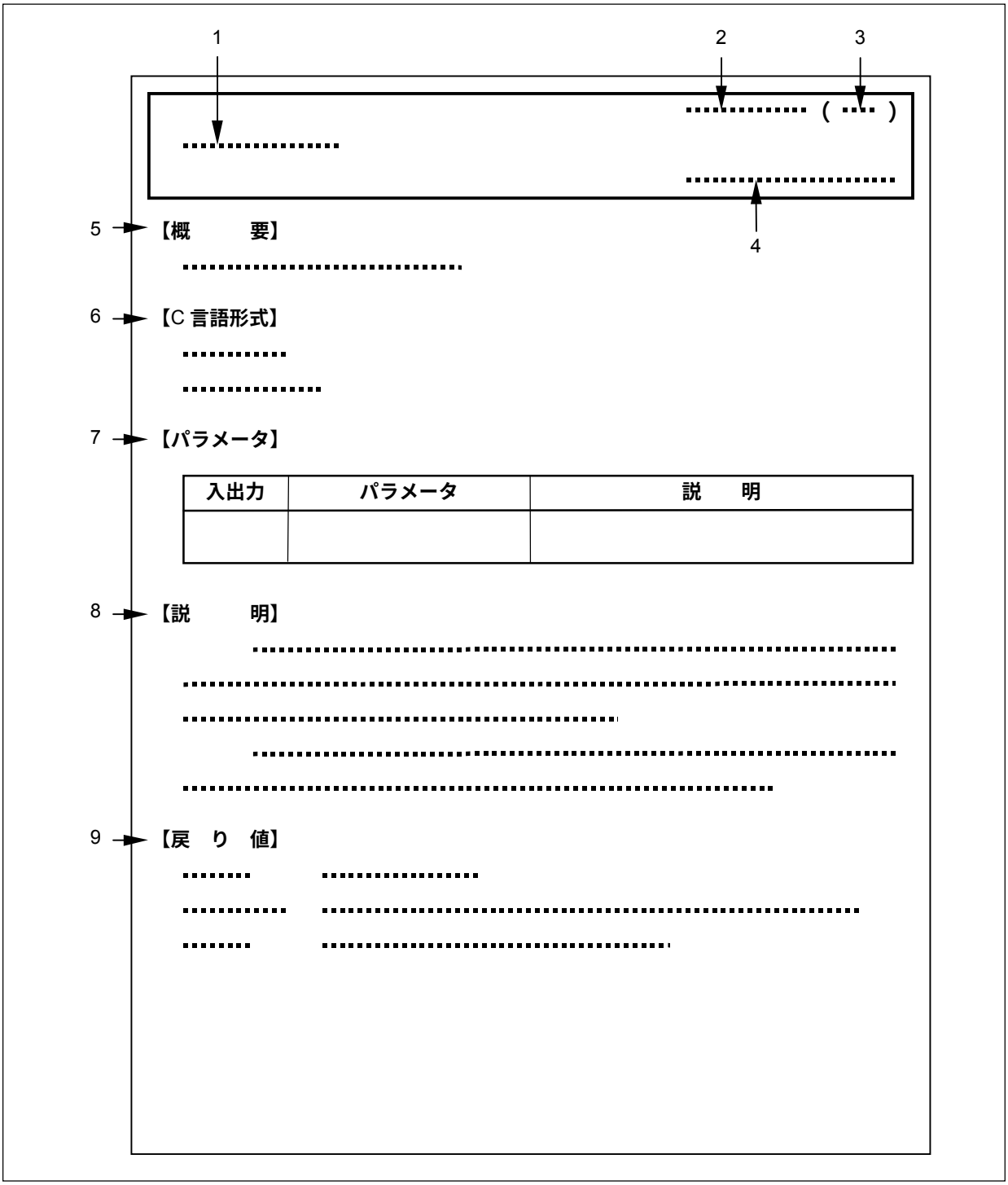
ただし，拡張システム・コール内に標準システム・コールを含ませた場合には，標準システム・コールの処理が終了した際にもスケジューラが起動されるため，拡張システム・コールの処理途中で，他タスクに制御が移ることがあります。

保守／廃止

11.8　システム・コールの解説

この節では，RX850 Pro が提供するシステム・コールについて，次の記述フォーマットに従って解説します。

図11－1　システム・コールの記述フォーマット



保守／廃止

(1) 名称

システム・コールの名称を示しています。

(2) 名称の由来

システム・コール名称の由来を示しています。

(3) 機能コード

システム・コールの機能コードを示しています。

(4) 発行可能な状態

システム・コールの発行が可能な状態を示しています。

タスク	： タスクからのみ発行可能
非タスク	： 非タスク（直接起動割り込みハンドラ，間接起動割り込みハンドラ，周期起動ハンドラ）からのみ発行可能
タスク／非タスク	： タスク，非タスクのどちらからも発行可能
直接起動割り込みハンドラ	： 直接起動割り込みハンドラからのみ発行可能
周期起動ハンドラ	： 周期起動ハンドラからのみ発行可能

(5) 【概　　要】

システム・コールの機能概要を示しています。

(6) 【C言語形式】

システム・コールを C 言語で発行する際の記述形式を示しています。

(7) 【パラメータ】

システム・コールのパラメータを次の形式で示しています。

入出力	パラメータ	説　　明
A	B	C

A：パラメータの種類

入 … RX850 Pro への入力パラメータ

出 … RX850 Pro からの出力パラメータ

B：パラメータのデータ・タイプ

C：パラメータの説明

(8) 【説　　明】

システム・コールの機能について示しています。

(9) 【戻　り　値】

システム・コールからの戻り値をマクロと数値で示しています。

*付きの戻り値 … 「パラメータ・チェック機能あり / なし」の両方の場合に返す値

*なしの戻り値 … 「パラメータ・チェック機能あり」の場合にだけ返す値

保守／廃止

11.8.1　タスク管理機能システム・コール

ここでは，タスクの状態操作を行うシステム・コールのグループ（タスク管理機能システム・コール）について説明します。

表 11－5 に，タスク管理機能システム・コールの一覧を示します。

表11－5　タスク管理機能システム・コール

システム・コール	機　　能
cre_tsk	他タスクを生成する
del_tsk	他タスクを削除する
sta_tsk	他タスクを起動する
ext_tsk	自タスクを終了する
exd_tsk	自タスクを終了したあと，削除する
ter_tsk	他タスクを強制的に終了する
dis_dsp	ディスパッチ処理を禁止する
ena_dsp	ディスパッチ処理を再開する
chg_pri	タスクの優先度を変更する
rot_rdq	タスクのレディ・キューを回転する
rel_wai	他タスクの wait 状態を強制的に解除する
get_tid	自タスクの ID 番号を獲得する
ref_tsk	タスク情報を獲得する
vget_tid	タスクの ID 番号を獲得する

保守／廃止

create task (－17)

cre_tsk

タスク

【概　　要】

タスクを生成する。

【C 言語形式】

・ID 番号を指定する場合

```
#include <stdrx85p.h>
ER       ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk);
```

・ID 番号を指定しない場合

```
#include <stdrx85p.h>
ER       ercd = cre_tsk(ID_AUTO, T_CTSK *pk_ctsk, ID *p_tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*tskid*;	タスクの ID 番号
入	T_CTSK　**pk_ctsk*;	タスク生成情報を格納したパケットの先頭アドレス
出	ID　　**p_tskid*;	ID 番号を格納する領域のアドレス

・タスク生成情報 T_CTSK の構造

```
typedef struct t_ctsk {
    VP    exinf;          /* 拡張情報 */
    ATR   tskatr;         /* タスクの属性 */
    FP    task;           /* タスクの起動アドレス */
    PRI   itskpri;        /* タスクの起動時優先度（初期優先度） */
    INT   stksz;          /* タスクのスタック・サイズ */
    VP    gp;             /* タスクの固有 GP レジスタ値 */
    VP    tp;             /* タスクの固有 TP レジスタ値 */
    ID    keyid;          /* タスクのキーID 番号 */
} T_CTSK;
```

保守／廃止

【説　　明】

RX850 Pro では，タスクの生成において「ID 番号を指定して生成する」，「ID 番号を指定しないで生成する」の 2 種類のインタフェースを用意しています。

●ID 番号を指定する場合

pk_ctsk で指定された情報を基に，*tskid* で指定された ID 番号を持つタスクを生成します。

これにより，対象タスクは non-existent 状態から dormant 状態へと遷移し，RX850 Pro の管理対象となります。

●ID 番号を指定しない場合

pk_ctsk で指定された情報を基に，タスクを生成します。

これにより，対象タスクは non-existent 状態から dormant 状態へと遷移し，RX850 Pro の管理対象となります。

ID 番号の割り付けは RX850 Pro により行われ，割り付けられた ID 番号は *p_tskid* で指定される領域に格納されます。

次に，タスク生成情報の詳細を示します。

exinf …　拡張情報

対象タスクに関する “ ユーザ独自の情報 ” を格納するための領域で，ユーザが自由に利用できます。

exinf に設定された情報は，処理プログラム（タスク，非タスク）から ref_tsk システム・コールを発行することにより，ダイナミックに獲得できます。

tskatr …　タスクの属性

ビット 0 …　タスクの記述言語

TA_ASM（0）　：アセンブリ言語

TA_HLNG（1）　：C 言語

ビット 8 …　キーID 番号指定の有無

TA_KEYID（1）　：キーID 番号を指定

ビット 9 …　メモリ領域の指定

TA_SPOL0（0）　：システム・メモリ領域 0 番からスタックの領域を確保

TA_SPOL1（1）　：システム・メモリ領域 1 番からスタックの領域を確保

ビット 10 …　固有 GP レジスタ値指定の有無

TA_DPID（1）　：固有 GP レジスタ値を指定

ビット 11 …　固有 TP レジスタ値指定の有無

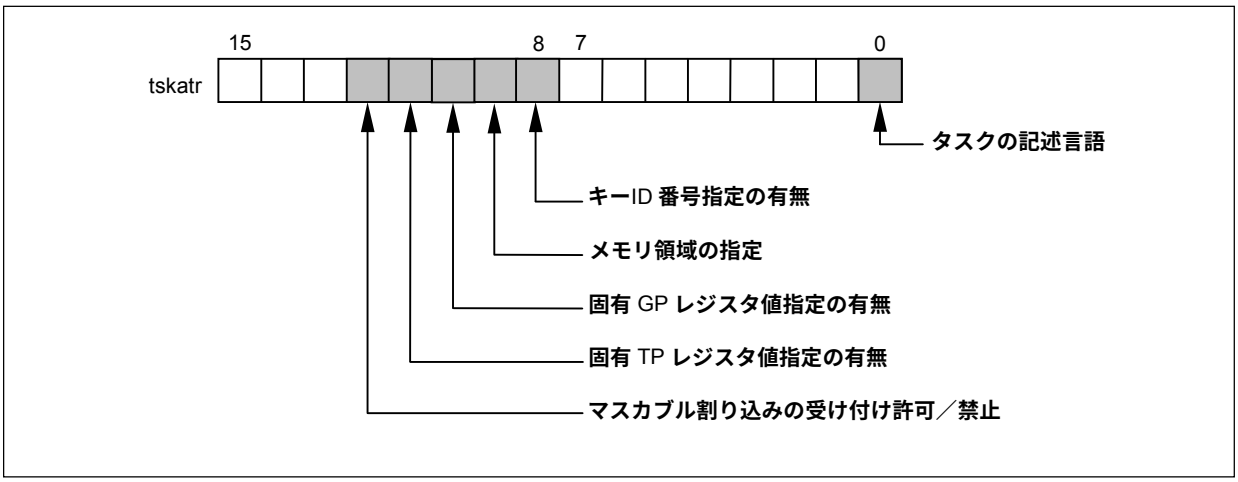
TA_DPIC（1）　：固有 TP レジスタ値を指定

ビット 12 …　マスカブル割り込みの受け付け許可／禁止

TA_ENAINT（0）：タスク起動時，マスカブル割り込みの受け付けは許可状態

TA_DISINT（1）：タスク起動時，マスカブル割り込みの受け付けは禁止状態

保守／廃止



task …　**タスクの起動アドレス**

itskpri …　**タスクの起動時優先度（初期優先度）**

stksz …　**タスクのスタック・サイズ（単位：バイト）**

gp …　**タスクの固有** GP **レジスタ値**

tp …　**タスクの固有** TP **レジスタ値**

keyid …　**タスクのキー**ID **番号**

備考　tskatr **のビット** 8 **の値が** 1（TA_KEYID）**以外の場合，**keyid **の内容は意味を持ちません。**
tskatr **のビット** 10 **の値が** 1（TA_DPID）**以外の場合，**gp **の内容は意味を持ちません。**
tskatr **のビット** 11 **の値が** 1（TA_DPIC）**以外の場合，**tp **の内容は意味を持ちません。**

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOMEM	－10	**タスク管理ブロックの領域が確保できない**
*E_NOSPT	－17	cre_tsk **システム・コールが** CF **定義されていない**
E_RSATR	－24	**属性** tskatr **の指定が不正である**
E_PAR	－33	**パラメータの指定が不正である** **・タスク生成情報を格納したパケットの先頭アドレスが不正（***pk_ctsk***＝0）である** **・起動アドレスの指定が不正（**task＝0**）である** **・起動時優先度の指定が不正（**itskpri≦0**，最大優先度＜**itskpri**）である** **・キー**ID **番号の指定が不正（**keyid＝0**）である（**TA_KEYID **属性指定時）** **・**ID **番号を格納する領域のアドレスが不正（***p_tskid***＝0）である（**ID **番号を指定しないで生成する場合）**
E_ID	－35	ID **番号の指定が不正（最大タスク生成数＜***tskid***）である**
*E_OBJ	－63	**指定した** ID **番号を持つタスクがすでに生成されている**
E_OACV	－66	**アクセス権のない** ID **番号（***tskid***≦0）を指定した**
E_CTX	－69	**非タスクから** cre_tsk **システム・コールを発行した**

保守／廃止

delete task (－18)

del_tsk

タスク

【概　　要】

他タスクを削除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = del_tsk(ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*tskid*;	タスクの ID 番号

【説　　明】

tskid で指定されたタスクを dormant 状態から non-existent 状態へと遷移させます。

これにより，対象タスクは RX850 Pro の管理下から除外されます。

なお，自タスクを削除する場合には，exd_tsk システム・コールを発行します。

注意　このシステム・コールでは，削除要求のキューイングが行われません。このため，対象タスクが dormant **状態以外の場合には，戻り値として** E_OBJ **を返します。**

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	del_tsk システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大タスク生成数＜*tskid*）である
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	対象タスクが dormant 状態でない
	E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した
	E_CTX	－69	非タスクから del_tsk システム・コールを発行した

保守／廃止

start task (－23)

sta_tsk

タスク / 非タスク

【概　　要】

他タスクを起動する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = sta_tsk(ID tskid, INT stacd);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　*tskid*;	タスクの ID 番号
入	INT　*stacd*;	起動コード

【説　　明】

tskid で指定されたタスクを dormant 状態から ready 状態へと遷移させます。

これにより，対象タスクは RX850 Pro のスケジューリング対象となります。

stacd には，対象タスクに引き渡す起動コードを指定します。対象タスクは，起動コードを関数パラメータと同様に取り扱うことで操作可能となります。

注意　このシステム・コールでは，起動要求のキューイングが行われません。このため，対象タスクが dormant **状態以外の場合には，戻り値として** E_OBJ **を返します。**

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	sta_tsk システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大タスク生成数＜*tskid*）である
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	対象タスクが dormant 状態でない
	E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した

保守／廃止

exit task (－21)

ext_tsk

タスク

【概　　要】

自タスクを終了する。

【C 言語形式】

```
#include  <stdrx85p.h>
void      ext_tsk();
```

【パラメータ】

なし

【説　　明】

自タスクを run 状態から dormant 状態へと遷移させます。
これにより，自タスクは RX850 Pro のスケジューリング対象から除外されます。

備考 1. このシステム・コールでは，タスク生成時（コンフィギュレーション時または cre_tsk システム・コール発行時）に指定された“ タスク生成情報 ”は初期化されます。

2. タスクをアセンブリ言語で記述する場合，自タスクの終了は次のように記述してください。

```
        jr      _ext_tsk
```

注意 1. このシステム・コールを非タスクまたはディスパッチ禁止状態から発行した場合，動作は保証されません。

2. このシステム・コールでは，自タスクが終了する以前に獲得していた資源（メモリ・ブロック，セマフォ・カウントなど）の解放処理が行われません。したがって，このシステム・コールを発行する前に，ユーザ側で資源の解放処理を行ってください。

【戻　り　値】

なし

保守／廃止

exit and delete task (－22)

exd_tsk

タスク

【概　　要】

自タスクを終了したあと，削除する。

【C 言語形式】

```
#include  <stdrx85p.h>
void      exd_tsk();
```

【パラメータ】

なし

【説　　明】

自タスクを run 状態から non-existent 状態へと遷移させます。
これにより，自タスクは RX850 Pro の管理下から除外されます。

備考　タスクをアセンブリ言語で記述する場合，自タスクの終了，および削除は次のように記述してください。

```
        jr      _exd_tsk
```

注意 1. このシステム・コールを非タスクまたはディスパッチ禁止状態から発行した場合，動作は保証されません。
2. このシステム・コールでは，自タスクが終了する以前に獲得していた資源（メモリ・ブロック，セマフォ・カウントなど）の解放処理が行われません。したがって，このシステム・コールを発行する前に，ユーザ側で資源の解放処理を行ってください。

【戻　り　値】

なし

保守／廃止

terminate task (－25)

ter_tsk

タスク

【概　　要】

他タスクを強制的に終了する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ter_tsk(ID tskid);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　　*tskid*;	タスクの ID 番号

【説　　明】

tskid で指定されたタスクを強制的に dormant 状態へと遷移させます。

備考　このシステム・コールでは，タスク生成時（コンフィギュレーション時または cre_tsk システム・コール発行時）に指定された“ タスク生成情報 ”は初期化されます。

注意 1. このシステム・コールでは，終了要求のキューイングが行われません。このため，対象タスクが ready 状態，wait 状態，suspend 状態，wait_suspend 状態以外の場合には，戻り値として E_NOEXS または E_OBJ を返します。

2. このシステム・コールでは，自タスクが終了する以前に獲得していた資源（メモリ・ブロック，セマフォ・カウントなど）の解放処理が行われません。したがって，このシステム・コールを発行する前にユーザ側で，資源の解放処理を行ってください。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	ter_tsk システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大タスク数＜*tskid*）である
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	対象タスクが dormant 状態，または自タスクである
	E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した
	E_CTX	－69	非タスクから ter_tsk システム・コールを発行した

保守／廃止

disable dispatch (－30)

dis_dsp

タスク

【概　　要】

ディスパッチ処理を禁止する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = dis_dsp();
```

【パラメータ】

なし

【説　　明】

ディスパッチ処理（タスクのスケジューリング処理）を禁止します。

これにより，ena_dsp システム・コールが発行されるまでの間，ディスパッチ処理が禁止されます。

RX850 Pro では，dis_dsp システム・コールの発行から ena_dsp システム・コールの発行までの間に，タスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_sem など）が発行された場合は，待ちキューのキュー操作などの処理を行うだけで，実際のスケジューリング処理は ena_dsp システム・コールが発行されるまで遅延され，一括して行われます。

注意 1. このシステム・コールでは，禁止要求のキューイングが行われません。このため，すでに dis_dsp システム・コールが発行され，ディスパッチ処理が禁止されていた場合には，何も処理は行わず，エラーとしても扱いません。

2. RX850 Pro では，dis_dsp システム・コールの発行から ena_dsp システム・コールの発行までの間に自タスクを wait 状態へと遷移させる可能性のあるシステム・コール（wai_sem，wai_flg システム・コールなど）を発行した場合は，待ち条件の即時成立／不成立にかかわらず，戻り値として E_CTX を返します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	dis_dsp システム・コールが CF 定義されていない
*E_CTX	－69	コンテキスト・エラー ・非タスクから dis_dsp システム・コールを発行した ・loc_cpu システム・コールを発行後，dis_dsp システム・コールを発行した

保守／廃止

enable dispatch (－29)

ena_dsp

タスク

【概　　要】

ディスパッチ処理を許可する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ena_dsp();
```

【パラメータ】

なし

【説　　明】

ディスパッチ処理（タスクのスケジューリング処理）を許可します。

これにより，dis_dsp システム・コールの発行により禁止されていたディスパッチ処理が再開されます。

RX850 Pro では，dis_dsp システム・コールの発行から ena_dsp システム・コールの発行までの間に，タスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_sem など）が発行された場合は，待ちキューのキュー操作などの処理を行うだけで，実際のスケジューリング処理は ena_dsp システム・コールが発行されるまで遅延され，一括して行われます。

注意　このシステム・コールでは，再開要求のキューイングが行われません。このため，すでに ena_dsp **システム・コールが発行され，ディスパッチ処理が再開されていた場合には，何も処理は行わず，エラーとしても扱いません。**

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	ena_dsp システム・コールが CF 定義されていない
*E_CTX	－69	コンテキスト・エラー ・非タスクから ena_dsp システム・コールを発行した ・loc_cpu システム・コールを発行後，ena_dsp システム・コールを発行した

保守／廃止

change priority (－27)

chg_pri

タスク / 非タスク

【概　　要】

タスクの優先度を変更する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = chg_pri(ID tskid, PRI tskpri);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*tskid;*	タスクの ID 番号 TSK_SELF（0）：自タスク 数値　　：タスクの ID 番号
入	PRI　　*tskpri;*	タスクの優先度 TPRI_INI（0）　：タスクの起動時優先度 数値　　：タスクの優先度

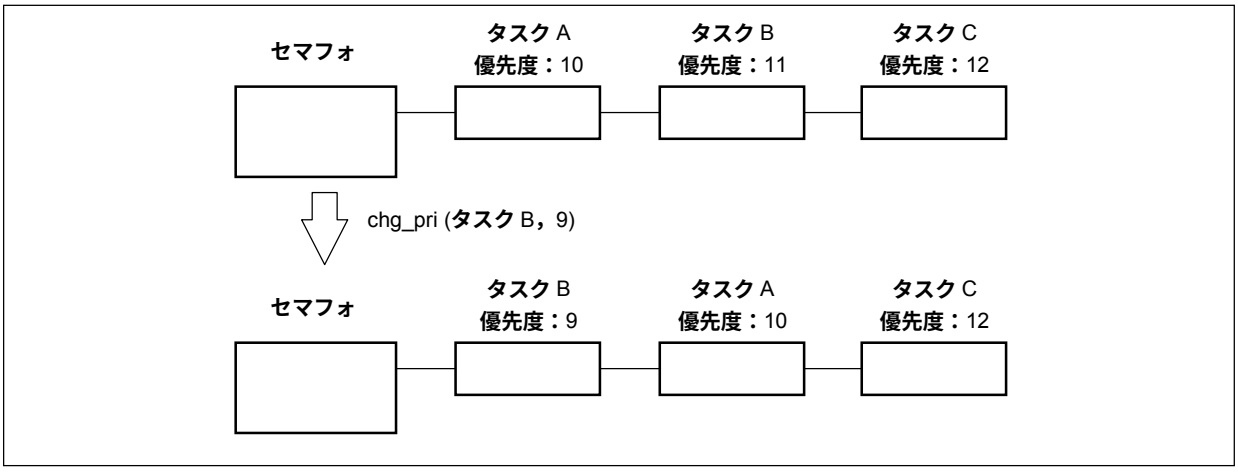
【説　　明】

tskid で指定されたタスクの優先度を *tskpri* で指定された値に変更します。

対象タスクが run 状態または ready 状態であった場合には，優先度の変更処理とともに，対象タスクを優先度に応じたレディ・キューの最後尾にキューイングしなおします。

備考 1. 対象タスクが何らかの待ちキューに優先度順でキューイングされている場合には，chg_pri システム・コールの発行により，待ち順序が変わることがあります。

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



保守／廃止

備考 2. *tskpri* **で指定された値は，次に** chg_pri **システム・コールが発行されるまで，または対象タスクが** dormant **状態へと遷移するまで有効になります。**

3. RX850 Pro **におけるタスクの優先度は，その値が小さいほど高い優先度であることを示します。**

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	chg_pri **システム・コールが** CF **定義されていない**
E_PAR	－33	**優先度の指定が不正（***tskpri***＜0，最大優先度＜***tskpri***）である**
E_ID	－35	ID **番号の指定が不正** **・最大タスク生成数＜***tskid* **・非タスクから** chg_pri **システム・コールを発行したとき，***tskid* **に** TSK_SELF **を指定した**
*E_NOEXS	－52	**対象タスクが存在していない**
*E_OBJ	－63	**対象タスクが** dormant **状態である**
E_OACV	－66	**アクセス権のない** ID **番号（***tskid***＜0）を指定した**

保守／廃止

rotate ready queue (－28)

rot_rdq

タスク / 非タスク

【概　　要】

タスクのレディ・キューを回転する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = rot_rdq(PRI tskpri);
```

【パラメータ】

入出力	パラメータ	説　　明
入	PRI　　*tskpri*;	タスクの優先度 TPRI_RUN（0）：run 状態のタスクの優先度 数値　　：タスクの優先度

【説　　明】

tskpri で指定された優先度に応じたレディ・キューの先頭タスクを最後尾にキューイングしなおします。

備考 1. 対象優先度のレディ・キューにタスクが 1 つもキューイングされていない場合には，何も処理は行わず，エラーとしても扱いません。

2. rot_rdq システム・コールを一定周期で発行することにより，ラウンドロビン・スケジューリングを実現できます。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	rot_rdq システム・コールが CF 定義されていない
E_PAR	－33	優先度の指定が不正（*tskpri*＜0，最大優先度＜*tskpri*）である

保守／廃止

release wait (－31)

rel_wai

タスク / 非タスク

【概　　要】

他タスクの wait 状態を強制的に解除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = rel_wai(ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*tskid*;	タスクの ID 番号

【説　　明】

tskid で指定されたタスクの wait 状態を強制的に解除します。

これにより，対象タスクは待ちキューから外れ，wait 状態から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

rel_wai システム・コールの発行により wait 状態を解除されたタスクには，wait 状態へと遷移するきっかけとなったシステム・コール（slp_tsk，wai_sem など）の戻り値として E_RLWAI が返されます。

注意　rel_wai システム・コールでは，suspend 状態の解除は行われません。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	rel_wai システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大タスク生成数＜*tskid*）である
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	対象タスクが wait 状態または wait_suspend 状態でない
	E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した

保守／廃止

get task identifier (－24)

get_tid

タスク / 非タスク

【概　　要】

タスクの ID 番号を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = get_tid(ID *p_tskid);
```

【パラメータ】

入出力	パラメータ	説　明
出	ID　*p_tskid;	ID 番号を格納する領域のアドレス

【説　　明】

自タスクの ID 番号を *p_tskid* で指定される領域に格納します。

注意　このシステム・コールを非タスクから発行した場合，*p_tskid* で指定される領域には FALSE（0）**が格納されます。**

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	get_tid システム・コールが CF 定義されていない
E_PAR	－33	ID 番号を格納する領域のアドレスが不正（*p_tskid*＝0）である

保守／廃止

refer task status (－20)

ref_tsk

タスク / 非タスク

【概　　要】

タスク情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ref_tsk(T_RTSK *pk_rtsk, ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
出	T_RTSK　*pk_rtsk;	タスク情報を格納するパケットの先頭アドレス
入	ID　　tskid;	タスクの ID 番号 TSK_SELF（0）：自タスク 数値　　：タスクの ID 番号

・タスク情報 T_RTSK の構造

```
typedef struct t_rtsk {
    VP    exinf;          /* 拡張情報 */
    PRI   tskpri;         /* 現在の優先度 */
    UINT  tskstat;        /* タスク状態 */
    UINT  tskwait;        /* 待ち要因 */
    ID    wid;            /* 待ちオブジェクト ID 番号 */
    INT   wupcnt;         /* 起床要求数 */
    INT   suscnt;         /* サスペンド要求数 */
    ID    keyid;          /* キーID 番号 */
} T_RTSK;
```

保守／廃止

【説　　明】

tskid で指定されたタスクの情報（拡張情報，現在の優先度など）を *pk_rtsk* で指定されるパケットに格納します。

次に，タスク情報の詳細を示します。

exinf … 拡張情報

tskpri … 現在の優先度

tskstat … タスクの状態

TTS_RUN（H'01）： run 状態

TTS_RDY（H'02）： ready 状態

TTS_WAI（H'04）： wait 状態

TTS_SUS（H'08）： suspend 状態

TTS_WAS（H'0c）： wait_suspend 状態

TTS_DMT（H'10）： dormant 状態

tskwait … wait 状態の種類

TTW_SLP（H'0001）： 起床待ち状態

TTW_DLY（H'0002）： 時間経過待ち状態

TTW_FLG（H'0010）： イベント・フラグ待ち状態

TTW_SEM（H'0020）： 資源待ち状態

TTW_MBX（H'0040）： メッセージ待ち状態

TTW_MPL（H'1000）： メモリ・ブロック待ち状態

wid … 待ちオブジェクト（セマフォ，イベント・フラグなど）の ID 番号

wupcnt … 起床要求数

suscnt … サスペンド要求数

keyid … キーID 番号

FALSE（0）： 生成時にキーID 番号が指定されていない

数値　　： キーID 番号

備考 1. tskstat の値が TTS_WAI，TTS_WAS 以外の場合，tskwait の内容は不定です。

2. tskwait の値が TTW_FLG，TTW_SEM，TTW_MBX，TTW_MPF 以外の場合，wid の内容は不定です。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	ref_tsk システム・コールが CF 定義されていない
E_PAR	－33	タスク情報を格納するパケットの先頭アドレスが不正（*pk_rtsk*＝0）である
E_ID	－35	ID 番号の指定が不正である ・最大タスク生成数＜*tskid* ・非タスクから ref_tsk システム・コールを発行したとき，*tskid* に TSK_SELF を指定した
*E_NOEXS	－52	対象タスクが存在していない
E_OACV	－66	アクセス権のない ID 番号（*tskid*＜0）を指定した

保守／廃止

get task Identifier (－248)

vget_tid

タスク / 非タスク

【概　　要】

タスクの ID 番号を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = vget_tid(ID *p_tskid, ID keyid);
```

【パラメータ】

入出力	パラメータ	説　明
出	ID　*p_tskid;	ID 番号を格納する領域のアドレス
入	ID　keyid;	タスクのキーID 番号

【説　　明】

keyid で指定されたタスクの ID 番号を *p_tskid* で指定される領域に格納します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	vget_tid システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・ID 番号を格納する領域のアドレスが不正（*p_tskid*＝0）である ・キーID 番号の指定が不正（*keyid*＝0）である
*E_NOEXS	－52	対象タスクが存在していない

保守／廃止

11.8.2　タスク付属同期機能システム・コール

ここでは，タスクに従属した同期操作を行うシステム・コールのグループ（タスク付属同期機能システム・コール）について説明します。

表 11－6 に，タスク付属同期機能システム・コールの一覧を示します。

表11－6　タスク付属同期機能システム・コール

システム・コール	機　　能
sus_tsk	他タスクを suspend 状態へ移行する
rsm_tsk	suspend 状態のタスクを再開する
frsm_tsk	suspend 状態のタスクを強制的に再開する
slp_tsk	自タスクを起床待ち状態へ移行する
tslp_tsk	自タスクを起床待ち状態へ移行する（タイムアウトあり）
wup_tsk	他タスクを起床する
can_wup	タスクの起床要求を無効化する

保守／廃止

suspend task (－33)

sus_tsk

タスク / 非タスク

【概　　要】

他タスクを suspend 状態へ移行する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = sus_tsk(ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*tskid*;	タスクの ID 番号

【説　　明】

tskid で指定されたタスクにサスペンド要求を発行（サスペンド要求カウンタに 0x1 を加算）します。

このシステム・コールを発行した際，対象タスクが ready 状態または wait 状態であった場合には，サスペンド要求の発行（サスペンド要求カウンタの加算処理）とともに，対象タスクを ready 状態から suspend 状態へ，または wait 状態から wait-suspend 状態へと遷移させます。

注意　RX850 Pro が管理するサスペンド要求カウンタは，7 ビット幅で構成されています。このため，sus_tsk システム・コールの発行でサスペンド要求数が 127 回を越えた場合には，サスペンド要求カウンタの加算処理は行わず，戻り値として E_QOVR を返します。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	sus_tsk システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大タスク生成数＜*tskid*）である
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	指定したタスクの状態が不適当である ・対象タスクが dormant 状態である ・タスクから sus_tsk システム・コールを発行した際，対象タスクに自タスクを指定した
	E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した
*	E_QOVR	－73	サスペンド要求数が 127 回を越えた

保守／廃止

resume task (－35)

rsm_tsk

タスク / 非タスク

【概　　要】

suspend 状態のタスクを再開する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = rsm_tsk(ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　*tskid*;	タスクの ID 番号

【説　　明】

tskid で指定されたタスクに発行されているサスペンド要求を 1 回分だけ解除（サスペンド要求カウンタから 0x1 を減算）します。

このシステム・コールの発行により対象タスクのサスペンド要求カウンタが 0x0 となった場合には，対象タスクを suspend 状態から ready 状態へ，または wait_suspend 状態から wait 状態へと遷移させます。

注意　このシステム・コールでは，解除要求のキューイングが行われません。このため，対象タスクが suspend **状態または** wait_suspend **状態でない場合には，サスペンド要求カウンタの減算処理は行わず，戻り値として** E_OBJ **を返します。**

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	rsm_tsk システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大タスク生成数＜*tskid*）である
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	対象タスクが suspend 状態または wait_suspend 状態でない
	E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した

