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。

NEC

RENESAS

ユーザーズ・マニュアル

保守／廃止

RX830 （μITRON Ver.3.0）

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

V830 ファミリ™

資料番号 U13152JJ2V0UM00（第2版）
発行年月 June 1998 CP(K)

保守／廃止

〔メ　モ〕

保守／廃止

保守／廃止

M7A 94.11

はじめに

マイクロプロセッサは半導体技術の進歩にともなって急速に普及し，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合わせた固有のプログラムをその都度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）の需要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されます。つまり，各々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 では，このような市場状況を考慮し V800 シリーズ ™，V830 ファミリ ™ 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，高機能なマイクロプロセッサが持つ機能を十分引き出すため，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，RX830 を開発，発売しました。

RX830 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，一層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。

保守／廃止

〔メ　モ〕

保守／廃止

目次

保守／廃止

表目次

図目次

〔メ　モ〕

第１章　概説

1.1　概要

RX830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おもな目的としています。

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に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としたOSがマルチタスクOSです。

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

1.4 特徴

以下に，RX830 の特徴を示します。

(1) μITRON3.0 仕様に準拠

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

μITRON3.0 仕様で規定されているシステム・コール (66 種類) のほかに，RX830 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，RX830 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 を構築することができます。

(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
- 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

V830 ファミリ™ 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 RAM の活用

V830 ファミリ™ の内蔵 RAM(内蔵命令メモリ，内蔵データ・メモリ) 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) 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 CF830
- 高級言語インタフェース・ライブラリ

(9) C コンパイラ

以下に示す V830 ファミリ™ 用 C コンパイラに対応しています。

- CA830　NEC 製
- CCV830　米国 Green Hills Software, Inc. 製

1.5 構成

RX830 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，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。
以下に，これらサブシステムの概要を示します。

(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 RX830 の核となる部分であり，以下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，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，および，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。

(2) システム初期化処理

システム初期化処理は，RX830 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，および，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，RX830 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），および，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，様々なターゲット・システムへの移植性を向上させるとともに，カスタイ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・コールの発行形式，および，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，外部関数形式で発行されたシステム・コール，または，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，RX830 が提供するインタフェース・ライブラリは，NEC 製 V830 ファミリ™ 用 C コンパイラ CA830 用，および，米国 Green Hills Software, Inc. 製 C クロス V800™ コンパイラ CCV830 用の 2 種類が用意されています。

(4) コンフィギュレータ CF830

RX830 を使ったシステムを構築する場合，RX830 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，RX830 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（CF 定義ファイル）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 CF830”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 CF 定義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

以下に，RX830 の適用分野を示します。

- 高速なデータ処理が必要なシステム

  例　ゲーム
  　　カー・ナビゲーション
  　　高機能テレビ
  　　カラー FAX

- 低消費電力が必要なシステム

  例　PDA
  　　PHS

1.7 実行環境

以下に，RX830 が処理を実行するうえで必要となる環境を示します。

- プロセッサ

  V830 ファミリ ™

- 周辺コントローラ

  RX830 では，実行環境のハードウエア構成に依存する部分（システム初期化処理：ブート処理，ハードウエア初期化部）については，サンプル・ソース・ファイルを提供しています。

  このため，システム初期化処理を各ターゲット・システム用に書き替えることで，特定の周辺コントローラは要求しません。

- メモリ容量

  以下に，RX830 が処理を実行するうえで必要となるメモリ容量を示します。

  ニュークリアス・テキスト部 ： 約 4 ～ 11 K バイト
  ニュークリアス・データ部 ： 約 1 ～ 2 K バイト

  なお，上記に示したメモリ容量の最大値は，コンフィギュレーション時に，「優先度の指定範囲を最大に設定する」，「RX830 が提供する機能（システム・コール）をすべて使用する」を指定した場合のものであり，優先度の指定範囲や機能に制限を設けることにより，必要となるメモリ容量を小さくすることが可能です。

1.8 開発環境

以下に，アプリケーション・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，および，ソフトウエア環境を示します。

- ハードウエア環境
  - ホスト・マシン
    - * PC-9800 シリーズ　Windows Ver 3.1，または，Windows 95
    - * IBM-PC/AT 互換機　Windows Ver 3.1，または，Windows 95
    - * SPARC station™　SunOS™ Rel 4.1.x
    - * SPARC station™　Solaris™ Rel 2.x
  - インサーキット・エミューレータ
    - * IE-705100-MC-EM1
  - ネットワーク・モジュール
    - * IE-70000-MC-SV2
- ソフトウエア環境
  - C コンパイラ
    - * CA830　NEC 製
    - * CCV830　米国 Green Hills Software, Inc. 製
  - ディバッガ
    - * ID830　NEC 製
    - * RD830　NEC 製
    - * MULTI　米国 Green Hills Software, Inc. 製
  - システム・パフォーマンス・アナライザ
    - * AZ830　NEC 製

1.9　システム構築手順

システム構築とは，RX830の提供媒体（CGMT，または，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のち，ターゲット・システムへ組み込むことです。

以下に，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，「RX830　ニュークリアス　インストレーション編」を参照してください。

(1) CF定義ファイルの作成

(2) 情報ファイルの作成

- 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

(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

(5) 初期化データの退避領域の作成

(6) リンク・ディレクティブ・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

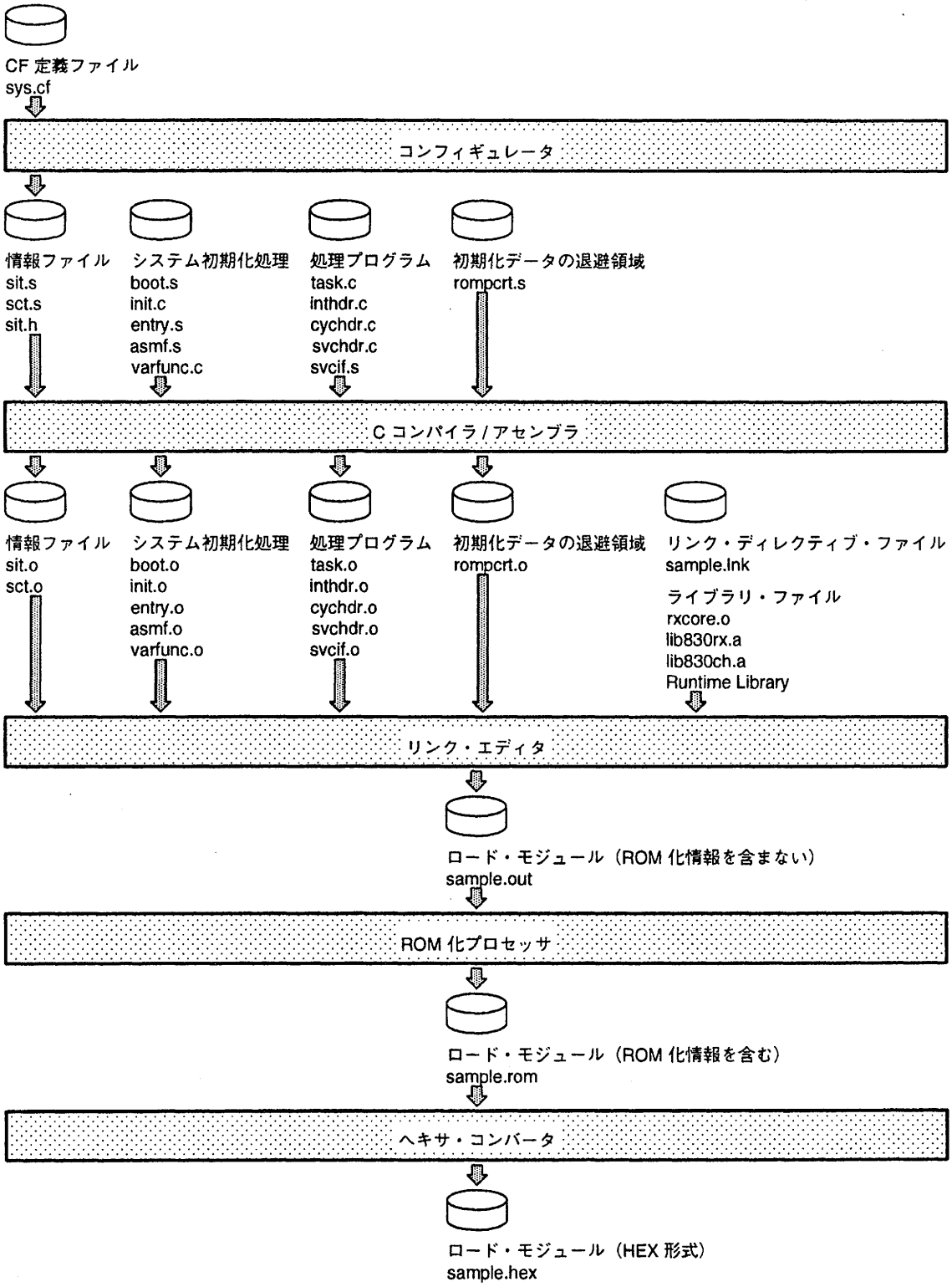
注意　米国Green Hills Software, Inc.製CクロスV800™コンパイラCCV830を使用した場合，“初期化データの退避領域”を作成する必要はありません。

図1-1～図1-2に，システムを構築する際の手順を示します。

ただし，図1-1は，NEC製V830ファミリ™用Cコンパイラ CA830を使用した場合のシステム構築手順であり，図1-2は，米国Green Hills Software, Inc.製CクロスV800™コンパイラCCV830を使用した場合のシステム構築手順です。

保守／廃止

図 1-1　システム構築手順 CA830

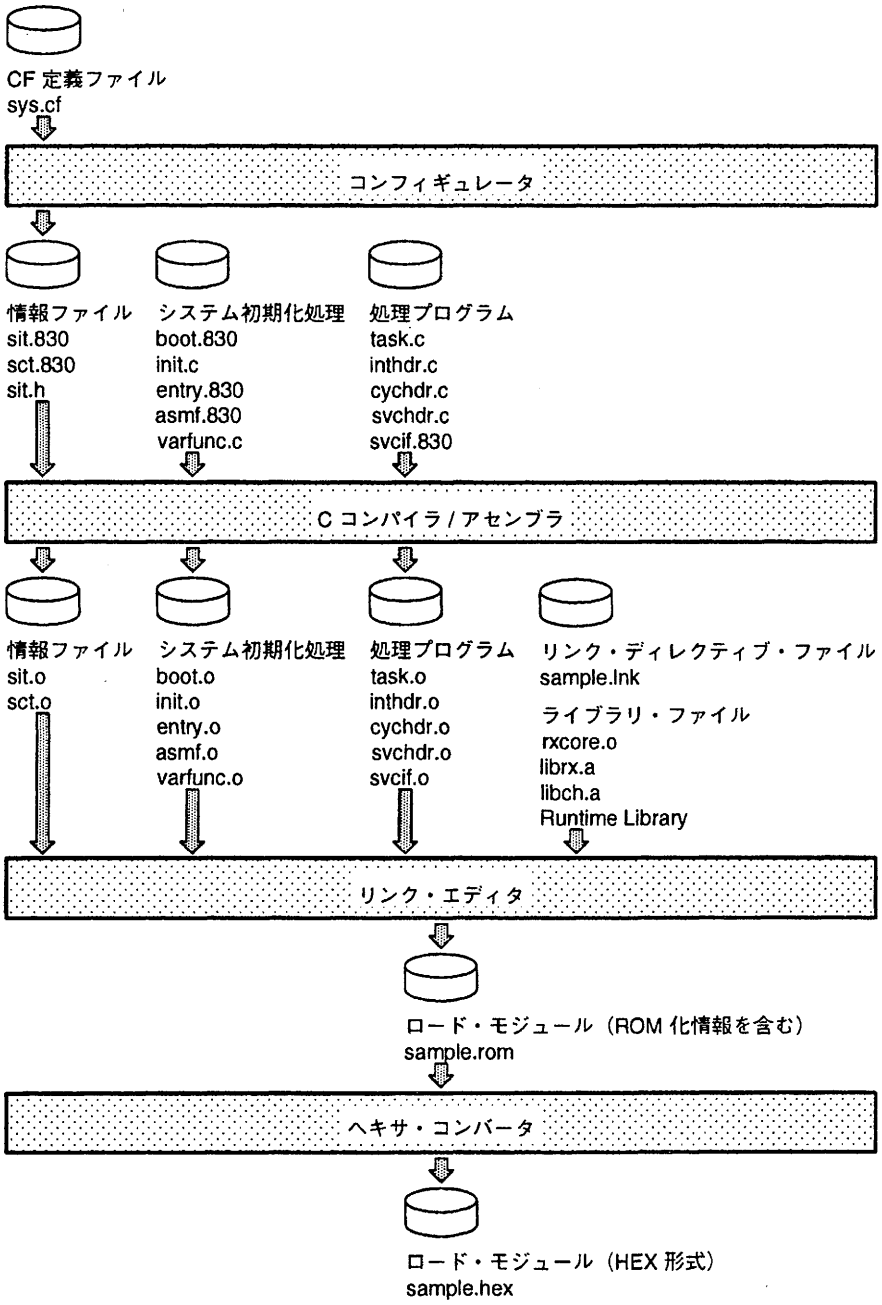


注意　以下に，図1-1に示した各種ファイルの概要を示します。

- CF 定義ファイル
  - `sys.cf`　：CF 定義ファイル
- 情報ファイル
  - `sit.s`　：システム情報テーブル
  - `sct.s`　：ブランチ・テーブル
  - `sit.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
  - `inti.c`　：ハードウエア初期化部
  - `entry.s`　：ハードウエア初期化部（割り込み／例外エントリ）
  - `asmf.s`　：ハードウエア初期化部（ポート操作）
  - `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`　：ニュークリアス共通部
  - `lib830rx.a`：ニュークリアス・ライブラリ
  - `lib830ch.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

保守／廃止

図 1-2　システム構築手順　CCV830



注意　以下に，図1-2に示した各種ファイルの概要を示します。

- CF 定義ファイル
  - sys.cf　：CF 定義ファイル
- 情報ファイル
  - sit.830　：システム情報テーブル
  - sct.830　：ブランチ・テーブル
  - sit.h　：システム情報ヘッダ・ファイル
- システム初期化処理
  - boot.830　：ブート処理
  - inti.c　：ハードウエア初期化部
  - entry.830　：ハードウエア初期化部（割り込み／例外エントリ）
  - asmf.830　：ハードウエア初期化部（ポート操作）
  - 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.830　：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　：ニュークリアス共通部
  - librx.a　：ニュークリアス・ライブラリ
  - libch.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.rom：ROM 化情報を含む
  - sample.hex：HEX 形式

第 2 章　ニュークリアス

本章では，ニュークリアスについて示しています。

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 RX830 の核となる部分であり，以下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，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，および，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。

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。
以下に，各種管理モジュール，および，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

RX830 の処理単位である，「タスクの生成，起動，終了，削除」などといったタスクの状態操作，および，タスクの状態管理を行っています。

(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，以下に挙げる 3 種類の機能を提供しています。

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っています。

(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，以下に挙げる 2 つの領域に分けて管理しています。

- RX830 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，RX830 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となった際に獲得し，不要と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可能となります。

(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）などを提供しています。

(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ています。

なお，RX830 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，および，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，および，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ています。

注意　RX830 におけるタスクの優先度は，その値が小さいほど，高い優先度を意味します。

第 3 章　タスク管理機能

本章では，RX830 が行うタスク管理機能について示しています。

3.1　概要

タスクは実行実体であり，サイズなどが一様でなく，直接管理することが困難です。そこで，RX830 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。

注意　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，処理を実行するうえで必要となる資源の獲得状況，および，事象発生の有無などにより，様々な状態へと遷移します。
そこで，RX830 では，タスクが遷移する状態を，以下に示す 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ながらも，RX830 の管理下にない状態です。

(2) 休止 (dormant) 状態

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

なお，dormant 状態のタスクは，RX830 のスケジューリング対象から除外されています。

また，wait 状態との相違点は，

- 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

などといった点が挙げられます。

(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となる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) を持つタスクが処理を実行中のため，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，RX830 のスケジューリング対象となります。

(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

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

なお，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の再開となります。したがって，処理を再開するうえで必要となるタスク・コンテキストは，中断直前の値が復元されます。

また，RX830 では，wait 状態へと遷移する要因となった条件の種類により，以下に示す 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の獲得が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の受信が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・ブロックの獲得が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処理の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。

注意　RX830 では，suspend 状態をネストさせることも可能です。

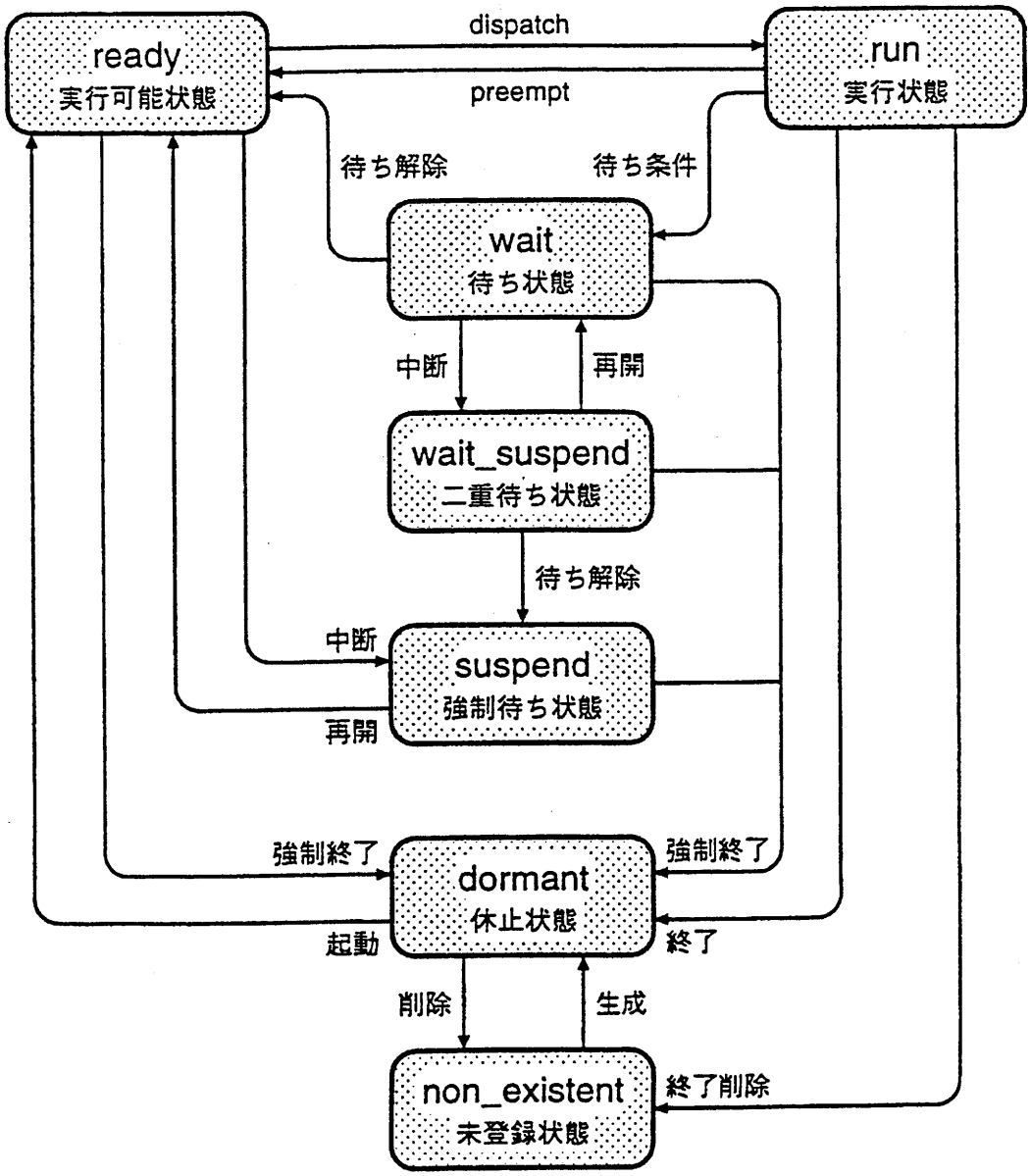
(7) 二重待ち (wait_suspend) 状態

wait 状態と suspend 状態が複合した状態です。

したがって，wait 状態が解除された際には suspend 状態へ，suspend 状態が解除された際には wait 状態へと遷移します。

図 3-1に，タスクが取り得る状態の関係を示します。

図 3-1　タスクの状態遷移



3.3 タスクの生成

RX830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，RX830におけるタスクの生成は，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・メモリ領域から確保したのち，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。

- 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することにより行われます。

  RX830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もとにタスクの生成処理を行い，管理対象とします。

- 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・コールを発行することにより行われます。

  RX830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もと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

RX830におけるタスクの起動は，タスクの状態をdormant状態からready状態へと遷移させ，RX830の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，sta_tskシステム・コールを発行することにより行われます。

- 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

RX830に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状態へと遷移させ，RX830のスケジューリング対象から除外することです。

なお，RX830では，タスクの終了形態として，以下に示す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，自ら終了する形態です。

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。

なお，タスクの終了は，ext_tsk，exd_tsk，または，ter_tsk システム・コールを発行することにより行われます。

- 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。

3.6 タスクの削除

RX830 に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，RX830 の管理下から除外することです。

