User's Manual

RI600/PX V.1.00

ユーザーズマニュアル

RX ファミリ MPU(Memory Protection Unit)対応リアルタイム OS

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

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00 2011.09

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。

■はじめに

本マニュアルは、MPU(Memory Protection Unit)を搭載した RX ファミリマイコン用リアルタイム OS 製品「RI600/PX」の使用方法を述べたものです。ご使用になる前に本マニュアルを良く読んで理解してください。

■表記上の注意事項

- 数値のプリフィックス
  "0x"は 16 進数を意味します。プリフィックスの無い場合は 10 進数です。
- ディレクトリ区切り記号は"¥"です。
- "cfg ファイル"は、コンフィギュレーションファイルを意味します。
- "system.stack_size"など、ピリオドでつながれた表記は、以下のいずれかの意味です。
  (1) cfg ファイルの設定項目
  (2) 構造体の要素
  (3) レジスタ等の特定ビット
- [メニュー->メニューオプション]
  "->"はメニューオプションを示します。(例[ファイル->保存])
- $(xxxx)
  $(xxxx)は、High-performance Embedded Workshop のカスタムプレースホルダを示します。

■商標等

すべての商標および登録商標は、それぞれの所有者に帰属します。

- TRON は、"The Real-time Operating system Nucleus" の略称です。ITRON は、"Industrial TRON"の略称です。μITRON は、"Micro Industrial TRON" の略称です。TRON、ITRON、および μITRON は、コンピュータの仕様に対する名称であり、特定の商品ないし商品群を指すものではありません。 μITRON4.0 仕様は、T-Engine フォーラムが策定したオープンなリアルタイムカーネル仕様です。μITRON4.0 仕様の仕様書は、T-Engine フォーラムのホームページ(http://www.t-engine.org/)から入手が可能です。 μITRON 仕様の著作権は T-Engine フォーラムに属しています。
- Microsoft、Windows は米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。Windows の正式名称は、Microsoft Windows Operating System です。
- その他、本書で登場するシステム名、製品名は各社の登録商標または商標です。

■ホームページ

弊社ホームページにて各種サポート情報をお知らせしておりますので、あわせてご利用ください。

http://japan.renesas.com/

目次

RI600/PX V.1.00
RX ファミリ MPU(Memory Protection Unit)対応リアルタイム OS
R20UT0606JJ0100
Rev.1.00
2011.09.01

1. 概要

1.1 特長

(1) µITRON4.0 仕様に準拠

RI600/PX は、µITRON 仕様の最新バージョンである µITRON4.0 仕様に基づいて開発された RI600/4 をベースとし、µITRON4.0 仕様保護機能拡張のメモリ保護機能を追加した製品です。このため、µITRON 仕様関連の各種出版物やセミナー等で得た知識を、そのまま役立てることができます。また、µITRON 仕様に準拠した他のリアルタイム OS を用いて開発したアプリケーションプログラムを、比較的容易に RI600/PX に移植することができます。

(2) 高速処理を実現

RX600 シリーズマイコンのアーキテクチャを活用し、高速処理を実現しています。

(3) 必要モジュールのみを自動選択することにより常に最小サイズのシステムを構築

RI600/PX のカーネルは、RX600 シリーズオブジェクトライブラリ形式で供給されています。したがって、リンケージエディタの持つ機能により、数あるカーネルの機能モジュールの中からアプリケーションが使用しているモジュールのみが自動選択されます。このため、常に最小サイズのシステムを生成できます。

(4) 統合開発環境を利用して効率の良い開発が可能

ルネサス統合開発環境 High-performance Embedded Workshop を使用して、開発を行うことができます。High-performance Embedded Workshop では、RI600/PX 対応アプリケーション用のワークスペースを生成する機能をサポートしています。

(5) コンフィギュレータによる容易なカーネル構築

RI600/PX では、コマンドラインコンフィギュレータ cfg600px を装備しています。各種カーネル構築情報をテキスト形式の cfg ファイルに記述することで、カーネルを構築することができます。テキスト形式なので、変更管理も容易です。

1.2 提供ソフトウェア構成

(1) カーネル

リアルタイム OS 本体です。

(2) cfg600px(コマンドラインコンフィギュレータ)

カーネルを構築するためのツールです。ユーザが作成した cfg ファイルを入力とし、カーネル定義ファイルを出力します。

(3) mkritblpx(テーブル生成ユーテリティ)

アプリケーションが使用しているサービスコール情報を収集し、最適なサービスコールテーブルと割り込みベクタテーブルを生成するコマンドラインツールです。

1.3 動作環境

表 1.1に、動作環境を示します。

表1.1 動作環境

項目	動作環境
対象 CPU	MPU(Memory Protection Unit)を搭載した RX600 シリーズマイコン
ホストマシン	下記 OS が稼動している IBM-PC/AT 互換機 Windows®XP , WIndows Vista®, Windows® 7
コンパイラ	ルネサス製「RX ファミリ用 C/C++コンパイラパッケージ V.1.01 以降」

2. カーネル入門

2.1 カーネルの動作原理

カーネルとは、リアルタイム OS の中核となるプログラムのことです。カーネルは、1 つの CPU を、あたかも複数の CPU が動作しているように見せることのできるソフトウェアです。では、1 つの CPU をどのように複数あるように見せかけているのでしょうか？それは、カーネルは図 2.1に示すようにそれぞれのタスクを必要に応じて切り替えながら動作させるからです。