保守／廃止

force resume task (－36)

frsm_tsk

タスク / 非タスク

【概　　要】

suspend 状態のタスクを強制的に再開する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = frsm_tsk(ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*tskid*;	タスクの ID 番号

【説　　明】

tskid で指定されたタスクに発行されているサスペンド要求をすべて解除（サスペンド要求カウンタに 0x0 を設定）します。

これにより，対象タスクは suspend 状態から ready 状態へ，または wait_suspend 状態から wait 状態へと遷移します。

注意　このシステム・コールでは，解除要求のキューイングが行われません。このため，対象タスクが suspend **状態または** wait_suspend **状態でない場合には，サスペンド要求カウンタの設定処理は行わず，戻り値として** E_OBJ **を返します。**

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	frsm_tsk システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大タスク生成数＜*tskid*）である
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	対象タスクが suspend 状態または wait_suspend 状態でない
	E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した

保守／廃止

sleep task (－38)

slp_tsk

タスク

【概　　要】

自タスクを起床待ち状態へ移行する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = slp_tsk();
```

【パラメータ】

なし

【説　　明】

自タスクに発行されている起床要求を 1 回分だけ解除（起床要求カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，自タスクの起床要求カウンタが 0x0 であった場合には，起床要求の解除（起床要求カウンタの減算処理）は行わず，自タスクを run 状態から wait 状態（起床待ち状態）へと遷移させます。

なお，起床待ち状態は wup_tsk，ret_wup，rel_wai システム・コールが発行されると解除され，自タスクは ready 状態へと遷移します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	slp_tsk システム・コールが CF 定義されていない
E_CTX	－69	コンテキスト・エラー ・非タスクから slp_tsk システム・コールを発行した ・ディスパッチ禁止状態から slp_tsk システム・コールを発行した
*E_RLWAI	－86	rel_wai システム・コールにより起床待ち状態を強制的に解除された

保守／廃止

sleep task with timeout (－37)

tslp_tsk

タスク

【概　　要】

自タスクを起床待ち状態へ移行する（タイムアウトあり）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = tslp_tsk(TMO tmout);
```

【パラメータ】

入出力	パラメータ	説　　明
入	TMO　*tmout*;	待ち時間（単位：ms） TMO_POL（0）　：即時リターン TMO_FEVR（－1）：永久待ち 数値　：待ち時間

【説　　明】

自タスクに発行されている起床要求を 1 回分だけ解除（起床要求カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，自タスクの起床要求カウンタが 0x0 であった場合には，起床要求の解除（起床要求カウンタの減算処理）は行わず，自タスクを run 状態から wait 状態（起床待ち状態）へと遷移させます。

なお，起床待ち状態は *tmout* で指定された待ち時間が経過するか，または wup_tsk，ret_wup，rel_wai システム・コールが発行されると解除され，自タスクは ready 状態へと遷移します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	tslp_tsk システム・コールが CF 定義されていない
E_PAR	－33	待ち時間の指定が不正（*tmout*＜TMO_FEVR）である
E_CTX	－69	コンテキスト・エラー ・非タスクから tslp_tsk システム・コールを発行した ・ディスパッチ禁止状態から tslp_tsk システム・コールを発行した
*E_TMOUT	－85	待ち時間が経過した
*E_RLWAI	－86	rel_wai システム・コールにより起床待ち状態を強制的に解除された

保守／廃止

wakeup task (－39)

wup_tsk

タスク / 非タスク

【概　　要】

他タスクを起床する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = wup_tsk(ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*tskid*;	タスクの ID 番号

【説　　明】

tskid で指定されたタスクに起床要求を発行（起床要求カウンタに 0x1 を加算）します。

ただし，このシステム・コールを発行した際，対象タスクが wait 状態（起床待ち状態）であった場合には，起床要求の発行（起床要求カウンタの加算処理）は行わず，対象タスクを起床待ち状態から ready 状態へと遷移させます。

注意　RX850 Pro が管理する起床要求カウンタは，7 ビット幅で構成されています。このため，wup_tsk システム・コールの発行により起床要求数が 127 回を越えた場合には，起床要求カウンタの加算処理は行わず，戻り値として E_QOVR を返します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	wup_tsk システム・コールが CF 定義されていない
E_ID	－35	ID 番号の指定が不正（最大タスク生成数＜*tskid*）である
*E_NOEXS	－52	対象タスクが存在していない
*E_OBJ	－63	指定したタスクの状態が不適当である ・対象タスクが dormant 状態である ・タスクから wup_tsk システム・コールを発行した際，対象タスクに自タスクを指定した
E_OACV	－66	アクセス権のない ID 番号（*tskid*≦0）を指定した
*E_QOVR	－73	起床要求数が 127 回を越えた

保守／廃止

cancel wakeup task (－40)

can_wup

タスク / 非タスク

【概　　要】

タスクの起床要求を無効化する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = can_wup(INT *p_wupcnt, ID tskid);
```

【パラメータ】

入出力	パラメータ	説　　明
出	INT　*p_wupcnt;	起床要求数を格納する領域のアドレス
入	ID　tskid;	タスクの ID 番号 TSK_SELF（0）：自タスク 数値　：タスクの ID 番号

【説　　明】

tskid で指定されたタスクに発行されている起床要求をすべて解除（起床要求カウンタに 0x0 を設定）します。

なお，このシステム・コールの発行により解除された起床要求数は，*p_wupcnt* で指定される領域に格納されます。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	can_wup システム・コールが CF 定義されていない
	E_PAR	－33	起床要求数を格納する領域のアドレスが不正（*p_wupcnt* ＝0）である
	E_ID	－35	ID 番号の指定が不正である ・最大タスク生成数＜*tskid* ・非タスクから can_wup システム・コールを発行した際，*tskid* に TSK_SELF を指定した
*	E_NOEXS	－52	対象タスクが存在していない
*	E_OBJ	－63	対象タスクが dormant 状態である
	E_OACV	－66	アクセス権のない ID 番号（*tskid*＜0）を指定した

保守／廃止

11.8.3　同期通信機能システム・コール

ここでは，タスク間の同期（排他制御，待ち合わせ），および通信を行うシステム・コールのグループ（同期通信機能システム・コール）について説明します。

表 11－7 に，同期通信機能システム・コールの一覧を示します。

表11－7　同期通信機能システム・コール

システム・コール	機　　能
cre_sem	セマフォを生成する
del_sem	セマフォを削除する
sig_sem	資源を返却する
wai_sem	資源を獲得する
preq_sem	資源を獲得する（ポーリング）
twai_sem	資源を獲得する（タイムアウトあり）
ref_sem	セマフォ情報を獲得する
vget_sid	セマフォの ID 番号を獲得する
cre_flg	イベント・フラグを生成する
del_flg	イベント・フラグを削除する
set_flg	ビット・パターンをセットする
clr_flg	ビット・パターンをクリアする
wai_flg	ビット・パターンをチェックする
pol_flg	ビット・パターンをチェックする（ポーリング）
twai_flg	ビット・パターンをチェックする（タイムアウトあり）
ref_flg	イベント・フラグ情報を獲得する
vget_fid	イベント・フラグの ID 番号を獲得する
cre_mbx	メールボックスを生成する
del_mbx	メールボックスを削除する
snd_msg	メッセージを送信する
rcv_msg	メッセージを受信する
prcv_msg	メッセージを受信する（ポーリング）
trcv_msg	メッセージを受信する（タイムアウトあり）
ref_mbx	メールボックス情報を獲得する
vget_mid	メールボックスの ID 番号を獲得する

保守／廃止

create semaphore (－49)

cre_sem

タスク

【概　　要】

セマフォを生成する。

【C 言語形式】

・ID 番号を指定する場合

```
#include <stdrx85p.h>
ER       ercd = cre_sem(ID semid, T_CSEM *pk_csem);
```

・ID 番号を指定しない場合

```
#include <stdrx85p.h>
ER       ercd = cre_sem(ID_AUTO, T_CSEM *pk_csem, ID *p_semid);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　*semid;*	セマフォの ID 番号
入	T_CSEM　**pk_csem;*	セマフォ生成情報を格納したパケットの先頭アドレス
出	ID　**p_semid;*	ID 番号を格納する領域のアドレス

・セマフォ生成情報 T_CSEM の構造

```
typedef struct t_csem {
    VP    exinf;          /* 拡張情報 */
    ATR   sematr;         /* セマフォの属性 */
    INT   isemcnt;        /* セマフォの初期資源数 */
    INT   maxsem;         /* セマフォの最大資源数 */
    ID    keyid;          /* セマフォのキーID 番号 */
} T_CSEM;
```

【説　　明】

RX850 Pro では，セマフォの生成において「ID 番号を指定して生成する」，「ID 番号を指定しないで生成する」の 2 種類のインタフェースを用意しています。

● ID 番号を指定する場合

pk_csem で指定された情報を基に，*semid* で指定された ID 番号を持つセマフォを生成します。

● ID 番号を指定しない場合

pk_csem で指定された情報を基に，セマフォを生成します。

ID 番号の割り付けは RX850 Pro により行われ，割り付けられた ID 番号は，*p_semid* で指定される領域に格納されます。

保守／廃止

次に，セマフォ生成情報の詳細を示します。

exinf …　**拡張情報**

対象セマフォに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。

exinf **に設定された情報は，処理プログラム（タスク，非タスク）から** ref_sem **システム・コールを発行することにより，ダイナミックに獲得できます。**

sematr …　**セマフォの属性**

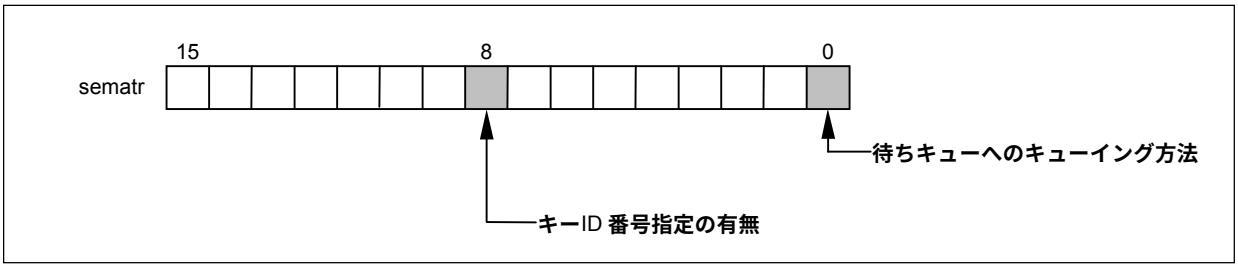
ビット 0 …　**待ちキューへのキューイング方法**

TA_TPRI（0）　**：優先度順**

TA_TFIFO（1）**：** FIFO **順**

ビット 8 …　**キー**ID **番号指定の有無**

TA_KEYID（1）**：キー**ID **番号を指定**



isemcnt …　**セマフォの初期資源数**

maxsem …　**セマフォの最大資源数**

keyid　…　**セマフォのキー**ID **番号**

備考　sematr **のビット** 8 **の値が** TA_KEYID **以外の場合，**keyid **の内容は意味を持ちません。**

保守／廃止

【戻　り　値】

*E_OK	0	正常終了
*E_NOMEM	－10	セマフォ管理ブロックの領域が確保できない
*E_NOSPT	－17	cre_sem システム・コールが CF 定義されていない
E_RSATR	－24	属性 sematr の指定が不正である
E_PAR	－33	パラメータの指定が不正である ・セマフォ生成情報を格納したパケットの先頭アドレスが不正（*pk_csem* ＝ 0）である ・初期資源数の指定が不正（isemcnt＜0）である ・最大資源数の指定が不正（maxsem≦0，maxsem＜isemcnt）である ・キーID 番号の指定が不正（keyid＝0）である（TA_KEYID 属性指定時） ・ID 番号を格納する領域のアドレスが不正（*p_semid*＝0）である（ID 番号を指定しないで生成する場合）
E_ID	－35	ID 番号の指定が不正（最大セマフォ生成数＜*semid*）である
*E_OBJ	－63	指定した ID 番号を持つセマフォがすでに生成されている
E_OACV	－66	アクセス権のない ID 番号（*semid*≦0）を指定した
E_CTX	－69	非タスクから cre_sem システム・コールを発行した

保守／廃止

delete semaphore (－50)

del_sem

タスク

【概　　要】

セマフォを削除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = del_sem(ID semid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*semid*;	セマフォの ID 番号

【説　　明】

semid **で指定されたセマフォを削除します。**

これにより，対象セマフォは RX850 Pro **の管理下から除外されます。**

del_sem **システム・コールの発行により** wait **状態（資源待ち状態）を解除されたタスクには，**wait **状態へと遷移するきっかけとなったシステム・コール（**wai_sem，twai_sem**）の戻り値として** E_DLT **が返されます。**

【戻　り　値】

*	E_OK	0	**正常終了**
*	E_NOSPT	－17	del_sem **システム・コールが** CF **定義されていない**
	E_ID	－35	ID **番号の指定が不正（最大セマフォ生成数＜***semid***）である**
*	E_NOEXS	－52	**対象セマフォが存在していない**
	E_OACV	－66	**アクセス権のない** ID **番号（***semid*≦0**）を指定した**
	E_CTX	－69	**非タスクから** del_sem **システム・コールを発行した**

保守／廃止

signal semaphore (－55)

sig_sem

タスク / 非タスク

【概　　要】

資源を返却する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = sig_sem(ID semid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*semid*;	セマフォの ID 番号

【説　　明】

semid で指定されたセマフォに資源を返却（セマフォ・カウンタに 0x1 を加算）します。

ただし，このシステム・コールを発行した際，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却処理（セマフォ・カウンタの加算処理）は行わず，該当タスク（待ちキューの先頭タスク）に資源を渡します。

これにより，該当タスクは待ちキューから外れ，wait 状態（資源待ち状態）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

注意　RX850 Pro が管理するセマフォ・カウンタは，生成時に指定された資源数としてとりうる最大資源数までしかカウントしません。したがって，sig_sem システム・コールの発行により，資源数が最大資源数を越えるような場合には，セマフォ・カウンタの加算処理は行われず，戻り値として E_QOVR が返されます。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	sig_sem システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大セマフォ生成数＜*semid*）である
*	E_NOEXS	－52	対象セマフォが存在していない
	E_OACV	－66	アクセス権のない ID 番号（*semid*≦0）を指定した
*	E_QOVR	－73	資源数が生成時に指定した最大資源数を越えた

保守／廃止

wait on semaphore (－53)

wai_sem

タスク

【概　　要】

資源を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = wai_sem(ID semid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*semid;*	セマフォの ID 番号

【説　　明】

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，対象セマフォから資源を獲得できなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューにキューイングしたあと，run 状態から wait 状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態は sig_sem，del_sem，rel_wai システム・コールが発行されると解除され，自タスクは ready 状態へと遷移します。

備考　自タスクを対象セマフォの待ちキューにキューイングする際は，対象セマフォ生成時（コンフィギュレーション時または cre_sem システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	wai_sem システム・コールが CF 定義されていない
E_ID	－35	ID 番号の指定が不正（最大セマフォ生成数＜*semid*）である
*E_NOEXS	－52	対象セマフォが存在していない
E_OACV	－66	アクセス権のない ID 番号（*semid*≦0）を指定した
E_CTX	－69	コンテキスト・エラー ・非タスクから wai_sem システム・コールを発行した ・ディスパッチ禁止状態から wai_sem システム・コールを発行した
*E_DLT	－81	del_sem システム・コールにより対象セマフォが削除された
*E_RLWAI	－86	rel_wai システム・コールにより資源待ち状態が強制的に解除された

保守／廃止

poll and request semaphore (－107)

preq_sem

タスク / 非タスク

【概　　要】

資源を獲得する（ポーリング）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = preq_sem(ID semid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*semid;*	セマフォの ID 番号

【説　　明】

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，対象セマフォから資源を獲得できなかった（空き資源が存在しなかった）場合には，戻り値として E_TMOUT を返します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	preq_sem システム・コールが CF 定義されていない
E_ID	－35	ID 番号の指定が不正（最大セマフォ生成数＜*semid*）である
*E_NOEXS	－52	対象セマフォが存在していない
E_OACV	－66	アクセス権のない ID 番号（*semid*≦0）を指定した
*E_TMOUT	－85	対象セマフォの資源数が 0x0 である

保守／廃止

wait on semaphore with timeout (－171)

twai_sem

タスク

【概　　要】

資源を獲得する（タイムアウトあり）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = twai_sem(ID semid, TMO tmout);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　*semid;*	セマフォの ID 番号
入	TMO　*tmout;*	待ち時間（単位：ms） TMO_POL（0）　：即時リターン TMO_FEVR（－1）　：永久待ち 数値　：待ち時間

【説　　明】

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，このシステム・コールを発行した際，対象セマフォから資源を獲得できなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューにキューイングしたあと，run 状態から wait 状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態は *tmout* で指定された時間が経過したとき，または sig_sem，del_sem，rel_wai システム・コールが発行されたときに解除され，自タスクは ready 状態へと遷移します。

備考　自タスクを対象セマフォの待ちキューにキューイングする際は，対象セマフォ生成時（コンフィギュレーション時または cre_sem システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	twai_sem システム・コールが CF 定義されていない
E_PAR	－33	待ち時間の指定が不正（*tmout*＜TMO_FEVR）である
E_ID	－35	ID 番号の指定が不正（最大セマフォ生成数＜*semid*）である
*E_NOEXS	－52	対象セマフォが存在していない
E_OACV	－66	アクセス権のない ID 番号（*semid*≦0）を指定した
E_CTX	－69	コンテキスト・エラー ・非タスクから twai_sem システム・コールを発行した ・ディスパッチ禁止状態から twai_sem システム・コールを発行した
*E_DLT	－81	del_sem システム・コールにより対象セマフォが削除された
*E_TMOUT	－85	待ち時間が経過した
*E_RLWAI	－86	rel_wai システム・コールにより資源待ち状態が強制的に解除された

保守／廃止

refer semaphore status (－52)

ref_sem

タスク / 非タスク

【概　　要】

セマフォ情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ref_sem(T_RSEM *pk_rsem, ID semid);
```

【パラメータ】

入出力	パラメータ	説　　明
出	T_RSEM　*pk_rsem;	セマフォ情報を格納するパケットの先頭アドレス
入	ID　　semid;	セマフォの ID 番号

・セマフォ情報 T_RSEM の構造

```
typedef struct t_rsem {
    VP        exinf;              /* 拡張情報 */
    BOOL_ID   wtsk;               /* 待ちタスクの有無 */
    INT       semcnt;             /* 現在の資源数 */
    INT       maxsem;             /* 最大資源数 */
    ID        keyid;              /* キーID 番号 */
} T_RSEM;
```

【説　　明】

semid で指定されたセマフォのセマフォ情報（拡張情報，待ちタスクの有無など）を *pk_rsem* で指定されるパケットに格納します。

次に，セマフォ情報の詳細を示します。

exinf …　拡張情報

wtsk …　待ちタスクの有無

FALSE（0）：待ちタスクなし

数値　　：待ちキューの先頭タスクの ID 番号

semcnt …　現在の資源数

maxsem …　生成時に指定した最大資源数

keyid …　キーID 番号

FALSE（0）：生成時にキーID 番号が指定されていない

数値　　：キーID 番号

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	ref_sem **システム・コールが** CF **定義されていない**
E_PAR	－33	**セマフォ情報を格納するパケットの先頭アドレスが不正（***pk_rsem* ＝0**）である**
E_ID	－35	ID **番号の指定が不正（最大セマフォ生成数＜***semid***）である**
*E_NOEXS	－52	**対象セマフォが存在していない**
E_OACV	－66	**アクセス権のない** ID **番号（***semid*≦0**）を指定した**

保守／廃止

get semaphore identifier (－246)

vget_sid

タスク / 非タスク

【概　　要】

セマフォの ID 番号を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = vget_sid(ID *p_semid, ID keyid);
```

【パラメータ】

入出力	パラメータ	説　明
出	ID　*p_semid;	ID 番号を格納する領域のアドレス
入	ID　keyid;	セマフォのキーID 番号

【説　　明】

keyid で指定されたセマフォの ID 番号を *p_semid* で指定される領域に格納します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	ref_sem システム・コールが CF 定義されていない
E_PAR	－33	セマフォ情報を格納するパケットの先頭アドレスが不正（*pk_rsem* ＝0）である ・キーID 番号の指定が不正（*keyid*＝0）である ・ID 番号を格納する領域のアドレスが不正（*p_semid*＝0）である
*E_NOEXS	－52	対象セマフォが存在していない

保守／廃止

create event flag (－41)

cre_flg

タスク

【概　　要】

イベント・フラグを生成する。

【C 言語形式】

・ID 番号を指定する場合

```
#include <stdrx85p.h>
ER       ercd = cre_flg(ID flgid, T_CFLG *pk_cflg);
```

・ID 番号を指定しない場合

```
#include <stdrx85p.h>
ER       ercd = cre_flg(ID_AUTO, T_CFLG *pk_cflg, ID *p_flgid);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　*flgid*;	イベント・フラグの ID 番号
入	T_CFLG　**pk_cflg*;	イベント・フラグ生成情報を格納したパケットの先頭アドレス
出	ID　**p_flgid*;	ID 番号を格納する領域のアドレス

・イベント・フラグ生成情報 T_CFLG の構造

```
typedef struct t_cflg {
    VP    exinf;            /* 拡張情報 */
    ATR   flgatr;           /* イベント・フラグの属性 */
    UINT  iflgptn;          /* イベント・フラグの初期ビット・パターン */
    ID    keyid;            /* イベント・フラグのキーID 番号 */
} T_CFLG;
```

【説　　明】

RX850 Pro では，イベント・フラグの生成において「ID 番号を指定して生成する」，「ID 番号を指定しないで生成する」の 2 種類のインタフェースを用意しています。

● ID 番号を指定する場合

pk_cflg で指定された情報を基に，*flgid* で指定された ID 番号を持つイベント・フラグを生成します。

● ID 番号を指定しない場合

pk_cflg で指定された情報を基に，イベント・フラグを生成します。

ID 番号の割り付けは RX850 Pro により行われ，割り付けられた ID 番号は，*p_flgid* で指定される領域に格納されます。

保守／廃止

次に，イベント・フラグ生成情報の詳細を示します。

exinf …　拡張情報

対象イベント・フラグに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。

exinf に設定された情報は，処理プログラム（タスク，非タスク）から ref_flg システム・コールを発行することにより，ダイナミックに獲得できます。

flgatr …　イベント・フラグの属性

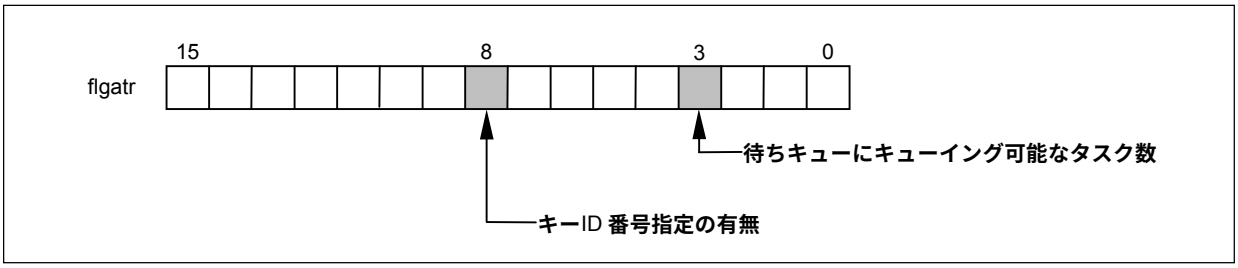
ビット 3 …　待ちキューにキューイング可能なタスク数

TA_WSGL（0）：1 タスクのみ

TA_WMUL（1）：複数タスク

ビット 8 …　キーID 番号指定の有無

TA_KEYID（1）：キーID 番号を指定



iflgptn …　イベント・フラグの初期ビット・パターン

keyid …　イベント・フラグのキーID 番号

備考　flgatr のビット 8 の値が TA_KEYID 以外の場合，keyid の内容は意味を持ちません。

【戻　り　値】

*E_OK	0	正常終了
*E_NOMEM	−10	イベント・フラグ管理ブロックの領域が確保できない
*E_NOSPT	−17	cre_flg システム・コールが CF 定義されていない
E_RSATR	−24	属性 flgatr の指定が不正である
E_PAR	−33	パラメータの指定が不正である ・イベント・フラグ生成情報を格納したパケットの先頭アドレスが不正（*pk_cflg*＝0）である ・キーID 番号の指定が不正（keyid＝0）である（TA_KEYID 属性指定時） ・ID 番号を格納する領域のアドレスが不正（*p_flgid*＝0）である（ID 番号を指定しないで生成する場合）
E_ID	−35	ID 番号の指定が不正（最大イベント・フラグ生成数＜*flgid*）である
*E_OBJ	−63	指定した ID 番号を持つイベント・フラグがすでに生成されている
E_OACV	−66	アクセス権のない ID 番号（*flgid*≦0）を指定した
E_CTX	−69	非タスクから cre_flg システム・コールを発行した

保守／廃止

delete event flag (－42)

del_flg

タスク

【概　　要】

イベント・フラグを削除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = del_flg(ID flgid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*flgid*;	**イベント・フラグの** ID **番号**

【説　　明】

flgid **で指定されたイベント・フラグを削除します。**

これにより，対象イベント・フラグは RX850 Pro **の管理下から除外されます。**

なお，このシステム・コールの発行により wait **状態（イベント・フラグ待ち）を解除されたタスクには，**wait **状態へと遷移するきっかけとなったシステム・コール（**wai_flg，twai_flg**）の戻り値として** E_DLT **が返されます。**

【戻　り　値】

*	E_OK	0	**正常終了**
*	E_NOSPT	－17	del_flg **システム・コールが** CF **定義されていない**
	E_ID	－35	ID **番号の指定が不正（最大イベント・フラグ生成数＜***flgid***）である**
*	E_NOEXS	－52	**対象イベントフラグが存在していない**
	E_OACV	－66	**アクセス権のない** ID **番号（***flgid*≦0**）を指定した**
	E_CTX	－69	**非タスクから** del_flg **システム・コールを発行した**

保守／廃止

set event flag (－48)

set_flg

タスク / 非タスク

【概　　要】

ビット・パターンをセットする。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = set_flg(ID flgid, UINT setptn);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　*flgid;*	イベント・フラグの ID 番号
入	UINT　*setptn;*	セットするビット・パターン（32 ビット幅）

【説　　明】

flgid で指定されたイベント・フラグのビット・パターンと *setptn* で指定されたビット・パターンの論理和（OR）をとり，その結果を対象イベント・フラグに設定します。

たとえば，このシステム・コールを発行するとき，対象イベント・フラグのビット・パターンが B’1100，*setptn* で指定されたビット・パターンが B’1010 の場合，対象イベント・フラグのビット・パターンは B’1110 となります。

なお，このシステム・コールを発行した際，対象イベント・フラグの待ちキューにキューイングされているタスクの待ち条件を満足した場合には，該当タスクを待ち状態から外します。

これにより，該当タスクは wait 状態（イベント・フラグ待ち）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	set_flg システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大イベント・フラグ生成数＜*flgid*）である
*	E_NOEXS	－52	対象イベントフラグが存在していない
	E_OACV	－66	アクセス権のない ID 番号（*flgid*≦0）を指定した

保守／廃止

clear event flag (－47)

clr_flg

タスク / 非タスク

【概　　要】

ビット・パターンをクリアする。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = clr_flg(ID flgid, UINT clrptn);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　*flgid;*	イベント・フラグの ID 番号
入	UINT　*clrptn;*	クリアするビット・パターン（32 ビット幅）

【説　　明】

flgid で指定されたイベント・フラグのビット・パターンと *clrptn* で指定されたビット・パターンの論理積（AND）をとり，その結果を対象イベント・フラグに設定します。

たとえば，このシステム・コールを発行するとき，対象イベント・フラグのビット・パターンが B’1100，*clrptn* で指定されたビット・パターンが B’1010 の場合，対象イベント・フラグのビット・パターンは B’1000 となります。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	clr_flg システム・コールが CF 定義されていない
E_ID	－35	ID 番号の指定が不正（最大イベント・フラグ生成数＜*flgid*）である
*E_NOEXS	－52	対象イベントフラグが存在していない
E_OACV	－66	アクセス権のない ID 番号（*flgid*≦0）を指定した

保守／廃止

wait event flag (－46)

wai_flg

タスク

【概　　要】

ビット・パターンをチェックする。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = wai_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
```

【パラメータ】

入出力	パラメータ	説　明
出	UINT *p_flgptn;	条件成立時のビット・パターンを格納する領域のアドレス
入	ID flgid;	イベント・フラグの ID 番号
入	UINT waiptn;	要求するビット・パターン（32 ビット幅）
入	UINT wfmode;	待ち条件／条件成立時の指定 TWF_ANDW（0）　：AND 待ち TWF_ORW（2）　：OR 待ち TWF_CLR（1）　：ビット・パターンをクリアする

【説　　明】

waiptn で指定された要求ビット・パターンと *wfmode* で指定された待ち条件を満足するビット・パターンが，*flgid* で指定されたイベント・フラグに設定されているかどうかをチェックします。

待ち条件を満足するビット・パターンが対象イベント・フラグに設定されていた場合には，対象イベント・フラグのビット・パターンを *p_flgptn* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象イベント・フラグのビット・パターンが待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ちキューの最後尾にキューイングしたあと，run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ちの状態は，set_flg システム・コールの発行により待ち条件を満足するようなビット・パターンが設定されたとき，または del_flg，rel_wai システム・コールが発行されたときに解除され，自タスクは ready 状態へと遷移します。

次に，*wfmode* の指定形式を示します。

- *wfmode*＝TWF_ANDW

  waiptn で“1”を設定している全ビットが対象イベント・フラグに設定されているかどうかをチェックします。

- *wfmode*＝（TWF_ANDW｜TWF_CLR）

  waiptn で“1”を設定している全ビットが対象イベント・フラグに設定されているかどうかをチェックします。

  なお，待ち条件が満足されたときには，対象イベント・フラグのビット・パターンがクリア（B’0000 を設定）されます。

保守／廃止

● *wfmode*＝TWF_ORW

waiptn で“1”を設定しているビットのうち，いずれかのビットが対象イベント・フラグに設定されているかどうかをチェックします。

● *wfmode*＝（TWF_ORW｜TWF_CLR）

waiptn で“1”を設定しているビットのうち，いずれかのビットが対象イベント・フラグに設定されているかどうかをチェックします。

なお，待ち条件が満足されたときには，対象イベント・フラグのビット・パターンがクリア（B'0000を設定）されます。

注意 1. RX850 Pro では，イベント・フラグの待ちキューにキューイング可能なタスク数を，生成時（コンフィギュレーション時または cre_flg システム・コール発行時）に次のいずれかに指定します。

TA_WSGL 属性：1 タスクのみキューイング可能
TA_WMUL 属性：複数タスクをキューイング可能

このため，すでに待ちタスクがキューイングされている TA_WSGL 属性のイベント・フラグに対して wai_flg システム・コールを発行した場合には，ビット・パターンのチェック処理は行わず，戻り値として E_OBJ を返します。

2. del_flg，rel_wai システム・コールの発行によりイベント・フラグ待ち状態が強制的に解除された場合には，*p_flgptn* で指定される領域の内容は不定になります。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	wai_flg システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・条件成立時のビット・パターンを格納する領域のアドレスが不正（*p_flgptn*＝0）である ・要求するビット・パターンの指定が不正（*waiptn*＝0）である ・待ち条件／条件成立時 *wfmode* の指定が不正である
E_ID	－35	ID 番号の指定が不正（最大イベント・フラグ生成数＜*flgid*）である
*E_NOEXS	－52	対象イベントフラグが存在していない
*E_OBJ	－63	すでに待ちタスクがキューイングされている TA_WSGL 属性のイベント・フラグに対して，wai_flg システム・コールを発行した
E_OACV	－66	アクセス権のない ID 番号（*flgid*≦0）を指定した
E_CTX	－69	コンテキスト・エラー ・非タスクから wai_flg システム・コールを発行した ・ディスパッチ禁止状態から wai_flg システム・コールを発行した
*E_DLT	－81	del_flg システム・コールにより対象イベント・フラグが削除された
*E_RLWAI	－86	rel_wai システム・コールによりイベント・フラグ待ち状態が強制的に解除された