なお，タスクの削除は，exd_tsk，または，del_tsk システム・コールを発行することにより行われます。

- 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

RX830 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。
そこで，タスクの処理を記述する際には，以下に示す注意事項があります。

(1) レジスタの退避／復帰

RX830 では，タスクを切り替える際，C コンパイラ（CA830，または，CCV830）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がありません。

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。

注意　RX830 では，タスクを切り替える際，gp，tp レジスタの切り替えを行いません。

(2) スタックの切り替え

RX830 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，および，終了部分でスタックの切り替え処理を記述する必要がありません。

(3) システム・コールの発行制限

タスク内で発行可能なシステム・コールの一覧を，以下に示します。

- 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	loc_cpu	unl_cpu	chg_ilv	ref_ilv

- 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。

  以下に，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
  - 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

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。

3.8　システム・タスク

3.8.1　アイドル・タスク

アイドル・タスクは，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 状態でなくなった際，すなわち，RX830 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。

(1) アイドル・タスクの生成／起動

アイドル・タスクは，システム初期化処理（ニュークリアス初期化部）で生成，起動され，ready 状態となります。

なお，アイドル・タスクは，RX830 が定義するシステム・タスクであり，ユーザの処理プログラム（タスク，非タスク）からアイドル・タスクに対して操作（生成，起動，終了，削除など）を行うことは禁止されています。

(2) アイドル・タスク内での処理

アイドル・タスクの役割は，プロセッサを HALT 状態にすることです。そこで，アイドル・タスクでは，その処理として，HALT 命令の発行を行っています。

なお，プロセッサの HALT 状態を解除する要因には，以下の 2 つが挙げられます。

- 外部割り込み（マスカブル割り込み，ノンマスカブル割り込み）の発生

  割り込みが発生した場合，該当する割り込みハンドラが起動され，HALT 状態は解除されます。

  ただし，ノンマスカブル割り込みに対応した割り込みハンドラは，システム・コールの発行が禁止されているため，処理終了時には，再び HALT 状態となります。

- ハードウエア・リセット

  ハードウエア・リセットが発生した場合，初期化処理（ブート処理）からの処理再開となり，HALT 状態は解除されます。

第 4 章　同期通信機能

本章では，RX830 が行う同期通信機能について示しています。

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るとか，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処理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。

RX830 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，RX830 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。

RX830 では，このような機能を「通信機能」と呼びます。なお，RX830 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。RX830 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，以下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

注意 RX830 では，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。

4.2.1 セマフォの生成

RX830 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。

なお，RX830 におけるセマフォの生成は，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のち，初期化することです。

- 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することにより行われます。

  RX830 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もとにセマフォの生成処理を行い，管理対象とします。

- 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・コールを発行することにより行われます。

  RX830 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もと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われます。

- 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，RX830 の管理対象から除外されます。

  ただし，本システム・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 状態へと遷移させています。

  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われます。

- 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，本システム・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。

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われます。

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

  なお，資源待ち状態の解除は，以下の場合に行われ，資源待ち状態から ready 状態へと遷移します。

- 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，対象セマフォを削除した
- rel_wai システム・コールが発行され，強制的に待ち状態を解除した

注意　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方法は，対象セマフォ生成時（コンフィギュレーション時，cre_sem システム・コール発行時）に指定された順（FIFO 順，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，本システム・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，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，本システム・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューにキューイングしたのち，run 状態から wait 状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態の解除は，以下の場合に行われ，資源待ち状態から 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
- 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，対象セマフォを削除した
- rel_wai システム・コールが発行され，強制的に待ち状態を解除した

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

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。

  以下に，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。

4.2.7 セマフォによる排他制御

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，以下に示します。

（前提条件）

- 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

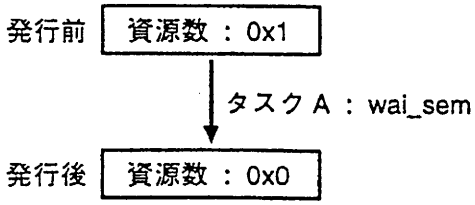
(1) タスク A が wai_sem システム・コールを発行します。

現在，RX830 が管理している対象セマフォの資源数は「0x1」です。このため，RX830 は，対象セマフォから 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

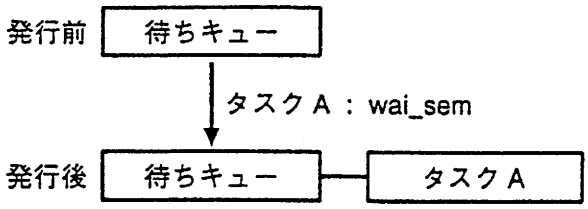


(2) タスク A が wai_sem システム・コールを発行します。

現在，RX830 が管理している対象セマフォの資源数は「0x0」です。このため，RX830 は，タスク A を run 状態から資源待ち状態へと遷移させたのち，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。

なお，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



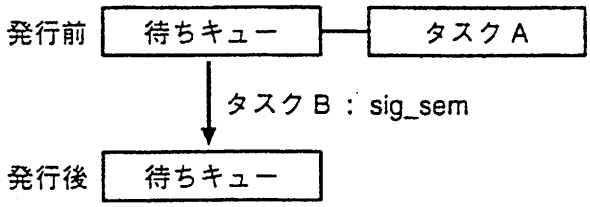
(3) タスク A の資源待ち状態への遷移にともない，タスク B が ready 状態から run 状態へと遷移します。

(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。

なお，対象セマフォの待ちキューは，図 4-3のようになります。

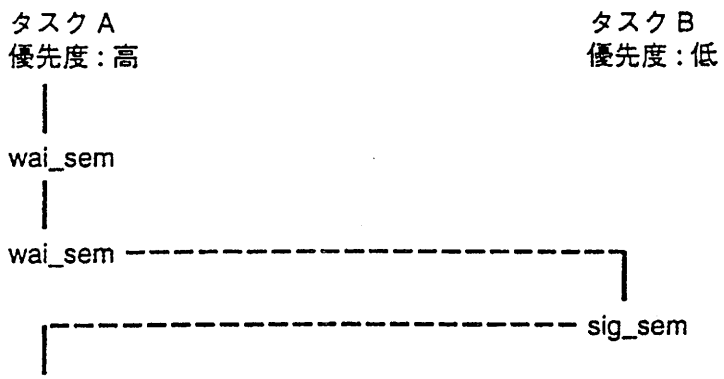
図 4-3　待ちキューの状態



(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。
なお，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）～（5）の排他制御の流れを示します。

図 4-4　セマフォによる排他制御



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，RX830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あり，RX830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することができます。

なお，イベント・フラグに対するダイナミックな操作は，以下に示すイベント・フラグ関連のシステム・コールを用いて行います。

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_fid ： 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

RX830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，RX830におけるイベント・フラグの生成は，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・メモリ領域から確保したのち，初期化することです。

- 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することにより行われます。

  RX830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もとにイベント・フラグの生成処理を行い，管理対象とします。

- 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・コールを発行することにより行われます。

  RX830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もとにイベント・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われます。

- 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，RX830 の管理対象から除外されます。

  ただし，本システム・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 状態へと遷移させています。

  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_flg，twai_flg）の戻り値として E_DLT が返されます。

4.3.3 ビット・パターンのセット

ビット・パターンのセットは，set_flg システム・コールを発行することにより行われます。

- 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，本システム・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

ビット・パターンのクリアは，clr_flg システム・コールを発行することにより行われます。

- 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，本システム・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 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

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・コールを発行することにより行われます。

- 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否かをチェックします。

  ただし，本システム・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のち，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・フラグ待ち状態の解除は，以下の場合に行われ，イベント・フラグ待ち状態から ready 状態へと遷移します。

  - set_flg システム・コールが発行され，要求する待ち条件をセットした
  - del_mbx システム・コールが発行され，対象イベント・フラグを削除した
  - rel_wai システム・コールが発行され，強制的に待ち状態を解除した

- 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否かをチェックします。

  ただし，本システム・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，戻り値として E_TMOUT が返されます。

- twai_flg

  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否かをチェックします。

  ただし，本システム・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のち，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・フラグ待ち状態の解除は，以下の場合に行われ，イベント・フラグ待ち状態から 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
  - set_flg システム・コールが発行され，要求する待ち条件をセットした
  - del_mbx システム・コールが発行され，対象イベント・フラグを削除した
  - rel_wai システム・コールが発行され，強制的に待ち状態を解除した

また，RX830では，イベント・フラグの待ち条件として，以下の2条件を指定することができます。

(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・パターンをクリアする

  待ち条件を満足した際，対象イベント・フラグのビット・パターンのクリアを行います。

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。

  以下に，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。

4.3.8 イベント・フラグによる待ち合わせ

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，以下に示します。

（前提条件）

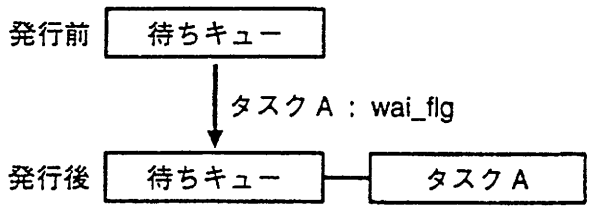
- タスクの優先度
  タスク A ＞ タスク B
- タスクの状態
  タスク A ： run 状態
  タスク B ： ready 状態
- イベント・フラグの属性
  初期ビット・パターン　　： 0x0
  待ちキューにキューイング可能なタスク数 ： 1 タスクのみ

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

現在，RX830 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。このため，RX830 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）へと遷移させたのち，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。

なお，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



(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。

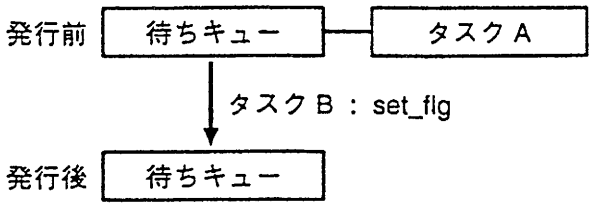
(3) タスク B が set_flg システム・コールを発行します。なお，設定ビット・パターンは「0x1」です。

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。

なお，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。

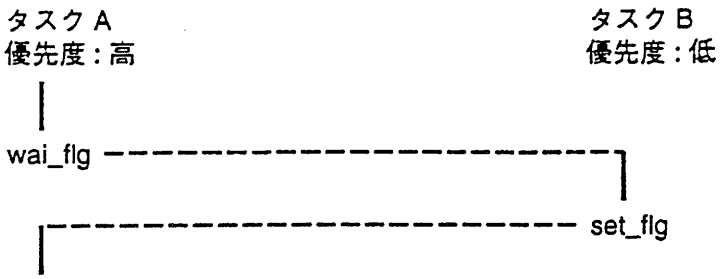
図 4-6　待ちキューの状態



(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) ～ (4) の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，他タスクの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。RX830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。

RX830におけるメールボックスは，タスク専用の待ちキュー（タスク用待ちキュー）と，メッセージ専用の待ちキュー（メッセージ用待ちキュー）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使用することもできます。

なお，メールボックスに対するダイナミックな操作は，以下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

RX830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。

なお，RX830におけるメールボックスの生成は，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のち，初期化することです。

- 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することにより行われます。

  RX830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もとにメールボックスの生成処理を行い，管理対象とします。

- 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・コールを発行することにより行われます。

  RX830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もと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，RX830 の管理対象から除外されます。

  ただし，本システム・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 状態へと遷移させています。

  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。

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，本システム・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。

  注意　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のキューイング方法は，対象メールボックス生成時（コンフィギュレーション時，cre_mbx システム・コール発行時）に指定された順（FIFO 順，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，本システム・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することができなかった（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のち，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の解除は，以下の場合に行われ，メッセージ待ち状態から ready 状態へと遷移します。

- 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を削除した
- rel_wai システム・コールが発行され，強制的に待ち状態を解除した

注意　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のキューイング方法は，対象メールボックス生成時（コンフィギュレーション時，cre_mbx システム・コール発行時）に指定された順（FIFO 順，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，本システム・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することができなかった（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，本システム・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することができなかった（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のち，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の解除は，以下の場合に行われ，メッセージ待ち状態から 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
- 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を削除した
- rel_wai システム・コールが発行され，強制的に待ち状態を解除した

注意　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のキューイング方法は，対象メールボックス生成時（コンフィギュレーション時，cre_mbx システム・コール発行時）に指定された順（FIFO 順，優先度順）に行われます。

4.4.5 メッセージ

RX830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，任意のタスクに送信することができますが，RX830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。

- 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，RX830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。このため，メモリ・プール以外の領域からメッセージ用の領域を確保する場合は，4バイト・アラインされた領域を使用しなければなりません。

- メッセージ内容の作成

  RX830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特にありません。したがって，先頭4バイト以降の長さ，内容については，メールボックスを使用してタスク間通信を行うタスク間で規定します。

  注意　RX830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。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メッセージの先頭4バイトに“0x0”を設定する必要があります。
  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，RX830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を返しています。

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われます。

- 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。

  以下に，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。

4.4.8 メールボックスによるタスク間通信

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，以下に示します。

（前提条件）

- 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

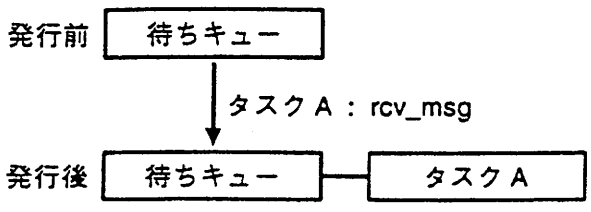
  メッセージのキューイング順序 ： FIFO 順

(1) タスク A が rcv_msg システム・コールを発行します。

現在，RX830 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。このため，RX830 は，タスク A を run 状態から wait 状態（メッセージ待ち状態）へと遷移させたのち，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。

なお，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



(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。

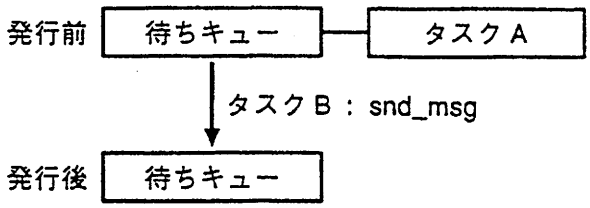
(4) タスクBがメモリ・ブロックにメッセージを書き込みます。

(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るタスクAがメッセージ待ち状態からready状態へと遷移します。

なお，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。

図 4-9　タスク用待ちキューの状態



(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。

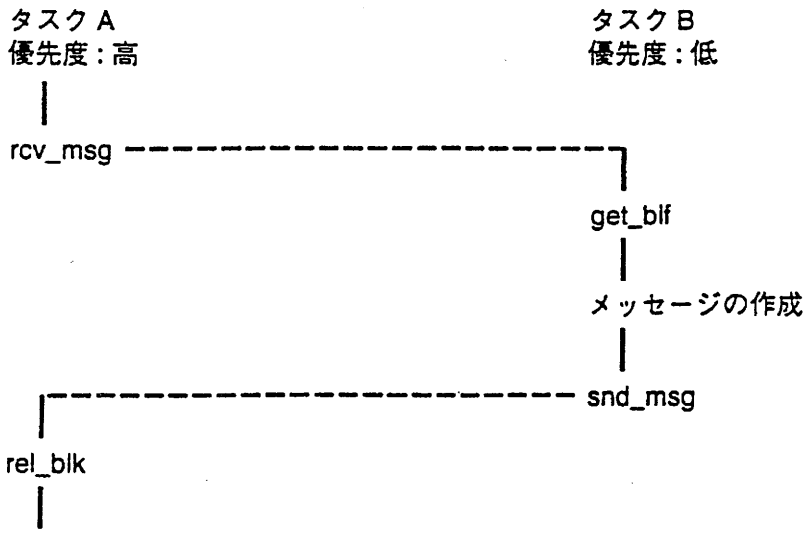
なお，タスクBはrun状態からready状態へと遷移します。

(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)～(7)の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

本章では，RX830 が行う割り込み管理機能について示しています。

5.1　概要

RX830 における割り込み管理では，

- 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

などといった処理が行われます。

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あり，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，RX830 では，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

  RX830 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

  RX830 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，RX830 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，RX830 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，待ちキューのキュー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，一括して行うようにしています。

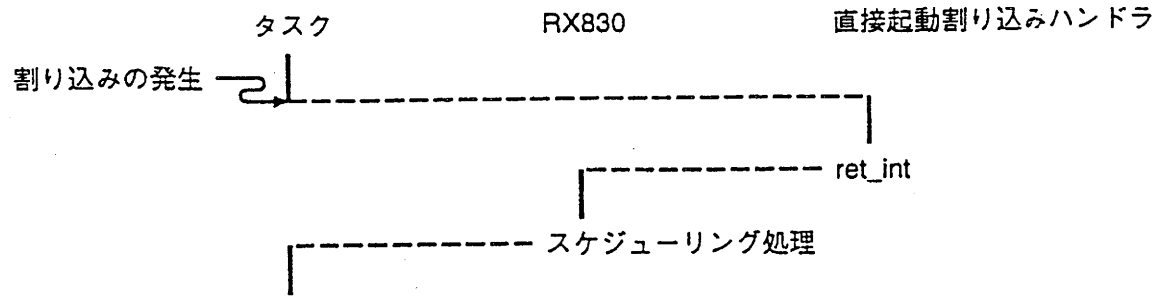
5.3 直接起動割り込みハンドラ

直接起動割り込みハンドラは，割り込みが発生した際，RX830 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，直接起動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。

ただし，NEC 製 V830 ファミリ ™C コンパイラ CA830 を使用し，`#pragma rtos_interrupt` 指令を用いて直接起動割り込みハンドラを登録した場合は，登録に関する処理を記述する必要がありません。

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は，割り込みが発生した際，RX830 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。

そこで，直接起動割り込みハンドラの処理を記述する際には，以下に示す注意事項があります。

(1) レジスタの退避／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。

ただし，NEC 製 V830 ファミリ ™C コンパイラ CA830 を使用し，`#pragma rtos_interrupt` 指令を用いて直接起動割り込みハンドラを登録した場合は，作業用レジスタの退避処理／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

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。

(3) システム・コールの発行制限

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，以下に示します。

- 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　ret_int　ret_wup　vret_clk　chg_ilv
  ref_ilv

- 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，直接起動割り込みハンドラの終了部分で，ret_int，ret_wup，または，vret_clk 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のち，直接起動割り込みハンドラから復帰します。

- vret_clk システム・コール

  システム・クロック処理を呼び出したのち，直接起動割り込みハンドラから復帰します。

なお，RX830 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) が発行された際には，待ちキューのキュー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，直接起動割り込みハンドラからの復帰処理 (ret_int，ret_wup，または，vret_clk システム・コールの発行) まで遅延され，一括して行うようにしています。

注意 ret_int，ret_wup，vret_clk 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) を行っていません。このため，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，本システム・コールを発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。

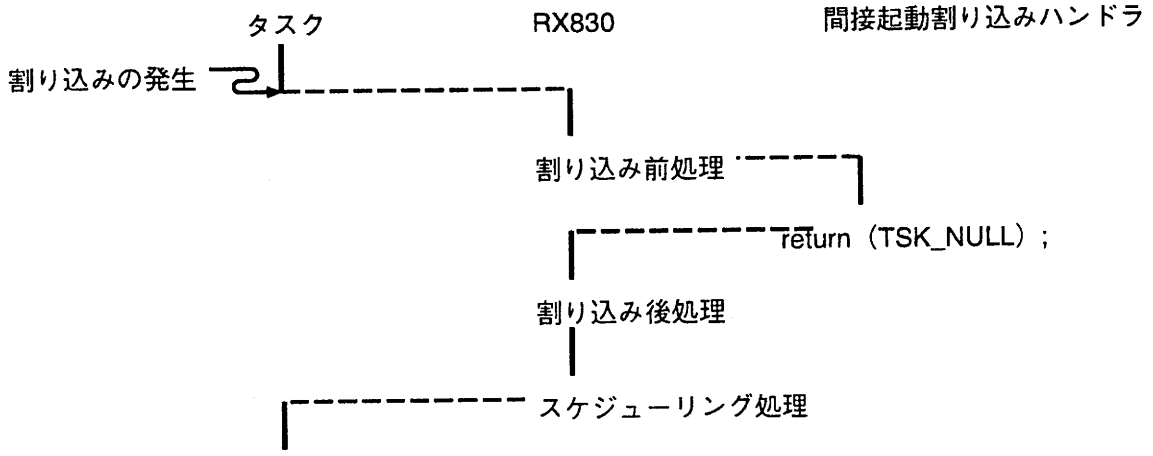
5.4 間接起動割り込みハンドラ

間接起動割り込みハンドラは，割り込みが発生した際，RX830 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，RX830 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

RX830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，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。

なお，RX830における間接起動割り込みハンドラの登録は，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・メモリ領域から確保したのち，初期化することです。

- 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することにより行われます。

  RX830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もと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。

- 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・コールを発行することにより行われます。

  RX830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もと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。

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は，割り込みが発生した際，RX830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)を行わせたのち起動される割り込み処理専用ルーチンです。

そこで，間接起動割り込みハンドラの処理を記述する際には，以下に示す注意事項があります。

(1) レジスタの退避／復帰

RX830では，間接起動割り込みハンドラに制御を移す際，および，間接起動割り込みハンドラから復帰する際，Cコンパイラ(CA830，または，CCV830)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，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。

注意　RX830では，間接起動割り込みハンドラに制御を移す際，gp，tpレジスタの切り替えを行っていません。

(2) スタックの切り替え

RX830では，間接起動割り込みハンドラに制御を移す際，および，間接起動割り込みハンドラから復帰する際，スタックの切り替え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が行われないため，割り込み発生時のスタックが使用されることになります。

(3) システム・コールの発行制限

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，以下に示します。

- 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` `chg_ilv` `ref_ilv`

- 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) 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，間接起動割り込み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のち，間接起動割り込みハンドラから復帰します。

- return (CLK_START) 命令

  システム・クロック処理を呼び出したのち，間接起動割り込みハンドラから復帰します。

なお，RX830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) が発行された際には，待ちキューのキュー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，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，一括して行うようにしています。

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) が行われません。このため，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，本命令を発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。

5.5　マスカブル割り込みの受け付け禁止／再開

RX830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。

なお，この機能は，以下に示したシステム・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実現されます。

- 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のち，ディスパッチ処理 (タスクのスケジューリング処理) も禁止します。

  これにより，本システム・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，他タスク，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のち，ディスパッチ処理 (タスクのスケジューリング処理) も再開します。

  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～図 5-4に，通常の場合，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 5-3　通常の場合

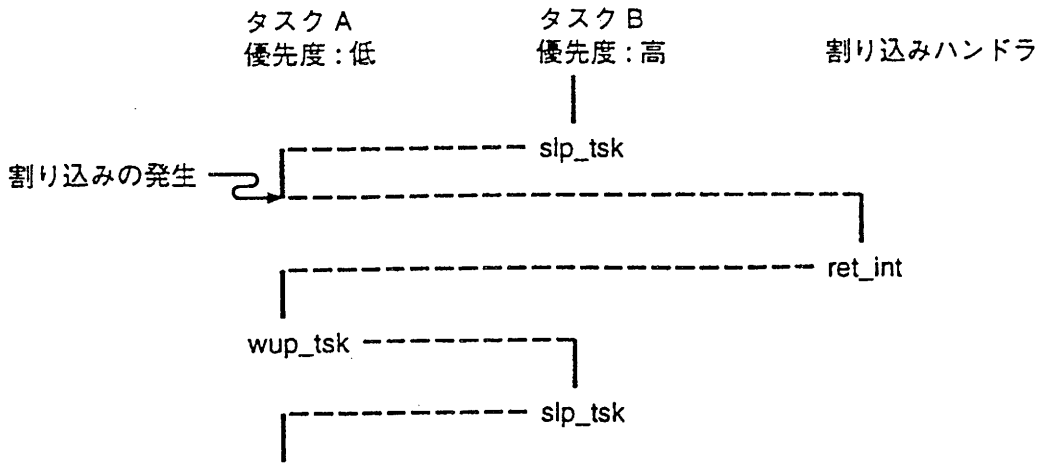
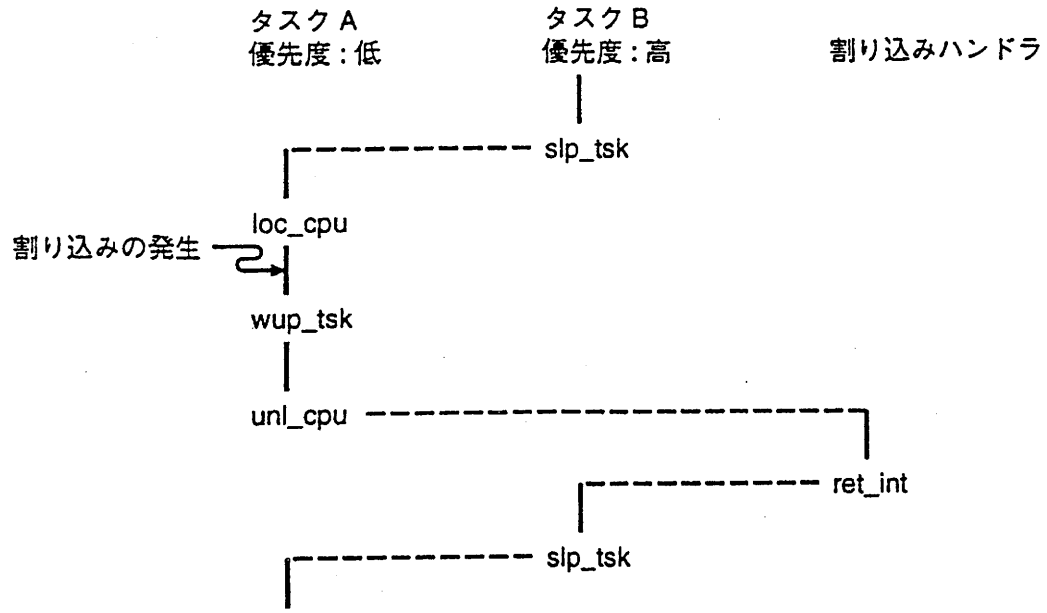


図 5-4　loc_cpu システム・コールを発行した場合



5.6 割り込み許可レベルの変更／獲得

割り込み許可レベルの内容の変更／獲得は，chg_ilv，または，ref_ilv システム・コールを発行することにより行われます。

- chg_ilv システム・コール

  プロセッサ (V830 ファミリ ™) の割り込み許可レベルをパラメータで指定された値に変更します。

- ref_ilv システム・コール

  プロセッサ (V830 ファミリ ™) の割り込み許可レベルを獲得します。

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状態 (PSW の ID フラグをセット) にしても受け付けられる割り込みです。

このため，RX830 の処理中，および，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ることになります。

そこで，RX830 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，その動作を保証していません。

5.8 クロック割り込み

RX830 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際には，RX830 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。

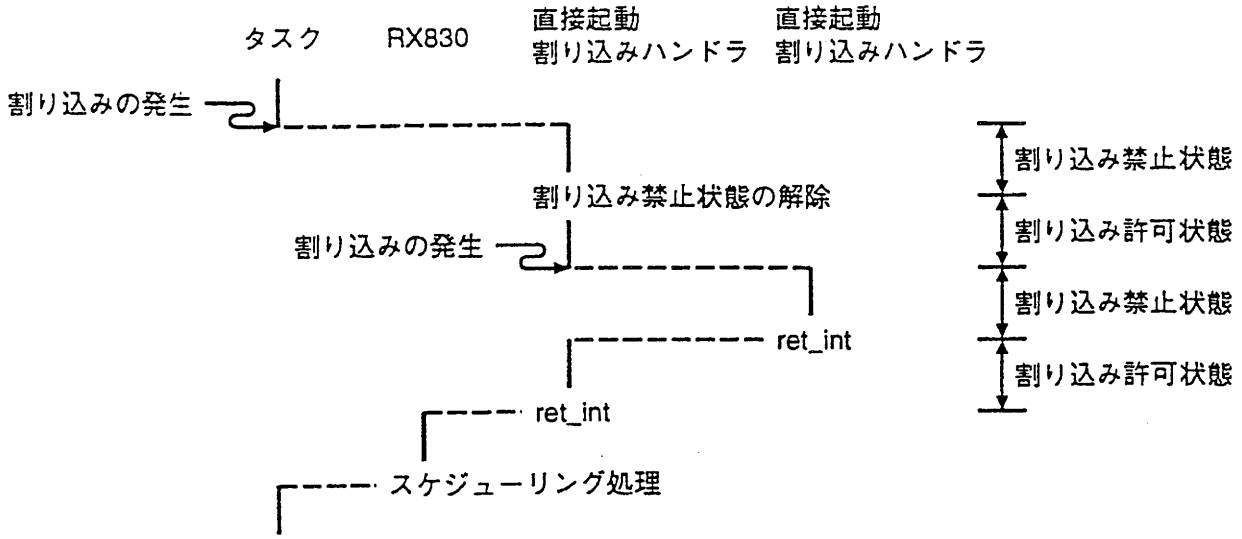
5.9 多重割り込み

RX830 では，割り込みハンドラ内で再び割り込みが発生することを，「多重割り込み」と呼びます。

ただし，割り込みハンドラは，割り込み禁止状態 (PSW の ID フラグをセット) でその処理が開始されるため，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



第 6 章　メモリ・プール管理機能

本章では，RX830 が行うメモリ・プール管理機能について示しています。

6.1　概要

RX830 では，システム全体を管理するための情報テーブル，各種機能（セマフォ，イベント・フラグなど）を実現するための管理ブロックなどといった管理オブジェクトをシステム初期化処理時にスタティックに生成／初期化しています。

また，RX830 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となった際に獲得し，不要となった際には解放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可能となります。

6.2　管理オブジェクト

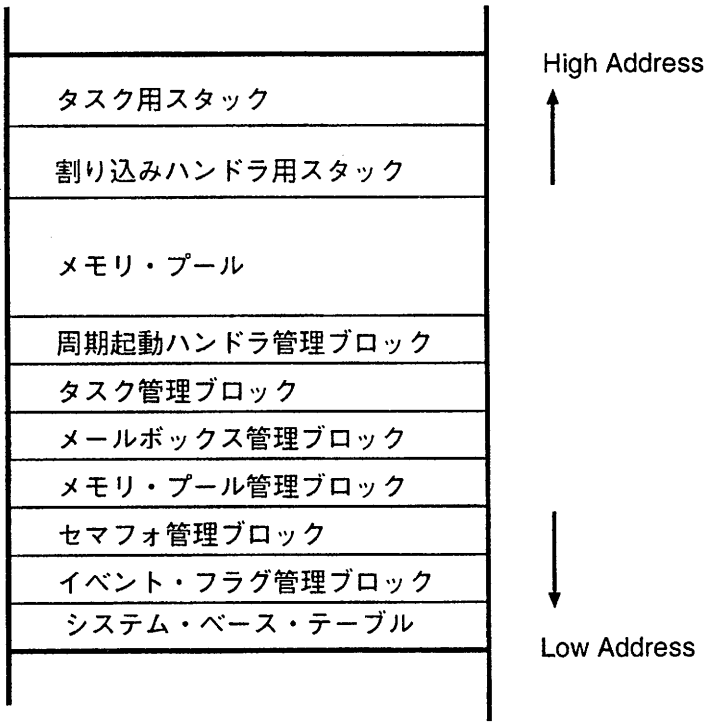
以下に，RX830 が提供する機能を実現するうえで必要となる管理オブジェクトを示します。

なお，これら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（タスク情報，セマフォ情報など）をもとに，システム初期化処理時に生成／初期化されます。

- システム・ベース・テーブル
- タスク管理ブロック
- タスク実行権管理ブロック
- 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
- 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

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／メモリ・ブロック

RX830では，ダイナミックなメモリ・プール管理も行っており，処理プログラム（タスク，ハンドラなど）からメモリ領域に対する要求が行われた際に使用する領域としてメモリ・プールを用意しています。

なお，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。

また，メモリ・プールに対するダイナミックな操作は，以下に示すメモリ・プール関連のシステム・コールを用いて行います。

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

注意　RX830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。

6.3.1 メモリ・プールの生成

RX830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。

なお，RX830におけるメモリ・プールの生成は，メモリ・プールを管理するための領域（管理オブジェクト），および，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のち，初期化することです。

- 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することにより行われます。

  RX830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もとにメモリ・プールの生成処理を行い，管理対象とします。

- 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・コールを発行することにより行われます。

  RX830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もとにメモリ・プールの生成処理を行い，管理対象とします。

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，対象はメモリ・プール，RX830 の管理対象から除外されます。

  ただし，本システム・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 状態へと遷移させています。

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

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われます。

- 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，本システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得することができなかった（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のち，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・ブロック待ち状態の解除は，以下の場合に行われ，メモリ・ブロック待ち状態から 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・ブロックを返却した
  - del_mpl システム・コールが発行され，対象メモリ・プールを削除した
  - rel_wai システム・コールが発行され，強制的に待ち状態を解除した

  注 1　RX830 では，メモリ・ブロックを獲得する際，メモリ・クリアを行っていません。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。

  注 2　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のキューイング方法は，対象メモリ・プール生成時（コンフィギュレーション時，cre_mpl システム・コール発行時）に指定された順（FIFO 順，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，本システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得することができなかった（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　RX830 では，メモリ・ブロックを獲得する際，メモリ・クリアを行っていません。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。

- 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，本システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得することができなかった（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のち，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・ブロック待ち状態の解除は，以下の場合に行われ，メモリ・ブロック待ち状態から 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・ブロックを返却した
  - del_mpl システム・コールが発行され，対象メモリ・プールを削除した
  - rel_wai システム・コールが発行され，強制的に待ち状態を解除した

  注 1　RX830 では，メモリ・ブロックを獲得する際，メモリ・クリアを行っていません。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。

  注 2　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のキューイング方法は，対象メモリ・プール生成時（コンフィギュレーション時，cre_mpl システム・コール発行時）に指定された順（FIFO 順，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

  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，，本システム・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（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１　RX830 では，メモリ・ブロックを返却する際，メモリ・クリアを行っていません。したがって，返却したメモリ・ブロックの内容は不定となります。

注２　メモリ・ブロックの返却は，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・プールと同一でなければなりません。

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。

  以下に，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。

第 7 章　時間管理機能

本章では，RX830 が行う時間管理機能について示しています。

7.1　概要

RX830 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。

つまり，クロック割り込みが発生した際には，RX830 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，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。

注意　RX830 のシステム・クロック処理は，ユーザが記述したクロック割り込み用割り込みハンドラから vret_clk システム・コールを発行することにより呼び出すことができます。

7.2　システム・クロック

システム・クロックは，RX830 が時間管理を行う際に使用する時刻（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。

なお，システム・クロックは，システム初期化処理で「0x0」に設定されたのち，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　RX830 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，RX830 では，桁あふれした数値（48 ビットでは表せない数値）については無視しているので注意が必要です。

7.2.1　システム・クロックの設定／読み出し

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

- 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，RX830 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。

7.4 タスクの遅延起床

タスクの遅延起床は，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，時間が経過した際には，wait 状態を解除し，ready 状態へと遷移させるものです。

なお，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われます。

- 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。

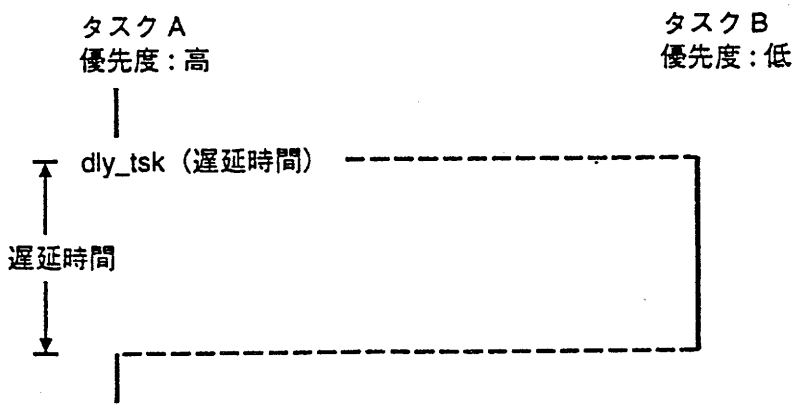
  なお，時間経過待ち状態の解除は，以下の場合に行われ，時間経過待ち状態から ready 状態へと遷移します。

  – 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，強制的に待ち状態を解除した

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，時間が経過した際には，wait 状態を解除し，ready 状態へと遷移させるものです。

なお，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。

- 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，本システム・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，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から 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
  - 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，強制的に待ち状態を解除した

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

  なお，資源待ち状態の解除は，以下の場合に行われ，資源待ち状態から 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
  - 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，対象セマフォを削除した
  - rel_wai システム・コールが発行され，強制的に待ち状態を解除した

- 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否かをチェックします。

  ただし，本システム・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のち，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・フラグ待ち状態の解除は，以下の場合に行われ，イベント・フラグ待ち状態から 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
  - set_flg システム・コールが発行され，要求する待ち条件をセットした
  - del_flg システム・コールが発行され，対象イベント・フラグを削除した
  - rel_wai システム・コールが発行され，強制的に待ち状態を解除した

- 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，本システム・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することができなかった（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のち，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の解除は，以下の場合に行われ，メッセージ待ち状態から 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
  - 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を削除した
  - rel_wai システム・コールが発行され，強制的に待ち状態を解除した

- 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，本システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得することができなかった（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のち，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・ブロック待ち状態の解除は，以下の場合に行われ，メモリ・ブロック待ち状態から 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，要求するサイズのメモリ・ブロックを返却した
- del_mpl システム・コールが発行され，対象メモリ・プールを削除した
- rel_wai システム・コールが発行され，強制的に待ち状態を解除した

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な処理プログラムです。

また，周期起動ハンドラに対するダイナミックな操作は，以下に示す周期起動ハンドラ関連のシステム・コールを用いて行います。

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
rcv_msg ： メッセージを受信する

7.6.1 周期起動ハンドラの登録

RX830 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。

なお，RX830 における周期起動ハンドラの登録は，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のち，初期化することです。

- 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することにより行われます。

  RX830 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もと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。

- 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・コールを発行することにより行われます。

  RX830 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もと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，RX830 が周期起動ハンドラを起動するか否かを判定する際の基準の 1 つです。

なお，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／def_cyc システム・コール発行時）に設定されますが，RX830 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。

RX830 では，周期起動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／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。そこで，1 回目の起動要求が周期起動ハンドラ登録時に指定した起動時間間隔で発行されるためには，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 を併せて指定する必要があります。

図 7-2～図 7-3に，処理プログラムから act_cyc システム・コールを発行することにより，周期起動ハンドラの活性状態を OFF 状態から ON 状態に変更した際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周期起動ハンドラの起動時間間隔であるものとし，図 7-2における Δt と ΔT の関係は $\Delta t \leq \Delta T$ であるものとします。

図 7-2　act_cyc(TCY_ON) 発行時の処理の流れ

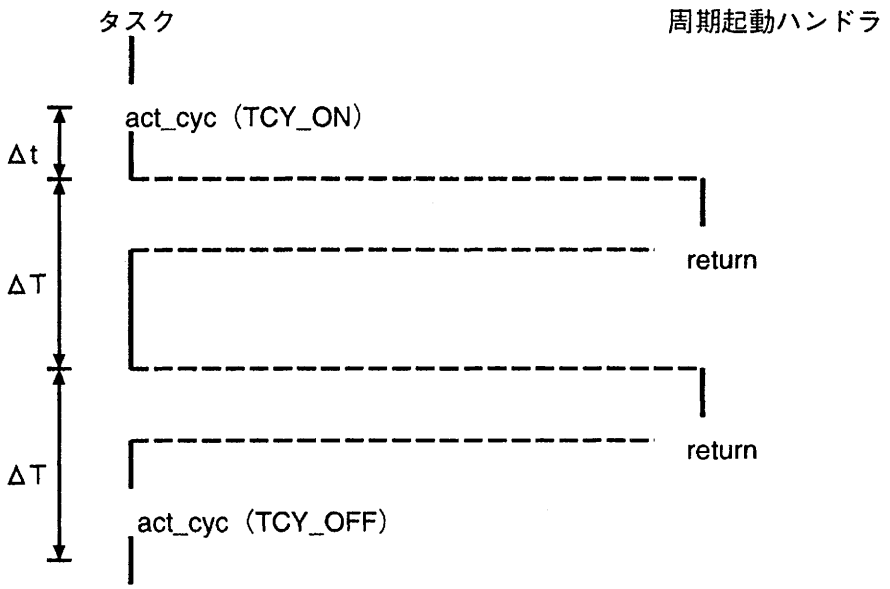
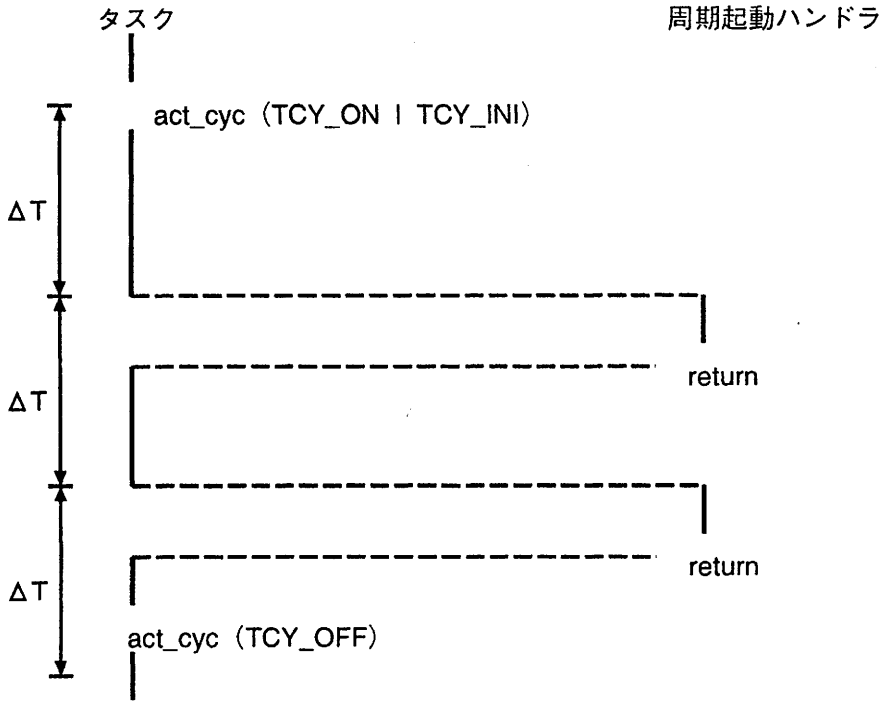