保守／廃止

poll event flag (－106)

pol_flg

タスク / 非タスク

【概　　要】

ビット・パターンをチェックする（ポーリング）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = pol_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
```

【パラメータ】

入出力	パラメータ	説　　明
出	UINT　*p_flgptn;	条件成立時のビット・パターンを格納する領域のアドレス
入	ID　flgid;	イベント・フラグの ID 番号
入	UINT　waiptn;	要求するビット・パターン（32 ビット幅）
入	UINT　wfmode;	待ち条件／条件成立時の指定 TWF_ANDW（0）　：AND 待ち TWF_ORW（2）　：OR 待ち TWF_CLR（1）　：ビット・パターンをクリアする

【説　　明】

waiptn で指定された要求ビット・パターンと *wfmode* で指定された待ち条件が *flgid* で指定されたイベント・フラグに設定されているかどうかをチェックします。

待ち条件を満足するピット・パターンが対象イベント・フラグに設定されていた場合には，対象イベント・フラグのビット・パターンを *p_flgptn* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象イベント・フラグのビット・パターンが待ち条件を満足していなかった場合には，戻り値として E_TMOUT が返されます。

次に，*wfmode* の指定形式を示します。

- *wfmode*＝TWF_ANDW

  waiptn で“ 1 ”を設定している全ビットが対象イベント・フラグに設定されているかどうかをチェックします。

- *wfmode*＝（TWF_ANDW｜TWF_CLR）

  waiptn で“ 1 ”を設定している全ビットが対象イベント・フラグに設定されているかどうかをチェックします。

  なお，待ち条件が満足されていたときには，対象イベント・フラグのビット・パターンがクリア（B’0000 を設定）されます。

保守／廃止

● *wfmode*＝TWF_ORW

waiptn で“1”を設定しているビットのうち，いずれかのビットが対象イベント・フラグに設定されているかどうかをチェックします。

● *wfmode*＝（TWF_ORW｜TWF_CLR）

waiptn で“1”を設定しているビットのうち，いずれかのビットが対象イベント・フラグに設定されているかどうかをチェックします。

なお，待ち条件が満足されていたときには，対象イベント・フラグのビット・パターンがクリア（B’0000を設定）されます。

注意　RX850 Pro では，イベント・フラグの待ちキューにキューイング可能なタスク数を，生成時（コンフィギュレーション時または cre_flg システム・コール発行時）に次のいずれかに指定します。

TA_WSGL 属性 ：1 タスクのみキューイング可能
TA_WMUL 属性 ：複数タスクをキューイング可能

このため，すでに待ちタスクがキューイングされている TA_WSGL 属性のイベント・フラグに対して wai_flg システム・コールを発行した場合には，ビット・パターンのチェック処理は行わず，戻り値として E_OBJ を返します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	pol_flg システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・条件成立時のビット・パターンを格納する領域のアドレスが不正（*p_flgptn*＝0）である ・要求するビット・パターンの指定が不正（*waiptn*＝0）である ・待ち条件／条件成立時 *wfmode* の指定が不正である
E_ID	－35	ID 番号の指定が不正（最大イベント・フラグ生成数＜*flgid*）である
*E_NOEXS	－52	対象イベント・フラグが存在していない
*E_OBJ	－63	すでに待ちタスクがキューイングされている TA_WSGL 属性のイベント・フラグに対して pol_flg システム・コールを発行した
E_OACV	－66	アクセス権のない ID 番号（*flgid*≦0）を指定した
*E_TMOUT	－85	対象イベント・フラグのビット・パターンが待ち条件を満足していない

保守／廃止

wait event flag with timeout (－170)

twai_flg

タスク

【概　　要】

ビット・パターンをチェックする（タイムアウトあり）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER    ercd = twai_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode, TMO tmout);
```

【パラメータ】

入出力	パラメータ	説　　明
出	UINT　*p_flgptn;	条件成立時のビット・パターンを格納する領域のアドレス
入	ID　flgid;	イベント・フラグの ID 番号
入	UINT　waiptn;	要求するビット・パターン（32 ビット幅）
入	UINT　wfmode;	待ち条件／条件成立時の指定 TWF_ANDW（0）　：AND 待ち TWF_ORW（2）　：OR 待ち TWF_CLR（1）　：ビット・パターンをクリアする
入	TMO　tmout;	待ち時間（単位：ms） TMO_POL（0）　：即時リターン TMO_FEVR（－1）：永久待ち 数値　：待ち時間

【説　　明】

waiptn で指定された要求ビット・パターンと *wfmode* で指定された待ち条件が *flgid* で指定されたイベント・フラグに設定されているかどうかをチェックします。

待ち条件を満足するピット・パターンが対象イベント・フラグに設定されていた場合には，対象イベント・フラグのビット・パターンを *p_flgptn* で指定される領域に格納します。

ただし，このシステム・コール発行した際，対象イベント・フラグのビット・パターンが待ち条件を満たしていなかった場合には，自タスクを対象イベント・フラグの待ちキューの最後尾にキューイングしたあと，run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させます。

なお，イベント・フラグ待ちの状態は，*tmout* で指定された待ち時間が経過したとき，set_flg システム・コールの発行により待ち条件を満足するようなビット・パターンが設定されたとき，または del_flg，rel_wai システム・コールが発行されたときに解除され，自タスクは ready 状態へと遷移します。

次に，*wfmode* の指定形式を示します。

保守／廃止

● *wfmode*＝TWF_ANDW

waiptn で“1”を設定している全ビットが対象イベント・フラグに設定されているかどうかをチェックします。

● *wfmode*＝（TWF_ANDW｜TWF_CLR）

waiptn で“1”を設定している全ビットが対象イベント・フラグに設定されているかどうかをチェックします。

なお，待ち条件が満足されたときには，対象イベント・フラグのビット・パターンがクリア（B’0000を設定）されます。

● *wfmode*＝TWF_ORW

waiptn で“1”を設定しているビットのうち，いずれかのビットが対象イベント・フラグに設定されているかどうかをチェックします。

● *wfmode*＝（TWF_ORW｜TWF_CLR）

waiptn で“1”を設定しているビットのうち，いずれかのビットが対象イベント・フラグに設定されているかどうかをチェックします。

なお，待ち条件が満足されたときには，対象イベント・フラグのビット・パターンがクリア（B’0000を設定）されます。

注意 1. RX850 Pro では，イベント・フラグの待ちキューにキューイング可能なタスク数を，生成時（コンフィギュレーション時または cre_flg システム・コール発行時）に次のいずれかに指定します。

TA_WSGL 属性：1 タスクのみキューイング可能
TA_WMUL 属性：複数タスクをキューイング可能

このため，すでに待ちタスクがキューイングされている TA_WSGL 属性のイベント・フラグに対して wai_flg システム・コールを発行した場合には，ビット・パターンのチェック処理は行わず，戻り値として E_OBJ を返します。

2. del_flg，rel_wai システム・コールの発行によりイベント・フラグ待ち状態が強制的に解除された場合には，*p_flgptn* で指定される領域の内容は不定になります。

保守／廃止

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	twai_flg システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・条件成立時のビット・パターンを格納する領域のアドレスが不正（*p_flgptn*＝0）である ・要求するビット・パターンの指定が不正（*waiptn*＝0）である ・待ち条件／条件成立時 *wfmode* の指定が不正である ・待ち時間の指定が不正（*tmout*＜TMO_FEVR）である
E_ID	－35	ID 番号の指定が不正（最大イベント・フラグ生成数＜*flgid*）である
*E_NOEXS	－52	対象イベント・フラグが存在していない
*E_OBJ	－63	すでに待ちタスクがキューイングされている TA_WSGL 属性のイベント・フラグに対して twai_flg システム・コールを発行した
E_OACV	－66	アクセス権のない ID 番号（*flgid*≦0）を指定した
E_CTX	－69	コンテキスト・エラー ・非タスクから twai_flg システム・コールを発行した ・ディスパッチ禁止状態から twai_flg システム・コールを発行した
*E_DLT	－81	del_flg システム・コールにより対象イベント・フラグが削除された
*E_TMOUT	－85	待ち時間が経過した
*E_RLWAI	－86	rel_wai システム・コールによりイベント・フラグ待ち状態が強制的に解除された

保守／廃止

refer event flag status (－44)

ref_flg

タスク / 非タスク

【概　　要】

イベント・フラグ情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ref_flg(T_RFLG *pk_rflg, ID flgid);
```

【パラメータ】

入出力	パラメータ	説　　明
出	T_RFLG　*pk_rflg;	イベント・フラグ情報を格納するパケットの先頭アドレス
入	ID　　flgid;	イベントフラグの ID 番号

・イベント・フラグ情報 T_RFLG の構造

```
typedef struct t_rflg {
    VP          exinf;          /* 拡張情報 */
    BOOL_ID     wtsk;           /* 待ちタスクの有無 */
    UINT        flgptn;         /* 現在のビット・パターン */
    ID          keyid;          /* キーID 番号 */
} T_RFLG;
```

【説　　明】

flgid で指定されたイベント・フラグのイベント・フラグ情報（拡張情報，待ちタスクの有無など）を *pk_rflg* で指定されるパケットに格納します。

次に，イベント・フラグ情報の詳細を示します。

exinf … 拡張情報

wtsk … 待ちタスクの有無

FALSE（0）：待ちタスクなし

数値　　：待ちキューの先頭タスクの ID 番号

flgptn … 現在のビット・パターン

keyid … キーID 番号

FALSE（0）：生成時にキーID 番号が指定されていない

数値　　：キーID 番号

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	ref_flg **システム・コールが** CF **定義されていない**
E_PAR	－33	**イベント・フラグ情報を格納するパケットの先頭アドレスが不正（***pk_rflg***＝0）である**
E_ID	－35	ID **番号の指定が不正（最大イベント・フラグ生成数＜***flgid***）**
*E_NOEXS	－52	**対象イベント・フラグが存在していない**
E_OACV	－66	**アクセス権のない** ID **番号（***flgid***≦0）を指定した**

保守／廃止

get event flag identifier (－247)

vget_fid

タスク / 非タスク

【概　　要】

イベント・フラグの ID 番号を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = vget_fid(ID *p_flgid, ID keyid);
```

【パラメータ】

入出力	パラメータ	説　　明
出	ID　*p_flgid;	ID 番号を格納する領域のアドレス
入	ID　keyid;	イベント・フラグのキーID 番号

【説　　明】

keyid で指定されたイベント・フラグの ID 番号を *p_flgid* で指定される領域に格納します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	vget_fid システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・キーID 番号の指定が不正（*keyid*＝0）である ・ID 番号を格納する領域のアドレスが不正（*p_flgid*＝0）である
*E_NOEXS	－52	対象イベント・フラグが存在していない

保守／廃止

create mailbox (－57)

cre_mbx

タスク

【概　　要】

メールボックスを生成する。

【C 言語形式】

・ID 番号を指定する場合

```
#include <stdrx85p.h>
ER       ercd = cre_mbx(ID mbxid, T_CMBX *pk_cmbx);
```

・ID 番号を指定しない場合

```
#include <stdrx85p.h>
ER       ercd = cre_mbx(ID_AUTO, T_CMBX *pk_cmbx, ID *p_mbxid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*mbxid*;	メールボックスの ID 番号
入	T_CMBX　　**pk_cmbx*;	メールボックス生成情報を格納したパケットの先頭アドレス
出	ID　　**p_mbxid*;	ID 番号を格納する領域のアドレス

・メールボックス生成情報 T_CMBX の構造

```
typedef struct t_cmbx {
    VP    exinf;          /* 拡張情報 */
    ATR   mbxatr;         /* メールボックスの属性 */
    ID    keyid;          /* メールボックスのキーID 番号 */
} T_CMBX;
```

【説　　明】

RX850 Pro では，メールボックスの生成において「ID 番号を指定して生成する」，「ID 番号を指定しないで生成する」の 2 種類のインタフェースを用意しています。

● ID 番号を指定する場合

pk_cmbx で指定された情報を基に，*mbxid* で指定された ID 番号を持つメールボックスを生成します。

● ID 番号を指定しない場合

pk_cmbx で指定された情報を基に，メールボックスを生成します。

ID 番号の割り付けは RX850 Pro により行われ，割り付けられた ID 番号は，*p_mbxid* で指定される領域に格納されます。

保守／廃止

次に，メールボックス生成情報の詳細を示します。

exinf …　拡張情報

exinf は対象メールボックスに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。

exinf に設定された情報は，処理プログラム（タスク，非タスク）から ref_mbx システム・コールを発行することにより，ダイナミックに獲得できます。

mbxatr …　メールボックスの属性

ビット 0 …　タスク用待ちキューへのキューイング方法

TA_TPRI（0）　：　優先度順

TA_TFIFO（1）：　FIFO 順

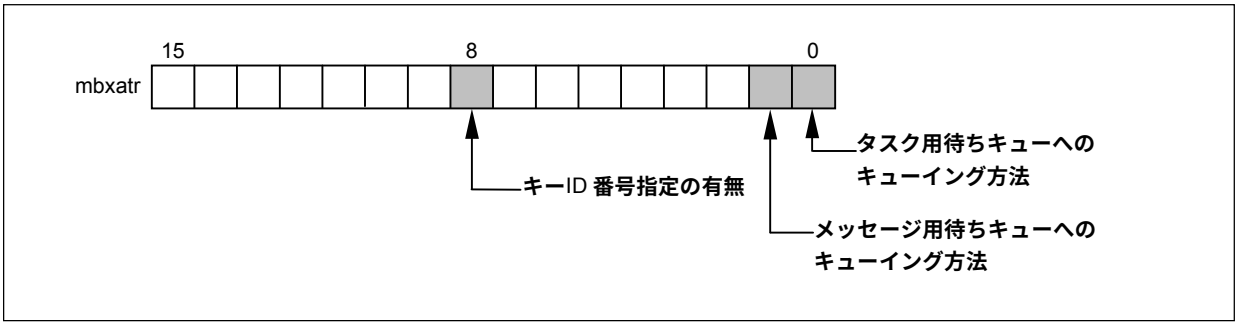
ビット 1 …　メッセージ用待ちキューへのキューイング方法

TA_MPRI（0）　：　優先度順

TA_MFIFO（1）：　FIFO 順

ビット 8 …　キーID 番号指定の有無

TA_KEYID（1）：　キーID 番号を指定



keyid …　メールボックスのキーID 番号

備考　mbxatr のビット 8 の値が TA_KEYID 以外の場合，keyid の内容は意味を持ちません。

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOMEM	－10	**メールボックス管理ブロックの領域が確保できない**
*E_NOSPT	－17	cre_mbx **システム・コールが** CF **定義されていない**
E_RSATR	－24	**属性** mbxatr **の指定が不正である**
E_PAR	－33	**パラメータの指定が不正である** **・メールボックス生成情報を格納したパケットの先頭アドレスが不正（***pk_cmbx***＝0）である** **・キー**ID **番号の指定が不正（***keyid***＝0）である（**TA_KEYID **指定時）** **・**ID **番号を格納する領域のアドレスが不正（***p_mbxid***＝0）である（**ID **番号を指定しないで生成する場合）**
E_ID	－35	ID **番号の指定が不正（最大メールボックス生成数＜***mbxid***）である**
*E_OBJ	－63	**指定した** ID **番号を持つメールボックスがすでに生成されている**
E_OACV	－66	**アクセス権のない** ID **番号（***mbxid***≦0）を指定した**
E_CTX	－69	**非タスクから** cre_mbx **システム・コールを発行した**

保守／廃止

delete mailbox (－58)

del_mbx

タスク

【概　　要】

メールボックスを削除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = del_mbx(ID mbxid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　　*mbxid;*	メールボックスの ID 番号

【説　　明】

mbxid で指定されたメールボックスを削除します。

これにより，対象メールボックスは RX850 Pro の管理下から除外されます。

なお，このシステム・コールの発行により wait 状態（メッセージ待ち状態）を解除されたタスクには，wait 状態へと遷移するきっかけとなったシステム・コール（rcv_msg，trcv_msg）の戻り値として E_DLT が返されます。

備考　このシステム・コールを発行したときに，対象メールボックスのメッセージ用待ちキューにメモリ・プールから獲得したメモリ･ブロックを使用したメッセージがキューイングされていた場合，メッセージ（メモリ・ブロック）は該当メモリ・プールに返却されます。
このため，メモリ・プールから獲得したメモリ・ブロック以外の領域をメッセージとして使用していた場合には，動作保証外となりますので，このシステム・コールは発行しないでください。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	del_mbx システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大メールボックス生成数＜*mbxid*）である
*	E_NOEXS	－52	対象メールボックスが存在していない
	E_OACV	－66	アクセス権のない ID 番号（*mbxid*≦0）を指定した
	E_CTX	－69	非タスクから del_mbx システム・コールを発行した

保守／廃止

send message (－63)

snd_msg

タスク / 非タスク

【概　　要】

メッセージを送信する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = snd_msg(ID mbxid, T_MSG *pk_msg);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　*mbxid;*	メールボックスの ID 番号
入	T_MSG　**pk_msg;*	メッセージを格納したパケットの先頭アドレス

・メッセージ T_MSG の構造

```
typedef struct t_msg {
    VW    msgrfu;          /* メッセージ管理領域 */
    PRI   msgpri;          /* メッセージの優先度 */
    VB    msgcont[];       /* メッセージ本体 */
} T_MSG;
```

【説　　明】

mbxid で指定されたメールボックスに *pk_msg* で指定されたメッセージを送信（メッセージ用待ちキューにメッセージをキューイング）します。

ただし，このシステム・コールを発行した際，対象メールボックスのタスク用待ちキューにタスクがキューイングされていた場合には，メッセージのキューイング操作は行われず，該当タスク（タスク用待ちキューの先頭タスク）にメッセージを渡します。

これにより，該当タスクはタスク用待ちキューから外れ，wait 状態（メッセージ待ち状態）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

備考　メッセージを対象メールボックスのメッセージ用待ちキューにキューイングする際は，対象メールボックス生成時（コンフィギュレーション時または cre_mbx システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

保守／廃止

注意　RX850 Pro **では，メッセージの先頭** 4 **バイト（メッセージ管理領域** msgrfu**）をメッセージ用待ちキューにキューイングする際のリンク領域として使用します。このため，メッセージを対象メールボックスに送信する場合は，**snd_msg **システム・コールを発行する前に** msgrfu **に“** 0x0 **”を設定する必要があります。**

snd_msg **システム・コールを発行した際，**msgrfu **に“** 0x0 **”以外の値が設定されていた場合には，**RX850 Pro **が「対象メッセージはすでにメッセージ用待ちキューにキューイングされている」と判断し，メッセージの送信処理は行わず，戻り値として** E_OBJ **を返します。**

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	snd_msg **システム・コールが** CF **定義されていない**
E_PAR	－33	**メッセージを格納したパケットの先頭アドレスが不正（***pk_msg***＝0）である**
E_ID	－35	ID **番号の指定が不正（最大メールボックス生成数＜***mbxid***）である**
*E_NOEXS	－52	**対象メールボックスが存在していない**
E_OBJ	－63	**指定したメッセージ用の領域がすでにメッセージとして使用されている**
E_OACV	－66	**アクセス権のない** ID **番号（***mbxid***≦0）を指定した**

保守／廃止

receive message from mailbox (－61)

rcv_msg

タスク

【概　　要】

メッセージを受信する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = rcv_msg(T_MSG **ppk_msg, ID mbxid);
```

【パラメータ】

入出力	パラメータ	説　明
出	T_MSG　**ppk_msg;	メッセージの先頭アドレスを格納する領域のアドレス
入	ID　mbxid;	メールボックスの ID 番号

【説　　明】

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象メールボックスからメッセージを受信できなかった（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク待ち用待ちキューにキューイングしたあと，run 状態から wait 状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態は snd_msg，del_mbx，rel_wai システム・コールが発行されると解除され，自タスクは ready 状態へと遷移します。

備考　自タスクを対象メールボックスのタスク用待ちキューにキューイングする際は，対象メールボックス生成時（コンフィギュレーション時または cre_mbx システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	rcv_msg システム・コールが CF 定義されていない
	E_PAR	－33	メッセージの先頭アドレスを格納する領域のアドレスが不正（*ppk_msg*＝0）である
	E_ID	－35	ID 番号の指定が不正（最大メールボックス生成数＜*mbxid*）である
*	E_NOEXS	－52	対象メールボックスが存在していない
	E_OACV	－66	アクセス権のない ID 番号（*mbxid*≦0）を指定した
	E_CTX	－69	コンテキスト・エラー ・非タスクから rcv_msg システム・コールを発行した ・ディスパッチ禁止状態から rcv_msg システム・コールを発行した
*	E_DLT	－81	del_mbx システム・コールにより対象メールボックスが削除された
*	E_RLWAI	－86	rel_wai システム・コールによりメッセージ待ち状態が強制的に解除された

保守／廃止

poll and receive message from mailbox (－108)

prcv_msg

タスク / 非タスク

【概　　要】

メッセージを受信する（ポーリング）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = prcv_msg(T_MSG **ppk_msg, ID mbxid);
```

【パラメータ】

入出力	パラメータ	説　　明
出	T_MSG　**ppk_msg;	メッセージの先頭アドレスを格納する領域のアドレス
入	ID　mbxid;	メールボックスの ID 番号

【説　　明】

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象メールボックスからメッセージを受信できなかった（メッセージ用待ちキューにメッセージが存在しなかった）場合には，戻り値として E_TMOUT が返されます。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	prcv_msg システム・コールが CF 定義されていない
	E_PAR	－33	メッセージの先頭アドレスを格納する領域アドレスが不正（*ppk_msg*＝0）である
	E_ID	－35	ID 番号の指定が不正（最大メールボックス生成数＜*mbxid*）である
*	E_NOEXS	－52	対象メールボックスが存在していない
	E_OACV	－66	アクセス権のない ID 番号（*mbxid*≦0）を指定した
*	E_TMOUT	－85	対象メールボックスにメッセージが存在していない

保守／廃止

receive message from mailbox with timeout (－172)

trcv_msg

タスク

【概　　要】

メッセージを受信する（タイムアウトあり）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = trcv_msg(T_MSG **ppk_msg, ID mbxid, TMO tmout);
```

【パラメータ】

入出力	パラメータ	説　　明
出	T_MSG　**ppk_msg;	メッセージの先頭アドレスを格納する領域のアドレス
入	ID　mbxid;	メールボックスの ID 番号
入	TMO　tmout;	待ち時間（単位：基本クロック周期） TMO_POL（0）　：即時リターン TMO_FEVR（－1）：永久待ち 数値　：待ち時間

【説　　明】

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象メールボックスからメッセージを受信できなかった（メッセージ用待ちキューにメッセージが存在しなかった）場合には，自タスクを対象メールボックスのタスク用待ちキューにキューイングしたあと，run 状態から wait 状態（メッセージ待ち状態）へと遷移させます。

なお，メッセージ待ち状態は *tmout* で指定された待ち時間が経過したとき，または snd_msg，del_mbx，rel_wai システム・コールが発行発行されたときに解除され，自タスクは ready 状態へ遷移します。

備考　自タスクを対象メールボックスのタスク用待ちキューにキューイングする際は，対象メールボックス生成時（コンフィギュレーション時または cre_mbx システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

保守／廃止

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	trcv_msg システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・メッセージの先頭アドレスを格納する領域のアドレスが不正（*ppk_msg*＝0）である
E_ID	－35	ID 番号の指定が不正（最大メールボックス生成数＜*mbxid*）である
*E_NOEXS	－52	対象メールボックスが存在していない
E_OACV	－66	アクセス権のない ID 番号（*mbxid*≦0）を指定した
E_CTX	－69	コンテキスト・エラー ・非タスクから trcv_msg システム・コールを発行した ・ディスパッチ禁止状態から trcv_msg システム・コールを発行した
*E_DLT	－81	del_mbx システム・コールによりメールボックスが削除された
*E_TMOUT	－85	待ち時間が経過した
*E_RLWAI	－86	rel_wai システム・コールによりメッセージ待ち状態が強制的に解除された

保守／廃止

refer mailbox status (－60)

ref_mbx

タスク / 非タスク

【概　　要】

メールボックス情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ref_mbx(T_RMBX *pk_rmbx, ID mbxid);
```

【パラメータ】

入出力	パラメータ		説　　明
出	T_RMBX	*pk_rmbx;	メールボックス情報を格納するパケットの先頭アドレス
入	ID	mbxid;	メールボックスの ID 番号

・メールボックス情報 T_RMBX の構造

```
typedef struct t_rmbx {
    VP          exinf;          /* 拡張情報 */
    BOOL_ID     wtsk;           /* 待ちタスクの有無 */
    T_MSG       *pk_msg;        /* 待ちメッセージの有無 */
    ID          keyid;          /* キーID 番号 */
} T_RMBX;
```

【説　　明】

mbxid で指定されたメールボックスのメールボックス情報（拡張情報，待ちタスクの有無など）を *pk_rmbx* で指定されたパケットに格納します。

次に，メールボックス情報の詳細を示します。

exinf …　拡張情報

wtsk …　待ちタスクの有無

- FALSE（0）　：待ちタスクなし
- 数値　　：待ちキューの先頭タスクの ID 番号

pk_msg …　待ちメッセージの有無

- NADR（－1）：待ちメッセージなし
- 数値　　：待ちキューの先頭メッセージのアドレス

keyid …　キーID 番号

- FALSE（0）　：生成時にキーID 番号が指定されていない
- 数値　　：キーID 番号

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	ref_mbx **システム・コールが** CF **定義されていない**
E_PAR	－33	**メールボックス情報を格納するパケットの先頭アドレスが不正（***pk_rmbx***＝0）である**
E_ID	－35	ID **番号指定が不正（最大メールボックス生成数＜***mbxid***）である**
*E_NOEXS	－52	**対象メールボックスが存在していない**
E_OACV	－66	**アクセス権のない** ID **番号（***mbxid***≦0）を指定した**

保守／廃止

get mailbox identifier (－245)

vget_mid

タスク / 非タスク

【概　　要】

メールボックスの ID 番号を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = vget_mid(ID *p_mbxid, ID keyid);
```

【パラメータ】

入出力	パラメータ	説　明
出	ID　*p_mbxid;	ID 番号を格納する領域のアドレス
入	ID　keyid;	メールボックスのキーID 番号

【説　　明】

keyid で指定されたメールボックスの ID 番号を *p_mbxid* で指定される領域に格納します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	vget_mid システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・キーID 番号の指定が不正（*keyid*＝0）である ・ID 番号を格納する領域のアドレスが不正（*p_mbxid*＝0）である
*E_NOEXS	－52	対象メールボックスが存在していない

保守／廃止

11.8.4　割り込み管理機能システム・コール

ここでは，マスカブル割り込みに依存した処理を行うシステム・コールのグループ（割り込み管理機能システム・コール）について説明します。

表 11－8 に，割り込み管理機能システム・コールの一覧を示します。

表11－8　割り込み管理機能システム・コール

システム・コール	機　　能
def_int	間接起動割り込みハンドラを登録／登録解除する
ret_int	直接起動割り込みハンドラから復帰する
ret_wup	他タスクの起床と直接起動割り込みハンドラからの復帰を行う
ena_int	マスカブル割り込みの受け付けを再開する
dis_int	マスカブル割り込みの受け付けを禁止する
loc_cpu	マスカブル割り込みの受け付けとディスパッチ処理を禁止する
unl_cpu	マスカブル割り込みの受け付けとディスパッチ処理を再開する
chg_icr	割り込み制御レジスタの内容を変更する
ref_icr	割り込み制御レジスタの内容を獲得する

保守／廃止

define interrupt handler (－65)

def_int

タスク / 非タスク

【概　　要】

間接起動割り込みハンドラを登録／登録解除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = def_int(UINT eintno, T_DINT *pk_dint);
```

【パラメータ】

入出力	パラメータ	説　　明
入	UINT　　*eintno*;	間接起動割り込みハンドラの割り込み要求番号
入	T_DINT　**pk_dint*;	間接起動割り込みハンドラ登録情報を格納したパケットの先頭アドレス

・間接起動割り込みハンドラ登録情報 T_DINT の構造

```
typedef struct t_dint {
    ATR   intatr;          /* 間接起動割り込みハンドラの属性 */
    FP    inthdr;          /* 間接起動割り込みハンドラの起動アドレス */
    VP    gp;              /* 間接起動割り込みハンドラの固有 GP レジスタ値 */
    VP    tp;              /* 間接起動割り込みハンドラの固有 TP レジスタ値 */
} T_DINT;
```

【説　　明】

pk_dint **で指定された情報を基に，***eintno* **で指定された割り込み要求番号のマスカブル割り込みが発生した際に起動する間接起動割り込みハンドラを登録します。**

次に，間接起動割り込みハンドラ登録情報の詳細を示します。

intatr …　**間接起動割り込みハンドラの属性**

ビット 0 …　**間接起動割り込みハンドラの記述言語**

TA_ASM（0）：　**アセンブリ言語**

TA_HLNG（1）：　**C 言語**

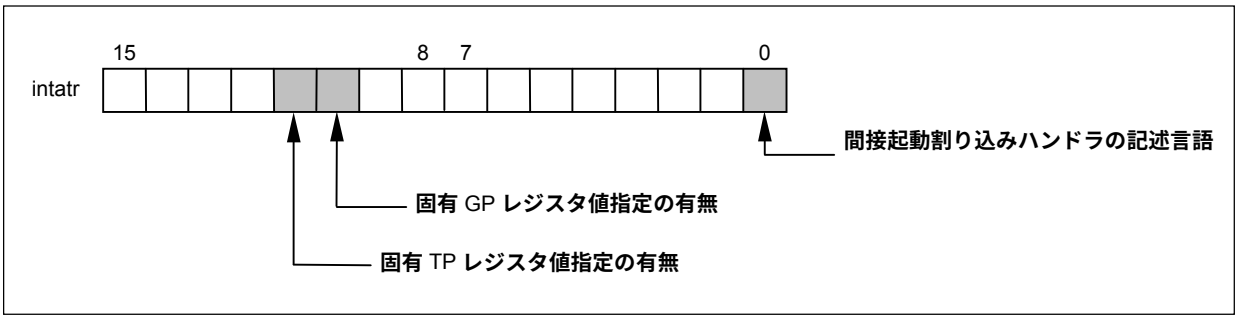
ビット 10 …　**固有 GP レジスタ値指定の有無**

TA_DPID（1）：　**固有 GP レジスタ値を指定**

ビット 11 …　**固有 TP レジスタ値指定の有無**

TA_DPIC（1）：　**固有 TP レジスタ値を指定**

保守／廃止



inthdr …　間接起動割り込みハンドラの起動アドレス

gp …　間接起動割り込みハンドラの固有 GP レジスタ値

tp …　間接起動割り込みハンドラの固有 TP レジスタ値

このシステム・コールを発行した際，対象割り込み要求番号に対応した間接起動割り込みハンドラがすでに登録されていた場合には，エラーとしては扱わず，このシステム・コールで指定された間接起動割り込みハンドラを新規に登録します。

また，このシステム・コールを発行した際，*pk_dint* で指定される領域に NADR（－1）を設定した場合，*eintno* で指定された間接起動割り込みハンドラの登録が解除されます。

備考 1. intatr のビット 10 の値が 1（TA_DPID）以外の場合，gp の内容は意味を持ちません。

2. intatr のビット 11 の値が 1（TA_DPIC）以外の場合，tp の内容は意味を持ちません。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	def_int システム・コールが CF 定義されていない
E_RSATR	－24	属性 intatr の指定が不正である
E_PAR	－33	パラメータの指定が不正である ・割り込み要求番号の指定が不正（*eintno*＜0，最大割り込み要因番号＜*eintno*）である ・間接起動割り込みハンドラ登録情報を格納したパケットの先頭アドレスが不正（*pk_dint*＝0）である ・起動アドレスの指定が不正（inthdr＝0）である

保守／廃止

return from interrupt handler (－69)

ret_int

直接起動割り込みハンドラ

【概　　要】

直接起動割り込みハンドラから復帰する。

【C 言語形式】

```
#include  <stdrx85p.h>
void      ret_int ();
```

【パラメータ】

なし

【説　　明】

直接起動割り込みハンドラから復帰します。

RX850 Pro では，直接起動割り込みハンドラ内でタスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューのキュー操作などの処理を行うだけであり，実際のスケジューリング処理は，直接起動割り込みハンドラからの復帰処理（ret_int または ret_wup システム・コールの発行）まで遅延され，一括して行われます。

注意 1. このシステム・コールでは，外部割り込みコントローラに対する処理終了通知（EOI コマンドの発行など）を行いません。したがって，外部割り込み要求によって起動した直接起動割り込みハンドラから復帰する場合は，このシステム・コールを発行する前に外部割り込みコントローラに対して終了通知を行ってください。

2. 直接起動割り込みハンドラをアセンブリ言語で記述した場合，直接起動割り込みハンドラからの復帰は，次のように記述してください。

```
jr      _ret_int
```

ただし，RX850 Pro で用意しているマクロ RTOS_IntReturn を使用して復帰する場合，マクロ内で ret_int システム・コールを発行しているので，記述する必要はありません。

3. 間接起動割り込みハンドラを C 言語で記述した場合，間接起動割り込みハンドラからの復帰は，次のように記述してください。

```
return (TSK_NULL);
```

4. 間接起動割り込みハンドラをアセンブリ言語で記述した場合，間接起動割り込みハンドラからの復帰は，次のように記述してください。

```
mov     TSK_NULL,r10
jmp     [lp]
```

【戻　り　値】

なし

保守／廃止

return and wakeup task (－70)

ret_wup

直接起動割り込みハンドラ

【概　　要】

他タスクの起床と直接起動割り込みハンドラからの復帰を行う。

【C 言語形式】

```
#include  <stdrx85p.h>
void      ret_wup (ID tskid);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　　*tskid;*	タスクの ID 番号

【説　　明】

tskid **で指定されたタスクに起床要求を発行（起床要求カウンタに** 0x1 **を加算）したあと，直接起動割り込みハンドラから復帰します。**

このシステム・コールを発行した際，対象タスクが wait **状態（起床待ち状態）であった場合には，起床要求の発行（起床要求カウンタの加算処理）は行わず，対象タスクを起床待ち状態から** ready **状態へと遷移させます。**

RX850 Pro **では，直接起動割り込みハンドラ内でタスクのスケジューリング処理が必要なシステム・コール（**chg_pri**，**sig_sem **など）が発行された場合，待ちキューのキュー操作などの処理を行うだけであり，実際のスケジューリング処理は，直接起動割り込みハンドラからの復帰処理（**ret_wup**，または** ret_int **システム・コールの発行）まで遅延され，一括して行われます。**

注意 1. このシステム・コールでは，外部割り込みコントローラに対する処理終了通知（EOI コマンドの発行など）を行いません。したがって，外部割り込み要求によって起動した割り込みハンドラから復帰する場合は，このシステム・コールを発行する前に外部割り込みコントローラに対して終了通知を行ってください。

2. このシステム・コールでは，次のようなエラーが発生した場合，直接起動割り込みハンドラからの復帰処理だけを行います。

- **ID 番号の指定が不正（最大タスク生成数＜*tskid*）である**
- **対象タスクが存在していない**
- **対象タスクが dormant 状態である**
- **起床要求数が 127 を越えた**