図 7-3　act_cyc(TCY_ON | TCY_INI) 発行時の処理の流れ



7.6.3 周期起動ハンドラ内での処理

RX830では，クロック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。

このため，周期起動ハンドラの処理を記述する際には，以下に示す注意事項があります。

(1) レジスタの退避／復帰

RX830では，周期起動ハンドラに制御を移す際，および，周期起動ハンドラから復帰する際，Cコンパイラ（CA830，または，CCV830）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，周期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

RX830では，周期起動ハンドラに制御を移す際，および，周期起動ハンドラから復帰する際，スタックの切り替え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せん。

注意　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，スタックの切り替え処理が行われないため，割り込み発生時のスタックが使用されることになります。

(3) システム・コールの発行制限

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，以下に示します。

- 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	chg_ilv	ref_ilv

- 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，周期起動ハンドラの終了部分で，**return** とすることにより行われます。

なお，RX830 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) が発行された際には，待ちキューのキュー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，周期起動ハンドラからの復帰処理 (**return**) まで遅延され，一括して行うようにしています。

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。

  以下に，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

第 8 章　スケジューラ

本章では，RX830 が行うスケジューリング処理について示しています。

8.1　概要

RX830 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ています。

8.2　駆動方式

RX830 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，RX830 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。

したがって，ユーザの処理プログラム内からタスクの状態遷移を引き起こす可能性のあるシステム・コールが発行された際，ハンドラからの復帰命令が発行された際，および，クロック割り込みが発生した際には，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。

以下に，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

RX830 のスケジューリング方式は，優先度方式，および，FCFS 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，および，ready 状態) にあるタスクの優先度を参照し，その中から最適なタスクを選び出し，プロセッサの使用権を与えています。

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，および，ready 状態) にあるタスクの優先度を参照し，その中から最も高い優先度 (最高優先度) を持つタスクを選び出し，プロセッサの使用権を与えています。

注意 RX830 におけるタスクの優先度は，その値が小さいほど，高い優先度を意味します。

8.3.2 FCFS (First Come First Service) 方式

RX830 では，同一の優先度を複数のタスクに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，先に実行可能状態となったタスク (ready 状態と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ています。

8.4 ラウンドロビン方式の実現

RX830では，優先度方式，および，FCFS方式のスケジューリングを行っていますが，この方式では，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獲得したタスクが他の状態へ遷移，または，プロセッサの使用権を放棄しない限り，他タスクが「同一の優先度でありながらも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，RX830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などといったシステム・コールを提供しています。

以下に，ラウンドロビン方式の実現例を示します。

（前提条件）

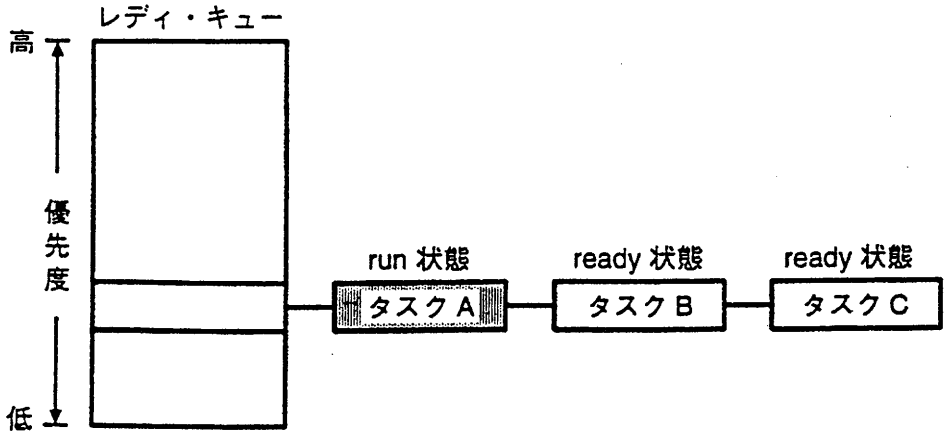
- タスクの優先度
  タスクA ＝ タスクB ＝ タスクC
- タスクの状態
  タスクA ： run状態
  タスクB ： ready状態
  タスクC ： ready状態
- 周期起動ハンドラの属性
  活性状態　　： ON状態
  起動時間間隔 ： ΔT（単位：基本クロック周期）
  処理内容　　： レディ・キューの回転（rot_rdqシステム・コールの発行）

(1) 現在，タスクAが処理を実行しています。

なお，他タスク（タスクB，タスクC）もタスクAと同一の優先度を持ちますが，タスクAが他の状態へ遷移，または，プロセッサの使用権を放棄しない限り，処理を実行することはできません。

図8-1に，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

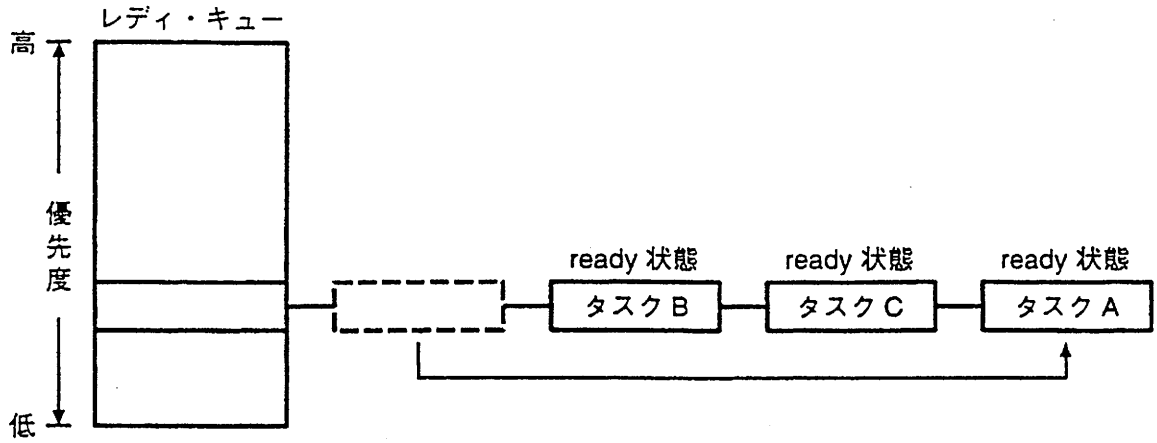


(2) 時間の経過により周期起動ハンドラが起動し，rot_rdq システム・コールを発行します。

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，レディ・キューの状態を示します。

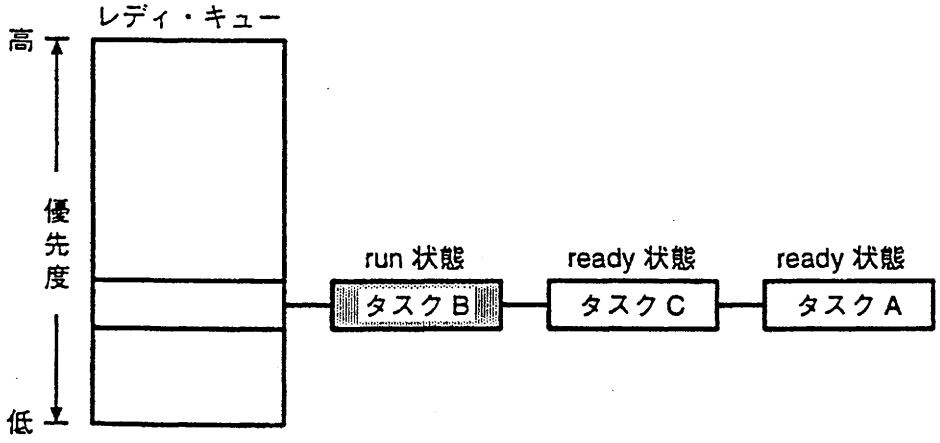
図 8-2　レディ・キューの状態



(3) タスク A が ready 状態へと遷移したことにともない，タスク B が ready 状態から run 状態へと遷移します。

図 8-3に，レディ・キューの状態を示します。

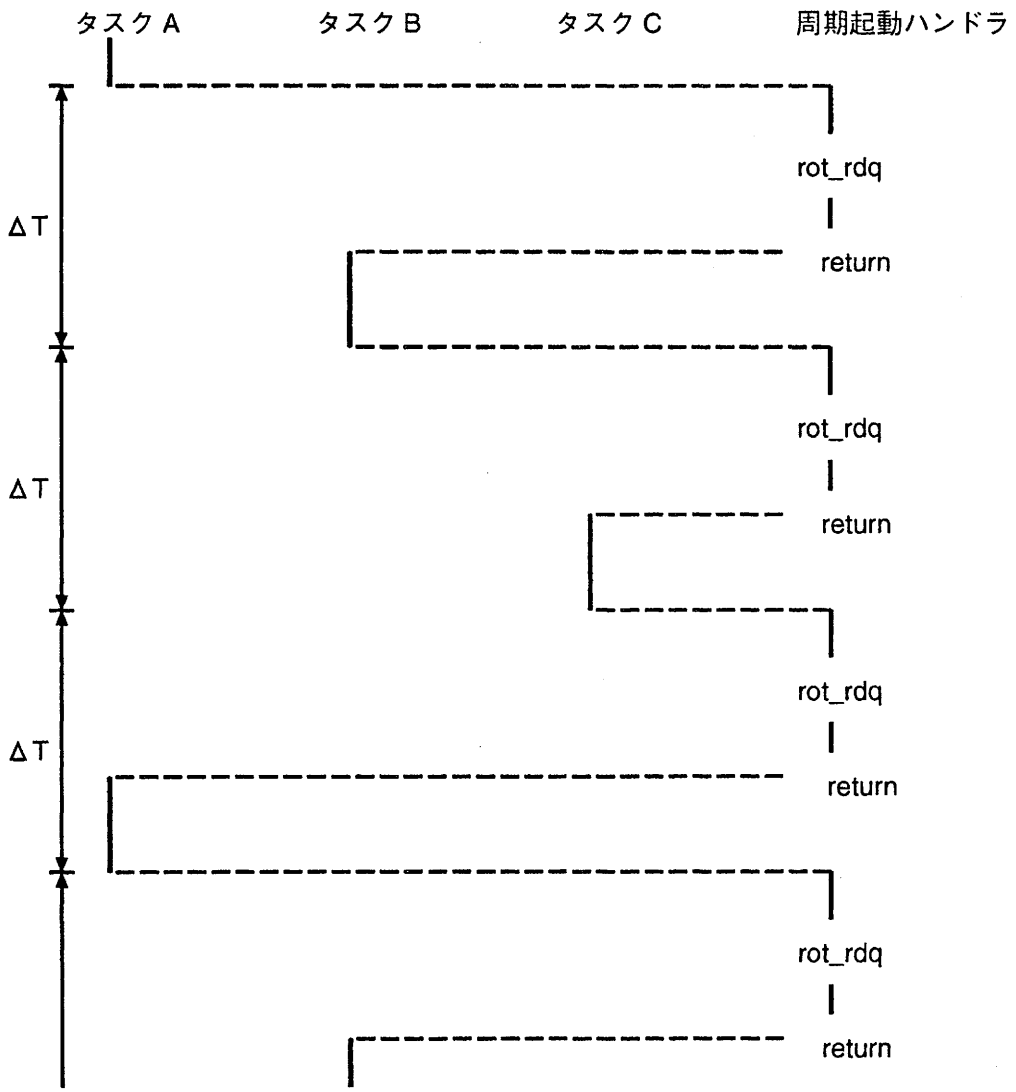
図 8-3　レディ・キューの状態



(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内から rot_rdq システム・コールを発行することにより，一定の時間（ΔT）が経過した際には，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）を実現することができます。

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

RX830では，ユーザの処理プログラム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，この機能は，以下に示したシステム・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，本システム・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，他タスクに制御が移ることはありません。

- 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。

  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。

  なお，RX830では，dis_dsp システム・コールの発行から本システム・コールが発行されるまでの間に，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，待ちキューのキュー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，本システム・コールが発行されるまで遅延され，一括して行うようにしています。

- 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のち，ディスパッチ処理（タスクのスケジューリング処理）も禁止します。

  これにより，本システム・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，他タスク，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のち，ディスパッチ処理（タスクのスケジューリング処理）も再開します。

  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。

  なお，RX830では，loc_cpu システム・コールの発行から本システム・コールが発行されるまでの間に，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，本システム・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，本システム・コールが発行されるまで遅延され，一括して行うようにしています。

図 8-5～図 8-7に，通常の場合，dis_dsp システム・コールを発行した場合，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　通常の場合

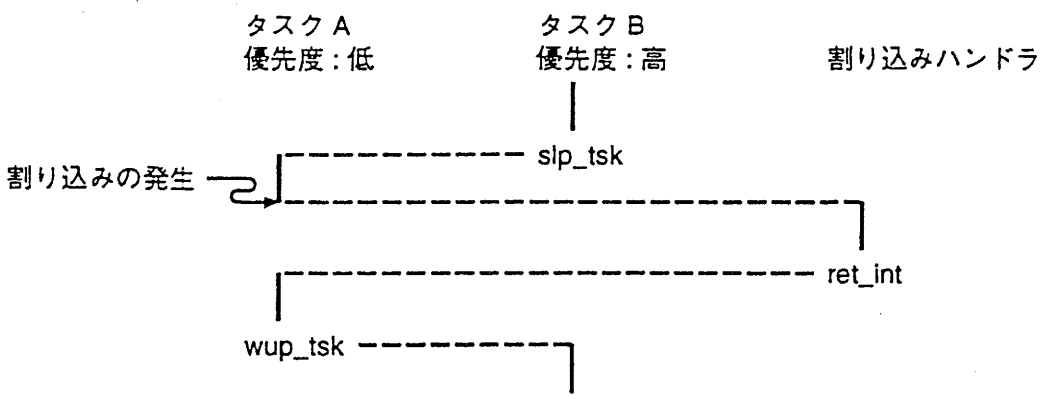


図 8-6　dis_dsp システム・コールを発行した場合

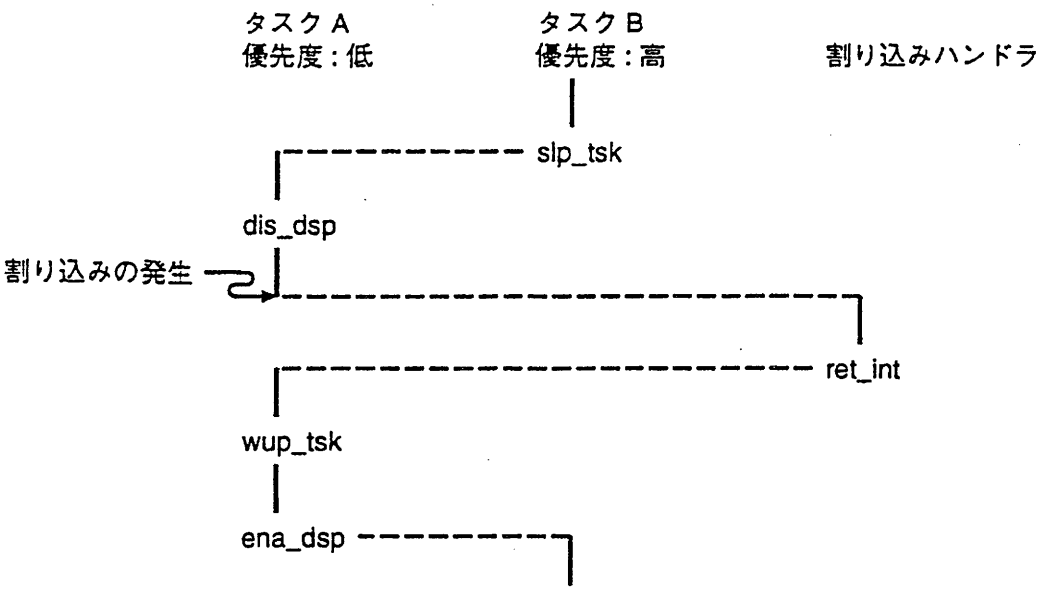
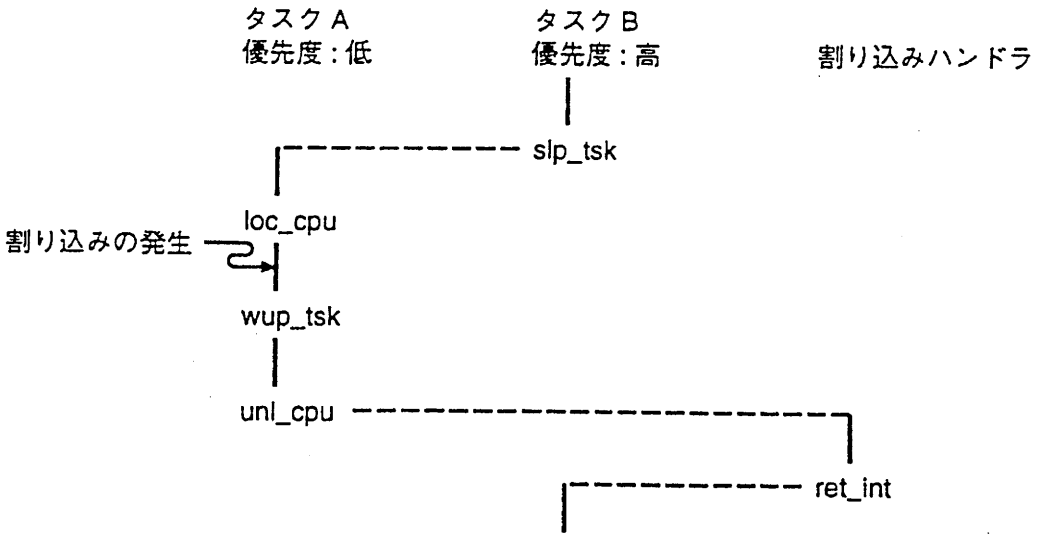


図 8-7　loc_cpu システム・コールを発行した場合



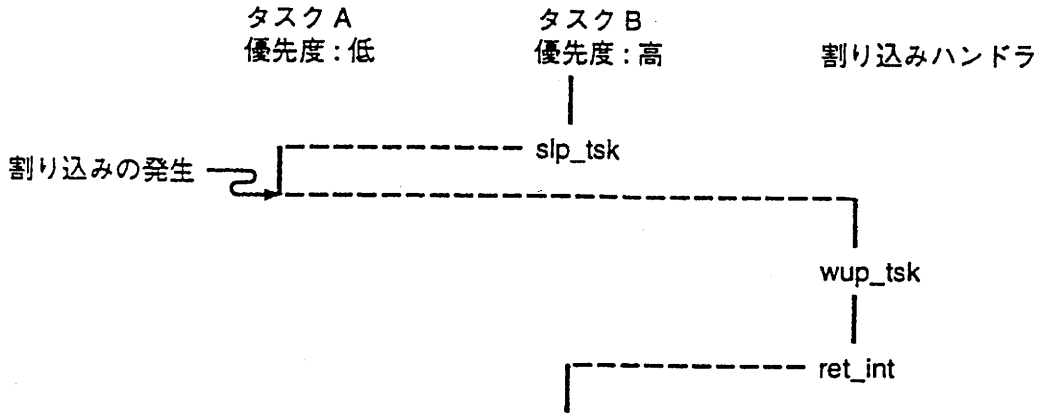
8.6　ハンドラ内でのスケジューリング

RX830 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ています。

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，ハンドラからの復帰処理（ret_int，ret_tmr など）まで遅延され，一括して行うようにしています。

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。

図 8-8　wup_tsk システム・コールを発行した場合



第 9 章　システム初期化処理

本章では，RX830 が行うシステム初期化処理について示しています。

なお，システム初期化処理についての詳細は，「RX830 ニュークリアス インストレーション編」を参照してください。

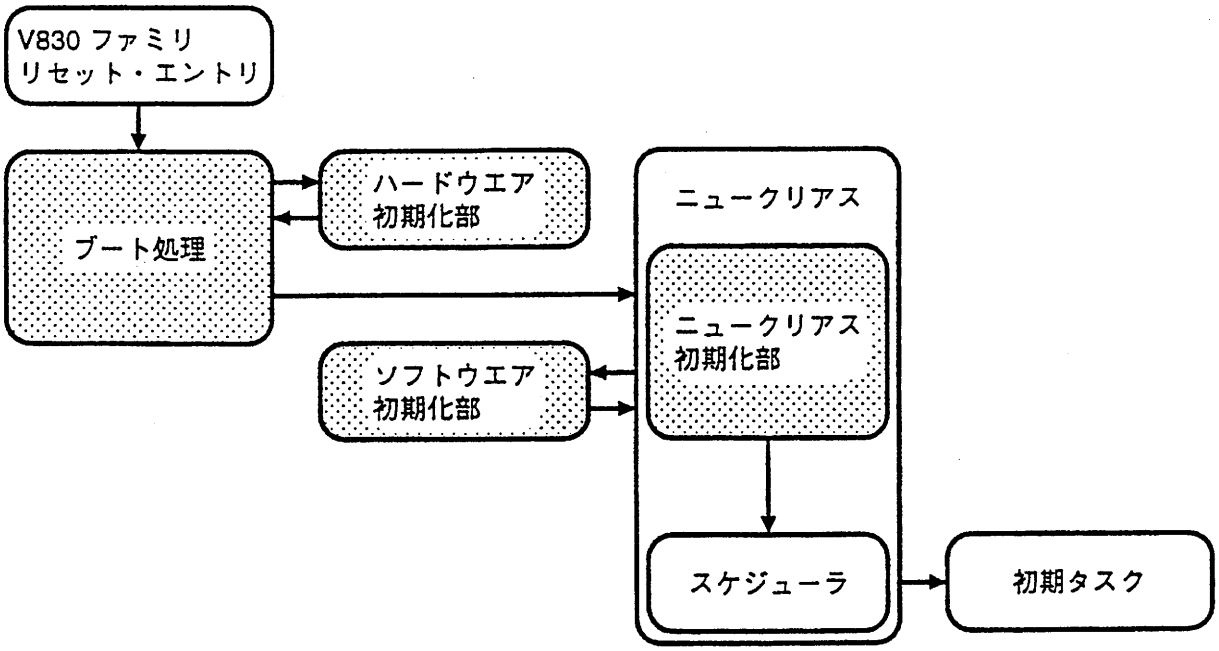
9.1　概要

システム初期化処理は，RX830 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，および，ソフトウエアの初期化処理から構成されています。

したがって，RX830 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。

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



9.2　ブート処理

ブート処理は，V830 ファミリ™ のリセット・エントリ（ハンドラ・アドレス：0xffffffff0）に割り付けられた関数であり，システム初期化処理の中でも，まず最初に処理が実行されるものです。

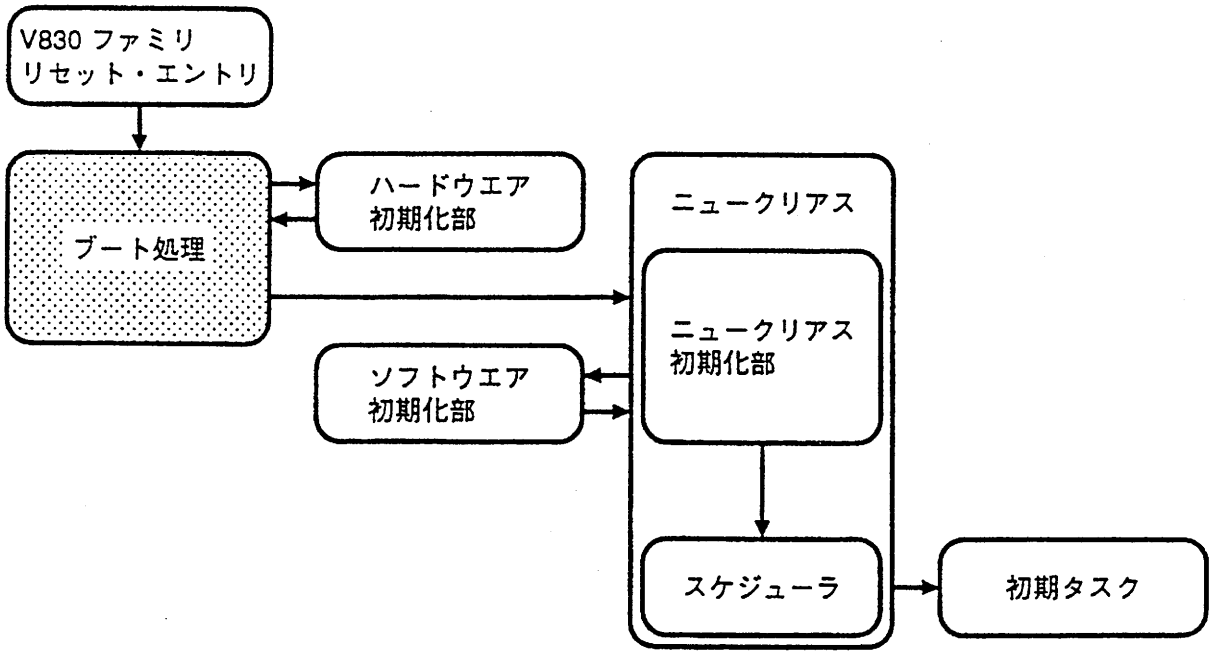
なお，ブート処理では，以下のような処理を行います。

- gp，tp 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

注意　標準添付のブート処理は，その処理内容を，ユーザのニーズにあわせて書き替えることができます。

図 9-2に，ブート処理の位置付けを示します。

図 9-2　ブート処理の位置付け



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。
なお，ハードウエア初期化部では，以下のような処理を行います。

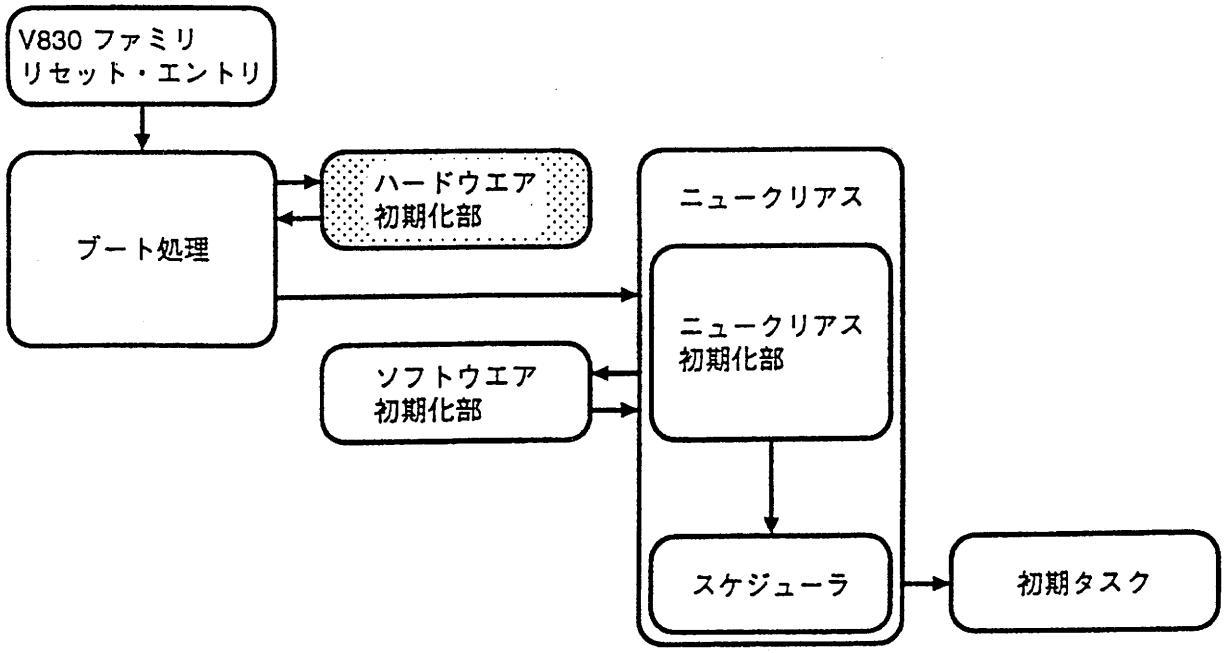
- 内部ユニットの初期化
  - 割り込みコントローラの初期化
  - クロック・コントローラの初期化
- 周辺コントローラの初期化
- ブート処理に制御を戻す

ハードウエア初期化部は，実行環境のハードウエア構成に依存するため，サンプル・ソース・ファイルを提供しています。
これにより，様々なターゲット・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。

注意　標準添付のハードウエア初期化部は，ユーザの実行環境として，（株）電産製 V830 ファミリ ™ ボード [DVE-V830/20] を想定した処理が記述してあります。
このため，ユーザの実行環境が標準構成でない場合や，周辺コントローラのモード設定を変更する場合には，ハードウエア初期化部を書き替える必要があります。

図 9-3に，ハードウエア初期化部の位置付けを示します。

図 9-3　ハードウエア初期化部の位置付け



9.4　ニュークリアス初期化部

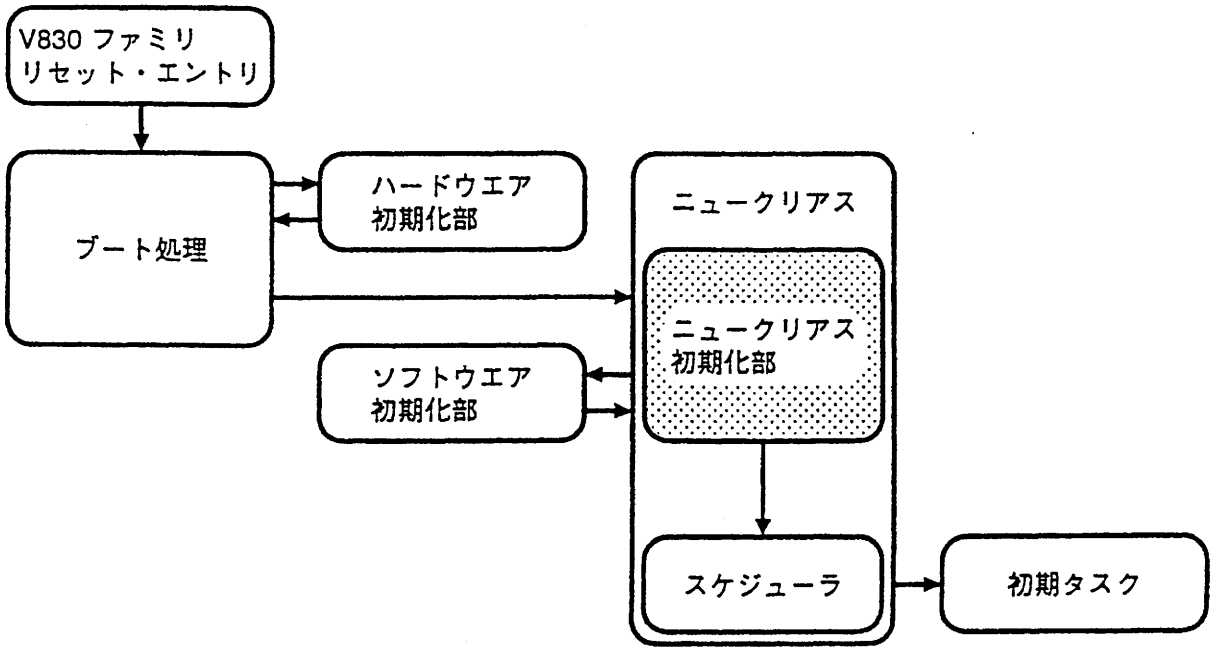
ニュークリアス初期化部は，ブート処理から呼び出される関数であり，情報ファイル（システム情報テーブル，システム情報ヘッダ・ファイル）に記述された情報（タスク情報，セマフォ情報など）をもとに，各種管理オブジェクトを生成／初期化するために用意されたものです。

なお，ニュークリアス初期化部では，以下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を呼び出す
- スケジューラに制御を移す

図 9-4に，ニュークリアス初期化部の位置付けを示します。

図 9-4　ニュークリアス初期化部の位置付け



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あり，ユーザのソフトウエア環境を快適なものとするために用意されたものです。

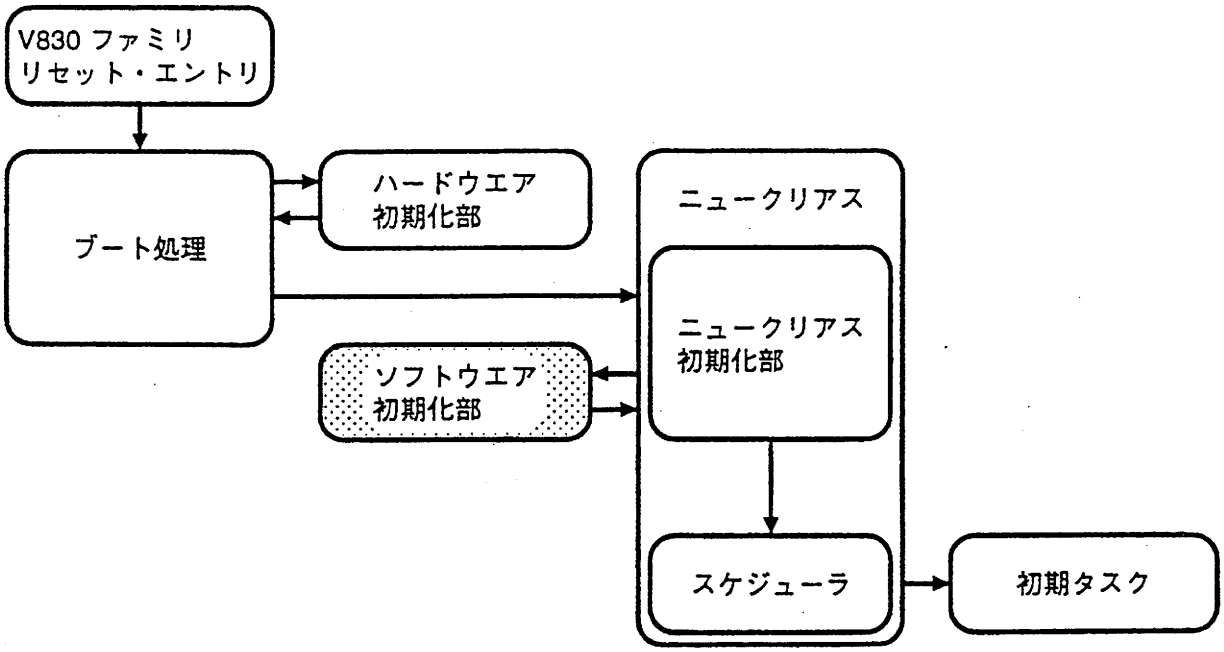
なお，ソフトウエア初期化部では，以下のような処理を行います。

- 初期化データのコピー
- クロック割り込みの許可
- ニュークリアス初期化部に制御を戻す

注意　標準添付のソフトウエア初期化部は，その処理内容を，ユーザのニーズにあわせて書き替えることができます。

図 9-5に，ソフトウエア初期化部の位置付けを示します。

図 9-5　ソフトウエア初期化部の位置付け



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示しています。

なお，インタフェース・ライブラリについての詳細は，「RX830 ユーザーズ・マニュアル インストレーション編」を参照してください。

10.1　概要

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，および，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，外部関数形式で発行されたシステム・コール，または，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インタフェース・ライブラリが必要となります。

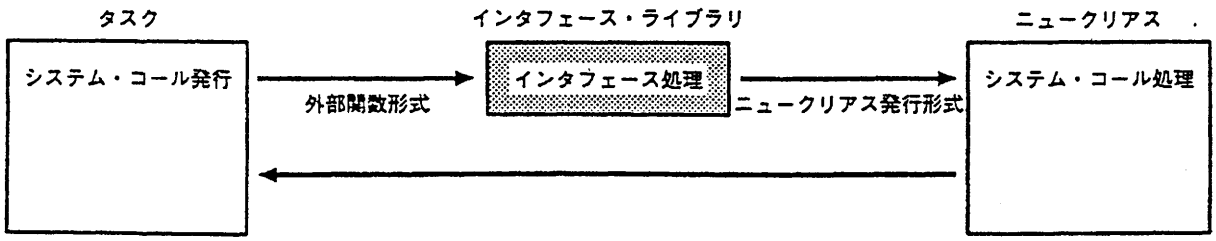
つまり，インタフェース・ライブラリは，処理プログラムとニュークリアスの中間に位置し，ニュークリアスが対応した処理を行ううえで必要となる各種情報を設定したのち，ニュークリアスに制御を移す機能を持ちます。

なお，RX830が提供するインタフェース・ライブラリは，NEC製V830ファミリ™用Ｃコンパイラ CA830用，および，米国Green Hills Software, Inc.製Ｃクロス V800™ コンパイラ CCV830用の2種類が用意されています。

また，RX830が提供するシステム・コール用インタフェース・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インタフェース・ライブラリ（パラメータ・チェック機能あり）と，システム・コールのパラメータをチェックしないインタフェース・ライブラリ（パラメータ・チェック機能なし）の2種類があり，用途に応じて使い分けることができます。

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



第 11 章　システム・コール

本章では，RX830 が提供するシステム・コールについて示しています。

11.1　概要

システム・コールは，ユーザの処理プログラム（タスク，非タスク）から RX830 のサービス・ルーチンを呼び出すための手続き／機能であり，RX830 が直接管理している資源（カウンタ，待ちキューなど）を間接的に操作することができます。

RX830 では，μITRON3.0 仕様で規定されているシステム・コール（66 種類）のほかに，RX830 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，これらシステム・コールは，その機能により，以下に示す 7 グループに分類することができます。

(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。

なお，このグループには，タスクを生成／起動／終了／削除する機能，ディスパッチ処理を禁止／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などが含まれます。

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。

なお，このグループには，タスクを suspend 状態に移行する／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／起床待ち状態のタスクを起床する機能，タスクの起床要求を無効化する機能などが含まれます。

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），および，通信を行うシステム・コールのグループです。

なお，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などが含まれます。

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　8 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。

なお，このグループには，間接起動割り込みハンドラを登録／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／参照する機能などが含まれます。

def_int	ret_int	ret_wup	vret_clk	loc_cpu
unl_cpu	chg_ilv	ref_ilv		

(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。

なお，このグループには，メモリ・プールを生成／削除する機能，メモリ・ブロックを獲得／返却する機能，メモリ・プールの状態を参照する機能などが含まれます。

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　7 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。

なお，このグループには，システム・クロックに時刻を設定する／システム・クロックの時刻を参照する機能，タスクを時間経過待ち状態に移行する機能，周期起動ハンドラを登録／登録解除する機能，周期起動ハンドラの活性状態を制御する機能，周期起動ハンドラから復帰する機能，周期起動ハンドラの状態を参照する機能などが含まれます。

set_tim	get_tim	dly_tsk	def_cyc	act_cyc
ref_cyc	ret_tmr			

(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。

なお，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／登録解除する機能，拡張 SVC ハンドラを呼び出す機能などが含まれます。

get_ver	ref_sys	def_svc	viss_svc

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を行い，そのパラメータは引き数として渡されます。

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，および，戻り番地の設定を行ったのち，jal命令により呼び出しを行います。

注意 RX830では，システム・コールのプロト・タイプ宣言をstdrx.hファイルで行っています。このため，処理プログラムからシステム・コールを発行する際には，本ヘッダ・ファイルのインクルード処理

```
#include  <stdrx.h>
```

を記述する必要があります。

11.3 システム・コールの機能コード

RX830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の割り当てが行われています。

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，RX830では，1以上の値については，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	意　　味
-256 ～ -225	RX830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

11.4 パラメータのデータ・タイプ

RX830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	符号無し8ビット整数
UH	unsigned short	符号無し16ビット整数
UINT	unsigned int	符号無し32ビット整数
UW	unsigned long	符号無し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	オブジェクトの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

11.5 パラメータの値域

RX830 が提供するシステム・コールのパラメータには，指定可能な値として範囲があるもの，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 ～ 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 ～ 注 1
拡張 SVC ハンドラの拡張機能コード	0x1 ～ 注 1
オブジェクトの優先度	0x1 ～ 注 1
セマフォの最大資源数	0x1 ～ 注 1
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 ～ 0x7fffffffffff
待ち時間	-0x1 ～ 0x7fffffff
遅延時間	0x0 ～ 0x7fffffff
周期起動ハンドラの起動時間間隔	0x1 ～ 0x7fffffff
タスクのスタック・サイズ	0x0 ～ 0x7fffffff
メモリ・プールのサイズ	0x1 ～ 0x7fffffff
メモリ・ブロックのサイズ	0x1 ～ 0x7fffffff
メッセージの優先度	0x1 ～ 0x7fff

注 1　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となります。

注 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。

11.6　システム・コールからの戻り値

RX830 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_NOMEM	-10	オブジェクト用の領域が確保できない
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	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

RX830 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 (RX830 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「タスクからのみ発行可」に制限されます。

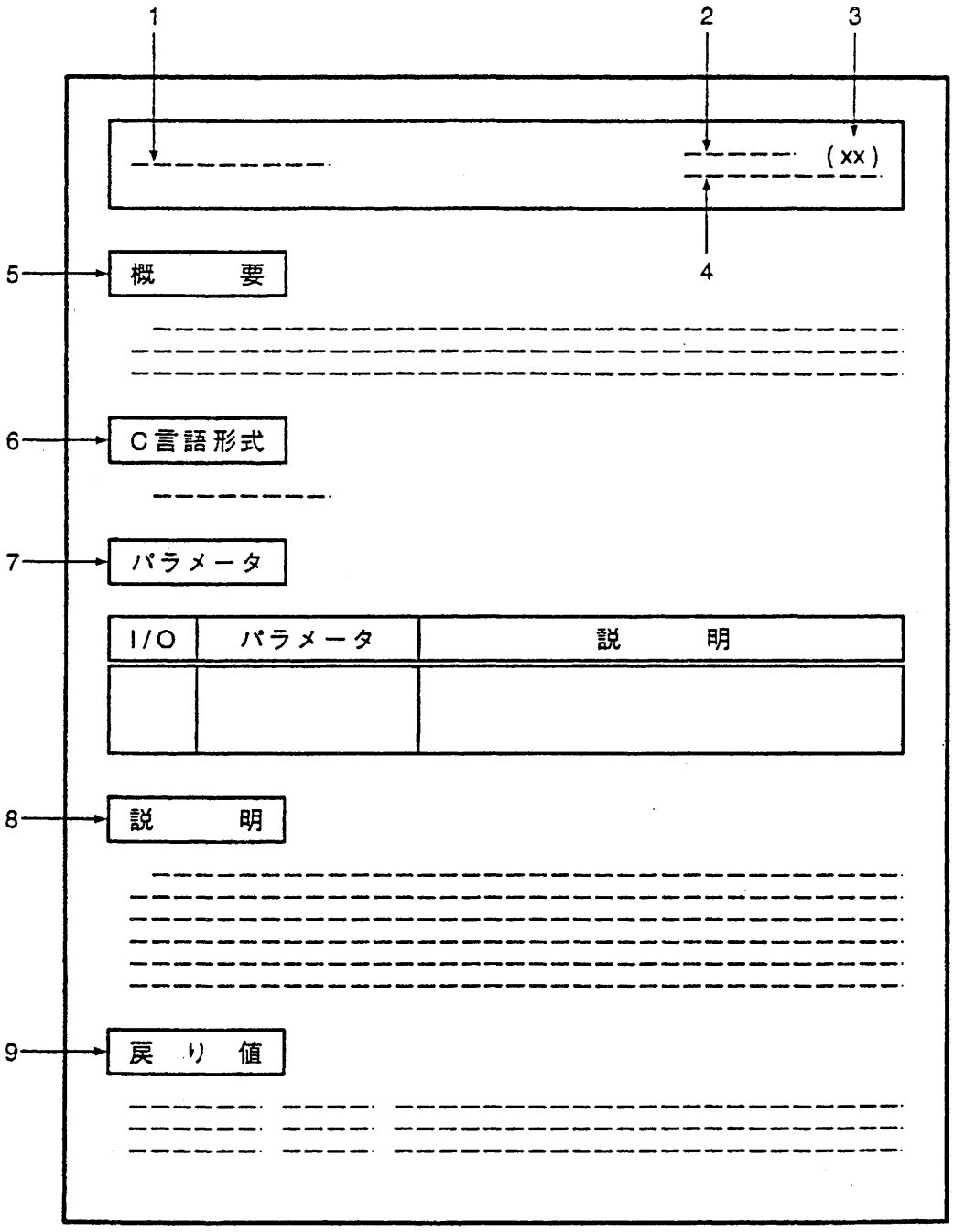
また，拡張システム・コールは，ユーザ定義のシステム・コールとして位置付けられる一方で，タスクに準じた部分を有します。つまり，標準システム・コールと同様に，処理が終了した際にはスケジューラが起動され，最適なタスクの選択処理が行われます。