保守／廃止

3. 直接起動割り込みハンドラをアセンブリ言語で記述した場合，直接起動割り込みハンドラからの復帰は，次のように記述してください。

```
mov     tskid,r10
jr      _ret_wup
```

ただし，RX850 Pro で用意しているマクロ RTOS_IntReturnWakeup を使用して復帰する場合，マクロ内で ret_wup システム・コールを発行しているので，記述する必要はありません。
このマクロを使用する場合，次のように記述してください。

```
RTOS_IntReturnWakeup r10
```

（ r10 は，タスクの ID になります。）

4. 間接起動割り込みハンドラを C 言語で記述した場合，間接起動割り込みハンドラからの復帰は，次のように記述してください。

```
return (ID tskid);
```

5. 間接起動割り込みハンドラをアセンブリ言語で記述した場合，他タスクの起床と間接起動割り込みハンドラからの復帰は，次のように記述してください。

```
mov     tskid,r10
jmp     [lp]
```

【戻　り　値】

なし

保守／廃止

enable interrupt (－71)

ena_int

タスク / 非タスク

【概　　要】

マスカブル割り込みの受け付けを再開する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ena_int();
```

【パラメータ】

なし

【説　　明】

dis_int **システム・コールの発行により禁止されていたマスカブル割り込みの受け付けを再開します。**

なお，RX850 Pro **では，**dis_int **システム・コールの発行から** ena_int **システム・コールの発行までの間にマスカブル割り込みが発生していた場合，該当する割り込み処理（直接起動割り込みハンドラ，間接起動割り込みハンドラ）への移行は** ena_int **システム・コールが発行されるまで遅延されます。**

注意　このシステム・コールでは，再開要求のキューイングは行われません。したがって，すでにこのシステム・コールが発行され，マスカブル割り込みの受け付けが許可されていた場合には，なにも処理は行わず，エラーとしても扱いません。

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	ena_int **システム・コールが** CF **定義されていない**

保守／廃止

disable interrupt (－72)

dis_int

タスク / 非タスク

【概　　要】

マスカブル割り込みの受け付けを禁止する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        dis_int();
```

【パラメータ】

なし

【説　　明】

マスカブル割り込みの受け付けを禁止します。

これにより，ena_int システム・コールが発行されるまでの間，マスカブル割り込みの受け付けが禁止されます。

RX850 Pro では，dis_int システム・コールの発行から ena_int システム・コールの発行までの間に，マスカブル割り込みが発行していた場合，該当する割り込み処理（直接起動割り込みハンドラ，間接起動割り込みハンドラ）への移行は ena_int システム・コールが発行されるまで遅延されます。

注意　このシステム・コールでは，禁止要求のキューイングが行われません。このため，すでに dis_int システム・コールが発行され，マスカブル割り込みの受け付けが禁止されていた場合には，何も処理は行わず，エラーとしても扱いません。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	dis_int システム・コールが CF 定義されていない

保守／廃止

lock CPU (－8)

loc_cpu

タスク

【概　　要】

マスカブル割り込みの受け付けとディスパッチ処理を禁止する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = loc_cpu();
```

【パラメータ】

なし

【説　　明】

マスカブル割り込みの受け付けとディスパッチ処理（タスクのスケジューリング処理）を禁止します。

したがって，このシステム・コールの発行から unl_cpu システム・コールが発行されるまでの間は，ほかのハンドラやタスクに制御が移ることはありません。

なお，RX850 Pro では，このシステム・コールの発行から unl_cpu システム・コールの発行までの間にマスカブル割り込みが発生していた場合，該当する割り込み処理（割り込みハンドラへの移行）は unl_cpu システム・コールが発行されるまで遅延されます。また，タスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューのキュー操作などの処理を行うだけであり，実際のスケジューリング処理は unl_cpu システム・コールが発行されるまで遅延され，一括して行われます。

注意　このシステム・コールでは，禁止要求のキューイングは行われません。したがって，すでにこのシステム・コールが発行され，マスカブル割り込みの受け付けとディスパッチ処理が禁止されていた場合には，何も処理は行わず，エラーとしても扱いません。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	loc_cpu システム・コールが CF 定義されていない
E_CTX	－69	非タスクから loc_cpu システム・コールを発行した

保守／廃止

unlock CPU (－7)

unl_cpu

タスク

【概　　要】

マスカブル割り込みの受け付けとディスパッチ処理を許可する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = unl_cpu();
```

【パラメータ】

なし

【説　　明】

loc_cpu システム・コールの発行により禁止されていたマスカブル割り込みの受け付けとディスパッチ処理（タスクのスケジューリング処理）を再開します。

なお，RX850 Pro では，loc_cpu システム・コールの発行からこのシステム・コールの発行までの間にマスカブル割り込みが発生していた場合，該当する割り込み処理（割り込みハンドラ）への移行はこのシステム・コールが発行されるまで遅延されます。また，タスクのスケジューリング処理が必要なシステム・コール（chg_pri，sig_sem など）が発行された場合，待ちキューのキュー操作などの処理を行うだけであり，実際のスケジューリング処理はこのシステム・コールが発行されるまで遅延され，一括して行われます。

備考　dis_dsp システム・コールの発行により禁止されたディスパッチ処理は，このシステム・コールの発行により再開できます。

注意　**このシステム・コールでは，再開要求のキューイングは行われません。したがって，すでにこのシステム・コールが発行され，マスカブル割り込みの受け付けとディスパッチ処理が再開されていた場合には，何も処理は行わず，エラーとしても扱いません。**

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	unl_cpu システム・コールが CF 定義されていない
E_CTX	－69	非タスクから unl_cpu システム・コールを発行した

保守／廃止

change interrupt Control Register (－67)

chg_icr

タスク / 非タスク

【概　　要】

割り込み制御レジスタの内容を変更する。

【C 言語形式】

```
ER        chg_icr(UINT eintno, UB icrcmd);
```

【パラメータ】

入出力	パラメータ	説　　明
入	UINT　*eintno;*	割り込み要求番号
入	UB　*icrcmd;*	割り込み要求フラグの指定 ICR_CLRINT（0x20）：割り込み要求なし 割り込みマスク・フラグの指定 ICR_CLRMSK（0x10）：割り込み処理を許可 ICR_SETMSK（0x40）：割り込み処理を禁止 割り込み優先順位変更の指定 ICR_CHGLVL（0x08）：割り込み優先順位を変更 割り込み優先順位の指定 数値（0～7）：割り込み優先順位

【説　　明】

eintno **で指定された割り込み制御レジスタの内容を** *icrcmd* **で指定された値に変更します。**

次に，*icrcmd* **の指定形式を示します。**

- *icrcmd* = ICR_CLRINT

  割り込み制御レジスタの割り込み要求フラグを“ 0 **”に変更します。**

- *icrcmd* = ICR_CLRMSK

  割り込み制御レジスタの割り込みマスク・フラグを“ 0 **”に変更します。**

- *icrcmd* = ICR_SETMSK

  割り込み制御レジスタの割り込みマスク・フラグを“ 1 **”に変更します。**

- *icrcmd* =（ICR_CHGLVL｜**数値**）

  割り込み制御レジスタの割り込み優先順位を“数値”で指定された値に変更します。

  なお，数値は“ 0 **”がレベル** 0 **に，“** 7 **”がレベル** 7 **に対応しています。**

備考　**割り込み要求番号** *eintno* **には，「（対象割り込み要求番号の例外コード－**0x80）/ 0x10**」で算出される値を指定します。**

保守／廃止

注意　V850E コアで動作させる場合，chg_icr システム・コールを発行しても，期待した割り込み制御レジスタが操作されない場合があります。RX850 Pro では，割り込み要因番号から割り込み制御レジスタのアドレスを算出しています。しかし，V850E コアの場合，割り込み要因番号と割り込み制御レジスタの並びが，他の V850 シリーズと異なっているため，正しいレジスタ・アドレスが得られなくなっています。そのため，V850E コアにおける chg_icr の使用を制御します。割り込み制御レジスタをアプリケーションから操作したい場合は，このシステム・コールを使わず，直接レジスタを操作してください。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	chg_icr システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・割り込み要求番号の指定が不正 （*eintno*＜0，最大割り込み要因番号＜*eintno*）である ・割り込み要求フラグの指定が不正 （*eintno*＜0，最大割り込み要因番号＜*eintno*）である ・割り込み要求フラグとして（ICR_CLRMSK｜ICR_SETMSK）が指定されている

保守／廃止

refer interrupt Control Register Status (－68)

ref_icr

タスク / 非タスク

【概　　要】

割り込み制御レジスタの内容を獲得する。

【C 言語形式】

```
ER        ref_icr(UB *p_regptn, UINT eintno);
```

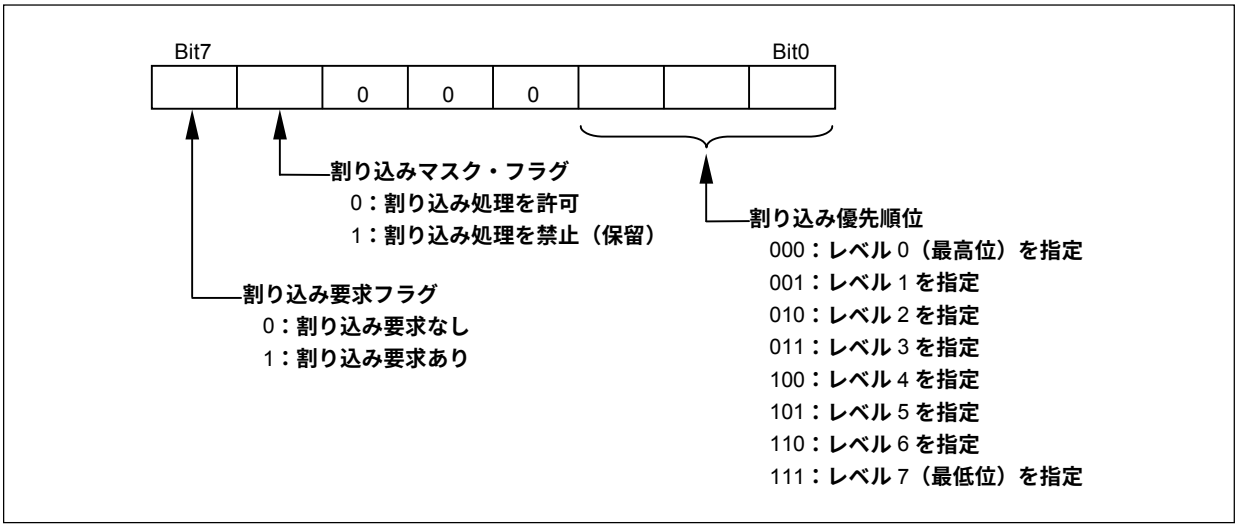
【パラメータ】

入出力	パラメータ	説　明
出	UB　*p_regptn;	割り込み制御レジスタの内容を格納する領域のアドレス
入	UINT　eintno;	割り込み要求番号

【説　　明】

eintno で指定された割り込み制御レジスタの内容を p_regptn で指定される領域に格納します。

次に，獲得した割り込み制御レジスタの内容を示します。



備考　割り込み要求番号 *eintno* には，「（対象割り込み要求番号の例外コード－0x80）/ 0x10」で算出される値を指定します。

注意　V850E コアで動作させる場合，ref_icr システム・コールを発行しても，期待した割り込み制御レジスタが操作されない場合があります。RX850 Pro では，割り込み要因番号から割り込み制御レジスタのアドレスを算出しています。しかし，V850E コアの場合，割り込み要因番号と割り込み制御レジスタの並びが，他の V850 シリーズと異なっているため，正しいレジスタ・アドレスが得られなくなっています。そのため，V850E コアにおける ref_icr の使用を制御します。割り込み制御レジスタをアプリケーションから操作したい場合は，このシステム・コールを使わず，直接レジスタを操作してください。

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	ref_icr **システム・コールが** CF **定義されていない**
E_PAR	－33	**パラメータの指定が不正である** **・割り込み制御レジスタの内容を格納する領域のアドレス（***p_regptn***）が 0 である** **・割り込み要求番号の指定が不正（***eintno***＜0，最大割り込み要因番号＜***eintno***）である**

保守／廃止

11.8.5　メモリ・プール管理機能システム・コール

ここでは，メモリ・ブロックの割り当てを行うシステム・コールのグループ（メモリ・プール管理機能システム・コール）について説明します。

表 11－9 に，メモリ・プール管理機能システム・コールの一覧を示します。

表11－9　メモリ・プール管理機能システム・コール

システム・コール	機　　能
cre_mpl	メモリ・プールを生成する
del_mpl	メモリ・プールを削除する
get_blk	メモリ・ブロックを獲得する
pget_blk	メモリ・ブロックを獲得する（ポーリング）
tget_blk	メモリ・ブロックを獲得する（タイムアウトあり）
rel_blk	メモリ・ブロックを返却する
ref_mpl	メモリ・プール情報を獲得する
vget_pid	メモリ・プールの ID 番号を獲得する

保守／廃止

create variable-size memory pool (－137)

cre_mpl

タスク

【概　　要】

メモリ・プールを生成する。

【C 言語形式】

・ID 番号を指定する場合

```
#include <stdrx85p.h>
ER       ercd = cre_mpl(ID mplid, T_CMPL *pk_cmpl);
```

・ID 番号を指定しない場合

```
#include <stdrx85p.h>
ER       ercd = cre_mpl(ID_AUTO, T_CMPL *pk_cmpl, ID *p_mplid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　*mplid;*	メモリ・プールの ID 番号
入	T_CMPL　**pk_cmpl;*	メモリ・プール生成情報を格納したパケットの先頭アドレス
出	ID　**p_mplid;*	ID 番号を格納する領域のアドレス

・メモリ・プール生成情報 T_CMPL の構造

```
typedef struct t_cmpl {
    VP    exinf;          /* 拡張情報 */
    ATR   mplatr;         /* メモリ・プールの属性 */
    INT   mplsz;          /* メモリ・プールのサイズ */
    ID    keyid;          /* メモリ・プールのキーID 番号 */
} T_CMPL;
```

【説　　明】

RX850 Pro では，メモリ・プールの生成において「ID 番号を指定して生成する」，「ID 番号を指定しないで生成する」の 2 種類のインタフェースを用意しています。

● ID 番号を指定する場合

pk_cmpl で指定された情報を基に，*mplid* で指定された ID 番号を持つメモリ・プールを生成します。

● ID 番号を指定しない場合

pk_cmpl で指定された情報を基に，メモリ・プールを生成します。

ID 番号の割り付けは RX850 Pro により行われ，割り付けられた ID 番号は，*p_mplid* で指定される領域に格納されます。

保守／廃止

次に，メモリ・プール生成情報の詳細を示します。

exinf …　拡張情報

exinf は対象メモリ・プールに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。

exinf に設定された情報は，処理プログラム（タスク，非タスク）から ref_mpl システム・コールを発行することにより，ダイナミックに獲得できます。

mplatr …　メモリ・プールの属性

ビット 0 …　待ちキューへのキューイング方法

TA_TPRI（0）　：　優先度順

TA_TFIFO（1）　：　FIFO 順

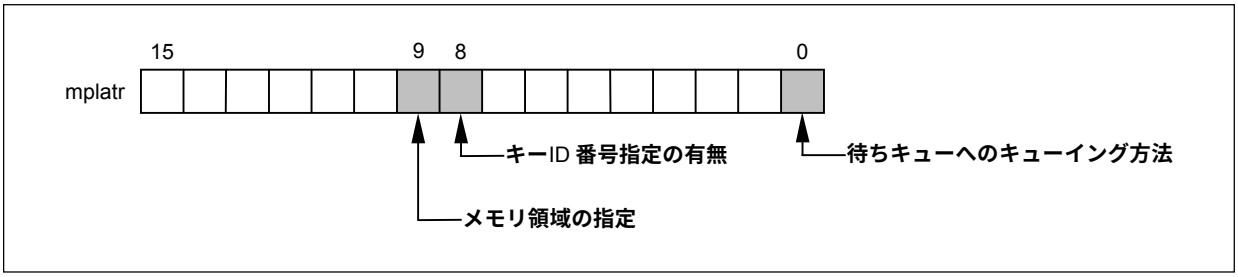
ビット 8 …　キーID 番号指定の有無

TA_KEYID（1）　：　キーID 番号を指定

ビット 9 …　メモリ領域の指定

TA_UPOL0（0）：　ユーザ・メモリ領域 0 番からメモリ・プール領域を確保

TA_UPOL1（1）：　ユーザ・メモリ領域 1 番からメモリ・プール領域を確保



mplsz …　メモリ・プールのサイズ（単位：バイト）

keyid …　メモリ・プールのキーID 番号

備考　mplatr のビット 8 の値が 1（TA_KEYID）以外の場合，keyid の内容は意味を持ちません。

保守／廃止

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	cre_mpl システム・コールが CF 定義されていない
*E_NOMEM	－10	メモリ・プール管理ブロック，またはメモリ・プール領域が確保できない
E_RSATR	－24	属性 mplatr の指定が不正である
E_PAR	－33	パラメータの指定が不正である ・メモリ・プール生成情報を格納したパケットの先頭アドレスが不正（*p_mplid*＝0）である ・サイズの指定が不正（mplsz≦0）である ・キーID 番号の指定が不正（keyid＝0）である（TA_KEYID 属性指定時） ・ID 番号を格納する領域のアドレスが不正（*p_mplid*＝0）である（ID 番号を指定しないで生成する場合）
E_ID	－35	ID 番号の指定が不正（最大メモリ・プール生成数＜*mplid*）である
*E_OBJ	－63	指定した ID 番号を持つメモリ・プールがすでに生成されている
E_OACV	－66	アクセス権のない ID 番号（*mplid*≦0）を指定した
E_CTX	－69	非タスクから cre_mpl システム・コールを発行した

保守／廃止

delete variable-size memory pool (－138)

del_mpl

タスク

【概　　要】

メモリ・プールを削除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = del_mpl(ID mplid);
```

【パラメータ】

入出力	パラメータ	説　　明
入	ID　*mplid;*	メモリ・プールの ID 番号

【説　　明】

mplid で指定されたメモリ・プールを削除します。

これにより，対象メモリ・プールは RX850 Pro の管理下から除外されます。

なお，このシステム・コールの発行により wait 状態（メモリ・ブロック待ち状態）を解除されたタスクは，wait 状態へと遷移するきっかけとなったシステム・コール（get_blk，tget_blk）の戻り値として E_DLT が返されます。

また，対象メモリ・プールが管理していたメモリ・ブロックをタスクが獲得していた場合に，このシステム・コールを発行すると，メモリ・ブロックについても RX850 Pro の管理下から除外されます。したがって，獲得していたメモリ・ブロックの内容については不定となります。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	del_mpl システム・コールが CF 定義されていない
	E_ID	－35	ID 番号の指定が不正（最大メモリ・プール生成数＜*mplid*）である
*	E_NOEXS	－52	対象メモリ・プールが存在していない
	E_OACV	－66	アクセス権のない ID 番号（*mplid*≦0）を指定した
	E_CTX	－69	非タスクから del_mpl システム・コールを発行した

保守／廃止

get variable-size memory block (－141)

get_blk

タスク

【概　　要】

メモリ・ブロックを獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd =get_blk(VP *p_blk, ID mplid, INT blksz);
```

【パラメータ】

入出力	パラメータ		説　　明
出	VP	*p_blk;	メモリ・ブロックの先頭アドレスを格納する領域のアドレス
入	ID	mplid;	メモリ・プールの ID 番号
入	INT	blksz;	メモリ・ブロックのサイズ（単位：バイト）

【説　　明】

mplid で指定されたメモリ・プールから *blksz* で指定されたサイズのメモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった（要求したサイズの空き領域が存在しなかった）場合には，自タスクを対象メモリ・プールの待ちキューにキューイングしたあと，run 状態から wait 状態（メモリ・ブロック待ち状態）へと遷移させます。

なお，メモリ・ブロック待ち状態は rel_blk システム・コールの発行により要求したサイズを満足するようなメモリ・ブロックが返却されるか，または del_mpl，rel_wai システム・コールが発行されると解除され，自タスクは ready 状態へと遷移します。

注意　RX850 Pro では，メモリ・ブロックを獲得する際，メモリ・クリアを行いません。したがって，獲得したメモリ・ブロックの内容は不定となります。

備考　自タスクを対象メモリ・プールの待ちキューにキューイングする際は，対象メモリ・プール生成時（コンフィギュレーション時または cre_mpl システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	get_blk **システム・コールが** CF **定義されていない**
E_PAR	－33	**パラメータの指定が不正である** **・メモリ・ブロックの先頭アドレスを格納する領域のアドレスが不正（***p_blk***＝0）である** **・メモリ・ブロック・サイズの指定が不正（***blksz***≦0）である**
E_ID	－35	ID **番号の指定が不正（最大メモリ・プール生成数＜***mplid***）である**
*E_NOEXS	－52	**対象メモリ・プールが存在していない**
E_OACV	－66	**アクセス権のない** ID **番号（***mplid***≦0）を指定した**
E_CTX	－69	**コンテキスト・エラー** **・非タスクから** get_blk **システム・コールを発行した** **・ディスパッチ禁止状態から** get_blk **システム・コールを発行した**
*E_DLT	－81	del_mpl **システム・コールにより対象メモリ・プールが削除された**
*E_RLWAI	－86	rel_wai **システム・コールによりメモリ・ブロック待ち状態が強制的に解除された**

保守／廃止

poll and get variable-size memory block (－104)

pget_blk

タスク / 非タスク

【概　　要】

メモリ・ブロックを獲得する（ポーリング）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = pget_blk(VP *p_blk, ID mplid, INT blksz);
```

【パラメータ】

入出力	パラメータ	説　　明
出	VP　*p_blk;	メモリ・ブロックの先頭アドレスを格納する領域のアドレス
入	ID　mplid;	メモリ・プールの ID 番号
入	INT　blksz;	メモリ・ブロックのサイズ（単位：バイト）

【説　　明】

mplid で指定されたメモリ・プールから，*blksz* で指定されたサイズのメモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった（要求したサイズの空き領域が存在しなかった）場合には，戻り値として E_TMOUT が返されます。

注意　RX850 Pro では，メモリ・ブロックを獲得する際，メモリ・クリアを行いません。したがって，獲得したメモリ・ブロックの内容は不定となります。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	pget_blk システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・メモリ・ブロックの先頭アドレスを格納する領域のアドレスが不正（*p_blk*＝0）である ・メモリ・ブロック・サイズの指定が不正（*blksz*≦0）である
E_ID	－35	ID 番号の指定が不正（最大メモリ・プール生成数＜*mplid*）である
*E_NOEXS	－52	対象メモリ・プールが存在していない
E_OACV	－66	アクセス権のない ID 番号（*mplid*≦0）を指定した
*E_TMOUT	－85	対象メモリ・プールに空き領域が存在しない

保守／廃止

get variable-size memory block with timeout (－168)

tget_blk

タスク

【概　　要】

メモリ・ブロックを獲得する（タイムアウトあり）。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = tget_blk(VP *p_blk, ID mplid, INT blksz, TMO tmout);
```

【パラメータ】

入出力	パラメータ	説　　明
出	VP　*p_blk;	メモリ・ブロックの先頭アドレスを格納する領域のアドレス
入	ID　mplid;	メモリ・プールの ID 番号
入	INT　blksz;	メモリ・ブロックのサイズ（単位：バイト）
入	TMO　tmout;	待ち時間（単位：ms） TMO_POL（0）　：即時リターン TMO_FEVR（－1）：永久待ち 数値　：待ち時間

【説　　明】

mplid で指定されたメモリ・プールから，*blksz* で指定されたサイズのメモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定される領域に格納します。

ただし，このシステム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得できなかった（要求したサイズの空き領域が存在しなかった）場合には，自タスクを対象メモリ・プールの待ちキューにキューイングしたあと，run 状態から wait 状態（メモリ・ブロック待ち状態）へと遷移させます。

なお，メモリ・ブロック待ち状態は *tmout* で指定された待ち時間が経過するか，rel_blk システム・コールの発行により要求したサイズを満足するようなメモリ・ブロックが返却されるか，または del_mpl，rel_wai システム・コールが発行されると解除され，自タスクは ready 状態へと遷移します。

注意　RX850 Pro では，メモリ・ブロックを獲得する際，メモリ・クリアを行いません。したがって，獲得したメモリ・ブロックの内容は不定となります。

備考　自タスクを対象メモリ・プールの待ちキューにキューイングする際は，対象メモリ・プール生成時（コンフィギュレーション時または cre_mpl システム・コール発行時）に指定された順（FIFO 順または優先度順）に行われます。

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	tget_blk **システム・コールが** CF **定義されていない**
E_PAR	－33	**パラメータの指定が不正である** **・メモリ・ブロックの先頭アドレスを格納する領域のアドレスが不正（***p_blk***＝0）である** **・メモリ・ブロック・サイズの指定が不正（***blksz***≦0）である**
E_ID	－35	ID **番号の指定が不正（最大メモリ・プール生成数＜***mplid***）である**
*E_NOEXS	－52	**対象メモリ・プールが存在していない**
E_OACV	－66	**アクセス権のない** ID **番号（***mplid***≦0）を指定した**
E_CTX	－69	**コンテキスト・エラー** **・非タスクから** tget_blk **システム・コールを発行した** **・ディスパッチ禁止状態から** tget_blk **システム・コールを発行した**
*E_DLT	－81	del_mpl **システム・コールにより対象メモリ・プールが削除された**
*E_TMOUT	－85	**待ち時間が経過した**
*E_RLWAI	－86	rel_wai **システム・コールによりメモリ・ブロック待ち状態が強制的に解除された**

保守／廃止

release variable-size memory block (－143)

rel_blk

タスク / 非タスク

【概　　要】

メモリ・ブロックを返却する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = rel_blk(ID mplid, VP blk);
```

【パラメータ】

入出力	パラメータ	説　明
入	ID　*mplid;*	メモリ・プールの ID 番号
入	VP　*blk;*	メモリ・ブロックの先頭アドレス

【説　　明】

blk で指定されたメモリ・ブロックを *mplid* で指定されたメモリ・プールに返却します。

このシステム・コールを発行した際，返却するメモリ・ブロックのサイズが対象メモリ・プールの待ちキューにキューイングされているタスク（待ちキューの先頭タスク）が要求したサイズを満足するような場合には，該当タスク（待ちキューの先頭タスク）にメモリ・ブロックを渡します。

これにより，該当タスクは待ちキューから外れ，wait 状態（メモリ・ブロック待ち状態）から ready 状態へ，または wait_suspend 状態から suspend 状態へと遷移します。

注意 1. RX850 Proでは，メモリ・ブロックを返却するときにメモリ・クリアを行いません。したがって，返却したメモリ・ブロックの内容は不定です。

★ **2. RX850 Proには2つの仕様のrel_blkシステム・コールがあります。**

（1）rel_blkシステム・コールでメモリ・ブロックを返却する際，メモリ・ブロックの先頭4バイトが0でなかった場合に，メモリ・ブロックを返却せずに戻り値E_OBJで終了する。

（2）rel_blkシステム・コールを発行すると，メモリ・ブロックの先頭4バイトが0でなくてもメモリ・ブロックを返却する（戻り値E_OK）。

（1）はメモリ・ブロックがメールボックスのメッセージ領域として使用されていた場合を考慮したもので，これまでのRX850 Proに搭載されていたrel_blkと同じ仕様のものです。
メモリ・ブロックがメールボックスのメッセージ領域として使用されていた場合，先頭4バイトがメッセージの待ちキューのリンク領域になります。つまりメッセージが，メールボックスにキューイングされているときにrel_blkシステム・コールを発行し，必ずメモリ・ブロックが返却する仕様にすると，キューにつながれていたメッセージ領域が返却されてしまうことになります。これを防ぐため，リンク領域である先頭4バイトが0でなければ，メッセージ領域として使用され

保守／廃止

ているメモリ・ブロックと判断し，メモリ・ブロックを返却せずに戻り値E_OBJで終了します。この仕様のrel_blkを使用してメモリ・ブロックを返却するときは，必ず先頭4バイトを0クリアする必要があります。

これらの仕様のrel_blkは，別々のライブラリに搭載されていますので，どちらか一方のrel_blkを使用することになります。使用する方のライブラリをリンクしてください。

(1) メモリ・ブロックの先頭4バイトを0でクリアする必要のあるrel_blkが搭載されたライブラリ　→　librxp.a

(2) メモリ・ブロックの先頭4バイトを0でクリアしなくてもよいrel_blkが搭載されたライブラリ　→　librxpm.a

3. **メモリ・ブロックを返却するメモリ・プールは，get_blk，pget_blk，tget_blkシステム・コールを発行した際に指定したメモリ・プールと同じにしてください。**

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	rel_blk **システム・コールが** CF **定義されていない**
*E_PAR	－33	**パラメータの指定が不正である** **・メモリ・ブロックの先頭アドレスの指定が不正（***blk*＝0）**である** **・獲得時に指定したメモリ・プールと** rel_blk **システム・コール発行時に指定したメモリ・プールが異なっている**
E_ID	－35	ID **番号の指定が不正（最大メモリ・プール生成数＜***mplid*）**である**
*E_NOEXS	－52	**対象メモリ・プールが存在していない**
E_OBJ	－63	**返却するメモリ・ブロックの先頭** 4 **バイトに** 0x0 **以外の値が入っている** **・メッセージ領域として使用されているメモリ・ブロックを返却しようとしたときに，この戻り値となります。**
E_OACV	－66	**アクセス権のない** ID **番号（***mplid*≦0）**を指定した**

保守／廃止

refer variable-size memory pool status (－140)

ref_mpl

タスク / 非タスク

【概　　要】

メモリ・プール情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ref_mpl(T_RMPL *pk_rmpl, ID mplid);
```

【パラメータ】

入出力	パラメータ	説　明
出	T_RMPL　*pk_rmpl;	メモリ・プール情報を格納するパケットの先頭アドレス
入	ID　mplid;	メモリ・プールの ID 番号

・メモリ・プール情報 T_RMPL の構造

```
typedef struct t_rmpl {
    VP              exinf;          /* 拡張情報 */
    BOOL_ID         wtsk;           /* 待ちタスクの有無 */
    INT             frsz;           /* 空き領域の合計サイズ */
    INT             maxsz;          /* 獲得可能な最大メモリ・ブロック・サイズ */
    ID              keyid;          /* キーID 番号 */
} T_RMPL;
```

【説　　明】

mplid で指定されたメモリ・プールのメモリ・プール情報（拡張情報，待ちタスクの有無など）を *pk_rmpl* で指定されるパケットに格納します。

次に，メモリ・プール情報の詳細を示します。

exinf …　拡張情報

wtsk …　待ちタスクの有無

FALSE（0）：待ちタスクなし

数値　　：待ちキューの先頭タスクの ID 番号

frsz …　空き領域の合計サイズ（単位：バイト）

maxsz …　獲得可能な最大メモリ・ブロック・サイズ（単位：バイト）

keyid …　キーID 番号

FALSE（0）：生成時にキーID 番号が指定されていない

数値　　：キーID 番号

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	ref_mpl **システム・コールが** CF **定義されていない**
E_PAR	－33	**メモリ・プール情報を格納するパケットの先頭アドレスが不正（***pk_rmpl***＝0）である**
E_ID	－35	ID **番号の指定が不正（最大メモリ・プール生成数＜***mplid***）である**
*E_NOEXS	－52	**対象メモリ・プールが存在していない**
E_OACV	－66	**アクセス権のない** ID **番号（***mplid***≦0）を指定した**

保守／廃止

get variable-size memory pool identifier (－242)

vget_pid

タスク / 非タスク

【概　　要】

メモリ・プールの ID 番号を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = vget_pid(ID *p_mplid, ID keyid);
```

【パラメータ】

入出力	パラメータ		説　　明
出	ID	*p_mplid;	ID 番号を格納する領域のアドレス
入	ID	keyid;	メモリ・プールのキーID 番号

【説　　明】

keyid で指定されたメモリ・プールの ID 番号を *p_mplid* で指定される領域に格納します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	vget_pid システム・コールが CF 定義されていない
E_PAR	－33	パラメータの指定が不正である ・キーID 番号の指定が不正（*keyid*＝0）である ・ID 番号を格納する領域のアドレスが不正（*p_mplid*＝0）である
*E_NOEXS	－52	対象メモリ・プールが存在していない

保守／廃止

11.8.6　時間管理機能システム・コール

ここでは，時間に依存した処理を行うシステム・コールのグループ（時間管理機能システム・コール）について説明します。

表 11－10 に，時間管理機能システム・コールの一覧を示します。

表11－10　時間管理機能システム・コール

システム・コール	機　　能
set_tim	システム・クロックに時刻を設定する
get_tim	システム・クロックの時刻を獲得する
dly_tsk	自タスクを時間経過待ち状態へ移行させる
def_cyc	周期起動ハンドラを登録／登録解除する
act_cyc	周期起動ハンドラの活性状態を制御する
ref_cyc	周期起動ハンドラ情報を獲得する

保守／廃止

set time (－83)

set_tim

タスク / 非タスク

【概　　要】

システム・クロックに時刻を設定する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = set_tim(SYSTIME *pk_tim);
```

【パラメータ】

入出力	パラメータ	説　　明
入	SYSTIME　*pk_tim;	時刻を格納したパケットの先頭アドレス

・システム・クロック SYSTIME の構造

★

```
typedef struct t_systime {
    UW    ltime;          /* 時刻 （下位 32 ビット） */
    H     utime;          /* 時刻 （上位 16 ビット） */
} SYSTIME;
```

【説　　明】

システム・クロックに *pk_tim* で指定された時刻を設定します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	set_tim システム・コールが CF 定義されていない
E_PAR	－33	時刻を格納したパケットの先頭アドレスが不正（*pk_tim*＝0）である

保守／廃止

get time (－84)

get_tim

タスク / 非タスク

【概　　要】

システム・クロックの時刻を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = get_tim(SYSTIME *pk_tim);
```