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るため注意が必要です。

11.8　システム・コール解説

次項から，RX830 が提供するシステム・コールについて，以下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



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・コールの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／非タスク	：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）のどちらからも発行可能
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
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，以下の形式で示しています。

I/O	パラメータ	説　　明
A	B	C

A ：パラメータの種類

I ･･･ RX830への入力パラメータ

O ･･･ RX830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RX830で返す値
＊無しの戻り値 ： “パラメータ・チェック機能あり”のRX830でのみ返す値

保守／廃止

11.8.1　タスク管理機能システム・コール

本項では，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）について説明しています。

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のち，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

Creat Task (-17)

cre_tsk

タスク

概　要

タスクを生成する。

C言語式形

ID番号を指定する場合

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

ID番号を指定しない場合

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

パラメータ

I/O	パラメータ	説　明
I	ID　*tskid*;	タスクのID番号
I	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
O	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;
```

説　明

RX830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もとに，*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，RX830 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もとに，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，RX830 の管理対象となります。

  なお，ID 番号の割り付けは RX830 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。

以下に，タスク生成情報の詳細を示します。

exinf ･･･ 拡張情報

exinf は，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あり，ユーザが自由に利用することができます。

なお，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することができます。

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

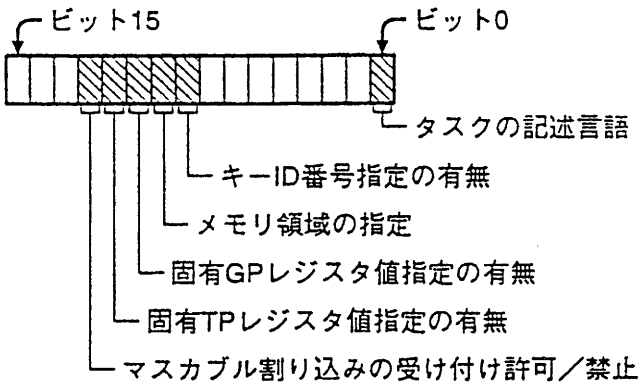
ビット 11 ･･ 固有 TP レジスタ値指定の有無

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の許可／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



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

注 1 tskatr のビット 8 の値が TA_KEYID 以外の場合，keyid の内容は意味をもちません。
注 2 tskatr のビット 10 の値が TA_DPID 以外の場合，gp の内容は意味をもちません。
注 3 tskatr のビット 11 の値が TA_DPIC 以外の場合，tp の内容は意味をもちません。

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_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，itskpri > 最大優先度）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 番号無指定生成時
E_ID	-35	ID 番号の指定が不正（*tskid* > 最大タスク生成数）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	非タスクから本システム・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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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，RX830 の管理下から除外されます。

注意　本システム・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 を返しています。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID番号の指定が不正（*tskid* > 最大タスク生成数）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	非タスクから本システム・コールを発行した

Start Task (-23)

sta_tsk

タスク／非タスク

概　要

他タスクを起動する。

C言語式形

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

パラメータ

I/O	パラメータ	説　明
I	ID　*tskid*;	タスクのID番号
I	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，RX830 のスケジューリング対象となります。

なお，*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　本システム・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 を返しています。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID 番号の指定が不正（*tskid* > 最大タスク生成数）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言語式形

```
#include  <stdrx.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，RX830 のスケジューリング対象から除外されます。

注１　本システム・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，動作の保証はありません。

注２　本システム・コールでは，タスク生成時（コンフィギュレーション時，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”の初期化が行われます。

注３　本システム・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（セマフォ・カウント，メモリ・ブロックなど）の解放処理が行われません。したがって，本システム・コールを発行する前に資源の解放処理を行うことは，ユーザの責任となります。

注４　タスクをアセンブリ言語で記述した場合，自タスクの終了は，以下のように記述します。

```
jr      _ext_tsk
```

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のち，削除する。

C言語式形

```
#include  <stdrx.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，RX830 の管理下から除外されます。

注1　本システム・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，動作の保証はありません。

注2　本システム・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（セマフォ・カウント，メモリ・ブロックなど）の解放処理が行われません。したがって，本システム・コールを発行する前に資源の解放処理を行うことは，ユーザの責任となります。

注3　タスクをアセンブリ言語で記述した場合，自タスクの終了，および，削除は，以下のように記述します。

```
jr      _exd_tsk
```

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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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。

注1　本システム・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 を返しています。

注2　本システム・コールでは，タスク生成時（コンフィギュレーション時，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”の初期化が行われます。

注3　本システム・コールでは，対象タスクが終了する以前に獲得していた資源（セマフォ・カウント，メモリ・ブロックなど）の解放処理が行われません。したがって，本システム・コールを発行する前に資源の解放処理を行うことは，ユーザの責任となります。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID番号の指定が不正（*tskid* > 最大タスク生成数）である
*E_NOEXS	-52	対象タスクが存在していない
*E_OBJ	-63	対象タスクが自タスク，または，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
E_CTX	-69	非タスクから本システム・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言語式形

```
#include  <stdrx.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，本システム・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，ディスパッチ処理が禁止されます。

なお，RX830 では，本システム・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，待ちキューのキュー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，**ena_dsp** システム・コールが発行されるまで遅延され，一括して行うようにしています。

注 1　本システム・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，すでに本システム・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。

注 2　RX830 では，本システム・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に，自タスクを **wait** 状態へと遷移させる可能のあるシステム・コール（**wai_sem**，**wai_flg** など）を発行した場合，待ち条件の即時成立／不成立を問わず，戻り値として **E_CTX** を返しています。

戻り値

*E_OK	0	正常終了
*E_CTX	-69	コンテキスト・エラー － 非タスクから本システム・コールを発行した － **loc_cpu** システム・コールを発行後，本システム・コールを発行した

Enable Dispatch (-29)

ena_dsp

タスク

概　要

ディスパッチ処理を再開する。

C言語式形

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

パラメータ

なし

説　明

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（タスクのスケジューリング処理）を再開します。

なお，RX830 では，dis_dsp システム・コールの発行から本システム・コールが発行されるまでの間に，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，待ちキューのキュー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，本システム・コールが発行されるまで遅延され，一括して行うようにしています。

注意　本システム・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，すでに本システム・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_CTX	-69	コンテキスト・エラー － 非タスクから本システム・コールを発行した － loc_cpu システム・コールを発行後，本システム・コールを発行した

保守／廃止

Change Task 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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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
I	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

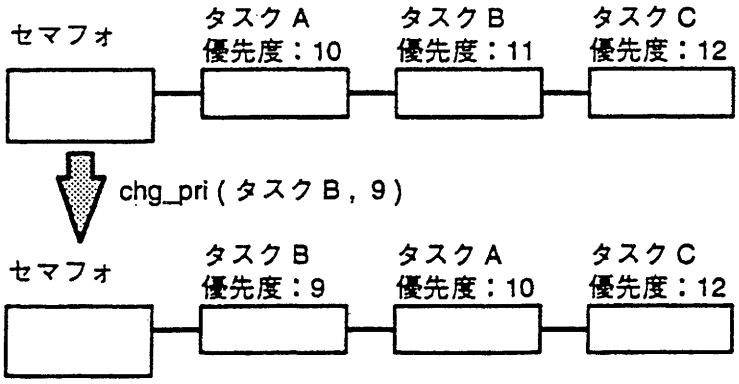
説　明

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。

ただし，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・キューの最後尾につなぎかえます。

注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た場合，本システム・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時，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，以下のように変更されます。



注２　本システム・コールが再発行されるまで，または，対象タスクがdormant状態へと遷移するまで，*tskpri*で指定された値が有効となります。

注３　RX830におけるタスクの優先度は，その値が小さいほど，高い優先度を意味します。

戻り値

*E_OK	0	正常終了
E_PAR	-33	優先度の指定が不正（*tskpri* < 0，*tskpri* > 最大優先度）である
E_ID	-35	ID番号の指定が不正である － *tskid* > 最大タスク生成数 － 非タスクから本システム・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

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言語式形

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

パラメータ

I/O	パラメータ	説　　明
I	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つなぎかえます。

注１　本システム・コールでは，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。

注２　本システム・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することが可能となります。

戻り値

*E_OK	0	正常終了
E_PAR	-33	優先度の指定が不正（*tskpri* < 0，*tskpri* > 最大優先度）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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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。

なお，本システム・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　本システム・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_ID	-35	ID番号の指定が不正 (*tskid* > 最大タスク生成数) 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言語式形

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

パラメータ

I/O	パラメータ	説　　明
O	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　本システム・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_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言語式形

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

パラメータ

I/O	パラメータ	説　　明
O	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
I	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;  /* タスクの状態              */
        UINT    tskwait;  /* 待ち要因                  */
        ID      wid;      /* 待ちオブジェクトの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;
```

説　明

tskid 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 *pk_rtsk* で指定されるパケットに格納します。

以下に，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

注 1 tskstsの値が TTS_WAI, TTS_WAS 以外の場合, tskwait の内容は不定となります。
注 2 tskwait の値が TTW_FLG, TTW_SEM, TTW_MBX, TTW_MPF 以外の場合, wid の内容は不定となります。

戻り値

*E_OK	0	正常終了
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 − *tskid* > 最大タスク生成数 − 非タスクから本システム・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_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言語式形

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

パラメータ

I/O	パラメータ	説　明
O	ID　*p_tskid;	ID番号を格納する領域のアドレス
I	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_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

11.8.2　タスク付属同期機能システム・コール

本項では，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）について説明しています。

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

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言語式形

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

パラメータ

I/O	パラメータ	説　　明
I	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。

ただし，本システム・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　RX830 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，本システム・コールでは，サスペンド要求数が 127 回を越えるような場合には，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 を返しています。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID 番号の指定が不正（*tskid* > 最大タスク生成数）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 － タスクから本システム・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

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言語式形

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

パラメータ

I/O	パラメータ	説　　明
I	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。

なお，本システム・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　本システム・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 状態以外の場合には，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 を返しています。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID 番号の指定が不正 (*tskid* > 最大タスク生成数) 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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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　本システム・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 状態以外の場合には，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 を返しています。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID 番号の指定が不正 (*tskid* > 最大タスク生成数) 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言語式形

```
#include  <stdrx.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，本システム・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，起床待ち状態の解除は， wup_tsk， ret_wup， rel_wai システム・コールが発行された際に行われ，起床待ち状態から ready 状態へと遷移します。

戻り値

*E_OK	0	正常終了
E_CTX	-69	コンテキスト・エラー － 非タスクから本システム・コールを発行した － ディスパッチ禁止状態から本システム・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言語式形

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

パラメータ

I/O	パラメータ	説　　明
I	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

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，本システム・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，起床待ち状態の解除は，*tmout* で指定された待ち時間が経過した際，または，wup_tsk，ret_wup，rel_wai システム・コールが発行された際に行われ，起床待ち状態から ready 状態へと遷移します。

戻り値

*E_OK	0	正常終了
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 － 非タスクから本システム・コールを発行した － ディスパッチ禁止状態から本システム・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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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，本システム・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　RX830 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，本システム・コールでは，起床要求数が 127 回を越えるような場合には，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 を返しています。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID 番号の指定が不正（*tskid* > 最大タスク生成数）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 － タスクから本システム・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言語式形

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

パラメータ

I/O	パラメータ	説　明
O	INT　*p_wupcnt;	起床要求数を格納する領域のアドレス
I	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，本システム・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_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 － *tskid* > 最大タスク生成数 － 非タスクから本システム・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

本項では，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）について説明しています。

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

Creat Semaphore (-49)

cre_sem

タスク

概　要

セマフォを生成する。

C言語式形

ID番号を指定する場合

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

ID番号を指定しない場合

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

パラメータ

I/O	パラメータ	説　明
I	ID　*semid*;	セマフォのID番号
I	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
O	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

RX830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もとに，*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もとに，セマフォを生成します。

  なお，ID 番号の割り付けは RX830 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。

以下に，セマフォ生成情報の詳細を示します。

exinf　･･･ 拡張情報

exinf は，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あり，ユーザが自由に利用することができます。

なお，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することができます。

sematr　･･･ セマフォの属性

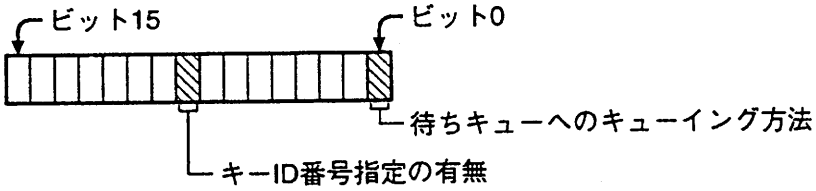
ビット 0 ･･ 待ちキューへのキューイング方法

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

注意　sematr のビット 8 の値が TA_KEYID 以外の場合，keyid の内容は意味をもちません。

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_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 番号無指定生成時

E_ID	-35	ID 番号の指定が不正（*semid* > 最大セマフォ生成数）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	非タスクから本システム・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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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，RX830の管理下から除外されます。

なお，本システム・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_ID	-35	ID番号の指定が不正（*semid* > 最大セマフォ生成数）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	非タスクから本システム・コールを発行した

Signal Semaphore (-55)

sig_sem

タスク／非タスク

概　要

資源を返却する。

C言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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，本システム・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　RX830 が管理するセマフォ・カウンタは，生成時に資源数としてとりえる最大資源数を指定します。このため，本システム・コールでは，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ず，戻り値として E_QOVR を返しています。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID 番号の指定が不正（*semid* > 最大セマフォ生成数）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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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，本システム・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，自タスクを対象セマフォの待ちキューにキューイングしたのち，run状態からwait状態（資源待ち状態）へと遷移させます。

なお，資源待ち状態の解除は，sig_sem，del_sem，rel_waiシステム・コールが発行された際に行われ，資源待ち状態からready状態へと遷移します。

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

戻り値

*E_OK	0	正常終了
E_ID	-35	ID番号の指定が不正（*semid* > 最大セマフォ生成数）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 － 非タスクから本システム・コールを発行した － ディスパッチ禁止状態から本システム・コールを発行した
*E_DLT	-81	del_semシステム・コールにより，対象セマフォを削除された
*E_RLWAI	-86	rel_waiシステム・コールにより，資源待ち状態を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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	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，本システム・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，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_ID	-35	ID番号の指定が不正（*semid* > 最大セマフォ生成数）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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	ID　*semid*;	セマフォのID番号
I	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

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

なお，資源待ち状態の解除は，*tmout* で指定された待ち時間が経過した際，または，sig_sem，del_sem，rel_wai システム・コールが発行された際に行われ，資源待ち状態から ready 状態へと遷移します。

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

戻り値

*E_OK	0	正常終了
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（*semid* > 最大セマフォ生成数）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

－ 非タスクから本システム・コールを発行した
－ ディスパッチ禁止状態から本システム・コールを発行した

*E_DLT	-81	del_sem システム・コールにより，対象セマフォを削除された
*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，資源待ち状態を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言語式形

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

パラメータ

I/O	パラメータ	説　　明
O	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
I	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。

以下に，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

戻り値

*E_OK	0	正常終了
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（*semid* > 最大セマフォ生成数）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言語式形

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

パラメータ

I/O	パラメータ	説　明
O	ID　*p_semid;	ID 番号を格納する領域のアドレス
I	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_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_semid* = 0) である − キー ID 番号の指定が不正 (*keyid* = 0) である
*E_NOEXS	-52	対象セマフォが存在していない

Creat Event Flag (-41)

cre_flg

タスク

概　要

イベント・フラグを生成する。

C言語式形

ID番号を指定する場合

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

ID番号を指定しない場合

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

パラメータ

I/O	パラメータ	説　明
I	ID　*flgid*;	イベント・フラグのID番号
I	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
O	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

RX830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もとに，*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もとに，イベント・フラグを生成します。

  なお，ID 番号の割り付けは RX830 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。

以下に，イベント・フラグ生成情報の詳細を示します。

exinf ・・・ 拡張情報

exinf は，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あり，ユーザが自由に利用することができます。

なお，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することができます。

flgatr ・・・ イベント・フラグの属性

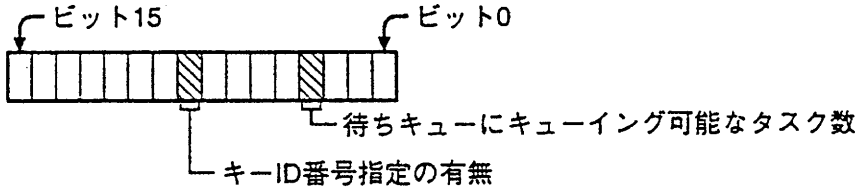
ビット 3 ‥ 待ちキューにキューイング可能なタスク数

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

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

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_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 番号無指定生成時
E_ID	-35	ID 番号の指定が不正 (*flgid* > 最大イベント・フラグ生成数) 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

保守／廃止

E_OACV	-66	アクセス権のないID番号 (*flgid* < 0) を指定した
E_CTX	-69	非タスクから本システム・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言語式形

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

パラメータ

I/O	パラメータ	説　　明
I	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，RX830の管理下から除外されます。

なお，本システム・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_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_ID	-35	ID番号の指定が不正（*flgid* > 最大イベント・フラグ生成数）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	非タスクから本システム・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言語式形

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

パラメータ

I/O	パラメータ	説　明
I	ID　*flgid*;	イベント・フラグのID番号
I	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。

なお，本システム・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。

例　本システム・コールを発行する際，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。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID番号の指定が不正 (*flgid* > 最大イベント・フラグ生成数) 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

clr_flg

Clear Event Flag (-47)

タスク／非タスク

概　要

ビット・パターンをクリアする。

C言語式形

```
#include  <stdrx.h>
ER        ercd = clr_flg(ID flgid, UINT clrptn);
```

パラメータ

I/O	パラメータ	説　明
I	ID　*flgid*;	イベント・フラグのID番号
I	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。

例　本システム・コールを発行する際，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_ID	-35	ID番号の指定が不正 (*flgid* > 最大イベント・フラグ生成数) 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

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言語式形

```
#include  <stdrx.h>
ER        ercd = wai_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	ID flgid;	イベント・フラグのID番号
I	UINT waiptn;	要求するビット・パターン (32ビット幅)
I	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否かをチェックします。

なお，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，本システム・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のち，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・フラグ待ち状態の解除は，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際，または，del_flg，rel_waiシステム・コールが発行された際に行われ，イベント・フラグ待ち状態からready状態へと遷移します。

以下に，*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否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否かをチェックします。

  なお，待ち条件が満足した際には，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否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否かチェックします。

  なお，待ち条件が満足した際には，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1　RX830 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，cre_flg システム・コール発行時) に指定します。

TA_WSGL 属性 ： 1 タスクのみキューイング可能とする

TA_WMUL 属性 ： 複数タスクをキューイング可能とする

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，本システム・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 を返しています。

注2　del_flg，rel_wai システム・コールの発行によりイベント・フラグ待ち状態を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となります。

戻り値

*E_OK	0	正常終了
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 (*flgid* > 最大イベント・フラグ生成数) 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，本システム・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

－ 非タスクから本システム・コールを発行した

－ ディスパッチ禁止状態から本システム・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・フラグを削除された
*E_RLWAI	-86	rel_wai システム・コールにより，イベント・フラグ待ち状態を強制的に解除された

保守／廃止

pol_flg

Poll Event Flag (-106)

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言語式形

```
#include  <stdrx.h>
ER        ercd = pol_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	ID flgid;	イベント・フラグのID番号
I	UINT waiptn;	要求するビット・パターン（32ビット幅）
I	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否かをチェックします。

なお，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，本システム・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。

以下に，*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否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否かをチェックします。

なお，待ち条件が満足した際には，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否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否かチェックします。

  なお，待ち条件が満足した際には，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　RX830 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，cre_flg システム・コール発行時）に指定します。

TA_WSGL 属性 ： 1 タスクのみキューイング可能とする

TA_WMUL 属性 ： 複数タスクをキューイング可能とする

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，本システム・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 を返しています。

戻り値

*E_OK	0	正常終了
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（*flgid* > 最大イベント・フラグ生成数）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，本システム・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言語式形

```
#include  <stdrx.h>
ER  ercd = twai_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode, TMO tmout);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	ID flgid;	イベント・フラグのID番号
I	UINT waiptn;	要求するビット・パターン（32ビット幅）
I	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
I	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

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否かをチェックします。

なお，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，本システム・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のち，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・フラグ待ち状態の解除は，*tmout* で指定された待ち時間が経過した際，または，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

設定された際，または，del_flg，rel_wai システム・コールが発行された際に行われ，イベント・フラグ待ち状態から ready 状態へと遷移します。

以下に，*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否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否かをチェックします。

  なお，待ち条件が満足した際には，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否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否かチェックします。

  なお，待ち条件が満足した際には，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 1　RX830 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，cre_flg システム・コール発行時) に指定します。

TA_WSGL 属性 ： 1 タスクのみキューイング可能とする

TA_WMUL 属性 ： 複数タスクをキューイング可能とする

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，本システム・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 を返しています。

注 2　del_flg，rel_wai システム・コールの発行によりイベント・フラグ待ち状態を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となります。

戻り値

*E_OK	0	正常終了
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（*flgid* > 最大イベント・フラグ生成数）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，本システム・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 − 非タスクから本システム・コールを発行した − ディスパッチ禁止状態から本システム・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・フラグを削除された
*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・フラグ待ち状態を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言語式形

```
#include  <stdrx.h>
ER        ercd = ref_flg(T_RFLG *pk_rflg, ID flgid);
```

パラメータ

I/O	パラメータ	説　　明
O	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
I	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。

以下に，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_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 (*flgid* > 最大イベント・フラグ生成数) 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

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言語式形

```
#include  <stdrx.h>
ER        ercd = vget_fid(ID *p_flgid, ID keyid);
```

パラメータ

I/O	パラメータ	説　明
O	ID　*p_flgid;	ID番号を格納する領域のアドレス
I	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_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_flgid* = 0) である － キーID番号の指定が不正 (*keyid* = 0) である
*E_NOEXS	-52	対象イベント・フラグが存在していない

保守／廃止

Creat Mailbox (-57)

cre_mbx

タスク

概　要

メールボックスを生成する。

C言語式形

ID番号を指定する場合

```
#include  <stdrx.h>
ER        ercd = cre_mbx(ID mbxid, T_CMBX *pk_cmbx);
```

ID番号を指定しない場合

```
#include  <stdrx.h>
ER        ercd = cre_mbx(ID_AUTO, T_CMBX *pk_cmbx, ID *p_mbxid);
```

パラメータ

I/O	パラメータ	説　明
I	ID　*mbxid*;	メールボックスのID番号
I	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
O	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

RX830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もとに，*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もとに，メールボックスを生成します。

なお，ID 番号の割り付けは RX830 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。

以下に，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あり，ユーザが自由に利用することができます。

なお，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することができます。

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

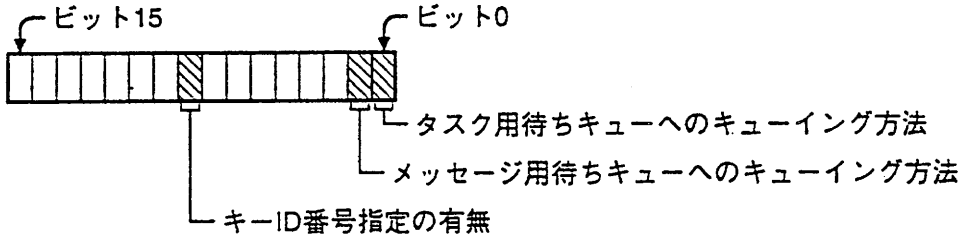
ビット 1 ‥ メッセージ用待ちキューへのキューイング方法

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

注意　mbxatr のビット 8 の値が TA_KEYID 以外の場合，keyid の内容は意味をもちません。

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_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 属性指定時

－ 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 番号無指定生成時

E_ID	-35	ID 番号の指定が不正 (*mbxid* > 最大メールボックス生成数) 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	非タスクから本システム・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言語式形

```
#include  <stdrx.h>
ER        ercd = del_mbx(ID mbxid);
```

パラメータ

I/O	パラメータ	説　明
I	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，RX830の管理下から除外されます。

なお，本システム・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。

注意　本システム・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。

戻り値

*E_OK	0	正常終了
E_ID	-35	ID番号の指定が不正（*mbxid* > 最大メールボックス生成数）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	非タスクから本システム・コールを発行した

snd_msg

Send Message to Mailbox (-63)

タスク／非タスク

概　要

メッセージを送信する。

C言語式形

```
#include  <stdrx.h>
ER        ercd = snd_msg(ID mbxid, T_MSG *pk_msg);
```

パラメータ

I/O	パラメータ	説　明
I	ID　*mbxid*;	メールボックスのID番号
I	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，本システム・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，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。

注1　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のキューイング方法は，対象メールボックス生成時（コンフィギュレーション時，cre_mbx システム・コール発行時）に指定された順（FIFO 順，優先度順）に行われます。

注2　RX830 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，本システム・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。

なお，本システム・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，RX830 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 を返しています。

注 3　メッセージの優先度の範囲は 0x1 ～ 0x7fff です。

戻り値

*E_OK	0	正常終了
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（*mbxid* > 最大メールボックス生成数）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

Receive Message from Mailbox (-61)

rcv_msg

タスク

概　要

メッセージを受信する。

C言語式形

```
#include  <stdrx.h>
ER        ercd = rcv_msg(T_MSG **ppk_msg, ID mbxid);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	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，本システム・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することができなかった（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のち，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の解除は，snd_msg，del_mbx，rel_waiシステム・コールが発行された際に行われ，メッセージ待ち状態からready状態へと遷移します。

注意　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のキューイング方法は，対象メールボックス生成時（コンフィギュレーション時，cre_mbxシステム・コール発行時）に指定された順（FIFO順，優先度順）に行われます。

戻り値

*E_OK	0	正常終了
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（*mbxid* > 最大メールボックス生成数）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

－ 非タスクから本システム・コールを発行した
－ ディスパッチ禁止状態から本システム・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を削除された
*E_RLWAI	-86	rel_wai システム・コールにより，メッセージ待ち状態を強制的に解除された

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言語式形

```
#include  <stdrx.h>
ER        ercd = prcv_msg(T_MSG **ppk_msg, ID mbxid);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	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，本システム・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することができなかった（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_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（*mbxid* > 最大メールボックス生成数）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言語式形

```
#include  <stdrx.h>
ER        ercd = trcv_msg(T_MSG **ppk_msg, ID mbxid, TMO tmout);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	ID mbxid;	メールボックスのID番号
I	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，本システム・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することができなかった（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のち，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の解除は，*tmout* で指定された待ち時間が経過した際，または，snd_msg，del_mbx，rel_waiシステム・コールが発行された際に行われ，メッセージ待ち状態からready状態へと遷移します。

注意　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のキューイング方法は，対象メールボックス生成時（コンフィギュレーション時，cre_mbxシステム・コール発行時）に指定された順（FIFO順，優先度順）に行われます。

戻り値

*E_OK	0	正常終了
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 (*mbxid* > 最大メールボックス生成数) 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 − 非タスクから本システム・コールを発行した − ディスパッチ禁止状態から本システム・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を削除された
*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を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言語式形

```
#include  <stdrx.h>
ER        ercd = ref_mbx(T_RMBX *pk_rmbx, ID mbxid);
```

パラメータ

I/O	パラメータ	説　　明
O	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
I	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*で指定されるパケットに格納します。

以下に，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

戻り値

*E_OK	0	正常終了
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番号の指定が不正（*mbxid* > 最大メールボックス生成数）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言語式形

```
#include  <stdrx.h>
ER        ercd = vget_mid(ID *p_mbxid, ID keyid);
```

パラメータ

I/O	パラメータ	説　明
O	ID　*p_mbxid;	ID番号を格納する領域のアドレス
I	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_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_mbxid* = 0）である － キーID番号の指定が不正（*keyid* = 0）である
*E_NOEXS	-52	対象メールボックスが存在していない

11.8.4　割り込み管理機能システム・コール

本項では，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）について説明しています。

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，および，直接起動割り込みハンドラからの復帰を行う
vret_clk	システム・クロック処理の呼び出し，および，直接起動割り込みハンドラからの復帰を行う
loc_cpu	マスカブル割り込みの受け付け，および，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，および，ディスパッチ処理を再開する
chg_ilv	割り込み許可レベルを変更する
ref_ilv	割り込み許可レベルを獲得する

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言語式形

```
#include  <stdrx.h>
ER        ercd = def_int(UINT dintno, T_DINT *pk_dint);
```

パラメータ

I/O	パラメータ	説　　明
I	UINT　*dintno*;	間接起動割り込みハンドラの割り込みレベル
I	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もとに，*dintno* で指定された割り込みレベル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。

以下に，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

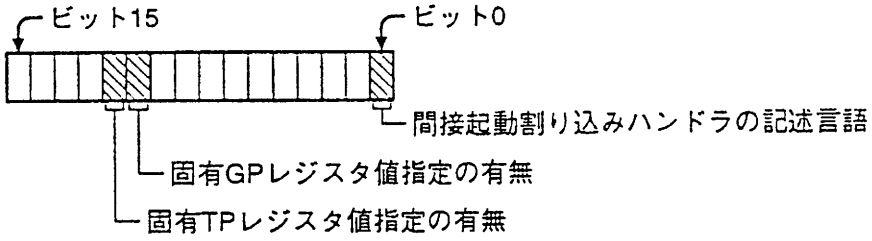
TA_HLNG (1) ： C 言語

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



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

注１ 本システム・コールを発行した際，すでに対象割り込みレベルに対応した間接起動割り込みハンドラが登録されていた場合には，エラーとして扱わず，本システム・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。

注２ 本システム・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，*dintno* で指定された間接起動割り込みハンドラの登録解除が行われます。

注３ intatr のビット 10 の値が TA_DPID 以外の場合，gp の内容は意味をもちません。

注４ intatr のビット 11 の値が TA_DPIC 以外の場合，tp の内容は意味をもちません。

戻り値

*E_OK	0	正常終了
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 － 割り込みレベルの指定が不正 (*dintno* < 0, *dintno* > 15) 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言語式形

```
#include  <stdrx.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。

なお，RX830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，直接起動割り込みハンドラからの復帰処理(本システム・コール，または，ret_wup，vret_clkシステム・コールの発行)まで遅延され，一括して行うようにしています。

注1　本システム・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)を行っていません。このため，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，本システム・コールを発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。

注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，以下のように記述します。

```
jr      _ret_int
```

注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，以下のように記述します。

```
return(TSK_NULL);
```

注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，以下のように記述します。

```
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，および，直接起動割り込みハンドラからの復帰を行う。

C言語式形

```
#include  <stdrx.h>
void      ret_wup(ID tskid);
```

パラメータ

I/O	パラメータ	説　　明
I	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）したのち，直接起動割り込みハンドラから復帰します。

ただし，本システム・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。

なお，RX830 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，直接起動割り込みハンドラからの復帰処理（本システム・コール，または，ret_int，vret_clk システム・コールの発行）まで遅延され，一括して行うようにしています。

注1　本システム・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）を行っていません。このため，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，本システム・コールを発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。

注2　本システム・コールでは，

- ID 番号の指定が不正（*tskid* > 最大タスク生成数）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

などといったエラーが生じた場合，直接起動割り込みハンドラからの復帰処理のみを行います。

保守／廃止

注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，他タスクの起床，および，直接起動割り込みハンドラからの復帰は，以下のように記述します。

```
mov     tskid, r10
jr      _ret_wup
```

注４　間接起動割り込みハンドラをＣ言語で記述した場合，他タスクの起床，および，間接起動割り込みハンドラからの復帰は，以下のように記述します。

```
return(ID tskid);
```

注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，および，間接起動割り込みハンドラからの復帰は，以下のように記述します。

```
mov     tskid, r10
jmp     [lp]
```

戻り値

なし

Return and Call System Clock Handler (-249)

vret_clk

直接起動割り込みハンドラ

概　要

システム・クロック処理の呼び出し，および，直接起動割り込みハンドラからの復帰を行う。

C言語式形

```
#include  <stdrx.h>
void      vret_clk();
```

パラメータ

なし

説　明

システム・クロック処理を呼び出したのち，直接起動割り込みハンドラから復帰します。

なお，RX830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，直接起動割り込みハンドラからの復帰処理 (本システム・コール，または，ret_int，ret_wup システム・コールの発行) まで遅延され，一括して行うようにしています。

注1　本システム・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) を行っていません。このため，外部割り込み要求によって直接起動起動した割り込みハンドラから復帰する場合は，本システム・コールを発行する前に外部割り込みコントローラに対する処理終了通知を行う必要があります。

注2　RX830では，システム・クロック処理として，以下の操作を行います。

- システム・クロックの更新
- 周期起動ハンドラの起動
- タスク (時間経過待ち状態など) のタイムアウト

注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，以下のように記述します。

```
mov     CLK_START, r10
jr      _vret_clk
```

注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，以下のように記述します。

```
return(CLK_START);
```

注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，間接起動割り込みハンドラからの復帰は，以下のように記述します。

```
mov     CLK_START, r10
jmp     [lp]
```

戻り値

なし

loc_cpu

Lock CPU (-8)

タスク

概　要

マスカブル割り込みの受け付け，および，ディスパッチ処理を禁止する。

C言語式形

```
#include  <stdrx.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，および，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，本システム・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，マスカブル割り込みの受け付け，および，ディスパッチ処理が禁止されます。

なお，RX830 では，本システム・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，unl_cpu システム・コールが発行されるまで遅延され，一括して行うようにしています。

注意　本システム・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，すでに本システム・コールが発行され，マスカブル割り込みの受け付け，および，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_CTX	-69	非タスクから本システム・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，および，ディスパッチ処理を再開する。

C言語式形

```
#include  <stdrx.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，および，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，RX830 では，loc_cpu システム・コールの発行から本システム・コールが発行されるまでの間に，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，本システム・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，本システム・コールが発行されるまで遅延され，一括して行うようにしています。

注１　本システム・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，すでに本システム・コールが発行され，マスカブル割り込みの受け付け，および，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。

注２　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，本システム・コールを発行することにより再開されます。

戻り値

*E_OK	0	正常終了
E_CTX	-69	非タスクから本システム・コールを発行した

保守／廃止

Change Interrupt Level (-67)

chg_ilv

タスク／非タスク

概　要

割り込み許可レベルを変更する。

C言語式形

```
#include  <stdrx.h>
ER        ercd = chg_ilv(UINT intlv);
```

パラメータ

I/O	パラメータ	説　明
I	UINT　*intlv*;	割り込み許可レベル

説　明

プロセッサ (V830 ファミリ™) の割り込み許可レベルを *intlv* で指定された値に変更します。

これにより，レベル 0 ～レベル (*intlv* - 1) はマスカブル割り込みの受け付けが禁止状態となり，レベル *intlv* ～レベル 15 はマスカブル割り込みの受け付けが許可状態となります。

注意　本システム・コールでは，全レベルのマスカブル割り込みの受け付けを禁止状態とすることができません。したがって，全レベルのマスカブル割り込みの受け付けを禁止状態とするには，処理プログラム (タスク，非タスク) からプロセッサのプログラム・ステータス・ワード PSW 内の割り込み禁止フラグ ID に “1” を設定します。

戻り値

*E_OK	0	正常終了
E_PAR	-33	割り込み許可レベルの指定が不正 (*intlv* < 0, *intlv* > 15) である

保守／廃止

Refer Interrupt Level (-68)

ref_ilv

タスク／非タスク

概　要

割り込み許可レベルを獲得する。

C言語式形

```
#include  <stdrx.h>
ER        ercd = ref_ilv(UINT *p_intlv);
```

パラメータ

I/O	パラメータ	説　　明
O	UINT　*p_intlv;	割り込み許可レベルを格納する領域のアドレス

説　明

プロセッサ (V830 ファミリ ™) の割り込み許可レベルを *p_intlv* 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_PAR	-33	割り込み許可レベルを格納する領域のアドレスが不正 (*p_intlv* = 0) 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

本項では，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）について説明しています。

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

Creat Variable-size Memory Pool (-137)

cre_mpl

タスク

概　要

メモリ・プールを生成する。

C言語式形

ID番号を指定する場合

```
#include  <stdrx.h>
ER        ercd = cre_mpl(ID mplid, T_CMPL *pk_cmpl);
```

ID番号を指定しない場合

```
#include  <stdrx.h>
ER        ercd = cre_mpl(ID_AUTO, T_CMPL *pk_cmpl, ID *p_mplid);
```

パラメータ

I/O	パラメータ	説　明
I	ID　*mplid*;	メモリ・プールのID番号
I	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
O	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

RX830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もとに，*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もとに，メモリ・プールを生成します。

  なお，ID 番号の割り付けは RX830 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。

以下に，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あり，ユーザが自由に利用することができます。

なお，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することができます。

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

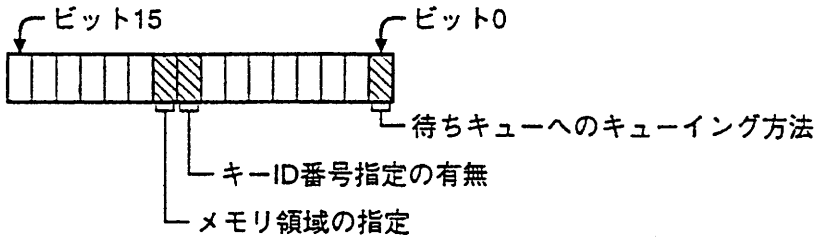
ビット 8 ・・ キー ID 番号指定の有無

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・プールの領域を確保

TA_UPOL1 (1) ： ユーザ・メモリ領域 1 番からメモリ・プールの領域を確保



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

注意　mplatr のビット 8 の値が TA_KEYID 以外の場合，keyid の内容は意味をもちません。

戻り値

*E_OK	0	正常終了
*E_NOMEM	-10	メモリ・プール管理ブロック，または，メモリ・プールの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（*pk_cmpl* = 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番号無指定生成時

E_ID	-35	ID番号の指定が不正（*mplid* > 最大メモリ・プール生成数）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	非タスクから本システム・コールを発行した

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言語式形

```
#include  <stdrx.h>
ER        ercd = del_mpl(ID mplid);
```

パラメータ

I/O	パラメータ	説　　明
I	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，RX830の管理下から除外されます。

なお，本システム・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（get_blk，tget_blk）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_ID	-35	ID番号の指定が不正（*mplid* > 最大メモリ・プール生成数）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	非タスクから本システム・コールを発行した

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言語式形

```
#include  <stdrx.h>
ER        ercd = get_blk(VP *p_blk, ID mplid, INT blksz);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	ID　mplid;	メモリ・プールのID番号
I	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，本システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得することができなかった（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のち，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・ブロック待ち状態の解除は，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た際，または，del_mpl，rel_wai システム・コールが発行された際に行われ，メモリ・ブロック待ち状態から ready 状態へと遷移します。

注1　RX830 では，メモリ・ブロックを獲得する際，メモリ・クリアを行っていません。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。

注2　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のキューイング方法は，対象メモリ・プール生成時（コンフィギュレーション時，cre_mpl システム・コール発行時）に指定された順（FIFO 順，優先度順）に行われます。

戻り値

*E_OK	0	正常終了

保守／廃止

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 ($blksz \leqq 0$) である
E_ID	-35	ID 番号の指定が不正 ($mplid$ > 最大メモリ・プール生成数) 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 ($mplid \leqq 0$) を指定した
E_CTX	-69	コンテキスト・エラー － 非タスクから本システム・コールを発行した － ディスパッチ禁止状態から本システム・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・プールを削除された
*E_RLWAI	-86	rel_wai システム・コールにより，メモリ・ブロック待ち状態を強制的に解除された

Pool 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言語式形

```
#include  <stdrx.h>
ER        ercd = pget_blk(VP *p_blk, ID mplid, INT blksz);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	ID mplid;	メモリ・プールのID番号
I	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，本システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得することができなかった（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　RX830では，メモリ・ブロックを獲得する際，メモリ・クリアを行っていません。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_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（*mplid* > 最大メモリ・プール生成数）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言語式形

```
#include  <stdrx.h>
ER        ercd = tget_blk(VP *p_blk, ID mplid, INT blksz, TMO tmout);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	ID mplid;	メモリ・プールのID番号
I	INT blksz;	メモリ・ブロックのサイズ（単位：バイト）
I	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

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，本システム・コールを発行した際，対象メモリ・プールからメモリ・ブロックを獲得することができなかった（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のち，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・ブロック待ち状態の解除は，*tmout* で指定された待ち時間が経過した際，または，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た際，または，del_mpl，rel_wai システム・コールが発行された際に行われ，メモリ・ブロック待ち状態から ready 状態へと遷移します。

注1　RX830 では，メモリ・ブロックを獲得する際，メモリ・クリアを行っていません。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。

注2　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のキューイング方法は，対象メモリ・プール生成時（コンフィギュレーション時，cre_mpl システム・コール発行時）に指定された順（FIFO 順，優先度順）に行われます。

戻り値

*E_OK	0	正常終了
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（*mplid* > 最大メモリ・プール生成数）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 － 非タスクから本システム・コールを発行した － ディスパッチ禁止状態から本システム・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・プールを削除された
*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・ブロック待ち状態を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言語式形