【パラメータ】

入出力	パラメータ	説　　明
出	SYSTIME　*pk_tim;	時刻を格納するパケットの先頭アドレス

・システム・クロック SYSTIME の構造

★

```
typedef struct t_systime {
    UW    ltime;          /* 時刻 （下位 32 ビット） */
    H     utime;          /* 時刻 （上位 16 ビット） */
} SYSTIME;
```

【説　　明】

システム・クロックの現時刻を *pk_tim* で指定されるパケットに格納します。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	get_tim システム・コールが CF 定義されていない
E_PAR	－33	時刻を格納するパケットの先頭アドレスが不正（*pk_tim*＝0）である

保守／廃止

delay task (－85)

dly_tsk

タスク

【概　　要】

自タスクを時間経過待ち状態へ移行させる。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = dly_tsk(DLYTIME dlytim);
```

【パラメータ】

入出力	パラメータ	説　　明
入	DLYTIME　*dlytim*;	遅延時間（単位：ms）

【説　　明】

自タスクを *dlytim* で指定された遅延時間だけ，run 状態から wait 状態（時間経過待ち状態）へと遷移させます。

なお，時間経過待ち状態は *dlytim* で指定された遅延時間が経過するか，または rel_wai システム・コールが発行されると解除され，自タスクは ready 状態へ遷移します。

注意　時間経過待ち状態は，wup_tsk，ret_wup システム・コールでは解除されません。

【戻　り　値】

*	E_OK	0	正常終了
*	E_NOSPT	－17	dly_tsk システム・コールが CF 定義されていない
	E_PAR	－33	遅延時間の指定が不正（*dlytim*＜0）である
	E_CTX	－69	コンテキスト・エラー ・非タスクから dly_tsk システム・コールを発行した ・ディスパッチ禁止状態から dly_tsk システム・コールを発行した
*	E_RLWAI	－86	rel_wai システム・コールにより時間経過待ち状態が強制的に解除された

保守／廃止

define cyclic handler (－90)

def_cyc

タスク / 非タスク

【概　　要】

周期起動ハンドラを登録／登録解除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = def_cyc(HNO cycno, T_DCYC *pk_dcyc);
```

【パラメータ】

入出力	パラメータ	説　　明
入	HNO　*cycno;*	周期起動ハンドラの指定番号
入	T_DCYC　**pk_dcyc;*	周期起動ハンドラ登録情報を格納したパケットの先頭アドレス

・周期起動ハンドラ登録情報 T_DCYC の構造

```
typedef struct t_dcyc {
    VP              exinf;          /* 拡張情報 */
    ATR             cycatr;         /* 周期起動ハンドラの属性 */
    FP              cychdr;         /* 周期起動ハンドラの起動アドレス */
    UINT            cycact;         /* 周期起動ハンドラの初期活性状態 */
    CYCTIME         cyctim;         /* 周期起動ハンドラの起動時間間隔 */
    VP              gp;             /* 周期起動ハンドラの固有 GP レジスタ値 */
    VP              tp;             /* 周期起動ハンドラの固有 TP レジスタ値 */
} T_DCYC;
```

【説　　明】

pk_dcyc で指定された情報を基に，*cycno* で指定された指定番号を持つ周期起動ハンドラを登録します。
次に，周期起動ハンドラ登録情報の詳細を示します。

exinf …　拡張情報

exinf は対象周期起動ハンドラに関する “ ユーザ独自の情報 ” を格納するための領域で，ユーザが自由に利用できます。

exinf に設定された情報は，処理プログラム（タスク，非タスク）から ref_cyc システム・コールを発行することにより，ダイナミックに獲得できます。

cycatr …　周期起動ハンドラの属性

ビット 0 …　周期起動ハンドラの記述言語

TA_ASM（0）　：　アセンブリ言語

TA_HLNG（1）　：　C 言語

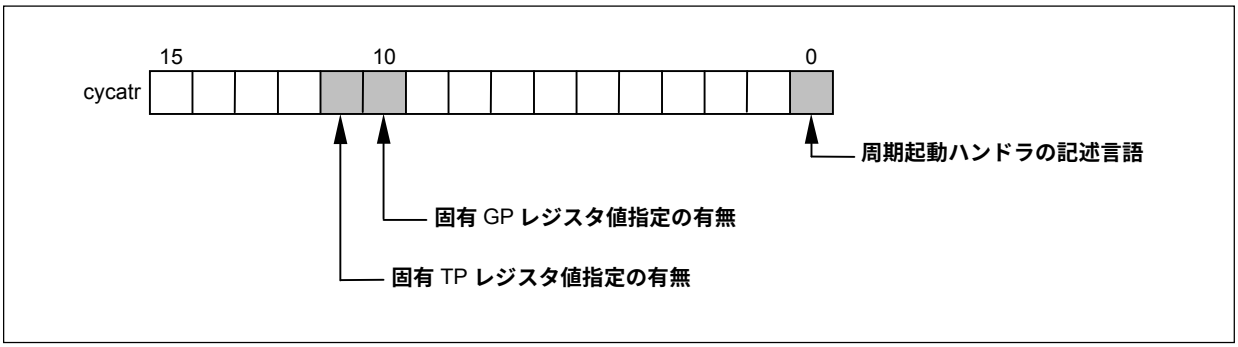
保守／廃止

ビット 10 …　固有 GP レジスタ値指定の有無

TA_DPID（1）　：　固有 GP レジスタ値を指定

ビット 11 …　固有 TP レジスタ値指定の有無

TA_DPIC（1）　：　固有 TP レジスタ値を指定



cychdr …　周期起動ハンドラの起動アドレス

cycact …　周期起動ハンドラの初期活性状態

TCY_OFF（0）：　初期活性状態は OFF 状態

TCY_ON（1）　：　初期活性状態は ON 状態

cyctim …　周期起動ハンドラの起動時間間隔（単位：基本クロック周期）

gp …　周期起動ハンドラの固有 GP レジスタ値

tp …　周期起動ハンドラの固有 TP レジスタ値

このシステム・コールを発行した際，対象指定番号に対応した周期起動ハンドラがすでに登録されていた場合には，エラーとしては扱わず，このシステム・コールで指定された周期起動ハンドラを新規に登録します。

また，このシステム・コールを発行した際，*pk_dcyc* で指定される領域に NADR（－1）を設定した場合，*cycno* で指定された周期起動ハンドラの登録が解除されます。

備考 1.　cycatr のビット 10 の値が 1（TA_DPID）以外の場合，gp の内容は意味を持ちません。

2.　cycatr のビット 11 の値が 1（TA_DPIC）以外の場合，tp の内容は意味を持ちません。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	def_cyc システム・コールが CF 定義されていない
E_RSATR	－24	属性 cycatr の指定が不正である
E_PAR	－33	パラメータの指定が不正である ・指定番号の指定が不正（*cycno*≦0，最大周期起動ハンドラ登録数＜*cycno*）である ・周期起動ハンドラ登録情報を格納したパケットの先頭アドレスの指定が不正（*pk_dcyc*＝0）である ・起動アドレスの指定が不正（cychdr＝0）である ・初期活性状態 cycact の指定が不正である ・起動時間間隔の指定が不正（cyctim≦0）である

保守／廃止

activate cyclic handler (－94)

act_cyc

タスク / 非タスク

【概　　要】

周期起動ハンドラの活性状態を制御する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = act_cyc(HNO cycno, UINT cycact);
```

【パラメータ】

入出力	パラメータ	説　明
入	HNO　*cycno;*	周期起動ハンドラの指定番号
入	UINT　*cycact;*	活性状態／周期カウンタの指定 TCY_OFF（0）：活性状態を OFF 状態へ変更 TCY_ON（1）：活性状態を ON 状態へ変更 TCY_INI（2）：周期カウンタを初期化

【説　　明】

cycno で指定され周期起動ハンドラの活性状態を *cycact* で指定された状態に変更します。

次に，*cycact* の指定形式を示します。

- *cycact*＝TCY_OFF

  対象周期起動ハンドラの活性状態を OFF 状態に変更します。

  これにより，起動時間に達しても，対象周期起動ハンドラは起動されません。

  注意　RX850 Pro では，周期起動ハンドラの活性状態が OFF 状態でも，周期カウンタのカウント処理は行われます。

- *cycact*＝TCY_ON

  対象周期起動ハンドラの活性状態を ON 状態に変更します。

  これにより，起動時間に達した際には，対象周期起動ハンドラが起動されます。

- *cycact*＝TCY_INI

  対象周期起動ハンドラの周期カウンタを初期化します。

- *cycact*＝（TCY_ON | TCY_INI）

  対象周期起動ハンドラの活性状態を ON 状態に変更したあと，周期カウンタを初期化します。

  これにより，起動時間に達した際には，対象周期起動ハンドラが起動されます。

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	act_cyc **システム・コールが** CF **定義されていない**
E_PAR	－33	**パラメータの指定が不正である** **・周期起動ハンドラの指定番号が不正（***cycno***≦0，最大周期起動ハンドラ登録数＜***cycno***）である** **・活性状態／周期カウンタ** *cycact* **の指定が不正である**
*E_NOEXS	－52	**対象周期起動ハンドラが登録されていない**

保守／廃止

refer cyclic handler status (－92)

ref_cyc

タスク / 非タスク

【概　　要】

周期起動ハンドラ情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ref_cyc(T_RCYC *pk_rcyc, HNO cycno);
```

【パラメータ】

入出力	パラメータ	説　　明
出	T_RCYC　*pk_rcyc;	周期起動ハンドラ情報を格納するパケットの先頭アドレス
入	HNO　cycno;	周期起動ハンドラの指定番号

・周期起動ハンドラ情報 T_RCYC の構造

```
typedef struct t_rcyc {
    VP              exinf;          /* 拡張情報 */
    CYCTIME         lfttim;         /* 残り時間 */
    UINT            cycact;         /* 現在の活性状態 */
} T_RCYC;
```

【説　　明】

cycno で指定された周期起動ハンドラの周期起動ハンドラ情報（拡張情報，残り時間など）を *pk_rcyc* で指定されるパケットに格納します。

次に，周期起動ハンドラ情報の詳細を示します。

exinf …　拡張情報

lfttim …　次に周期起動ハンドラを起動するまでの残り時間（単位：基本クロック周期）

cycact …　現在の活性状態

TCY_OFF（0）　：　活性状態は OFF 状態

TCY_ON（1）　：　活性状態は ON 状態

保守／廃止

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	ref_cyc **システム・コールが** CF **定義されていない**
E_PAR	－33	**パラメータの指定が不正である** **・周期起動ハンドラ情報を格納するパケットの先頭アドレスが不正（***pk_rcyc***＝0）である** **・周期起動ハンドラの指定番号が不正（***cycno***≦0，最大周期起動ハンドラ登録数＜***cycno***）である**
*E_NOEXS	－52	**対象周期起動ハンドラが登録されていない**

保守／廃止

11.8.7　システム管理機能システム・コール

ここでは，システムに依存した処理を行うシステム・コールのグループ（システム管理機能システム・コール）について説明します。

表 11－11 に，システム管理機能システム・コールの一覧を示します。

表11－11　システム管理機能システム・コール

システム・コール	機　　能
get_ver	RX850 Pro のバージョン情報を獲得する
ref_sys	システム情報を獲得する
def_svc	拡張 SVC ハンドラを登録／登録解除する
viss_svc	拡張 SVC ハンドラを呼び出す

保守／廃止

get version information (－16)

get_ver

タスク / 非タスク

【概　　要】

RX850 Pro のバージョン情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = get_ver(T_VER *pk_ver);
```

【パラメータ】

入出力	パラメータ	説　　明
出	T_VER　　*pk_ver;	バージョン情報を格納するパケットの先頭アドレス

・バージョン情報 T_VER の構造

```
typedef struct t_ver {
    UH    maker;          /* OS メーカ */
    UH    id;             /* OS 形式番号 */
    UH    spver;          /* 仕様書バージョン番号 */
    UH    prver;          /* OS 製品バージョン番号 */
    UH    prno[4];        /* 製品番号／製品管理情報 */
    UH    cpu;            /* CPU 情報 */
    UH    var;            /* バリエーション記述子 */
} T_VER;
```

【説　　明】

RX850 Pro のバージョン情報（OS メーカ，OS 形式番号など）を *pk_ver* で指定されるパケットに格納します。

次に，バージョン情報の詳細を示します。

maker …　OS メーカ
　　H’000d： NEC
id …　OS 形式番号
　　H’0000： 未使用
spver …　仕様書バージョン番号
　　H’5302： μITRON3.0 Ver.3.02
prver …　OS 製品バージョン番号
　　H’0300： RX850 Pro　Ver.3.00
prno[4] …　製品番号／製品管理情報
　　不定　： 出荷製品のシリアル番号（出荷製品ごとに異なります）

保守／廃止

cpu …　CPU 情報

H’0d37：　μPD703100

var …　バリエーション記述子

H’c000：　μITRON レベル E，ファイル・サポートなし

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	get_ver システム・コールが CF 定義されていない
E_PAR	－33	バージョン情報を格納するパケットの先頭アドレスが不正（*pk_ver*＝0）である

保守／廃止

refer system status (－12)

ref_sys

タスク / 非タスク

【概　　要】

システム情報を獲得する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = ref_sys(T_RSYS *pk_rsys);
```

【パラメータ】

入出力	パラメータ	説　　明
出	`T_RSYS  *pk_rsys;`	システム情報を格納するパケットの先頭アドレス

・システム情報 T_RSYS の構造

```
typedef struct t_rsys {
    INT   sysstat;          /* システムの状態 */
} T_RSYS;
```

【説　　明】

ダイナミックに変化するシステム情報（システムの状態）の現在値を *pk_rsys* で指定されるパケットに格納します。

次に，システム情報の詳細を示します。

sysstat … システムの状態

- TSS_TSK（0）　：タスクの処理を実行中
ディスパッチ処理は許可状態
- TSS_DDSP（1）：タスクの処理を実行中
ディスパッチ処理は禁止状態
- TSS_LOC（3）　：タスクの処理を実行中
マスカブル割り込みの受け付けとディスパッチ処理は禁止状態
- TSS_INDP（4）：非タスク（割り込みハンドラ，周期起動ハンドラなど）の処理を実行中

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	ref_sys システム・コールが CF 定義されていない
E_PAR	－33	システム情報を格納するパケットの先頭アドレスが不正（*pk_rsys*＝0）である

保守／廃止

define supervisor call handler (－9)

def_svc

タスク / 非タスク

【概　　要】

拡張 SVC ハンドラを登録／登録解除する。

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = def_svc(FN s_fncd, T_DSVC *pk_dsvc);
```

【パラメータ】

入出力	パラメータ	説　　明
入	FN　　*s_fncd*;	拡張 SVC ハンドラの拡張機能コード
入	T_DSVC　*pk_dsvc*;	拡張 SVC ハンドラ登録情報を格納したパケットの先頭アドレス

・拡張 SVC ハンドラ登録情報 T_DSVC の構造

```
typedef struct t_dsvc {
    ATR   svcatr;          /* 拡張 SVC ハンドラの属性 */
    FP    svchdr;          /* 拡張 SVC ハンドラの起動アドレス */
    VP    gp;              /* 拡張 SVC ハンドラの固有 GP レジスタ値 */
    VP    tp;              /* 拡張 SVC ハンドラの固有 TP レジスタ値 */
} T_DSVC;
```

【説　　明】

pk_dsvc で指定された情報を基に，*s_fncd* で指定された拡張機能コードを持つ拡張 SVC ハンドラを登録します。

次に，拡張 SVC ハンドラ登録情報の詳細を示します。

svcatr …　拡張 SVC ハンドラの属性

ビット 0 …　拡張 SVC ハンドラの記述言語

TA_ASM（0）　：　アセンブリ言語

TA_HLNG（1）：　C 言語

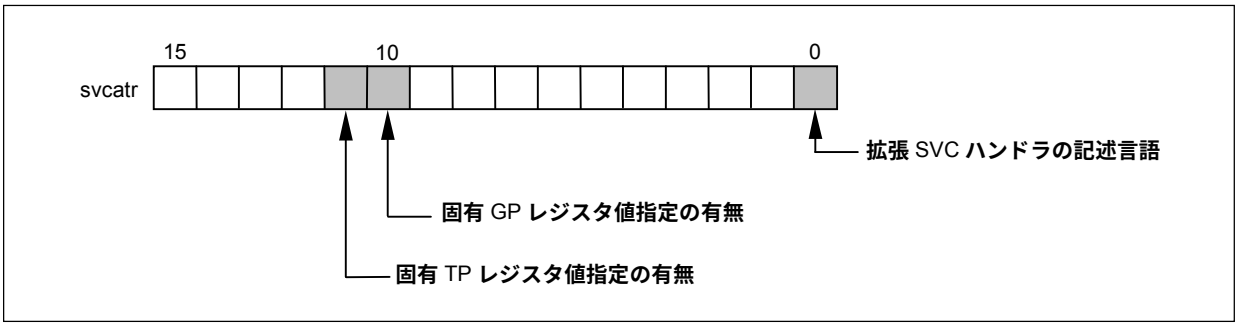
ビット 10 …　固有 GP レジスタ値指定の有無

TA_DPID（1）　：　固有 GP レジスタ値を指定

ビット 11 …　固有 TP レジスタ値指定の有無

TA_DPIC（1）　：　固有 TP レジスタ値を指定

保守／廃止



svchdr …　拡張 SVC ハンドラの起動アドレス

gp …　拡張 SVC ハンドラの固有 GP レジスタ値

tp …　拡張 SVC ハンドラの固有 TP レジスタ値

このシステム・コールを発行した際，すでに対象拡張機能コードに対応した拡張 SVC ハンドラが登録されていた場合には，エラーとしては扱わず，このシステム・コールで指定された拡張 SVC ハンドラを新規に登録します。

また，このシステム・コールを発行した際，*pk_dsvc* で指定される領域に NADR（－1）を設定した場合，*s_fncd* で指定された拡張 SVC ハンドラの登録が解除されます。

備考 1.　svcatr のビット 10 の値が 1（TA_DPID）以外の場合，gp の内容は意味を持ちません。

2.　svcatr のビット 11 の値が 1（TA_DPIC）以外の場合，tp の内容は意味を持ちません。

【戻　り　値】

*E_OK	0	正常終了
*E_NOSPT	－17	def_svc システム・コールが CF 定義されていない
E_RSATR	－24	属性 svcatr の指定が不正である
E_PAR	－33	パラメータの指定が不正である ・拡張機能コードの指定が不正 （*s_fncd*≦0，最大拡張 SVC ハンドラ登録数＜*s_fncd*）である ・拡張 SVC ハンドラ登録情報を格納したパケットの先頭アドレスが不正 （*pk_dsvc*＝0）である ・起動アドレスの指定が不正（svchdr＝0）である

保守／廃止

issued supervisor call handler (－250)

viss_svc

タスク / 非タスク

【概　　要】

拡張 SVC **ハンドラを呼び出す。**

【C 言語形式】

```
#include  <stdrx85p.h>
ER        ercd = viss_svc(FN s_fncd, VW prm1, VW prm2, VW prm3);
```

【パラメータ】

入出力	パラメータ	説　　明
入	FN　*s_fncd;*	**拡張** SVC **ハンドラの拡張機能コード**
入	VW　*prm1;*	**拡張** SVC **ハンドラへの引き渡しパラメータ** 1
入	VW　*prm2;*	**拡張** SVC **ハンドラへの引き渡しパラメータ** 2
入	VW　*prm3;*	**拡張** SVC **ハンドラへの引き渡しパラメータ** 3

【説　　明】

s_fncd **で指定された拡張機能コードを持つ拡張** SVC **ハンドラを呼び出します。**

備考　**このシステム・コールを利用して拡張** SVC **ハンドラを呼び出す場合には，拡張** SVC **ハンドラ用インタフェース・ライブラリの記述が不要となります。**

【戻　り　値】

*E_OK	0	**正常終了**
*E_NOSPT	－17	viss_svc **システム・コールが** CF **定義されていない，または登録されていない拡張** SVC **ハンドラを呼び出した**
E_PAR	－33	**拡張機能コードの指定が不正（***s_fncd***≦0，最大拡張** SVC **ハンドラ登録数＜***s_fncd***）である**
その他		**拡張** SVC **ハンドラからの戻り値**

付録 A　プログラミングのために

この章では，NEC エレクトロニクス製 V850 シリーズ用 C コンパイラ CA850，または米国 Green Hills Software, Inc.製 C クロス V800 コンパイラ CCV850 を使用した際の処理プログラムの記述方法について説明します。

A.1　概　　要

RX850 Pro では，処理プログラムを用途別に次のように区別しています。

●タスク

RX850 Pro の管理下で実行可能な処理プログラムの最小単位です。

●直接起動割り込みハンドラ

割り込みが発生した際，RX850 Pro を介在させることなく起動される割り込み処理専用ルーチンです。

このため，ハードウエアの限界に近い高速な応答性が期待されます。

●間接起動割り込みハンドラ

割り込みが発生した際，RX850 Pro に割り込み前処理（レジスタの退避処理，スタックの切り替え処理など）を行わせたあと起動される割り込み処理専用ルーチンです。

このため，直接起動割り込みハンドラに比べて応答性の面では劣りますが，RX850 Pro による割り込み前処理が行われているため，ハンドラ内での処理が簡素化されるといった利点があります。

●周期起動ハンドラ

一定の起動時間に達した際，ただちに起動される周期処理専用ルーチンであり，タスクとは独立したものとして扱われます。このため，起動時間に達した際には，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，周期起動ハンドラに制御が移ります。

なお，周期起動ハンドラは，ユーザが記述する周期的な処理プログラムの中で，実行開始までのオーバヘッドが最も小さい処理プログラムです。

●拡張 SVC ハンドラ

ユーザが拡張システム・コールとして登録した関数です。

なお，これらの処理プログラムは，一般的に，または RX850 Pro を使用する上での約束ごとなどにより，それぞれに基本型があります。

保守／廃止

A.2　キー・ワード

コンフィギュレータでは，次に示す文字列をキー・ワードとして予約しています。したがって，これらの文字列をほかの用途に使用することは禁止されています。

clkhdr	clktim	cyc	defstk	flg
flgsvc	ini	inthdr	intstk	intsvc
maxcyc	maxflg	maxint	maxintfactor	maxmbx
maxmpl	maxpri	maxsem	maxsvc	maxtsk
mbx	mbxsvc	mem	mpl	mplsvc
no_use	prtflg	prtmbx	prtmpl	prtsem
prttsk	RX850PRO	rxsers	sct_def	sem
semsvc	ser_def	sit_def	SPOL0	SPOL1
svc	syssvc	TA_ASM	TA_DISINT	TA_ENAINT
TA_HLNG	TA_MFIFO	TA_MPRI	TA_TFIFO	TA_TPRI
TA_WMUL	TA_WSGL	TCY_OFF	TCY_ON	timsvc
tsk	tsksvc	TTS_DMT	TTS_RDY	UPOL0
UPOL1				

A.3　予約語

RX850 Pro では，次に示す文字列を外部シンボルとして予約しています。したがって，これらの文字列をほかの用途に使用することは禁止されています。

_x_	_f_	_e_	_rx_

備考　これらの文字列を使用することが禁止されるのは，単一のロード・モジュールを作成した場合です。RX850 Pro とアプリケーションが分離しているロード・モジュールを作成している場合は，これらの文字列から始まるシンボルを使用しても問題ありません。

保守／廃止

A.4　タスク

A.4.1　CA850 対応版の場合

タスクを C 言語で記述する場合，プラグマ指令による関数宣言を行ったのち，INT 型の引数を 1 つ持った void 型の関数として記述します。

なお，引数（*stacd*）には，sta_tsk システム・コール発行時に指定された起動コードが設定されます。

次に，CA850 を使用したときのタスクの記述形式（C 言語）を示します。

図A－1　CA850を使用したときのタスクの記述形式（C言語）

```
#include <stdrx85p.h>

#pragma  rtos_task  func_task
void
func_task(INT stacd)
{

        /* タスク func_task の本体処理 */
        ..........................
        ..........................
        ..........................
        /* タスク func_task の終了 */
        ext_tsk( );
}
```

備考　プラグマ指令による関数宣言についての詳細は，CA850 **ユーザーズ・マニュアル** C **言語編**（U16054J）を参照してください。

また，タスクをアセンブリ言語で記述する場合は，CA850 の関数呼び出し規約に従った関数として記述します。

なお，引数（r6 レジスタ）には，sta_tsk システム・コール発行時に指定された起動コードが設定されます。

次に，CA850 を使用したときのタスクの記述形式（アセンブリ言語）を示します。

図A－2　CA850を使用したときのタスクの記述形式（アセンブリ言語）

```
.include "stdrx85p.inc"

          .text
          .align   4
          .globl   _func_task
_func_task :
          # タスク func_task の本体処理
          ........................
          ........................
          ........................
          # タスク func_task の終了
          jr       _ext_tsk
```

保守／廃止

A.4.2　CCV850 対応版の場合

タスクを C 言語で記述する場合，INT 型の引数を 1 つ持った void 型の関数として記述します。

なお，引数（stacd）には，sta_tsk システム・コール発行時に指定された起動コードが設定されます。

次に，CCV850 を使用したときのタスクの記述形式（C 言語）を示します。

図A－3　CCV850を使用したときのタスクの記述形式（C言語）

```
#include  <stdrx85p.h>

void
func_task(INT stacd)
{

          /* タスク func_task の本体処理 */
          ..........................
          ..........................
          ..........................
          /* タスク func_task の終了 */
          ext_tsk( );
}
```

また，タスクをアセンブリ言語で記述する場合は，CCV850 関数呼び出し規約に従った関数として記述します。

なお，引数（r6 レジスタ）には，sta_tsk システム・コール発行時に指定された起動コードが設定されます。

次に，CCV850 を使用したときのタスクの記述形式（アセンブリ言語）を示します。

図A－4　CCV850を使用したときのタスクの記述形式（アセンブリ言語）

```
#include <stdrx85p.h>

            .text
            .align    4
            .globl    _func_task
_func_task :
            # タスク func_task の本体処理
            ........................
            ........................
            ........................
            # タスク func_task の終了
            jr        _ext_tsk
```

注意 1. タスクをアセンブリ言語で記述する場合，ファイル名の拡張子には“.850”を指定してください。

2. アセンブリ言語で記述したタスクをコンパイルする場合，コンパイル時のオプションとして“-D_ _asm_ _”を指定してください。

保守／廃止

A.5　直接起動割り込みハンドラ

A.5.1　CA850 対応版の場合

直接起動割り込みハンドラを記述する場合は，アセンブリ言語を使用します。ただし，処理本体を C 言語で記述し，jarl 命令で呼び出す形を使うこともできます。

直接起動割り込みハンドラでは，その処理前にレジスタの退避，処理後に復帰の処理を行う必要があります。

しかし，RX850 Pro ではレジスタの退避，復帰処理を記述したマクロを用意しており，アセンブリ言語でのハンドラ記述におけるユーザ負担を軽減しています。

次に，CA850 を使用したときの直接起動割り込みハンドラの記述形式（アセンブリ言語）を示します。

図A－5　CA850を使用したときの直接起動割り込みハンドラの記述形式（アセンブリ言語）

```
.include "stdrx85p.inc"
          .section "int_name", text
          jr       _func_inthdr

          .text
          .align   4
          .globl   _func_inthdr
_func_inthdr:
          /* レジスタの退避，スタックの切り替え */
          RTOS_IntEntry

          /* 直接起動割り込みハンドラの本体処理 */
          .extern  _inthdr_body
          jarl     _inthdr_body, lp

          /* r10：ハンドラ復帰後に起床するタスクの ID */
          /* スタックの切り替え，レジスタの復帰 */
          /* 直接起動割り込みハンドラからの復帰と指定したタスクの起床 */
          RTOS_IntReturnWakeup r10
```

```
#include <stdrx85p.h>

ID
inthdr_body( )
{

        __asm("mov #__tp_TEXT, tp");
        __asm("mov #__gp_DATA, gp");
        /* 直接起動割り込みハンドラ func_inthdr の本体処理 */
        ....................................
        ....................................
        ....................................
        /* 直接起動割り込みハンドラ func_inthdr 本体からの復帰 */
        return    tskid
}
```

まず，ハンドラ・アドレスに割り込みハンドラのエントリ処理（jr 命令）を記述します。この例では，2，3 行目がこれにあたります。

次に，割り込みハンドラ処理の本体についてです。

マクロ RTOS_IntEntry では，RX850 Pro に対してハンドラ開始の通知，テンポラリ・レジスタと lp の退避，そしてスタックの切り替えなどを行います。そのあと，これら以外のレジスタ（r20～r30）の退避を行い，ハンドラ本体部へと処理が移ります。記述例では，ハンドラ本体部である C の関数 *inthdr_body* を呼び出しています。ハンドラ本体処理を行う前に，ハンドラで使用する TP（テキスト・ポインタ）と GP（グローバル・ポインタ）の設定を行います。5.3 **直接起動割り込みハンドラ**の節に説明があるように，直接起動割り込みハンドラで使用する TP と GP の値が不定になるためです。この設定は，アセンブリ言語で書かなくてはならないので，C 言語でハンドラ本体を書くときは，例のように__asm 命令を使用するか，#pragma asm ～ pragma endasm 指令を使って記述してください。ハンドラ本体部では，ユーザーズ・マニュアルに記載されている「ハンドラから発行可能なシステム・コール」が発行可能です。

ハンドラの発行処理が終了したら，ユーザが退避したレジスタの復帰と割り込みハンドラからの復帰を行います。割り込み復帰後に指定したタスクを起床させるときは，レジスタ r10 に起床させるタスクの ID をセットする必要があります。記述例では，*inthdr_body* から復帰するときに戻り値としてタスク ID を返しており，その値が r10 にコピーされます。これは，CA850 がこのようなコードを出力しています。

このあとに，他タスクを起床させて復帰するときは，マクロ RTOS_IntReturnWakeup r10 を記述します。単にハンドラから復帰するときは，マクロ RTOS_IntReturn を記述します。システム・コール jr_ret_int または jr_ret_wup でハンドラを終了することもできますが，その場合には退避したレジスタの復帰処理をシステム・コール発行前に行う必要があります。用意されたマクロを使用して，ハンドラからの復帰を行う方が容易です。

簡単な処理のみを行って復帰する場合，reti 命令を使うこともできます。そのときは，命令を発行する前に，レジスタの復帰処理を行う必要があります。この命令に相当し，レジスタの復帰処理まで行うマクロが RTOS_IntExit です。ただし，これらの命令，マクロを使用して復帰するときは，ハンドラ内でシステム・コールを発行しないでください。なお，システム・コールをハンドラ内で使用している場合には，RTOS_IntReturnWakeup や RTOS_IntReturn を使用して復帰してください。

また，ハンドラ処理中に多重割り込みを許可する場合は，RTOS_IntEntry 処理後から，

RTOS_IntReturnWakeup や RTOS_IntReturn，RTOS_IntExit までの間で ei/di を行ってください。

割り込みが発生した際にプロセッサが制御を移すハンドラ・アドレスに対して直接起動割り込みハンドラへの分岐命令などを設定する必要があります。図 A－5 内の .section 疑似命令がこれにあたります。

備考　割り込みが発生した際にプロセッサが制御を移すハンドラ・アドレスに対して直接起動割り込みハンドラへの分岐命令などを設定する必要があります。**図** A－5 内の **.section** 疑似命令がこれにあたります。.section 疑似命令についての詳細は，CA850 C **コンパイラ・パッケージ ユーザーズ・マニュアル アセンブリ言語編**（U16042J）を参照してください。なお，“ *int_name* ”には，デバイス・ファイルで定義されている割り込み要求名を指定します。

保守／廃止

A.5.2　CCV850 対応版の場合

直接起動割り込みハンドラを記述する場合は，アセンブリ言語を使用します。ただし，処理本体を C 言語で記述し，jarl 命令で呼び出す形を使うこともできます。

直接起動割り込みハンドラでは，その処理前にレジスタの退避，処理後に復帰の処理を行う必要があります。

しかし，RX850 Pro ではレジスタの退避，復帰処理を記述したマクロを用意しており，アセンブリ言語でのハンドラ記述におけるユーザ負担を軽減しています。

次に，CCV850 を使用したときの直接起動割り込みハンドラの記述形式（アセンブリ言語）を示します。

図A－6　CCV850を使用したときの直接起動割り込みハンドラの記述形式（アセンブリ言語）

```
.include "stdrx85p.inc"
          .org     handler_address_number
          jr       _func_inthdr

          .text
          .align   4
          .globl   _func_inthdr
_func_inthdr:
          /* レジスタの退避，スタックの切り替え */
          RTOS_IntEntry

          /* 直接起動割り込みハンドラの本体処理 */
          .extern  _inthdr_body
          jarl     _inthdr_body, lp

          /* r10：ハンドラ復帰後に起床するタスクの ID */
          /* スタックの切り替え，レジスタの復帰 */
          /* 直接起動割り込みハンドラからの復帰と指定したタスクの起床 */
           jr      RTOS_IntReturnWakeup r10
```

保守／廃止