```
#include  <stdrx.h>
ER        ercd = rel_blk(ID mplid, VP blk);
```

パラメータ

I/O	パラメータ	説　明
I	ID　*mplid*;	メモリ・プールのID番号
I	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。

なお，本システム・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（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1　RX830では，メモリ・ブロックを返却する際，メモリ・クリアを行っていません。したがって，返却したメモリ・ブロックの内容は不定となります。

注2　メモリ・ブロックの返却は，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・プールと同一でなければなりません。

戻り値

*E_OK	0	正常終了
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・プールと本システム・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（*mplid* > 最大メモリ・プール生成数）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

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言語式形

```
#include  <stdrx.h>
ER        ercd = ref_mpl(T_RMPL *pk_rmpl, ID mplid);
```

パラメータ

I/O	パラメータ	説　明
O	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
I	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。

以下に，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_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（*mplid* > 最大メモリ・プール生成数）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言語式形

```
#include  <stdrx.h>
ER        ercd = vget_pid(ID *p_mplid, ID keyid);
```

パラメータ

I/O	パラメータ	説　明
O	ID　*p_mplid;	ID番号を格納する領域のアドレス
I	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_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_mplid* = 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

本項では，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）について説明しています。

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する
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言語式形

```
#include  <stdrx.h>
ER        ercd = set_tim(SYSTIME *pk_tim);
```

パラメータ

I/O	パラメータ	説　　明
I	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  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_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言語式形

```
#include  <stdrx.h>
ER        ercd = get_tim(SYSTIME *pk_tim);
```

パラメータ

I/O	パラメータ	説　　明
O	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  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_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する。

C言語式形

```
#include  <stdrx.h>
ER        ercd = dly_tsk(DLYTIME dlytim);
```

パラメータ

I/O	パラメータ	説　　明
I	DLYTIME *dlytim*;	遅延時間（単位：基本クロック周期）

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，時間経過待ち状態の解除は，*dlytim* で指定された遅延時間が経過した際，または，rel_wai システム・コールが発行された際に行われ，時間経過待ち状態から ready 状態へと遷移します。

注意　時間経過待ち状態の解除は，wup_tsk，ret_wup システム・コールでは行われません。

戻り値

*E_OK	0	正常終了
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 － 非タスクから本システム・コールを発行した － ディスパッチ禁止状態から本システム・コールを発行した
*E_RLWAI	-86	rel_wai システム・コールにより，時間経過待ち状態を強制的に解除された

保守／廃止

def_cyc

Define Cyclic Handler (-90)

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言語式形

```
#include  <stdrx.h>
ER        ercd = def_cyc(HNO cycno, T_DCYC *pk_dcyc);
```

パラメータ

I/O	パラメータ	説　　明
I	HNO　　*cycno*;	周期起動ハンドラの指定番号
I	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もとに，*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。

以下に，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あり，ユーザが自由に利用することができます。

なお，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することができます。

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

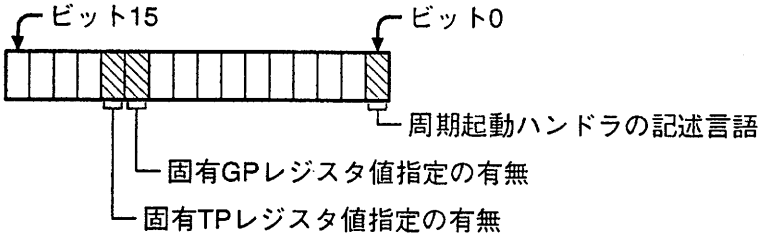
TA_HLNG (1) ： C 言語

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

注1 本システム・コールを発行した際，すでに対象指定番号に対応した周期起動ハンドラが登録されていた場合には，エラーとして扱わず，本システム・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。

注2 本システム・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解除が行われます。

注3 cycatr のビット 10 の値が TA_DPID 以外の場合，gp の内容は意味をもちません。

注4 cycatr のビット 11 の値が TA_DPIC 以外の場合，tp の内容は意味をもちません。

戻り値

*E_OK	0	正常終了
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, *cycno* > 最大周期起動ハンドラ登録数) 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言語式形

```
#include  <stdrx.h>
ER        ercd = act_cyc(HNO cycno, UINT cycact);
```

パラメータ

I/O	パラメータ	説　明
I	HNO　*cycno*;	周期起動ハンドラの指定番号
I	UINT　*cycact*;	活性状態／周期カウンタの指定 TCY_OFF (0)：活性状態を OFF 状態へ変更する TCY_ON (1)：活性状態を ON 状態へ変更する TCY_INI (2)：周期カウンタを初期化する

説　明

cycno で指定された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。以下に，*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　RX830 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が行われます。

- *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 状態に変更したのち，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_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, *cycno* > 最大周期起動ハンドラ登録数）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言語式形

```
#include  <stdrx.h>
ER        ercd = ref_cyc(T_RCYC *pk_rcyc, HNO cycno);
```

パラメータ

I/O	パラメータ	説　　明
O	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
I	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。

以下に，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

戻り値

*E_OK	0	正常終了
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，*cycno* > 最大周期起動ハンドラ登録数）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

本項では，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）について説明しています。

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	RX830 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

Get Version Information (-16)

get_ver

タスク／非タスク

概　要

RX830 のバージョン情報を獲得する。

C言語式形

```
#include  <stdrx.h>
ER        ercd = get_ver(T_VER *pk_ver);
```

パラメータ

I/O	パラメータ	説　　明
O	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 製造メーカ          */
        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

RX830 のバージョン情報 (OS 製造メーカ， 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。

以下に，バージョン情報の詳細を示します。

maker　… OS 製造メーカ
　　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 Ver3.02

prver　… OS 製品バージョン番号
　　H'0300 ： RX830 Ver3.00

保守／廃止

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る）
cpu　　･･･ CPU 情報
　　H'0d32 ：μPD705100
var　　･･･ バリエーション記述子
　　H'c000 ：μITRON レベル E，シングル・プロセッサ用，仮想記憶サポートなし，MMU サポートなし，ファイル・サポートなし

戻り値

*E_OK	0	正常終了
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

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言語式形

```
#include  <stdrx.h>
ER        ercd = ref_sys(T_RSYS *pk_rsys);
```

パラメータ

I/O	パラメータ	説　　明
O	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

動的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。

以下に，システム情報の詳細を示します。

sysstat ･･･ システムの状態

- TTS_TSK (0) ： タスクの処理を実行中。
  なお，ディスパッチ処理は許可状態。
- TTS_DDSP (1) ： タスクの処理を実行中。
  なお，ディスパッチ処理は禁止状態。
- TTS_LOC (3) ： タスクの処理を実行中。
  なお，マスカブル割り込みの受け付け，および，ディスパッチ処理は禁止状態。
- TTS_INDP (4) ： 非タスク（割り込みハンドラ，周期起動ハンドラ）の処理を実行中。

戻り値

*E_OK	0	正常終了

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 言語式形

```
#include  <stdrx.h>
ER        ercd = def_svc(FN s_fncd, T_DSVC *pk_dsvc);
```

パラメータ

I/O	パラメータ	説　　明
I	FN　　*s_fncd*;	拡張 SVC ハンドラの拡張機能コード
I	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もとに，*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。

以下に，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

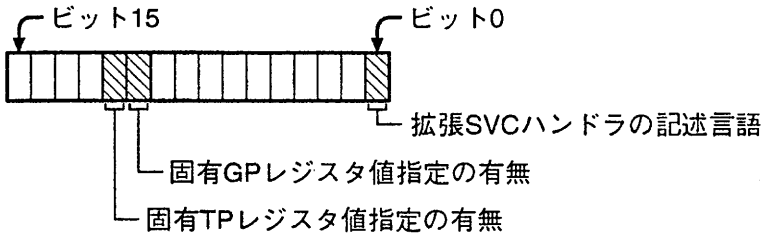
TA_HLNG (1) ： C 言語

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



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

注 1 本システム・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，エラーとして扱わず，本システム・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。

注 2 本システム・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 ハンドラの登録解除が行われます。

注 3 svcatr のビット 10 の値が TA_DPID 以外の場合，gp の内容は意味をもちません。

注 4 svcatr のビット 11 の値が TA_DPIC 以外の場合，tp の内容は意味をもちません。

戻り値

*E_OK	0	正常終了
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, *s_fncd* > 最大拡張 SVC ハンドラ登録数) 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言語式形

```
#include  <stdrx.h>
ER        ercd = viss_svc(FN s_fncd, VW prm1, VW prm2, VW prm3);
```

パラメータ

I/O	パラメータ	説　明
I	FN　*s_fncd*;	拡張 SVC ハンドラの拡張機能コード
I	VW　*prm1*;	拡張 SVC ハンドラへの引き渡しパラメータ 1
I	VW　*prm2*;	拡張 SVC ハンドラへの引き渡しパラメータ 2
I	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。

注意　本システム・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_PAR	-33	拡張機能コードの指定が不正（*s_fncd* ≦ 0，*s_fncd* > 最大拡張 SVC ハンドラ登録数）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 製 V830 ファミリ™ 用 C コンパイラ CA830，または，米国 Green Hills Software, Inc. 製 C クロス V800™ コンパイラ CCV830 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ています。

A.1　概要

RX830 では，処理プログラムを用途別に，以下のように区別しています。

なお，これらの処理プログラムは，一般的，あるいは，RX830 を使用するうえでの約束ごとなどにより，それぞれに基本型があります。

- タスク

  RX830 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，RX830 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，RX830 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，RX830 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。

- 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な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。

保守／廃止

A.2 キー・ワード

コンフィギュレータでは，以下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，これらの文字列を他の用途に使用することは禁止されています。

clktim	cyc	defstk	flg	flgsvc
ini	inthdr	intstk	intsvc	maxcyc
maxflg	maxmbx	maxmpl	maxpri	maxsem
maxsvc	maxtsk	mbx	mbxsvc	mem
mpl	mplsvc	no_use	prtflg	prtmbx
prtmpl	prtsem	prttsk	RX830	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	V300	

A.3 予約語

RX830 では，以下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，これらの文字列を他の用途に使用することは禁止されています。

RX830*	Sit*	SysIntEnt	Timer_Handler	_urx_start
start_kern	.pool0	.pool1		

注意　* は，1 文字以上の文字列を表しています。

A.4 タスク

A.4.1 CA830 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 型の引き数を 1 つ持った FP 型の関数として記述します。

なお，引き数 (*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。

図 A-1に，CA830 を使用した際のタスクの記述形式 (C 言語) を示します。

図 A-1 タスクの記述形式 CA830

```
#include <stdrx.h>

#pragma  rtos_task  func_task
FP
func_task(INT stacd)
{
        /* タスク func_task の本体処理 */
        .....................................
        .....................................
        .....................................

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

注意 プラグマ指令による関数宣言についての詳細は，「V800 シリーズ C コンパイラ・パッケージ ユーザーズ・マニュアル C 言語編」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，CA830 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，引き数 (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。

図 A-2に，CA830 を使用した際のタスクの記述形式 (アセンブリ言語) を示します。

図 A-2　タスクの記述形式 CA830

```
.include "stdrx.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
```

A.4.2 CCV830 対応版の場合

タスクを C 言語で記述する場合，INT 型の引き数を 1 つ持った FP 型の関数として記述します。

なお，引き数 (*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。

図 A-3に，CCV830 を使用した際のタスクの記述形式 (C 言語) を示します。

図 A-3　タスクの記述形式 CCV830

```
#include <stdrx.h>

FP
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，CCV830 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，引き数 (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。

図 A-4に，CCV830 を使用した際のタスクの記述形式 (アセンブリ言語) を示します。

図 A-4　タスクの記述形式 CCV830

```
#include <stdrx.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　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，“.830”を指定してください。

A.5　直接起動割り込みハンドラ

A.5.1　CA830 対応版の場合

直接起動割り込みハンドラを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，引き数を持たない INT 型の関数として記述します。

図 A-5に，CA830 を使用した際の直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-5　直接起動割り込みハンドラの記述形式 CA830

```
#include <stdrx.h>

#pragma  rtos_interrupt  int_name  func_inthdr
INT
func_inthdr()
{
        /* 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 からの復帰 */
        ret_int();
}
```

注１　プラグマ指令による関数宣言についての詳細は，「V800 シリーズ　C コンパイラ・パッケージ　ユーザーズ・マニュアル　C 言語編」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。

注２　直接起動割り込みハンドラをＣ言語で記述する場合，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せん。

また，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は，CA830 の関数呼び出し規約に従った関数として記述します。

図 A-6に，CA830 を使用した際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　直接起動割り込みハンドラの記述形式 CA830

```
.include "stdrx.inc"

        .section    "int_name", text
        .align      4
        .globl      _func_inthdr
_func_inthdr :
        # レジスタの退避，スタックの切り替え
        ..................................................

        # 直接起動割り込みハンドラ func_inthdr の本体処理
        ..................................................

        # 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 func_inthdr からの復帰
        reti
```

注 1　.section 疑似命令についての詳細は，「CA830　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」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。

注 2　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，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せん。

注 3　直接起動割り込みハンドラ内でシステム・コールを使用する場合は関数の先頭でマクロ「RTOS_IntEntry」を使用し、関数の最後で reti をマクロ「RTOS_IntReturn」(他タスクを起床しない場合) もしくは「RTOS_IntReturnWakeup a_tsk」(他タスクを起床する場合) に変更して下さい。

A.5.2 CCV830 対応版の場合

直接起動割り込みハンドラを C 言語で記述することはできません。したがって，CCV830 を使用する場合は，直接起動割り込みハンドラの記述はアセンブリ言語に限られます。

なお，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，CCV830 の関数呼び出し規約に従った関数として記述します。

図 A-7に，CCV830 を使用した際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-7　直接起動割り込みハンドラの記述形式 CCV830

```
#include <stdrx.h>

        .text
        .align      4
        .globl      _func_inthdr
_func_inthdr :
        # レジスタの退避，スタックの切り替え
        ..................................................

        # 直接起動割り込みハンドラ func_inthdr の本体処理
        ..................................................

        # 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 func_inthdr からの復帰
        reti
```

注 1　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，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。

注 2　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，“.830”を指定してください。

注 3　直接起動割り込みハンドラ内でシステム・コールを使用する場合は関数の先頭でマクロ「RTOS_IntEntry」を使用し、関数の最後で reti をマクロ「RTOS_IntReturn」(他タスクを起床しない場合) もしくは「RTOS_IntReturnWakeup a_tsk」(他タスクを起床する場合) に変更して下さい。

A.6 間接起動割り込みハンドラ

A.6.1 CA830 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，引き数を持たない FP 型の関数として記述します。

図 A-8に，CA830 を使用した際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-8 間接起動割り込みハンドラの記述形式 CA830

```
#include <stdrx.h>

FP
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);
}
```

注意 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。このため，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が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せん。

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，CA830 の関数呼び出し規約に従った関数として記述します。

図 A-9に，CA830 を使用した際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-9　間接起動割り込みハンドラの記述形式 CA830

```
.include  "stdrx.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]
```

注意　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。このため，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が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せん。

A.6.2 CCV830 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，引き数を持たない FP 型の関数として記述します。

図 A-10に，CCV830 を使用した際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-10 間接起動割り込みハンドラの記述形式 CCV830

```
#include <stdrx.h>

FP
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);
}
```

注意 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。このため，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が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せん。

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，CCV830 の関数呼び出し規約に従った関数として記述します。

図 A-11に，CCV830 を使用した際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-11　間接起動割り込みハンドラの記述形式 CCV830

```
#include <stdrx.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]
```

注 1　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。このため，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が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せん。

注 2　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，“.830”を指定してください。

A.7　周期起動ハンドラ

A.7.1　CA830 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，引き数を持たない FP 型の関数として記述します。

図 A-12に，CA830 を使用した際の周期起動ハンドラの記述形式 (C 言語) を示します。

図 A-12　周期起動ハンドラの記述形式　CA830

```
#include <stdrx.h>

FP
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;
}
```

注意　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，CA830 の関数呼び出し規約に従った関数として記述します。

図 A-13に，CA830 を使用した際を使用した際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-13　周期起動ハンドラの記述形式 CA830

```
.include "stdrx.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
          jr          jmp [lp]
```

注意　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 CCV830 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，引き数を持たない FP 型の関数として記述します。

図 A-14に，CCV830 を使用した際の周期起動ハンドラの記述形式 (C 言語) を示します。

図 A-14 周期起動ハンドラの記述形式 CCV830

```
#include <stdrx.h>

FP
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;
}
```

注意 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，CCV830の関数呼び出し規約に従った関数として記述します。

図A-15に，CCV830を使用した際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-15　周期起動ハンドラの記述形式 CCV830

```
#include <stdrx.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
        jr          jmp [lp]
```

注1　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2　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，".830"を指定してください。

A.8 拡張 SVC ハンドラ

A.8.1 CA830 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。
図 A-16に，CA830 を使用した際の拡張 SVC ハンドラの記述形式 (C 言語) を示します。

図 A-16 拡張 SVC ハンドラの記述形式 CA830

```
#include <stdrx.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，CA830の関数呼び出し規約に従った関数として記述します。

図A-17に，CA830を使用した際の拡張SVCハンドラの記述形式（アセンブリ言語）を示します。

図 A-17　拡張SVCハンドラの記述形式 CA830

```
.include "stdrx.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]
```

A.8.2 CCV830 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。
図 A-18に，CCV830 を使用した際の拡張 SVC ハンドラの記述形式 (C 言語) を示します。

図 A-18 拡張 SVC ハンドラの記述形式 CCV830

```
#include <stdrx.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，CCV830の関数呼び出し規約に従った関数として記述します。

図A-19に，CCV830を使用した際の拡張SVCハンドラの記述形式（アセンブリ言語）を示します。

図 A-19　拡張SVCハンドラの記述形式 CCV830

```
#include <stdrx.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，“.830”を指定してください。

保守／廃止

索引

S

T

U

V

W

あ

い

保守／廃止

は

ふ

ほ

ま

み

め

も

ゆ

よ

ら

り

わ

保守／廃止

［メ　モ］

保守／廃止

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン（インフォメーションセンター）

電　話　：044-548-8899
FAX　：044-548-7900
E-mail　：s-info@saed.tmg.nec.co.jp

【営業関係お問い合わせ先】

半導体第一販売事業部 半導体第二販売事業部 半導体第三販売事業部	〒108-8001	東京都港区芝５－７－１　（日本電気本社ビル）	(03)3454-1111
中部支社 半導体第一販売部 半導体第二販売部	〒460-8525	愛知県名古屋市中区錦１－17－１　（日本電気中部ビル）	(052)222-2170 (052)222-2190
関西支社 半導体第一販売部 半導体第二販売部 半導体第三販売部	〒540-8551	大阪府大阪市中央区城見１－４－24　（日本電気関西ビル）	(06) 945-3178 (06) 945-3200 (06) 945-3208

北海道支社	札　幌	(011)231-0161	宇都宮支店	宇都宮	(028)621-2281	北陸支社	金　沢	(076)232-7303
東北支社	仙　台	(022)267-8740	小山支店	小　山	(0285)24-5011	富山支店	富　山	(0764)31-8461
岩手支店	盛　岡	(019)651-4344	甲府支店	甲　府	(0552)24-4141	福井支店	福　井	(0776)22-1866
郡山支店	郡　山	(0249)23-5511	長野支社	松　本	(0263)35-1662	京都支社	京　都	(075)344-7824
いわき支店	いわき	(0246)21-5511	静岡支社	静　岡	(054)254-4794	神戸支社	神　戸	(078)333-3854
長岡支店	長　岡	(0258)36-2155	立川支社	立　川	(042)526-5981,6167	中国支社	広　島	(082)242-5504
水戸支店	水　戸	(029)226-1717	埼玉支社	大　宮	(048)649-1415	鳥取支店	鳥　取	(0857)27-5311
土浦支店	土　浦	(0298)23-6161	千葉支社	千　葉	(043)238-8116	岡山支店	岡　山	(086)225-4455
群馬支店	高　崎	(027)326-1255	神奈川支社	横　浜	(045)682-4524	松山支店	松　山	(089)945-4149
太田支店	太　田	(0276)46-4011	三重支店	津	(059)225-7341	九州支社	福　岡	(092)261-2806

C98.2

保守／廃止

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］RX830（μITRON Ver.3.0） ユーザーズ・マニュアル 基礎編
（U13152JJ2V0UM00（第2版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他） （ ）
ご住所 （ ）
お電話番号 （ ）
お仕事の内容 （ ）
お名前 （ ）

1. ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（ ）					
（ ）					

2. わかりやすい所（第　章，第　章，第　章，第　章，その他 ）
理由［ ］

3. わかりにくい所（第　章，第　章，第　章，第　章，その他 ）
理由［ ］

4. ご意見，ご要望

5. このドキュメントをお届けしたのは
NEC販売員，特約店販売員，NEC半導体ソリューション技術本部員，
その他（ ）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

NEC半導体テクニカルホットライン
FAX：（044）548-7900

キリトリ