```
#include <stdrx85p.h>

ID
inthdr_body( )
{

        __asm("__tp, tp");
        __asm("__gp, gp");
        /* 直接起動割り込みハンドラ func_inthdr の本体処理 */
        ....................................
        ....................................
        ....................................
        /* 直接起動割り込みハンドラ func_inthdr 本体からの復帰 */
        return    tskid
}
```

まず，ハンドラ・アドレスに割り込みハンドラのエントリ処理（jr 命令）を記述します。この例では，2，3**行目がこれにあたります。**

次に，割り込みハンドラ処理の本体についてです。

マクロ RTOS_IntEntry **では，**RX850 Pro **に対してハンドラ開始の通知，テンポラリ・レジスタと** lp **の退避，そしてスタックの切り替えなどを行います。そのあと，これら以外のレジスタ（**r20～r30**）の退避を行い，ハンドラ本体部へと処理が移ります。記述例では，ハンドラ本体部である** C **の関数** *inthdr_body* **を呼び出しています。ハンドラ本体処理を行う前に，ハンドラで使用する** TP**（テキスト・ポインタ）と** GP**（グローバル・ポインタ）の設定を行います。**5.3 **直接起動割り込みハンドラの節に説明があるように，直接起動割り込みハンドラで使用する** TP **と** GP **の値が不定になるためです。この設定は，アセンブリ言語で書かなくてはならないので，**C **言語でハンドラ本体を書くときは，例のように_ _asm 命令を使用するか，＃**pragma asm ～ pragma endasm **指令を使って記述してください。ハンドラ本体部では，ユーザーズ・マニュアルに記載されている「ハンドラから発行可能なシステム・コール」が発行可能です。**

ハンドラの発行処理が終了したら，ユーザが退避したレジスタの復帰と割り込みハンドラからの復帰を行います。割り込み復帰後に指定したタスクを起床させるときは，レジスタ r10 **に起床させるタスクの** ID **をセットする必要があります。記述例では，***inthdr_body* **から復帰するときに戻り値としてタスク** ID **を返しており，その値が** r10 **にコピーされます。これは，**CCV850 **がこのようなコードを出力しています。**

このあとに，他タスクを起床させて復帰するときは，マクロ RTOS_IntReturnWakeup r10 **を記述します。単にハンドラから復帰するときは，マクロ** RTOS_IntReturn **を記述します。システム・コール** jr_ret_int **または** jr_ret_wup **でハンドラを終了することもできますが，その場合には退避したレジスタの復帰処理をシステム・コール発行前に行う必要があります。用意されたマクロを使用して，ハンドラからの復帰を行う方が容易です。**

簡単な処理のみを行って復帰する場合，reti **命令を使うこともできます。そのときは，命令を発行する前に，レジスタの復帰処理を行う必要があります。この命令に相当し，レジスタの復帰処理まで行うマクロが** RTOS_IntExit **です。ただし，これらの命令，マクロを使用して復帰するときは，ハンドラ内でシステム・コールを発行しないでください。なお，システム・コールをハンドラ内で使用している場合には，**RTOS_IntReturnWakeup **や** RTOS_IntReturn **を使用して復帰してください。**

また，ハンドラ処理中に多重割り込みを許可する場合は，RTOS_IntEntry **処理後から，**

RTOS_IntReturnWakeup や RTOS_IntReturn，RTOS_IntExit までの間で ei/di を行ってください。

備考　割り込みが発生した際にプロセッサが制御を移すハンドラ・アドレスに対して，直接起動割り込みハンドラへの分岐命令などを設定する必要があります。図A－6内の.org命令部分が，これに当たります。

なお，*handler_address_number*には，割り込みのハンドラ・アドレスを指定します。

注意　直接起動割り込みハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には“.850”を指定してください。

保守／廃止

A.6　間接起動割り込みハンドラ

A.6.1　CA850 対応版の場合

間接起動割り込みハンドラを C 言語で記述する場合，引数を持たない ID 型の関数として記述します。

次に，CA850 を使用したときの間接起動割り込みハンドラの記述形式（C 言語）を示します。

図A－7　CA850を使用したときの間接起動割り込みハンドラの記述形式（C言語）

```
#include <stdrx85p.h>

ID
func_inthdr( )
{
        /* 間接起動割り込みハンドラ func_inthdr の本体処理 */
        ....................................
        ....................................
        ....................................
        /* 間接起動割り込みハンドラ func_inthdr からの復帰 */
        return(TSK_NULL);
}
```

備考　間接起動割り込みハンドラは，ニュークリアス内の割り込み前処理から呼び出されるサブルーチンですが，間接起動割り込みハンドラを記述する場合，割り込みが発生した際にプロセッサへ制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要があります。これはアセンブリ言語で記述する必要があります。

ただし，RX850 Proでは，この分岐命令として記述すべき処理をマクロで提供しているので，それを使用します。たとえば，INTP100（アドレス：0x100）というマスカブル割り込みを間接起動割り込みハンドラとして使用するには，

```
.section “INTP100”
RTOS_IntEntry_Indirect
```

と記述します。

なお，タイマ割り込みも間接起動割り込みハンドラとして扱われるので，同様の記述が必要です。

また，間接起動割り込みハンドラをアセンブリ言語で記述する場合は，CA850 の関数呼び出し規約に従った関数として記述します。

次に，CA850 を使用したときの間接起動割り込みハンドラの記述形式（アセンブリ言語）を示します。

図A－8　CA850を使用したときの間接起動割り込みハンドラの記述形式（アセンブリ言語）

```
.include “stdrx85p.inc”

          .text
          .align    4
          .globl    _func_inthdr
_func_inthdr :
          # 間接起動割り込みハンドラ func_inthdr の本体処理
          .....................................
          .....................................
          .....................................
          # 間接起動割り込みハンドラ func_inthdr からの復帰
          mov       TSK_NULL, r10
          jmp       [lp]
```

備考　間接起動割り込みハンドラは，ニュークリアス内の割り込み前処理から呼び出されるサブルーチンですが，間接起動割り込みハンドラを記述する場合，割り込みが発生した際にプロセッサへ制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要があります。これはアセンブリ言語で記述する必要があります。

ただし，RX850 Proでは，この分岐命令として記述すべき処理をマクロで提供しているので，それを使用します。たとえば，INTP100（アドレス：0x100）というマスカブル割り込みを間接起動割り込みハンドラとして使用するには，

```
.section “INTP100”
RTOS_IntEntry_Indirect
```

と記述します。

なお，タイマ割り込みも間接起動割り込みハンドラとして扱われるので，同様の記述が必要です。

A.6.2　CCV850 対応版の場合

間接起動割り込みハンドラを C 言語で記述する場合，引数を持たない ID 型の関数として記述します。

次に，CCV850 を使用したときの間接起動割り込みハンドラの記述形式（C 言語）を示します。

図A－9　CCV850を使用したときの間接起動割り込みハンドラの記述形式（C言語）

```
#include  <stdrx85p.h>


ID
func_inthdr( )
{

            /* 間接起動割り込みハンドラ func_inthdr の本体処理 */
            ....................................
            ....................................
            ....................................
            /* 間接起動割り込みハンドラ func_inthdr からの復帰 */
            return(TSK_NULL);
}
```

備考　間接起動割り込みハンドラは，ニュークリアス内の割り込み前処理から呼び出されるサブルーチンですが，間接起動割り込みハンドラを記述する場合，割り込みが発生した際にプロセッサへ制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要があります。これはアセンブリ言語で記述する必要があります。

ただし，RX850 Proでは，この分岐命令として記述すべき処理をマクロで提供しているので，それを使用します。たとえば，INTP100（アドレス：0x100）というマスカブル割り込みを間接起動割り込みハンドラとして使用するには，

```
.org 00000100
RTOS_IntEntry_Indirect
```

と記述します。

なお，タイマ割り込みも間接起動割り込みハンドラとして扱われるので，同様の記述が必要です。

また，間接起動割り込みハンドラをアセンブリ言語で記述する場合は，CCV850 の関数呼び出し規約に従った関数として記述します。

次に，CCV850 を使用したときの間接起動割り込みハンドラの記述形式（アセンブリ言語）を示します。

図A－10　CCV850を使用したときの間接起動割り込みハンドラの記述形式（アセンブリ言語）

```
#include <stdrx85p.h>

          .text
          .align    4
          .globl    _func_inthdr
_func_inthdr :
          # 間接起動割り込みハンドラ func_inthdr の本体処理
          .....................................
          .....................................
          .....................................
          # 間接起動割り込みハンドラ func_inthdr からの復帰
          mov       TSK_NULL, r10
          jmp       [lp]
```

備考　間接起動割り込みハンドラは，ニュークリアス内の割り込み前処理から呼び出されるサブルーチンですが，間接起動割り込みハンドラを記述する場合，割り込みが発生した際にプロセッサへ制御を移すハンドラ・アドレスに対して，間接起動割り込みハンドラへの分岐命令などを設定する必要があります。これはアセンブリ言語で記述する必要があります。

ただし，RX850 Proでは，この分岐命令として記述すべき処理をマクロで提供しているので，それを使用します。たとえば，INTP100（アドレス：0x100）というマスカブル割り込みを間接起動割り込みハンドラとして使用するには，

```
.org 00000100
RTOS_IntEntry_Indirect
```

と記述します。

なお，タイマ割り込みも間接起動割り込みハンドラとして扱われるので，同様の記述が必要です。

注意　間接起動割り込みハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には“.850”を指定してください。

保守／廃止

A.7　周期起動ハンドラ

A.7.1　CA850 対応版の場合

周期起動ハンドラを C 言語で記述する場合，引数を持たない void 型の関数として記述します。

次に，CA850 を使用したときの周期起動ハンドラの記述形式（C 言語）を示します。

図A－11　CA850を使用したときの周期起動ハンドラの記述形式（C言語）

```
#include <stdrx85p.h>

void
func_cychdr( )
{

        /* 周期起動ハンドラ func_cychdr の本体処理 */
        ....................................
        ....................................
        ....................................
        /* 周期起動ハンドラ func_cychdr からの復帰 */
        return;
}
```

備考　周期起動ハンドラは，ニュークリアス内のシステム・クロック処理から呼び出されるサブルーチンです。

また，周期起動ハンドラをアセンブリ言語で記述する場合は，CA850 の関数呼び出し規約に従った関数として記述します。

次に，CA850 を使用したときの周期起動ハンドラの記述形式（アセンブリ言語）を示します。

図A－12　CA850を使用したときの周期起動ハンドラの記述形式（アセンブリ言語）

```
.include "stdrx85p.inc"

        .text
        .align   4
        .globl   _func_cychdr
_func_cychdr :
        # 周期起動ハンドラ func_cychdr の本体処理
        .....................................
        .....................................
        .....................................
        # 周期起動ハンドラ func_cychdr からの復帰
        jmp      [lp]
```

備考　周期起動ハンドラは，ニュークリアス内のシステム・クロック処理から呼び出されるサブルーチンです。

A.7.2　CCV850 対応版の場合

周期起動ハンドラを C 言語で記述する場合，引数を持たない void 型の関数として記述します。

次に，CCV850 を使用したときの周期起動ハンドラの記述形式（C 言語）を示します。

図A－13　CCV850を使用したときの周期起動ハンドラの記述形式（C言語）

```
#include  <stdrx85p.h>

void
func_cychdr( )
{
        /* 周期起動ハンドラ func_cychdr の本体処理 */
        ....................................
        ....................................
        ....................................
        /* 周期起動ハンドラ func_cychdr からの復帰 */
        return;
}
```

備考　周期起動ハンドラは，ニュークリアス内のシステム・クロック処理から呼び出されるサブルーチンです。

また，周期起動ハンドラをアセンブリ言語で記述する場合は，CCV850 の関数呼び出し規約に従った関数として記述します。

次に，CCV850 を使用したときの周期起動ハンドラの記述形式（アセンブリ言語）を示します。

図A－14　CCV850を使用したときの周期起動ハンドラの記述形式（アセンブリ言語）

```
#include <stdrx85p.h>

            .text
            .align    4
            .globl    _func_cychdr
_func_cychdr :
            # 周期起動ハンドラ func_cychdr の本体処理
            ..................................
            ..................................
            ..................................
            # 周期起動ハンドラ func_cychdr からの復帰
            jmp       [lp]
```

備考　周期起動ハンドラは，ニュークリアス内のシステム・クロック処理から呼び出されるサブルーチンです。

注意　周期起動ハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には“.850”を指定してください。

A.8　拡張 SVC ハンドラ

A.8.1　CA850 対応版の場合

拡張 SVC ハンドラを C 言語で記述する場合，INT 型の関数として記述します。

次に，CA850 を使用したときの拡張 SVC ハンドラの記述形式（C 言語）を示します。

図A－15　CA850を使用したときの拡張SVCハンドラの記述形式（C言語）

```
#include <stdrx85p.h>

INT
func_svchdr(VW prm1, VW prm2, VW prm3)
{
        int     ret;

        /* 拡張 SVC ハンドラ func_svchdr の本体処理 */
        ....................................
        ....................................
        ....................................
        /* 拡張 SVC ハンドラ func_svchdr からの復帰 */
        return(INT ret);
}
```

また，拡張 SVC ハンドラをアセンブリ言語で記述する場合は，CA850 の関数呼び出し規約に従った関数として記述します。

次に，CA850 を使用したときの拡張 SVC ハンドラの記述形式（アセンブリ言語）を示します。

図A－16　CA850を使用したときの拡張SVCハンドラの記述形式（アセンブリ言語）

```
.include "stdrx85p.inc"

        .text
        .align   4
        .globl   _func_svchdr
_func_svchdr :
        # 拡張 SVC ハンドラ func_svchdr の本体処理
        ..................................
        ..................................
        ..................................
        # 拡張 SVC ハンドラ func_svchdr からの復帰
        mov      ret, r10
        jmp      [lp]
```

保守／廃止

A.8.2　CCV850 対応版の場合

拡張 SVC ハンドラを C 言語で記述する場合，INT 型の関数として記述します。

次に，CCV850 を使用したときの拡張 SVC ハンドラの記述形式（C 言語）を示します。

図A－17　CCV850を使用したときの拡張SVCハンドラの記述形式（C言語）

```
#include  <stdrx85p.h>

INT
func_svchdr(VW prm1, VW prm2, VW prm3)
{
            int       ret;

            /* 拡張 SVC ハンドラ func_svchdr の本体処理 */
            .....................................
            .....................................
            .....................................
            /* 拡張 SVC ハンドラ func_svchdr からの復帰 */
            return(INT ret);
}
```

また，拡張 SVC ハンドラをアセンブリ言語で記述する場合は，CCV850 の関数呼び出し規約に従った関数として記述します。

次に，CCV850 を使用したときの拡張 SVC ハンドラの記述形式（アセンブリ言語）を示します。

図A－18　CCV850を使用したときの拡張SVCハンドラの記述形式（アセンブリ言語）

```
#include <stdrx85p.h>

            .text
            .align    4
            .globl    _func_svchdr
_func_svchdr :
            # 拡張 SVC ハンドラ func_svchdr の本体処理
            ..................................
            ..................................
            ..................................
            # 拡張 SVC ハンドラ func_svchdr からの復帰
            mov       ret, r10
            jmp       [lp]
```

注意　拡張SVCハンドラをアセンブリ言語で記述する場合，ファイル名の拡張子には“.850”を指定してください。

付録B　Q & A

【製品概要】

Q.1

RX850 Proは，ITRON, μITRONどちらに準拠していますか？

A.1

RX850 Proは，μITRON3に準拠しています。

Q.2

RX850 Proのニュークリアス（カーネル）サイズは？

A.2

5～13 Kバイトです。

使用するシステム・コールの数によって異なります。

システム・コールをすべて使った場合，13 Kバイトになります。

見積もりの詳細は，NEC Electronics Microcomputerのホーム・ページ内のFAQにある「RX850 Proメモリ容量見積もりページ」を使用してください。

Q.3

RX850 Proが使用するRAMサイズは？

A.3

RX850 Proが使用するRAMサイズは，アプリケーションに依存します。

作成したタスク数やその他資源数により，各管理ブロックの数は変化します。

各管理ブロックのサイズや，使用RAMサイズなどの見積もり方については，RX850 Pro　ユーザーズ・マニュアル　インストレーション編（U13774J）の第5章　メモリとその容量の見積もりを参照するか，NEC Electronics Microcomputerのホーム・ページ内のFAQにある「RX850 Proメモリ容量見積もりページ」を使用してください。

Q.4

RX850 Proは，何言語で書かれていますか？

A.4

すべてアセンブリ言語で書かれています。

保守／廃止

Q.5

RX850 Proをアプリケーションに組み込むとは，どのようなイメージでしょうか？

A.5

RX850 Proは，システム・コール処理部などがライブラリ（librxp.a/librxpm.a）で，共通処理部はオブジェクト（rxcore.o/rxtmcore.o）として提供されます。

つまり，システム・コール処理部はリンク時にそのライブラリを参照，共通処理部はオブジェクトをリンクし，組み込むという形になります。

librxp.aとlibrxm.aの違いはQ.65, rxcore.oとrxtmcore.oの違いはQ.111を参照してください。

Q.6

RX850とRX850 Proのどちらを使ったらよいでしょうか？

A.6

RX850とRX850 Proはともにμ ITRON3仕様ですが，2つの間には仕様に差があります。

主な違いは次の通りです。

- RX850は，資源はすべて静的生成ですが，RX850 Proは静的生成と動的生成の両方が可能です。
- RX850のメモリ・プールには可変長と固定長の両方がありますが，RX850 Proは可変長のみです。

RX850は，RX850 Proのサブセットのような位置付けで，コードサイズ，使用RAMはRX850 Proに比べて少ないです。

また，処理速度も若干速いです。

ただし，速度優先のためにr0相対命令を使用しているので，メモリ配置に制限があります。

一方，RX850 Proは，RX850に比べてコードサイズ，使用RAMサイズは大きいですが，メモリ配置の制限がありません。

また，資源の動的生成ができるので，アプリケーション作成の自由度が大きいです。

どちらのRXもすべてのV850シリーズ上で動きますが，このような観点から，省メモリのV850シリーズをターゲットにしている場合はRX850, MIPS値が高く，多くのメモリを持つV850シリーズをターゲットにしている場合はRX850 Proを使用することを推奨しています。

仕様の差分についての詳細は，文書が発行されています（RX850/RX850 Pro 仕様比較 文書番号：SUD-T-4961）ので，こちらを参照されると便利です。資料の入手は特約店またはNECエレクトロニクス営業拠点にご連絡ください。

Q.7

RX850 Proが対応しているコンパイラには，どのようなものがありますか？

A.7

現在は，NECエレクトロニクス製コンパイラ“ CA850 ”とGHS社製コンパイラ“ CCV850E ”に対応しています。

RX850 Proの製品には，両方のコンパイラに対応したものが1パッケージに入っており，インストール時にどちらかを選択することになります。

保守／廃止

Q.8

タスク・ディバッガRD850 Pro**を使いたいのですが，別売りですか？**

A.8

RD850 Pro**は**RX850 Pro**に標準添付されています。**

使用方法については，RD850 Pro**のユーザーズ・マニュアルを参照してください。**

Q.9

システム・パフォーマンス・アナライザAZ850**を使いたいのですが，別売りですか？**

A.9

AZ850**は，**RX850 Pro**には付属しておりませんので別売りですが，**SP850**には含まれています。**

AZ850**を使用すると，タスク遷移の情報や**CPU**の使用率などを知ることができます。**

Q.10

RX850 Ver.3.13**から**RX850 Pro**に移行する際に，注意することはありますか？**

A.10

システム・コール・レベルでは，互換が保たれていますので，プログラムの見直し自体はさほど必要ありません。

ただし，システム・コールの引数の型の違いが若干あります。

また，システム情報などでリンク配置される場所や，メモリの管理の仕方がかなり異なるため，この辺りの見直しは必要となります。

移植手順に関しては，文書が発行されています（RX850 Pro RX850（Ver.3.1x）**からの移行資料 文書番号：**SUD-T-4816**）ので，こちらを参照してください。資料の入手は特約店または**NEC**エレクトロニクス営業拠点にご連絡ください。**

Q.11

コンフィギュレーション（CF**）定義ファイルとは何ですか？**

A.11

ニュークリアス（カーネル）に提供する各種データが保持されているファイルです。

ユーザが決められたフォーマットで各種情報（タスク情報など）を記述し，そのファイルをコンフィギュレータ（cf850pro**）に通すことによって，アセンブリ言語ファイルに変換されます。**

保守／廃止

Q.12

RX850 Proの，コンフィギュレーション（CF）定義ファイルで定義するものは何ですか？

A.12

システム情報（SIT）と使用システム・コール情報（SCT）です。

SIT（System Information Table）情報としては，

- システム情報
- システム最大値情報
- システム・メモリ情報
- タスク情報
- イベント・フラグ情報
- セマフォ情報
- メールボックス情報
- メモリ・プール情報
- 周期ハンドラ情報
- 優先度情報
- 間接起動割り込みハンドラ情報
- クロック割り込み情報
- 拡張SVCハンドラ情報

を記述します。

SCT（System Call Table）情報としては，

- タスク管理／タスク付属同期管理機能システム・コール情報
- 同期通信管理機能システム・コール情報
- メモリ・プール管理機能システム・コール情報
- 時間管理機能システム・コール情報
- システム管理機能システム・コール情報

を記述します。

Q.13

コンフィギュレーション（CF）定義ファイルはどのように作るのですか？

A.13

規定されたフォーマットに従って，テキストで記述します。

フォーマットについては，RX850 Pro　**ユーザーズ・マニュアル　インストレーション編**（U13774J）**の第6章　コンフィギュレーション・ファイル**を参照してください。

保守／廃止

Q.14

コンフィギュレータ（cf850pro）とは何ですか？

A.14

ユーザが作成したコンフィギュレーション（CF）定義ファイルから，リアルタイムOS（RX850 Pro）に必要なテーブル情報などを作成するアプリケーションです。

cf850pro.exe（RX850 Pro**の場合）という実行形式で，**MS-DOS**アプリケーションです。**

コンフィギュレータは，DOS**プロンプト上からコマンド入力して起動できます。**

Ver.3.15**より，**NEC**エレクトロニクス製の**PM plus**から起動も可能になりました。**

つまり，コンフィギュレーション・ファイルが更新されると，自動的にcf850pro**が起動され，**SIT**情報，**SCT**情報が作成されるようになります。**

Q.15

コンフィギュレータ（cf850pro）起動時のオプションは？

A.15

コンフィギュレータ起動時のオプションは，以下の通りです。

cf850pro -i *sit_file -c sct_file -d h_file cf_file* [return]

sit_file **：システム情報テーブル（**.sit**）**

sct_file **：システムコールテーブル（**.sct**）**

h_file **：システム情報ヘッダファイル（**.h**）**

cf_file **：コンフィギュレーション（**CF**）定義ファイル（**.cf**）**

Q.16

各資源をコンフィギュレーション（CF**）定義ファイルにて生成するときに指定する「キー**ID**」とは何ですか？**

A.16

キーIDは，資源を生成するときに「ID番号を自動的に振り当てる」という設定にする場合に必要になります。

自分でIDをつけるときは，気にする必要はありません。

“ KeyIDを使用しない ” 設定となる “ 0 ” を指定してください。

自動的にID番号を割り当てる指定にした場合，コンフィギュレータで自動的にIDを付けてしまうため，ユーザは資源のID番号を知ることができません。

そのために必要となるのが「キーID」です。

このキーIDは，資源生成の際に，一意に決定されます。

ユーザが指定したキーID番号を元に，システム・コール（vget_*id**）を使用して，その資源のID番号を取得することが可能です。**

保守／廃止

Q.17

コンフィギュレーション（CF）定義ファイルで

```
mem       SPOL0    0xffffd200        0x0000fe0
```

と設定して実行すると，コンフィギュレータ（CF850 Pro）で，エラー・メッセージ

```
E2259: enough system memorypool "SPOLO" block size
```

が表示されます。
メモリの見積もりは，以下のようにしました。
メモリ・プールは使っていないのですが，タスク・スタック，割り込みスタック，管理オブジェクト以外に，何か領域を確保する必要はありますか？

タスクAスタック		0x1f4 bytes
タスクBスタック		0x3e8 bytes
タスクCスタック		0x3e8 bytes
計		0x9c4 bytes
割り込みスタック		0x1f4 bytes
管理オブジェクト		
システム管理テーブル	504＋4＋16＋32	= 556 bytes
タスク管理ブロック	3×56	= 168 bytes
セマフォ管理ブロック	1×20	= 20 bytes
割り込みハンドラ	6×16＋104	= 200 bytes
周期起動時間管理	3×40	= 120 bytes
計		1064 bytes（0x428）
総計	0x9c4＋0x1f4＋0x428	= 0xfe0

A.17

タスクのスタックサイズは，コンフィギュレーション（cf）定義ファイルで指定したサイズに，それぞれ100バイト（タスク・スタック管理テーブル28バイト＋コンテキスト領域72バイト）加算する必要があります。

詳細については，RX850 Pro　ユーザーズ・マニュアル　インストレーション編（U13774J）の5.3　タスク・スタックの容量を参照するか，NEC Electronics Microcomputerのホーム・ページ内のFAQにある「RX850 Proメモリ容量見積もりページ」を使用してください。

タスクAスタック	0x1f4＋100	= 600 bytes
タスクBスタック	0x3e8＋100	= 1100 bytes
タスクCスタック	0x3e8＋100	= 1100 bytes
計		2800 bytes

また，割り込みハンドラスタック領域に割り込みハンドラ・スタックフレーム・サイズ（アイドル・タスク80バイト）を加算する必要があります。

詳細については，RX850 Pro　**ユーザーズ・マニュアル　インストレーション編**（U13774J）の5.4　**割り込みハンドラ用スタックの容量**を参照してください。

割り込みスタック	0x1f4＋80	= 580 bytes
総計	2800＋580＋1064 = 4444 bytes （0x115c）	

この場合，コンフィギュレーション（CF）定義ファイルで，0x115c以上の値をSPOL0に指定する必要があります。

Q.18

タスクはどうやって生成するのですか？

A.18

コンフィギュレーション（CF）定義ファイルで静的に生成，またはアプリケーション中から“ cre_tskシステム・コール ”を発行して生成します。

生成時には，タスクの名前（ID），タスクの起動アドレス，タスク用のスタックのサイズ，タスクの初期優先度，タスクの初期状態，タスクの起動コード，起動したときの割り込みの状態，固有GP・TP，キーIDを設定します。

Q.19

タスクはいくつまで生成できますか？

A.19

タスク最大生成数（maxtsk）で指定した数まで生成可能です。

ただし，最大生成数として指定できる値は32767ですので，最大32767個まで生成可能です。

Q.20

初期化ハンドラとは何ですか？

A.20

RX850 Proの処理として，スタート・アップ・ルーチンから，RX850 Proの初期化処理へ移行した後，スケジューラが起動されますが“ スケジューラが起動される前に実行されるハンドラ ”のことを初期化ハンドラといいます。

初期化ハンドラのアドレスは，コンフィギュレーション（CF）定義ファイルで指定できます（“ ini ”で記述します）。

サンプルでは「varfunc」というアドレスになっています。

この初期化ハンドラの中で，タスクの起動，ハードウエアの初期化など，システム起動前にしておきたいことを書いておくと便利です。

なお，初期化ハンドラを使用したくなければ，コンフィギュレーション（CF）定義ファイルに指定しないでください。

この指定がなければ，RX850 Proは初期化ハンドラがないものとして動作するので，若干のコード縮小につながります。

Q.21

RX850 Proにアイドル・タスクは存在しますか？

A.21

ディフォールトで用意されてはいませんが，ユーザがシステムで一番優先度の低いタスクを生成し，それをアイドル・タスクとすることができます。

アイドル・タスクもなく，実行すべきタスクがなくなった場合，OSはHALT命令を発行してCPUを停止します。

また，V850シリーズの「省電力モード」である“ STOP ”，“ IDLE ”状態を使いたい場合は，アイドル・タスクを作る必要があります。

つまり，“ STOP ”，“ IDLE ”状態にするコードをアイドル・タスク内に記述することによって，実現可能です。

その際，アイドル・タスクは，システムで一番優先度の低いタスクとして生成する必要があります。

Q.22

実行すべきタスクがなくなったとき，どうなりますか？

A.22

CPUがHALT状態になります。

つまり，RX850 Pro内でHALTにする処理を実行します。

Q.23

RX850 Proのタスクの優先度範囲はいくつですか？

A.23

コンフィギュレーション（CF）定義ファイルで指定する“ タスクの優先度範囲（maxpri） ”で指定した分だけ指定可能です。

ここで指定できる範囲は1～252です。

Q.24

タスク優先度範囲の指定で，注意することはありますか？

A.24

タスク優先度範囲は，きちんと使っている分だけを指定してください。

これは，RX850 Proが，レディ・キューをサーチする時間に関係してくるからです。

コンフィギュレーション（CF）定義ファイルで“ タスク優先度数 ”を指定しますが，この数だけレディ・キュー（1優先度につき4バイト）が作られます。

つまり，使用するRAMが増えます。

また，RX850 Proのスケジューラが起動するタスクをサーチする際，すぐに見つかれば問題ありませんが，最悪の場合，最高優先度から最低優先度まですべてサーチすることになります。

つまり，処理時間がかかります。

これらを踏まえて，使用する優先度数を見極めたほうがよいでしょう。

Q.25

タスクのスタックはどこに確保されますか？

A.25

コンフィギュレーション（CF）定義ファイルのタスク情報で指定された“タスクのスタックサイズ”分だけ，そのスタック領域から確保されます。

スタック領域は，システム・メモリ「SPOL0」または「SPOL1」のどちらかを指定します。

これらのアドレスは，コンフィギュレーション（CF）定義ファイル中に指定する“システム・メモリ情報（mem）”で指定します。

なお，SPOL0だけにタスク・スタックを割り当てても問題なく，逆にSPOL1だけに割り当てても問題ありません。

Q.26

タスクのスタックはどれくらい取ればよいのでしょうか？

A.26

タスクのスタックは，アプリケーションに依存します。

スタックは，次のような用途に使用されます。

- タスクがプリエンプションするとき，タスク・コンテキストをセーブ
- 割り込みが入ったときのレジスタ情報退避
- ローカル変数のセーブ

タスク・コンテキストのサイズと割り込み時のスタック・フレームのサイズは，はっきりしています。

これらの詳細については，RX850 Pro **ユーザーズ・マニュアル　インストレーション編の第5章　メモリとその容量の見積もり**を参照してください。

ローカル変数に関しては，使用している数とサイズによります。

要素数の多い配列などを使った場合は，かなりのスタックを消費します。

開発中など，まだスタックの見積もりがはっきりしない場合は，なるべく多く取っておき，見積もりができる時期になったら，切り詰めていく方がよいでしょう。

また，見積もりとしては，RD850 Proを使用し，初期状態のタスクのスタック・ポインタ値（SP値）とアプリケーション動作中（他のタスクに切り替わっている最中）のスタック・ポインタ値を比較するのも1つの手段です。

保守／廃止

Q.27

複数のタスクを生成するとき，タスクによって使用するスタックを，内部RAMと外部RAMにそれぞれ個別に割り当てられますか？

A.27

割り当てられます。

タスクのスタック領域は，コンフィギュレーション（CF）定義ファイルで，タスクを定義するときに同時に指定します。

タスクのスタックは，システム・メモリ・プール，つまりSPOL0**と**SPOL1**のどちらかに割り当てられますが，そのSPOL0，SPOL1を定義するときに，内部RAMと外部RAMのアドレスを割り当てて生成すればよいことになります。**

コンフィギュレーション（CF）定義ファイルでタスクを静的に生成するときを例にあげます。

例えば，SPOL0**を外部RAMに，**SPOL1**を内部RAMに割り当てた場合，**SPOL0**および**SPOL1**は次のように定義されます。**

```
mem SPOL0 0x200000 0x1000
mem SPOL1 0xffffe000 0x1000
```

そして，TASK01 **のスタック**0x200**バイトを外部**RAM，TASK02**のスタック**0x100**バイトを内部**RAM**に割り当てる場合，次のようにタスクを定義します。**

```
tsk TASK01 TTS_RDY 0x0 0x0 TA_ASM _task01 0x8 TA_ENAINT ¥
0x200:SPOL1 no_use no_use 0x0

tsk TASK02 TTS_RDY 0x0 0x0 TA_ASM _task02 0x6 TA_ENAINT ¥
0x100:SPOL0 no_use no_use 0x0
```

また，cre_tsk**システム・コールで動的にタスクを生成する場合は，**cre_tsk**の引数で指定します。**

Q.28

rel_waiシステム・コール（強制起床待ち解除）発行で他タスクの**wait**状態を強制的に解除した場合，対象となるタスクの起床要求数はクリアされますか？**

A.28

rel_waiシステム・コールでは，タスクの起床要求数のクリアは行いません。**

つまり，それまでに複数の起床要求があった場合，その要求カウントはそのままになります。

起床要求数を0**クリアするのは，**can_wup**システム・コールです。**

保守／廃止

Q.29

NECエレクトロニクス製コンパイラ（CA850）にある#pragma rtos_taskは使えますか？

A.29

使用できます。

この命令を使用することにより，指定された関数形式がリアルタイムOSのタスク用としてコンパイラが認識し，コードを出力します。

通常の関数では，関数のプロローグ処理，エピローグ処理が必要で，これらのコードをコンパイラは出力します。

しかし，このコードはリアルタイムOSには不要になります。

そこで，#pragma rtos_task命令で関数形式をタスクとして指定すると，関数のプロローグ処理，エピローグ処理が出力されず，コード縮小につながります。

Q.30

メールボックスはどうやって生成するのですか？

A.30

コンフィギュレーション（CF）定義ファイルにて静的に生成，またはアプリケーション中で“ cre_mbxシステム・コール ”を発行して生成します。

Q.31

メールボックスはいくつまで生成できますか？

A.31

メールボックス最大生成数で指定した数まで生成可能です。

ただし，最大生成数として指定できる値は32767ですので，最大32767個まで生成可能です。

Q.32

メールボックスを使用する際のメッセージのサイズに制限はありますか？

A.32

特に制限はありません。

メモリ容量の許す限り使用できます。

Q.33

メールボックスでは，メッセージそのものが通信されるのですか？

A.33

メッセージそのものが通信されるわけではなく，メッセージの先頭アドレス（ポインタ）だけが通信されます。

保守／廃止

Q.34

メッセージを書き込む領域は，どこを使えばいいですか？

A.34

メモリ領域の管理のしやすさという観点から，RX850 Pro**で管理されているメモリ・ブロック（**Q.54**参照）を使用することを推奨しています。**

具体的な順序は，メッセージで使うメモリ領域を獲得（get_blk**）し，その領域にメッセージを書き込み，そしてメールボックスに送信（**snd_msg**）します。**

受け取る側は，メッセージを受信（rcv_msg**）し，その内容を読み取り，そしてメッセージとして使用していたメモリ領域を解放（**rel_blk**）します。**

ただし，メッセージがキューイングしているメールボックスをdel_mbx**で動的に削除する際，メッセージ領域がメモリ・ブロックであれば，その領域は**RX850 Pro**が解放できますが，それが**RX850 Pro**の管理外の領域を使っていた場合，動作保証外になってしまいます。**

Q.35

メッセージに優先度をつけることはできますか？

A.35

できます。

メッセージ優先度順（TA_MPRI**属性）のメールボックスに，優先度付きメッセージを送信した場合，その優先度順にキューイングされます。**

優先度数は，メッセージ本体に格納します。

格納領域は，メッセージ先頭から5**バイト目の**2**バイト長の領域です。**

優先度付きメッセージのやり取りを行う場合は，この領域を書きつぶさないようにする必要があります。

なお，優先度付きメッセージ領域は，アラインメントを考え，先頭から8**バイト先に書くのが一般的です。**

Q.36

メールボックスでのメッセージのキューイングのされ方はどのようになっていますか？

A.36

メッセージのキューイングのされ方には2**種類あります。**

1つは「メッセージ到着順（FIFO**順）」，もう**1**つが「メッセージ優先度順（**PRI**順）」です。**

これらは，コンフィギュレーション（CF**）定義ファイルにて，メールボックスの属性を指定することで区別します。**

つまり，メールボックスごとにキューイング方式を決めることになります。

前者であればTA_MFIFO**属性，後者であれば**TA_MPRI**属性になります。**

保守／廃止

Q.37

特定のタスクに対して，メッセージを送ることは可能ですか？

A.37

タスクを指定してメッセージを送るといった機能は備えていません。

この機能は，受け取るタスクが使用するメールボックスを専用に1つ作成する（Q.30参照）ことにより，実現が可能です。

Q.38

複数のタスクがメッセージを待っている場合，どのような順番でメッセージが渡されますか？

A.38

メッセージ待ちになった順番，もしくはタスクの優先度順に渡されます。

これはメールボックスの属性によって決まります。

メールボックスを生成する際，TA_TFIFO属性であればメッセージ待ちになった順番，TA_TPRI属性であればタスクの優先度順にキューイングされます。

参考例）

--メールボックス情報

```
mbx    0x1    0x0    TA_TFIFO TA_MFIFO              0x1
mbx    0x2    0x0    TA_TPRI  TA_MFIFO(or TA_MPRI)  0x2
```

Q.39

ディバッガでprcv_msgシステム・コールの戻り値を参照すると，0xffffffabになります。

16進で参照すると“ −55 ”だと思われるのですが，ユーザーズ・マニュアルの戻り値に“ −55 ”はありません。

この戻り値は，どのように参照すればよいのでしょうか？

A.39

ユーザーズ・マニュアルに記載されているシステム・コールの戻り値は，10進数で記載されています。

0xffffffabを符号付き10進数として参照すると，“ −85 ”になります。

prcv_msgの戻り値“ −85 ”はE_TMOUTで，対象メールボックスにメッセージがなかった場合に返却されます。

Q.40

イベント・フラグは，どうやって生成するのですか？

A.40

コンフィギュレーション（CF）定義ファイルにて生成，もしくは，プログラム中でcre_flgシステム・コールを使用して生成します。

保守／廃止

Q.41

イベント・フラグは，いくつまで生成できますか？

A.41

イベント・フラグ最大生成数で指定した数まで生成可能です。

ただし，最大生成数として指定できる値は32767**ですので，最大**32767**個まで生成可能です。**

Q.42

イベント・フラグの大きさは，何ビットですか？

A.42

32**ビットです。**

Q.43

RX850 Pro**で，**1**つのイベント・フラグに対して，複数のタスクがイベントを待つことはできますか？**

A.43

はい，待てます。

イベント・フラグ生成の際に，「1**タスクだけ待ちを可能にする（**TA_WSGL**属性）」か「複数タスク待ちを可能にする（**TA_WMUL**属性）」のどちらかの属性を設定できます。**

Q.44

RX850 Ver.3.13**では**1**ビット・イベント・フラグがありましたが，**RX850 Pro**にはないのでしょうか？**

A.44

ありません。

RX850 Ver.3.13**にて**1**ビット・イベント・フラグを使用していて，**RX850 Pro**にプログラムを移植する場合は，すべてイベント・フラグにする必要があります。**

1**ビット・イベント・フラグ関連のシステム・コールも，イベント・フラグのものに変更してください。**

具体的な変更の仕方は，RX850（Ver.3.1x）**からの移行資料（**SUD-T-4816-2**）を参照してください。資料の入手は特約店または**NEC**エレクトロニクス営業拠点にご連絡ください。**

保守／廃止

Q.45

wai_flg, twai_flg, pol_flgの引数に“イベント・フラグのクリア指定”（TWF_CLR）がありますが，このクリア処理はどのタイミングで行われるのでしょうか？

A.45

イベント・フラグのクリアは，set_flgの後，wai_flg/twai_flg/pol_flg内のルーチンで処理されるのではなく，別ルーチンで行っています。

そこでは，イベント・フラグ待ちのタスクがあるかどうかをチェックして，あれば待ち条件のチェック，待ち解除，フラグのクリアを行います。

この別ルーチン内はdi命令で割り込み禁止状態で動作するので，割り込み内のset_flgによってフラグが変化することはありません。

また，タスク遷移に関しては，待ち解除などの処理が終わった後にスケジューラの起動要求を出すので，他タスクによってフラグを変化させられることもありません。

Q.46

1つのイベント・フラグの1個のビットに対して，複数のタスクがwai_flgでイベント発生を待っていた場合，イベント成立後のタスクの起動はどうなりますか？

A.46

複数タスクが1つのイベント・フラグに対して待っていて，その待ち解除の条件が同一だった場合，待ちが解除されると，待っていたタスクがすべてready状態になります。

その後，スケジューラが起動され，一番優先度の高いタスクがrun状態になります。

つまり，set_flgをした時点では，一番優先度の高いタスクが起床されるわけではなく，待っていたタスクすべてが起床されることになります。

すべてのタスク起床後，その時点で起床しているタスクの中で，一番優先度の高いタスクがrun状態へ移行していきます。

イベント・フラグ待ちであったタスクのうち，一番優先度の高いタスクが必ずrun状態になるとは限りません。

Q.47

set_flgシステム・コールを発行した時，そのイベント・フラグを待っているタスクのうち1つでもwai_flgシステム・コールでビット・パターンのクリア（TWF_CLR）を指定していた場合，このフラグはクリアされますか？

A.47

イベント・フラグの待ち解除は，イベント・フラグ待ちキュー（FIFO）の先頭からチェックを行い，待ち条件が成立したタスクを起床させています。

この時，タスクがクリア指定していた場合には，直ちにビット・パターンをクリアします。

したがって，待ち条件が成立したタスクがクリア指定していた場合，それ以降に待ちキューにキューイングしていたタスクは，待ち条件が成立していても起床しません。

ひとつのタスクが待ちになる例は，4. 4. 5　**メッセージ**を参照してください。

保守／廃止

Q.48

セマフォはどうやって生成するのですか？

A.48

コンフィギュレーション（CF）定義ファイルにて生成，もしくは，プログラム中でcre_semシステム・コールを使用して生成します。

Q.49

セマフォはいくつまで生成できますか？

A.49

セマフォ最大生成数で指定した数まで生成可能です。

ただし，最大生成数として指定できる値は32767ですので，最大32767個まで生成可能です。

Q.50

セマフォの初期資源数はどこで設定しますか？

A.50

コンフィギュレーション（CF）定義ファイルで，セマフォを生成するとき，またはcre_semでセマフォを生成するときに設定します。

Q.51

セマフォの資源数に上限はありますか？

A.51

上限はあります。

上限は，0x7fffffff（2147483647）です。

Q.52

セマフォに資源があるときに資源を返却する操作を行った場合，どうなりますか？

A.52

セマフォは資源のカウンタを持っており，そのカウンタがインクリメントされます。

保守／廃止

Q.53

セマフォなどの資源に関する管理オブジェクトは，SPOL0/SPOL1で資源ごとに別々に割り当てることはできますか？

また，SPOL0/SPOL1が，コンフィギュレーション（CF）定義ファイルで，両方とも定義されている場合，起動時にはどちらが先に使用されるのでしょうか？

A.53

セマフォなどの資源は，すべてシステムで管理されています。

つまり，すべてSPOL0に管理テーブルとして持っていますので，これはSPOL1に割り当てることはできません。

RX850 Pro起動前は，SPOL0やSPOL1の概念がありません。

つまり，スタックなどで使用したい場合は，ブート処理時にスタック領域を確保する必要があります。

このことに関しては，製品に添付されているサンプル・プログラムを参考にしてください。

RX850 Pro起動後は，SPOL0にシステム領域として，管理テーブルを作成していきます。

なお，SPOL0は必ず定義する必要があります。

SPOL1のみの定義はできないことに注意が必要です。

Q.54

メモリ管理はどのようになっていますか？

A.54

RX850 Proでは，メモリを「メモリ・プール」と「メモリ・ブロック」に分けて管理しています。

メモリ・プール内に複数のメモリ・ブロックがあるというイメージです。

タスクや割り込みハンドラからは，このメモリ・プールに対してメモリ領域の獲得・返却要求を行います。

そして，その獲得・返却されるメモリ領域を「メモリ・ブロック」と呼びます。

Q.55

メモリ・プールはどうやって生成するのですか？

A.55

コンフィギュレーション（CF）定義ファイル，またはcre_mplシステム・コールにて生成します。

コンフィギュレーション（CF）定義ファイルでは，メモリ・プールの名前（ID），タスクのキューイング方式，メモリ・プールのサイズ，キーIDを設定します。

Q.56

メモリ・プールはいくつまで生成できますか？

A.56

メモリ・プール最大生成数で指定した数まで生成可能です。

ただし，最大生成数として指定できる値は32767ですので，最大32767個まで生成可能です。

保守／廃止

Q.57

固定長メモリ・プールはありますか？

A.57

ありません。

すべて可変長メモリ・プールとなります。RX850**からアプリケーションを移行するとき，固定長メモリ・プールを使用していた場合は，すべて可変長メモリ・プールで代用する必要があります。**

Q.58

システム・メモリ・プール（SPOL0/1**）やユーザ・メモリ・プール（**UPOL0/1**）をコンフィギュレーション（**CF**）定義ファイルで指定する際，**.bss**や**.data**セクションなどの領域と同一配置を避ける必要はありますか？**
また，内部RAM**に必ず配置しなければならないメモリ・プールはありますか？**

A.58

SPOL0/1**およびUPOL0/1は，直接アドレス指定するもので，そこで指定されたアドレスから指定したサイズ分取られます。**

.bss**や**.data**セクションへの変数配置は，コンパイラ，リンカが行いますが，**RX850 Pro**が使用する上記のメモリ領域に関しては，コンパイラ，リンカは認識できません。**

よって，.bss**や**.data**セクションを避けて配置してください。**

リンカにて，RX850 Pro**が使用するセグメントを，リンク・ディレクティブであらかじめ確保しておくと，他の領域との重複による間違いは少なくなると思います（この場合のサンプルは**Q.63**を参照してください）。**

ただし，オーバフローなどの検出はできません。

また，RX850 Pro**に関しては，上記メモリを内部，または外部に配置しなければならないという制限はありません。**

Q.59

可変長メモリ・プールで，メモリ・ブロックの取得・解放を繰り返したときにできる，未使用領域の整理（ガベージ・コレクション）は行っていますか？

A.59

行っていません。

アプリケーションの記述言語がC言語で，ポインタなどにより，アプリケーションが直接アドレスを扱うことができるためです。

RX850 Pro**がガベージ・コレクションをして，実体（メモリ・ブロック）を移動してしまうと，アプリケーションがポインタに保持しているアドレスとその実体のアドレスとの間で矛盾が生じ，アプリケーションが正しく動作しなくなってしまいます。**

RX850 Pro**は，**rel_blk**でメモリ・ブロックが返却された時，返却されたメモリ・ブロックの前後を調べ，いずれか一方，あるいは，両方が空き（未使用の）領域である場合には，マージして大きな空き領域とし，メモリ・プールのフラグメンテーション（細分化）を少なくしています。**

しかし，空き領域が不連続な場合には，空き領域の中で最大のもの以上のメモリ・ブロックは，取得できません。

保守／廃止

Q.60

SPOL0, SPOL1, UPOL0, UPOL1は何を意味していますか？

A.60

それぞれの意味は，以下の通りです。

- SPOL0 … System Memory Pool 0
- SPOL1 … System Memory Pool 1
- UPOL0 … User Memory Pool 0
- UPOL1 … User Memory Pool 1

RX850 Proの資源として使用するメモリ領域は，すべて上記のどれかが使用されます。
具体的には，

- **割り込みスタック領域** … SPOL0 **または** SPOL1
- **タスク・スタック領域** … SPOL0 **または** SPOL1
- **メモリ・プール領域** … UPOL0 **または** UPOL1

というようになります。
このSPOL0, SPOL1, UPOL0, UPOL1の先頭アドレス，サイズは，コンフィギュレーション（CF）定義ファイルにおいて設定します。

設定例）

--メモリ情報

```
mem    SPOL0    0x1000    0x0000
mem    SPOL1    0x2000    0x1000
mem    UPOL0    0x3000    0x7000
mem    UPOL0    0x20000   0x2500
mem    UPOL1    0x30000   0x1500
```

Q.61

スタックは，SPOL0（System Pool 0）だけに割り当てても問題ありませんか？

A.61

問題ありません。

保守／廃止

Q.62

スタート・アップ・ルーチン（boot.s/boot.850）でスタック・ポインタ（sp）の設定を行っていますが，サイズはどのように決めればよいでしょうか？

A.62

RX850 Pro起動前に，スタックを使用することがあれば，そのスタック・ポインタを使うことになります。

たとえば，関数コールを行った場合，その時のlpの値を保存するときは，spの指すスタックに積まれます。

また，関数内でさらに関数コールする場合にも使用されます。

サンプルとして製品に添付しているスタート・アップ・ルーチンでは，RX850 Pro起動前，つまりjmp［lp］でカーネル初期化に移るまでの間に，NECエレクトロニクス版の場合はスタックを使用していません。

しかし，GHS版の場合はメモリ初期化処理を関数コールしているため，その部分のlp退避にスタックを使っています（Q.66参照）。

ただし，サンプルのように，0x28000バイトものサイズは使っていません。

サンプルで0x28000と指定している理由は，この領域は，リンク・ディレクティブ・ファイルでRX850 Proが使用するメモリ情報と重複させてあり，RX850 Pro起動後は，この部分をメモリ（memで指定）として使用させているからです。

サンプルのコンフィギュレーション（CF）定義ファイルのmemで指定しているサイズの合計が，0x28000になっています。

サンプルでスタックを規定しておかないと，サンプルを改造して使っているユーザが，スタックを設定せずに関数コールなどを行った場合に誤動作を起こすことが考えられるため，十分なサイズ分を取っています。

スタック・サイズの決定方法としては，スタート・アップ・ルーチンにもう手を加えないとなったときに，実際にディバッガでスタックの値がどの程度使われるかを調査し，その値をスタック・サイズとすればよいと思います。

または，memで指定しているサイズ分だけ取ってしまっても，無駄になることはないので，その値を使うという方法でもよいと思います。

Q.63

RX850 Proで，コンフィギュレーション（CF）定義ファイルのmem指定で作成するメモリ領域はリンカでは検知できないため，ユーザ・プログラムで使用するメモリ領域を重複してしまってもわかりません。
リンク時に「重複している」というエラーを検出するにはどうすればよいですか？

A.63

コンフィギュレーション（CF）定義ファイルとリンク・ディレクティブ・ファイルとの結合が取れていないため，リンク・ディレクティブ・ファイルに変更を加える必要があります。

RX850 Proのサンプル・プログラムを例として，そのやり方を説明します。

サンプルのコンフィギュレーション（CF）定義ファイル（sys.cf）で定義されているシステム・メモリ・プール，ユーザ・メモリ・プールは，次のようになっています。

```
-- memory information
mem SPOL0 0x00110000 0x00010000
mem SPOL1 0x00120000 0x00010000
mem UPOL0 0x00130000 0x00008000
```

この領域を，ダミーのセクションとして確保すればよいことになります。

【NECエレクトロニクス版の場合】

スタート・アップ・ルーチン（boot.s）に次の記述を追加します。

```
----------------------------------------
-- /*** SPOL0 area specifying ***/
.section ".spol0",  bss
.lcomm  __spol0_head,   0x10000,   4
.lcomm  __spol0_end,    0,   4

-- /*** SPOL1 area specifying ***/
.section ".spol1",  bss
.lcomm  __spol1_head,   0x10000,   4
.lcomm  __spol1_end,   0,   4

-- /*** UPOL0 area specifying ***/
.section ".upol0",  bss
.lcomm  __upol0_head,   0x8000,   4
.lcomm  __upol0_end,   0,   4
----------------------------------------
```

そして，リンク・ディレクティブに次の記述を追加します。

```
------------------------------------------------------------
SYSPOL0  : !LOAD ?RW            V0x00110000 {
        .spol0          = $NOBITS          ?AW       .spol0;
};

SYSPOL1  : !LOAD ?RW            V0x00120000 {
        .spol1          = $NOBITS          ?AW       .spol1;
};

USERPOL0 : !LOAD ?RW            V0x00130000 {
        .upol0                  = $NOBITS  ?AW       .upol0;
};
------------------------------------------------------------
```

【NECエレクトロニクス版の場合】

スタート・アップ・ルーチン（boot.850）に次の記述を追加します。

```
----------------------------------------
-- /*** SPOL0 area specifying ***/
.section ".spol0",  bss

-- /*** SPOL1 area specifying ***/
.section ".spol1",  bss

-- /*** UPOL0 area specifying ***/
.section ".upol0",  bss
----------------------------------------
```

そして，リンク・ディレクティブに次の記述を追加します。

```
----------------------------------------
        :
        :
.spol0  0x00110000    :
.spol1  0x00120000    :
.upol0  0x00130000    :
        :
        :
----------------------------------------
```

これで，sys.cfで確保したい領域がリンカにもわかります。

そのため，アプリケーションのリンク時に，他の領域がこの領域と重複すると，エラー検出することができます。

ただし，オーバフローなどの検出はできません。

Q.64

rel_blkの戻り値がマイナスになっています。

マイナスはあり得ないと思うのですが，なぜなのでしょうか？

A.64

RX850ProはμITRON3.0に準拠しており，μITRON3.0よりシステム・コール処理で発生したエラーは，マイナスの値として戻るように規定されています。

Q.65

rel_blkシステム・コールを実行すると，毎回返却値としてE_OBJ（－63）が返り，メモリ・ブロックを返却できません。

A.65

これは RX850 Proの特有の現象です。

rel_blkシステム・コールは，メモリ・ブロックを返却する際，メモリ・ブロックの先頭4バイトが0でなかった場合は，メモリ・ブロックを返却せずに戻り値E_OBJで終了しています。

これは，メモリ・ブロックがメールボックスのメッセージ領域として使用されていた場合を考慮した仕様です。

メモリ・ブロックがメールボックスのメッセージ領域として使用されていた場合，先頭4バイトがメッセージの待ちキューのリンク領域になります。

つまり，メッセージがメールボックスにキューイングされているときにrel_blkシステム・コールを発行し，必ずメモリ・ブロックが返却する仕様にすると，キューにつながれていたメッセージ領域が返却されてしまうことになります。

これを防ぐため，リンク領域である先頭4バイトが0でなければ，メッセージ領域として使用されているメモリ・ブロックと判別し，メモリ・ブロックを返却せずに戻り値E_OBJで終了しています。

そのため，メモリ・ブロックを返却するときは，必ず先頭4バイトを0クリアする必要があります。

しかし，Ver.3.15から，メモリ・ブロックがメッセージ領域として使用されていない場合を考慮し，先頭4バイトが0でない場合でも，メモリ・ブロックを返却できるrel_blkを別ライブラリとして加えました。

機　　能	ライブラリ名
メモリ・ブロックの先頭4バイトを0でクリアする必要のあるrel_blk（これまでと同じ仕様のrel_blk）の入ったライブラリ	librxp.a
メモリ・ブロックの先頭4バイトを0でクリアしなくてもよいrel_blk（新たな仕様のrel_blk）の入ったライブラリ	librxpm.a

それぞれ別々のライブラリに入っているため，どちらか一方のrel_blk を使用することになります。

使用する方のライブラリをリンクしてください。

保守／廃止

Q.66

GHS版で，サンプルにあるメモリ初期化処理を行うと無限ループに入ってしまいます。

A.66

GHS版のサンプルで，boot.850からmeminit.cを呼んでいる箇所があります。

meminit.cではRAMの初期化処理を行っていますが，ここから抜け出せない，または初期化処理から戻ろうとすると，boot.850の先頭から再び動作することがあります。

これは，スタック破壊によるものです。

スタート・アップ・ルーチン内で使用しているスタック領域もmeminit.c内で0クリアしてしまうため，このような現象が発生します。

対処としては，

・スタート・アップ・ルーチン内で使用しているスタック領域を初期化しない
・スタート・アップ・ルーチン内で使用するスタック領域を別の場所に一時的に確保する

などがあります。

スタート・アップ・ルーチン内のスタックに関しては，Q.62も参照してください。

Q.67

割り込みが入った時の処理は，どこに記述すればよいですか？

A.67

割り込みハンドラに記述します。

割り込みハンドラとは，割り込みが発生した際にただちに起動される割り込み処理専用ルーチンで，タスクとは独立したものとして扱われます。

RX850 Proでは，「直接起動割り込みハンドラ」と「間接起動割り込みハンドラ」の2種類が用意されています。

Q.68

直接起動割り込みハンドラと間接起動割り込みハンドラの違いは何ですか？

A.68

直接起動割り込みハンドラ

割り込みが発生した際，RX850 Proを介在させることなく起動される割り込みハンドラです。

ただし，レジスタの退避，スタックの切り替え処理は，ユーザが記述しなければなりません。

（なお，レジスタの退避・復帰に関しては，マクロが用意されています。）

間接起動割り込みハンドラ

割り込みが発生した際，RX850 Proによる割り込み前処理，すなわちレジスタの退避処理，スタックの切り替え処理を行わせた後，起動される割り込みハンドラです。

保守／廃止

Q.69

直接起動割り込みハンドラと間接起動割り込みハンドラのどちらを使えばよいですか？

A.69

直接起動割り込みハンドラは，割り込みが発生するとすぐにハンドラ処理に移るので，高速な処理が期待されます。

しかし，ユーザ自身でレジスタの退避処理やスタックの退避処理を行わなければならないため，ハンドラ自体の記述が多少複雑化します。

一方，間接起動割り込みハンドラは，割り込みが発生すると，RX850 Pro**に一度処理が移り，レジスタの退避やスタックの退避が行われた後に，ハンドラの処理が行われます。**

直接起動割り込みハンドラに比べ，多少応答性が落ちる可能性がありますが，ユーザが記述する部分はハンドラ処理のみなので簡潔になります。

このことをふまえて，ユーザ自身が選択することになります。

Q.70

直接起動割り込みハンドラは，どのようにして登録すればよいですか？

A.70

割り込みが発生した際に，プロセッサが制御を移すハンドラ・アドレスに直接起動割り込みハンドラを割り付ける，または，直接起動割り込みハンドラへの分岐命令を設定することにより登録されます。

詳しいプログラミングの仕方については，A.5　**直接起動割り込みハンドラ**を参照してください。

Q.71

間接起動割り込みハンドラは，どのようにして登録すればよいですか？

A.71

コンフィギュレーション（CF**）定義ファイルに記述することによって登録されます。**

また，“ def_int**システム・コール”で動的に登録することもできます。**

保守／廃止

Q.72

タイマ割り込みの登録方法を教えてください。

A.72

タイマは，間接起動割り込みハンドラの1**つとして起動されます。**

例えば，ソースの記述は次のようになります。

（CA850**の場合）**

```
-------------------------
.section  "INTCMD"
RTOS_IntEntry_Indirect
-------------------------
```

（CCV850**の場合）**

```
-------------------------
.org  0x00000240
RTOS_IntEntry_Indirect
-------------------------
```

このように，RTOS_IntEntry_Indirect**を使ってください。**

これにより，タイマ割り込みを入れることができます。

Q.73

多重割り込みには対応していますか？

A.73

対応しています。

割り込みハンドラ処理中に割り込みが入ることを多重割り込みといいます。

ただし，RX850 Pro**は，割り込みハンドラを“割り込みは禁止状態”で起動しますので，多重割り込みを受け付けたいときは，割り込みハンドラ内で割り込みを許可する必要があります。**

つまり，割り込みの許可（EI**または**ena_int**システムコール），禁止（**DI**または**dis_int**システム・コール）を発行し，ユーザ自身で管理する必要があります。**

Q.74

NEC**エレクトロニクス製コンパイラ（**CA850**）にある**#pragma rtos_interrupt**は使えますか？**

A.74

これは，リアルタイムOS**の直接起動割り込みハンドラの記述を簡潔にするために用意された**pragma**指令ですが，現在は，この指令を使用して直接起動割り込みハンドラを記述すると，正常に動作しません。**

そのため，制限事項になっていますので，使用しないでください。

なお，直接起動割り込みハンドラの記述方法は，A.5　**直接起動割り込みハンドラ**を参照してください。

保守／廃止

Q.75

割り込みハンドラから復帰する方法は？

A.75

直接起動割り込みハンドラと間接起動割り込みハンドラでは，復帰の方法が異なります。

直接起動割り込みハンドラからの復帰は，ハンドラの終了部分で用意されたマクロを使用して復帰します。

（マクロ内に割り込み復帰処理が書かれています。）

マクロの名前，および使用方法は，A.5　**直接起動割り込みハンドラ**を参照してください。

間接起動割り込みハンドラからの復帰は，ハンドラの終了部分で，間接起動割り込みハンドラから普通に復帰する場合は“return（TSK_NULL）**”，間接起動割り込みハンドラからの復帰，およびパラメータで指定されたタスクに対する起床要求を発行する場合は“**return（タスクのID番号）**”を記述することで行います。**

Q.76

間接割り込みハンドラは，以下のように戻り値を必要としますが，“return（TSK_NULL）**”の記述を削除し，戻り値がない関数をハンドラとして登録した場合，どうなるのでしょうか？**

```
ID
func_inthdr()
{
  /* 間接起動割り込みハンドラの本体処理 */
  ......................................
  ......................................
  ......................................

  /* 間接起動割り込みハンドラからの復帰 */
  return(TSK_NULL);
}
```

A.76

戻り値がない関数をハンドラとして登録すると，暴走する可能性があります。

戻り値はr10**に格納されて**RX850 Pro**の中に戻りますが，**r10**の値が**TSK_NULL（ = 0）**かどうかを判定し，もし**0**でなければそのID番号を持つタスクを起床させようとします。**

つまり，運良くr10**に**0**が格納されていれば正しい動作をしますが，そうでない場合，予期しないタスクが起床してレディ・キューにつながれたり，たまたま優先度の高いタスクが起床されると，そのタスクが起動されてしまうなどの動作をしてしまいます。**

このように，間接起動割り込みハンドラからの復帰を記述していないと，期待した動作をしないことがあります。

割り込みハンドラの最後では，必ず“return（TSK_NULL）**”，または“**return（ID tskid）**”を記述するようにしてください。**

保守／廃止

Q.77

割り込みハンドラのスタック領域はどこに確保されますか？

A.77

割り込みハンドラのスタック領域は，コンフィギュレーション（CF）定義ファイルにて指定した場所に確保されます。

コンフィギュレーション（CF）定義ファイルでは，スタック・サイズ，確保する場所（SPOL0 or SPOL1）を指定します。

記述例）

--システム情報

```
clktim  0x1
clkhdr  0x7
defstk  0x100
intstk  0x100:SPOL0 ← 割り込みハンドラ用スタック情報
prttsk  0x1
prtsem  0x1
prtflg  0x1
prtmbx  0x1
prtmpl  0x1
```

ただし，割り込みハンドラのスタックは，割り込みが入った直後から使われるわけではありません。

あるタスクの動作中に割り込みが入った場合，退避するレジスタ情報は割り込まれたタスクのスタックに積まれます。

その後，スタック・ポインタ（SP：r3）を割り込みハンドラのスタックに切り替えられます。

（正確には，タスクのlp, spの値は割り込みスタックに積まれます。）

多重割り込みの場合は，割り込み処理中に入るため，そのまま割り込みハンドラのスタックを使います。

つまり，多重割り込みが入れば入るほど，割り込みハンドラのスタックの消費量は多くなります。

Q.78

直接起動割り込みハンドラから戻ってくると，グローバル・ポインタ（GP）の値がおかしくなって暴走します。

A.78

直接起動割り込みハンドラから復帰するとGPの値が壊されてしまい，その後うまく動作しないことがあります。

これは，直接起動割り込みハンドラの場合，TP, GPの値は保証せず，ハンドラ内で使用する必要があれば，ハンドラの最初でセットする必要があるためです。

CA850対応版の場合の記述例）

```
------------------------------------------------------------
  #include

  ID
  inthdr_body( )
  {
      __asm("mov #__tp_TEXT, tp");
      __asm("mov #__gp_DATA, gp");

      /*直接起動割り込みハンドラ func_inthdr の本体処理 */
      ....................................
      ....................................
      ....................................

      /*直接起動割り込みハンドラ func_inthdr 本体からの復帰 */
      return tskid
  }
------------------------------------------------------------
```

詳細については，A.5　**直接起動割り込みハンドラ**を参照してください。

直接割り込みハンドラの最初と最後にマクロ（RTOS_IntEntry/RTOS_IntReturn）を使うことにより，GPの退避，復帰は行いますが，セットされることはありません。

それは，直接起動割り込みハンドラの性格上，RX850 Proの方で確実に管理できる割り込みハンドラではないためです。

タイマ・ハンドラ起動中は，GP（r4）の値をテンポラリ・レジスタとして使っています。

そのときに直接割り込みが入った場合，正しいGPの値がセットされておらず，正常に動作しません。

なお，間接起動割り込みハンドラの場合は，固有GP，固有TPのセットは，すべてRX850 Proによってセットされます。

具体的な記述方法については，A.6　**間接起動割り込みハンドラ**を参照してください。

Q.79

ノンマスカブル割り込み処理中にシステム・コールを発行した場合，動作はどうなりますか？

A.79

ノンマスカブル割り込みは，割り込み優先順位の対象外なので，すべての割り込みに対して優先して受け付けられます。

そのため，割り込み禁止状態で動作しなくてはならないところでも，NMIが受け付けられてしまいます。

RX850 Proでは，システム・コール発行によって管理情報やキューの書き換えなどを行います。

もし，RX850 Proが管理情報を触っているときにNMIが発生し，さらにその中でシステム・コールを発行すると，管理情報が正しく更新されずに動作を続けることになり，暴走につながります。

よって，RX850 Proでは，ノンマスカブル割り込みハンドラ内でシステム・コールを発行した場合，その動作の保証をしていません。

保守／廃止

Q.80

割り込みハンドラが起動しません。

A.80

次のような原因が考えられます。

・実際に割り込みが入っていますか？

ディバッガで“割り込みハンドラ・アドレス（ベクタ）”にブレーク・ポイントを設定し，目的の割り込みが入ってくるか確認します。

もし，入っていなければ，物理的に割り込みが入っていないことになります。

この場合は，ハードウエアの初期化処理を見直す必要があります。

割り込み制御レジスタの設定（割り込みマスクのオープンなど），割り込みトリガの設定を確認してください。

これらを設定しても動かない場合は，ターゲットに異常がある可能性があります。

・コンフィギュレーション（CF）定義ファイルの設定で，割り込み要因名（割り込み要因番号）は合っていますか？

間接起動割り込みハンドラを使う場合，コンフィギュレーション（CF）定義ファイルで割り込み要因名を指定します。

RX850**とは違い，割り込み要因名ではなく，「目的の割り込み例外コード－**0x80/0x10**」という式で算出した値を指定します。**

この値が間違っていないかどうかを確認してください。

また，その割り込みが入ったときに起動される割り込みハンドラの先頭アドレス（関数の先頭）も，確認してください。

直接起動割り込みハンドラを使う場合は，この限りではありません。

直接起動割り込みハンドラを使う場合，割り込みハンドラ・アドレスにjr**命令を直接記述し，その後にハンドラ本体へ飛びます。**

これがうまくいっていなければ，アプリケーションのダウンロードがうまくいっていないか，リンクがうまくいかなかったことが原因かもしれません。

Q.81

タイマ割り込み発生時に，直接起動割り込みハンドラのUART**の送信/受信割り込みが発生すると，プログラムが暴走してしまいます。**

UART**の送信/受信割り込みが発生した時点で，**TP = 0x00000400**に設定しているはずなのですが，調べたところ，**TP = 0xFFFFFFFF**となってしまいます。**

このため，割り込みベクタ領域に不正にジャンプしてしまい，暴走するようです。

A.81

次のような原因が考えられます。

直接起動割り込みハンドラをご使用の場合，RX850 Pro**では，**TP, GP**の値はハンドラの最初で設定する必要があります。**

保守／廃止

つまり，ハンドラで使用するTP, GPの値は，ハンドラ起動時に決める必要があるということになります。

［参考］Q.78

Q.82

直接起動割り込みハンドラで，RTOS_IntEntryマクロを使うと，リンク時に，メッセージ

Warning : register r1 used as source register

が表示されます。

A.82

RX850 Proに添付されているRTOS_IntEntryIndirectマクロで，予約レジスタのr1を使用しているために，このワーニングが表示されてしまいます。

しかし，誤った使い方をしているわけではないので，このワーニングが表示されても問題はありません。

どうしてもワーニングを表示したくない場合は，as850に-wオプションを指定することで，ワーニングの表示を抑制することはできます。

ただし，他のワーニングも抑制されてしまうので，macro.hを書き換える方がよいです。

macro.h内で，28行目の部分

```
----------------------------------------
/* go indirect interrupt handler */
ld.w   sbt_intent[r2], r1
jmp    [r1]
----------------------------------------
```

を次のように書き換えてください。

```
----------------------------------------
.option nowarning
ld.w    sbt_intent[r2], r1
jmp     [r1]
.option warning
----------------------------------------
```

Q.83

周期ハンドラとは何ですか？

A.83

ユーザの周期的な処理プログラムです。

周期的な処理プログラムの中で，実行開始までのオーバヘッドが最も小さいものです。

保守／廃止

Q.84

周期ハンドラはどこで生成するのですか？

A.84

コンフィギュレーション（CF）定義ファイル，または，def_cyc**システム・コールで登録することによって生成します。**

Q.85

周期ハンドラはいくつ生成することができますか？

A.85

周期ハンドラ最大登録数で指定した数まで生成できます。

ただし，最大登録数として指定できる値は32767**ですので，最大**32767**個まで生成可能です。**

Q.86

周期ハンドラはどのようにして起動するのですか？

A.86

コンフィギュレーション（CF）定義ファイルで，初期状態を「活性化」指定することによって起動します。

また，あるタスクの中から，act_cyc**システム・コールによって起動をかけたり，中断することもできます。**

Q.87

周期起動ハンドラは割り込み許可状態で起動されるのでしょうか？

禁止状態で起動されるのでしょうか？

A.87

RX850 Pro**の周期起動ハンドラは，割り込み許可状態で起動されます。**

周期起動ハンドラ内を割り込み禁止状態にしたい場合は，先頭でDI（dis_int**システム・コール発行）する必要があります。**

しかし，起動後とDI**命令の間に割り込みが入ると，割り込みが入ってしまいます。**

ただし，割り込みの優先度によります。

詳細については，Q.111**を参照してください。**

保守／廃止

Q.88

周期起動ハンドラから復帰するにはどうしたらいいのですか？

A.88

C言語の場合，周期起動ハンドラの終了部分でreturn命令を発行することによって復帰します。

アセンブラの場合はjmp［lp］命令で復帰します。

Q.89

クロック割り込みの要因は，どこで指定しますか？

A.89

コンフィギュレーション（CF）定義ファイルで指定します。

ただし，コンフィギュレーション（CF）定義ファイルですることは，クロック割り込みの登録だけで，実際に使うタイマのハード面の初期化は，初期化ハンドラや初期タスク，またはスタート・アップ・ルーチンに記述する必要があります。

記述例）

```
--システム情報
clktim  0x1         ← 基本クロック周期（単位：[ms]）
clkhdr  0x7         ← クロック割り込み要因番号：（割り込み例外コード - 0x80）/ 0x10
defstk  0x100
intstk  0x100:SPOL0
prttsk  0x1
prtsem  0x1
prtflg  0x1
prtmbx  0x1
prtmpl  0x1
```

Q.90

周期起動ハンドラの周期起動間隔は，最短いくつですか？

A.90

特に制限はありませんが，コンフィギュレーション（CF）定義ファイルで指定する“タイマの基本クロック周期（clktim）”以下の値は指定できません。

短い間隔で周期起動ハンドラを起動し，またハンドラ内の処理が多かった場合，周期ハンドラ本体の処理が終わる前にクロック割り込みが入ってしまうことも予想されます。

その場合，周期起動ハンドラ処理がネストし続け，他のタスク処理が行われないなど，期待した動作をしない可能性があります。

また，同時に起動する周期起動ハンドラが複数あった場合も，次にタイマ割り込みが入る前に処理が終わらなくなってしまう可能性があるので注意が必要です。

保守／廃止

Q.91

周期起動ハンドラの処理の最小単位は1 ms**ですが，**100μs**以下で処理したい場合はどうすればよいですか？**

A.91

RX850 Pro**の周期起動ハンドラは，**1 ms**程度で基本クロックが入ることを想定して設計されています。**

1 ms**以下の処理の場合は，直接起動割り込みハンドラで対応してください。**

ただし，100 μs**間隔で周期的な割り込み処理を行った場合は，**RX850 Pro**の処理がほとんど進まないような状況になってしまいます。**

Q.92

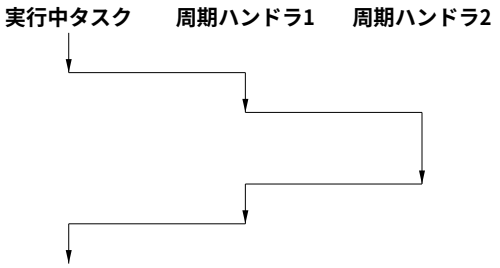
10 ms**間隔で起動する周期起動ハンドラ内で，**10 ms**以上かかる処理を実行した場合，どうなるのでしょうか？**

周期起動ハンドラ内は割り込み許可にしています。

A.92

RX850 Pro V3.13**では，周期起動ハンドラ実行中に，次の周期起動ハンドラが実行されるタイミングになった場合，ネストして実行されます。**

実行遷移を図で表すと，以下のようになります。



異なる周期起動ハンドラでも同じ周期起動ハンドラでも，同様にネストして実行しています。

Q.93

割り込み処理が周期起動ハンドラに割り込まれるのは正しいのでしょうか？

A.93

以下の場合は，割り込みハンドラ処理中でも周期起動ハンドラが起動されます。

- **該当割り込みハンドラが，タイマ割り込みよりも優先度の低い割り込みで起動する場合**
- **該当割り込みハンドラ内に**EI**区間がある場合**
- EI**区間内に達するまでにタイマ割り込みが入っていた，または**EI**区間内でタイマ割り込みが入った場合**
- **入ったタイマ割り込みが，周期起動ハンドラを起動するタイミングだった場合**

このように，周期起動ハンドラの優先度は，タイマ割り込みの優先度に依存すると考えてください。

保守／廃止

Q.94

def_cycシステム・コールで周期起動ハンドラを作成して起動させるとき，周期起動ハンドラが一番始めに起動するタイミングは，def_cycシステム・コールが発行された直後ですか？

def_cycが発行されてから指定時間経過後ですか？

A.94

def_cycが発行されてから，指定時間経過後になります。

これと似ていますが，act_cycで周期ハンドラを活性化するタイミングも，act_cycが発行されてから指定時間経過後になります。

Q.95

複数の周期起動ハンドラが周期起動ハンドラ起動要求キューにつながっている場合，すべての周期起動ハンドラが処理を実行し終わるまで，タスクは待たされますか？

A.95

複数の周期起動ハンドラが周期起動ハンドラ起動要求キューにつながっている場合，すべての周期起動ハンドラが処理を実行し終わるまで，タスクは待たされます。

詳細については，7.6.6　**周期起動ハンドラの起動順序**を参照してください。

Q.96

RX850 Proの割り込み管理機能のシステム・コールについてですが，以下の認識は正しいでしょうか？

	マスカブル割り込み	スケジューラ割り込み	ディスパッチ処理
loc_cpu	禁止	禁止	禁止
dis_int	禁止	禁止	呼出前状態を保持
unl_cpu	許可	許可	許可
ena_int	許可	許可	呼出前状態を保持

マスカブル割り込み　：割り込みコントローラへ接続される割り込み

スケジューラ割り込み　：スケジューリングを発生させる割り込み（一定周期割り込み）

ディスパッチ処理　：スケジューラ割り込み以外のシステム・コール呼出によるスケジューリング

A.96

はい，正しいです。

保守／廃止

Q.97

twai_flgシステム・コールなどで，起床条件を満たしてタスクが起床される前にタイムアウト時間が経過した場合，戻り値はE_OKですか？それともE_TMOUTですか？
また，起床条件を満たした場合とタイムアウトが逆の順序であった場合，どうなりますか？

A.97

起床条件を満たす前にタイムアウトが起こったのであれば，タイムアウトということでE_TMOUTになります。
つまり，起こったイベントの順序がそのまま反映されます。
従って，起床条件を満たした場合とタイムアウトが逆の順序であった場合，戻り値はE_OKになります。

Q.98

コンフィギュレーション（CF）定義ファイルで定義するclktimと，dly_tsk（dlytim）の遅延時間の関係は，どのように考えればよいでしょうか？
clktimに10, dly_tsk（100）と設定した場合に1秒の遅延となり，期待どおりの動作が得られません。

A.98

clktimで指定する値は，実際にハードウエアから入ってくる割り込み間隔を表し，これを元にして，RX850 Proがdly_tskなどで指定される時限待ちの時間を計算するために使われます。

つまり，dly_tsk（1000）と指定した場合，1000 msタスクの動作を遅延するという意味になりますが，RX850 Proではタイマ割り込み間隔の時間がわかりません。

そこで，clktimで，タイマ割り込みの間隔が何msなのかを指定します。

これによって，何回割り込みが入れば1000 msになるかがわかるようになります。

このため，実際に入るタイマの間隔とclktimで指定する値が違うと，期待した遅延時間が得られません。

Q.99

RX850 Proのタイマ精度は，どのくらいですか？

A.99

RX850 Proのタイマの精度は，最小で1 msです。

タイマの精度自体はハードウエアの設定に依存しますが，コンフィギュレーション（CF）定義ファイルで指定する“基本クロック周期（clktim）”で指定できる値の最小値が1 msです。

この値と実際のハード的なタイマの周期を一緒にする必要があります。

RX850 Proは，時間関係の処理をするとき，システム・コールの引数で指定されたms単位の数字を元に処理を行います。

例えば，dly_tsk（1000）とした場合，1000 msの間タスク処理を遅延させることになりますが，その1000 msという値をclktimで指定された値を元に計算するからです。

つまり，“clktim”が“1”と指定されていれば，1000回タイマ割り込みが入れば1000 msになるとRX850 Proが判断します。

“clktim”が“5”と指定されていれば，200回タイマ割り込みが入れば1000 msと判断します。

このカウント方法は，RX850の場合と違いますので，移植する際は注意が必要です。

保守／廃止

RX850の場合は，引数に指定する値は“タイマ割り込みが入った回数（tick）”になっています。

Q.100

クロック・ハンドラの周期についての質問です。

AZ850の実行遷移ウィンドウ（Analyze Window）において，出力されたINTCM40の周期は，約0.113 msでした。

reset.cのタイマは以下のように設定しました。

```
TMC40 = 0x86;
CM40 = 10;
CMIC40 = 0x0;
```

V850E/MS1で，内部カウント・クロックはφm/16，その中間クロックはφ/8，カウントは10を設定しているつもりなのですが，出力された値と計算した値が一致しません。

どのように解釈すればよいでしょうか？

A.100

レジスタの設定に誤りがあります。

V850E/MS1では，TMC40 = 0x86でレジスタを設定すると，内部カウント・クロックはφm/32になります。

内部システム・クロックを25 MHzで計算すると，INTCM40の周期（タイマ4のインターバル）は，約0.113 msになります。

具体的な計算手順は，以下の通りです。

1. タイマ・コントロール・レジスタ（TMC40）の設定値は，内部カウント・クロックと中間クロックの選択によって決まります。
   TMC40 = 0x86（10000110）と設定したということは，内部カウント・クロックにφm/32，中間クロックにφ/8を選択したことになります。

保守／廃止

【タイマ・コントロール・レジスタ（TMC40）の概要】

	7	6	5	4	3	2	1	0
TMC40	CE40	0	0	0	0	PRS400	PRM401	PRM400

ビット位置	ビット名	意　味
7	CE40	タイマの動作を制御 0：動作しない 1：カウント動作
2	PRS400	内部カウント・クロック（ϕm：中間クロック） 0：ϕm/16 1：ϕm/32
1, 0	PRM401, PRM400	中間クロックϕm（ϕ：内部システム・クロック） PRM401 PRM400 ϕm 0 0 ϕ/2 0 1 ϕ/4 1 0 ϕ/8 1 1 RFU（予約）

2. インターバル時間は，以下の式により決定します。

（インターバル時間）=（コンペア・レジスタ値＋1）×（カウント・クロック周期）

カウント・クロック周期は，以下の式により決定します。

（カウント・クロック周期）= 1/（内部カウント・クロック）

内部システム・クロックϕ = 25［MHz］= 25×10^6［Hz］とすると，カウント・クロック周期は，

1/（ϕ/8×1/32）= 1/｛25×10^6×1/8×1/32｝
= （8×32）/（25×10^6）

これにより，コンペア・レジスタCM40 = 10のときのインターバル時間は，以下のようにして求められます。

（10＋1）×｛（8×32）/（25×10^6）［Hz］｝　0.113［ms］

［参考］Q.101

保守／廃止

Q.101

タイマ4をインターバル・タイマとして使用したいのですが，設定はどのようにすればよいですか？

A.101

V850E/MS1**の場合を例にして，説明します。**

内部システム・クロックが25 MHz**のとき，インターバル時間が約**0.1 ms**となるように，タイマ・コントロール・レジスタ（**TMC40**）とコンペア・レジスタ（**CM40**）の値を設定します。**

1. **内部カウント・クロックと中間クロックを選択し，タイマ・コントロール・レジスタ（**TMC40**）の値を設定します。**

例）

内部カウント・クロック → ϕm/32
中間クロック → ϕ/8

	7	6	5	4	3	2	1	0
TMC40	1	0	0	0	0	1	1	0

【タイマ・コントロール・レジスタ（TMC40**）の概要】**

	7	6	5	4	3	2	1	0
TMC40	CE40	0	0	0	0	PRS400	PRM401	PRM400

ビット位置	ビット名	意　味
7	CE40	タイマの動作を制御 0：動作しない 1：カウント動作
2	PRS400	内部カウント・クロック（ϕm：中間クロック） 0：ϕm/16 1：ϕm/32
1, 0	PRM401, PRM400	中間クロックϕm（ϕ：内部システム・クロック） PRM401 PRM400 ϕm 0 0 ϕ/2 0 1 ϕ/4 1 0 ϕ/8 1 1 RFU（予約）

2. **コンペア・レジスタ（**CM40**）の値を設定します。**
インターバル時間は，以下の式により決定します。

（インターバル時間）=（コンペア・レジスタ値＋1**）×（カウント・クロック周期）**

保守／廃止

1）カウント・クロック周期を求めます。
カウント・クロック周期は，以下の式により決定します。

（カウント・クロック周期）= 1/（内部カウント・クロック）

内部システム・クロックϕ = 25［MHz］= 25×10^6［Hz］とすると，カウント・クロック周期は，

1/（ϕ/8×1/32）= 1/ {25×10^6×1/8×1/32}
=（8×32）/（25×10^6）

2）コンペア・レジスタ値を求めます（コンペア・レジスタ値：n）。
インターバル時間を0.1［ms］= 1×10^{-4}［s］とすると，コンペア・レジスタ値は，

1×10^{-4}［s］=（n＋1）×｛（8×32）/（25×10^6）［Hz］｝
n = 1×10^{-4}［s］×｛（25×10^6）［Hz］/（8×32）｝－1　9

従って，インターバル時間を0.1 msに近い値にするには，CM40レジスタに9を設定すればよいことになります。
CM40 = 9と設定した場合のインターバル時間は0.1024 msとなるので，0.1 msに近い値が得られると考えられます。

Q.102

エラー・メッセージ
fa01（F）：PC位置の行情報が見つかりませんでした。
が表示されます。

A.102

プログラムの停止時にソース・ウィンドウが開かれていると，停止時のプログラム・カウンタ（PC）値に対応するソース・ファイルを表示します。

このエラー・メッセージは，その時のプログラム・カウンタ（PC）の値に対応するソース・ファイルが見つからない場合に表示されます。

原因としては，

(1) ソース・ファイルがソース・パスが通っていない場所に存在する。
(2) ライブラリ，リアルタイムOSのシステム・コールなど，ソース・ファイルが存在しないところでプログラムを停止した。
(3) プログラムが暴走し，プログラムで使用していないアドレスへ実行が飛んでしまい，そこで停止した。
(4) ディバグ・モードでビルドしていないために，オブジェクトにディバグ情報が含まれていない。

などが考えられます。

保守／廃止

（1）の場合，ソース・パスは，ディフォールトでダウンロードしているロード・モジュール・ファイルが存在するディレクトリになっています。

それ以外のディレクトリにソースが置いてある場合には，［オプション］メニュー→［ディバッガ・オプション］で，ソース・パスを指定してください。

（2）のように，プログラムの構成上，このメッセージの表示を避けられない場合には，コンソール・ウィンドウを使用してエラー・ダイアログの表示をさせないことが可能です。

［参考］Q.114

Q.103

リンク時に，メッセージ

multiple inclusion of same file attempted, ignored.

が表示されます。

A.103

このメッセージは，スタート・アップ・ファイルとして登録するオブジェクトのソースが，アセンブルするソース対象として登録されている場合に表示されます。

つまり，スタート・アップ・ファイルとしてstart.oが登録されていて，さらにそのソースstart.sがアセンブルする対象として登録されているような場合です

（RX850 Proの場合はstart.oをboot.o, start.sをboot.sに読み替えてください）。

PMのプロジェクトの設定ダイアログで，“ソースファイル名”のリストからstart.sを削除し，かつ，リンカ・オプション設定ダイアログで，“スタートアップファイル”のstart.sをstart.oに変更してください。

また，ROM化用オブジェクトを作成する際，ROM化用確保コードを自分で作成する場合にも同じ現象が起きます。

対処方法はstart.oのときと同じく，ソース・ファイルとして登録しないでビルドすることです。

特に，リアルタイムOSを使用したアプリケーションでは，必ず独自のスタート・アップ・モジュールを使用することから，この作業を行っておく必要があります。

Q.104

リンク時に，エラー・メッセージ

undefined: ’__e_sysfnc’ referenced in ’c:¥nectools32¥lib850e_ghs¥r32¥rxcore.o’

が表示されます。

A.104

このメッセージは，svc.sをアセンブルして得られるsvc.oがリンクされていないために，表示されます。

このファイルは，コンフィギュレータcf850proによって作られるシステム・コール・テーブルです。

このテーブルの先頭アドレスが，__e_sysfuncとなります。

RX850 V3.1xでは必要のないファイルのために忘れがちなので，注意が必要です。

なお，使用するシステム・コールを，コンフィギュレーション定義（CF）ファイルに記述する必要があります。

記述方法，およびコンフィギュレータの起動方法については，RX850 Pro **ユーザーズ・マニュアル　インストレーション編**の6.6　SCT**情報の記述形式，および第**7, 8**章　コンフィギュレータの操作方法**を参照してください。

保守／廃止

Q.105

PM plusを使用してビルドをすると，hx850実行時に，メッセージ

Warning:address is too long

が表示されます。

A.105

これはインテル・ヘキサの制限なので，インテル・ヘキサ・フォーマットを指定し，かつ，1 Mバイト空間を越えるようなプログラムの場合，必ず表示されてしまいます。

実際の開発では，ロード・モジュールをヘキサ・フォーマットに変換する際に考慮すればよいことなので，このメッセージは無視して構いません。

どうしてもメッセージを表示したくない場合は，モトローラ・ヘキサ・フォーマットを選択すれば，表示されなくなります。

Q.106

システムが暴走してしまいます。

A.106

次のような原因が考えられます。

・タスクのスタック領域，および割り込みスタック領域（システム・スタック領域）は十分ありますか？

リアルタイムOSを使ったアプリケーションで，うまく動作しなくなる原因のほとんどがこれです。

つまり，タスク・スタックや割り込みスタックが，指定したサイズを突き抜け，他のタスクのスタックを破壊したり，RX850 Proのシステム管理領域を破壊したりして，結果的に暴走してしまいます。

RX850 Proは，アドレス情報を頼りに動くため，そのアドレスにおかしな値が入っていても疑わずに動作しますので，注意してください。

・間接起動割り込みハンドラの終了処理をきちんと書いていますか？

間接起動割り込みハンドラの終了処理，つまりreturnの戻り値がきちんと設定されていなかった場合，割り込み後の処理がおかしくなります。

うまくいっていたとしても，偶然うまく動いていたことになります。

単に，間接起動割り込みハンドラから抜けるときは，“ return（TSK_NULL）；”のように，引数に“ TSK_NULL ”を指定します。

もし，間接起動割り込みハンドラを抜けるときに，あるタスクを起床させたい場合は，“ return（TASK_ID）；”のように，引数に“ 起床させたいタスクのID ”を指定します。

・タスクが無限ループ記述（for（；；），while（1））されていないタスクは，ext_tskシステム・コールを発行して終了処理をしていますか？

タスクが無限ループ記述になっている場合は問題ありませんが，もしそうでないタスクがある場合，そのタスクは，最後でext_tskシステム・コールを発行して終了処理をする必要があります。

記述されていない場合，RX850 Proはそのタスクが終了したかどうかを判断できません。

よって，確実に暴走します。

保守／廃止

Q.107

ブート処理自体はうまく動作し，ブート処理の最後でRX850 Proに制御を移したのですが，その後，タスクがひとつも動かずに，システムがHALT状態になってしまいます。

A.107

原因は，RX850 Proの初期化部分で，初期化がうまくいかなかったためです。

ブートの最後でRX850 Proに制御を移すと，システム・ベース・テーブル（SBT）の作成などの初期化を始めます。

そこでは，各管理テーブルの作成，メモリ・プールの作成を行いますが，1つでも作成に失敗するとHALT状態にしています。

作成できない原因としては，SBTや各管理テーブルを生成するシステム・メモリ・プール（SPOL0）が確保できない（メモリが書き込めない，領域が足りない）というような原因が考えられます。

Q.108

システム・コールを発行しても，期待した動作をしません。

A.108

次のような原因が考えられます。

・タスクの優先度によるもの

待ちを解除するために，システム・コールを発行したときに，解除されるタスクの優先度が他のタスクの優先度よりも低かった場合，すぐに動作を開始しないため，期待した動作をしないことがあります。

・スタック破壊によるもの

wai_flgやwai_semなどの待ち系のシステム・コールを発行し，その待ちを解除しようとしてもうまくいかない場合，タスク・スタックや割り込みスタックによる領域破壊が原因である可能性があります。

タスク・スタックや割り込みスタックが設定値より伸び，それがRX850 Proの管理領域を破壊し，その中の待ち情報などが破壊されていることがあります。

Q.109

システム・コールの戻り値として，“－17”が返されます。

このエラー値は，何を意味するのですか？

A.109

エラーの内容としては，E_NOSPTを表します。

これは，システム・コールが，システム・コール・テーブルに登録されていないという意味です。

保守／廃止

Q.110

RX850 Proのニュークリアス・オブジェクトのrxcore.o（rxtmcore.o）もリンクするのはわかりますが，PM Plusで，オブジェクトをファイル一覧に登録することができません。
このファイルはどうすればリンクできるでしょうか？

A.110

PM Plusでリンクしたいオブジェクトを指定するときは，リンカオプションの設定ダイアログの“その他”にある“他のオプション（Y）”のところに，オブジェクト名を設定します。

フルパスで指定すると安全です。

rxcore.oをリンクしたい場合は，

c:¥nectools32¥lib850e¥r32¥rxcore.o

と設定してください（rxtmcore.oを使用する場合はc：nectools32¥lib850e¥r32¥rxtmcore.o）。

なお，複数のオブジェクトを設定する場合は，半角スペースで区切ってください。
“，（コロン）”や“；（セミコロン）”で区切らないでください。

Q.111

ニュークリアス・オブジェクトrxcore.oとrxtmcore.oの違いは何ですか？

A.111

タイマ割り込み処理中の割り込み受け付けの仕様が異なります。

オブジェクト名	仕　様
rxcore.o	周期起動ハンドラ処理中に，すべての割り込みレベルの割り込みが受付可能なニュークリアス・カーネル
rxtmcore.o	周期起動ハンドラ処理中に，タイマ割り込みよりも優先度の高い割り込みレベルの割り込みのみ受付可能なニュークリアス・カーネル

周期起動ハンドラは，タイマ・ハンドラから呼び出されます。

rxcore.oでは，タイマ・ハンドラ実行中に一度割り込み終了処理（reti）を行っています。

そのため，周期起動ハンドラ処理中は，すべての割り込みレベルの割り込みが受け付け可能になります。

一方，rxtmcore.oでは，タイマ・ハンドラ実行中に周期起動ハンドラを呼び出すため，タイマ割り込みよりも優先度の高い割り込みのみ，受け付け可能になります。

保守／廃止

Q.112

RX850 Ver.3.1では，.sitセクションを0番地±32 Kバイト以内に配置しなくてはなりませんでしたが，RX850 Proでも同じですか？

A.112

RX850 Proでは，この制限はありません。

配置場所に関する制限はありません。

Q.113

RX850 Pro本体だけをROMやFLASH ROMに焼くには，どうすればよいですか？

A.113

RX850 Pro本体を焼いて，その後はユーザ・アプリケーションだけを変更していく方法を取ることができます。

これは，ROM化するセクションを切り分けることで実現できます。

RX850 Pro本体において，ROM化可能なセクションは次の通りです。

- .system
- .system_int
- .system_cmn

.system に配置されるものは，

- rxcore.o内でRXが共通に使用するルーチン
- svc.o（システム・コール・テーブル）
- システム・コール本体（cretsk.oなど）
- システム・コールで共通に使用されるルーチン（f_memget.oなど）

.system_intに配置されるものは，

- rxcore.o内の割り込み処理部分

.system_cmnに配置されるものは，

- rxcore.o内のスケジューラ処理部分

となります。

ただし，システム・コール本体は，現在は使用していなくても，今後使用される可能性があるものは，含めて配置する必要があります。

実際にライブラリ（librxp.a/librxpm.a）中のどのオブジェクトを使っているかは，リンカのオプションでリンク・マップの出力（NECエレクトロニクス版：-m/GHS版：-map）を指定してリンク情報を取得し，そこに出力されるオブジェクト名を参照してください。

この対処をしていただければ，ROM化されるRX850 Pro本体の領域はユーザ・プログラムの変更によって影響は受けません。

なお，インタフェース・ライブラリ，および.sitセクションは，ユーザ・アプリケーション側にリンクしてください。

また，ユーザ・プログラム側のブート部分（boot.s/boot.850）で，

```
mov #__rx_start, lp
jmp [lp]
```

というコードがありますが，__rx_startというシンボルはROM化したRX850 Pro側にありますので，#__rx_startは実アドレスにしてjmpする必要があります。

以下に，該当部分のリンク・ディレクティブの例（NECエレクトロニクス版）をあげます。

【例1】ライブラリ内のオブジェクトを別々の出力セクションにする方法

```
SYSTEM : !LOAD ?RX {
  .system_svc = $PROGBITS ?AX .system { svc.o };
  .system_core = $PROGBITS ?AX .system { ..¥..¥..¥lib850e¥r32¥rxcore.o };
  .os_lib1 = $PROGBITS ?AX .system {udfsys.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib2 = $PROGBITS ?AX .system {relblk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib3 = $PROGBITS ?AX .system {getblk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib4 = $PROGBITS ?AX .system {gettim.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib5 = $PROGBITS ?AX .system {sndmsg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib6 = $PROGBITS ?AX .system {rcvmsg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib7 = $PROGBITS ?AX .system {sigsem.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib8 = $PROGBITS ?AX .system {waisem.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib9 = $PROGBITS ?AX .system {setflg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib10 = $PROGBITS ?AX .system {waiflg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib11 = $PROGBITS ?AX .system {wuptsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib12 = $PROGBITS ?AX .system {statsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib13 = $PROGBITS ?AX .system {exdtsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib14 = $PROGBITS ?AX .system {exttsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .os_lib15 = $PROGBITS ?AX .system {cretsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)};
  .system_cmn = $PROGBITS ?AX .system_cmn;
  .system_int = $PROGBITS ?AX .system_int;
};
TEXT : !LOAD ?RX {
 .text = $PROGBITS ?AX .text;
};
```

保守／廃止

【例2】ライブラリ内のオブジェクトを1つのセクションにまとめる方法

```
SYSTEM : !LOAD ?RX {
  .system_svc = $PROGBITS ?AX .system { svc.o };
  .system_core = $PROGBITS ?AX .system { ..¥..¥..¥lib850e¥r32¥rxcore.o };
  .system = $PROGBITS ?AX .system { svc.o ..¥..¥..¥lib850e¥r32¥rxcore.o
    udfsys.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    relblk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    getblk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    gettim.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    sndmsg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    rcvmsg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    sigsem.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    waisem.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    setflg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    waiflg.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    wuptsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    statsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    exdtsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    exttsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
    cretsk.o(c:¥nectools32¥lib850e¥r32¥librxp.a)
  };
  .system_cmn = $PROGBITS ?AX .system_cmn;
  .system_int = $PROGBITS ?AX .system_int;
};
TEXT : !LOAD ?RX {
  .text = $PROGBITS ?AX .text;
};
```

上記のいずれかの方法が一般的です。

なお，リンクの際は，librxp.a（librxpm.a）**を参照するオプションをつけてください。**

Q.114

自分達のターゲット・システムに搭載している外部RAM**は16ビット，32ビットアクセスのみ可能で，8ビット（1バイト）アクセスができません。**

このようなシステムでRX850 Pro**は使用できますか？**

A.114

RX850 Pro**の管理領域が配置される**SPOL0**領域は8ビット（1バイト）単位でデータ・アクセスできる必要があります。つまり外部**RAM**領域に8ビット・アクセスができないような場合，その領域に**SPOL0**領域を配置することができません。配置されると動作中にデータが欠落し，正常動作しません。**

スタックやメモリ・プールへのアクセスに関しては，8ビット・アクセスがないので，SPOL1**領域に関しては，配置しても問題ありません。**

ただしSPOL1**領域内に作ったメモリ・ブロックなどにデータ・アクセスするコードに関しては，コンパイル・オプションなどで8ビット・アクセスを抑制する必要があります。**

なお，この問題はSPOL0**領域を**V850**シリーズの内蔵**RAM**に配置することができれば解決できます。**

保守／廃止

付録 C　総合索引

C.1　50 音で始まる語句の索引

【な行】

【は行】

【ま行】

保守／廃止

C.2　記号，アルファベットで始まる語句の索引

【S】

【T】

【U】

【V】

【W】

保守／廃止

付録D　改版履歴

これまでの改版履歴を次に示します。なお，適用箇所は各版での章を示します。

箇　所	内　　容	適応箇所
第2版	戻り値E_NOSPTの数値「－11」を「－17」に修正	全般
	動作対象CPUの記述を変更	第1章　概　　説
	ハードウエア環境とソフトウエア環境の記述を変更	
	システム構築手順の説明を追加	
	直接起動割り込みハンドラからの復帰処理についての注意文を追加	第5章　割り込み管理機能
	割り込み制御レジスタの変更／獲得についての注意文を追加	
	メモリ管理機能の概要の記述を変更	第6章　メモリ・プール管理機能
	管理オブジェクトの記述を変更	
	メモリ・プールとメモリ・ブロックの記述を変更	
	メモリ・ブロックの返却についての注意文を追加	
	周期起動ハンドラ中の割り込みの説明を追加	第7章　時間管理機能
	周期起動ハンドラの起動順序の説明を追加	
	アイドル・ハンドラの説明を追加	第8章　スケジューラ
	システム初期化処理の説明を追加	第9章　システム初期化処理
	直接起動割り込みハンドラの説明を追加	付録A　プログラミングのために
	間接起動割り込みハンドラの説明を追加	
★ 第3版	V850**ファミリ**をV850**シリーズ**に変更	全般
	動作対象CPUを追加	第1章　概　　説
	周辺コントローラの記述を変更	
	インサーキット・エミュレータの対象のデバイスを追加	
	インサーキット・エミュレータ用I/Oボード，対象デバイスを追加	
	ソフトウエア環境のOSの記述を変更，ディバッガを追加	
	メモリ・ブロックの返却の注意の記述を変更	第6章　メモリ・プール管理機能
	周期起動ハンドラ中の割り込みの説明を変更	第7章　時間管理機能
	rel_blkの説明中の注意をメモリ・ブロックの返却のところの注意と同じにした	第11章　システム・コール
	set_timとget_timの**システム・クロック**SYSTIME**の構造**の説明中のsystimeをt_systime**に変更**	
	付録B　Q&A**を追加**	**付録**B　Q & A

保守／廃止

【発　行】

NECエレクトロニクス株式会社

〒211-8668　神奈川県川崎市中原区下沼部1753

電話（代表）：044(435)5111

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。

URL(アドレス)　http://www.necel.co.jp/

【営業関係お問い合わせ先】

下記のページに最新版のお問い合わせ先が記載されています。

URL(アドレス)　http://www.necel.com/ja/contact/contact_j.html

【技術的なお問い合わせ先】

半導体テクニカルホットライン

（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044−435−9494
FAX　：044−435−9608
E-mail　：info@lsi.nec.co.jp

【資料請求先】

NECエレクトロニクス特約店または上記ホームページ記載の営業関係お問い合わせ先へお申し付けください。