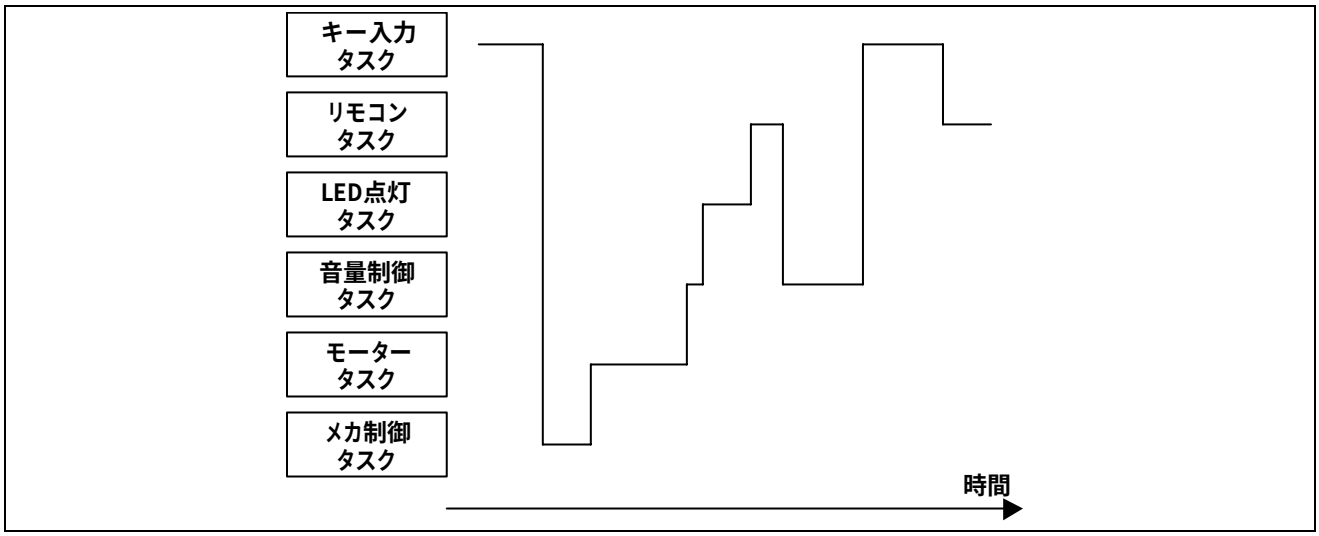

図2.1 **マルチタスク動作**

タスクを切り替えることをディスパッチと呼びます。ディスパッチが発生する要因として、以下のものがあります。

- タスクが自分自身で切り替えを要求する
- 割り込みなど、タスクから見て外部の要因で切り替わる

言い換えると、一定時間毎にタスクが切り替わる(タイムシェアリング)わけではありません。このようなスケジューリングを一般に「イベントドリブン」または「イベント駆動型」と呼びます。

ディスパッチ発生後、再度そのタスクを実行するときには、中断していたところから実行を再開します(図 2.2参照)。

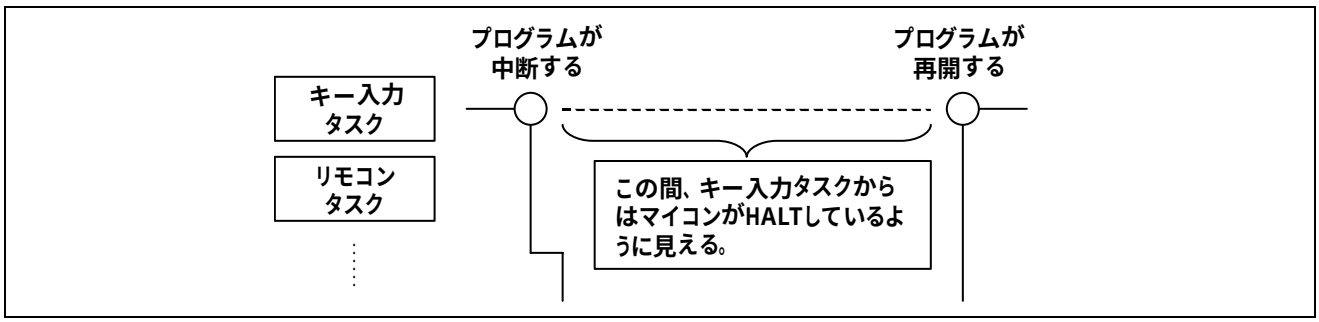

図2.2 **タスクの中断と再開**

図 2.2において、キー入力タスクは他のタスクに実行制御が移っている間、プログラマから見ればプログラムが中断し、そのマイコンが HALT しているように見えます。カーネルは、中断した時点の CPU レジスタ内

容を復帰することにより、タスクを中断した時点の状態から再開させます。すなわち、ディスパッチとは、現在実行中のタスクの CPU レジスタの内容を、そのタスクを管理するメモリ領域に退避し、切り替えるタスクの CPU レジスタ内容を復帰することです(図 2.3参照)。

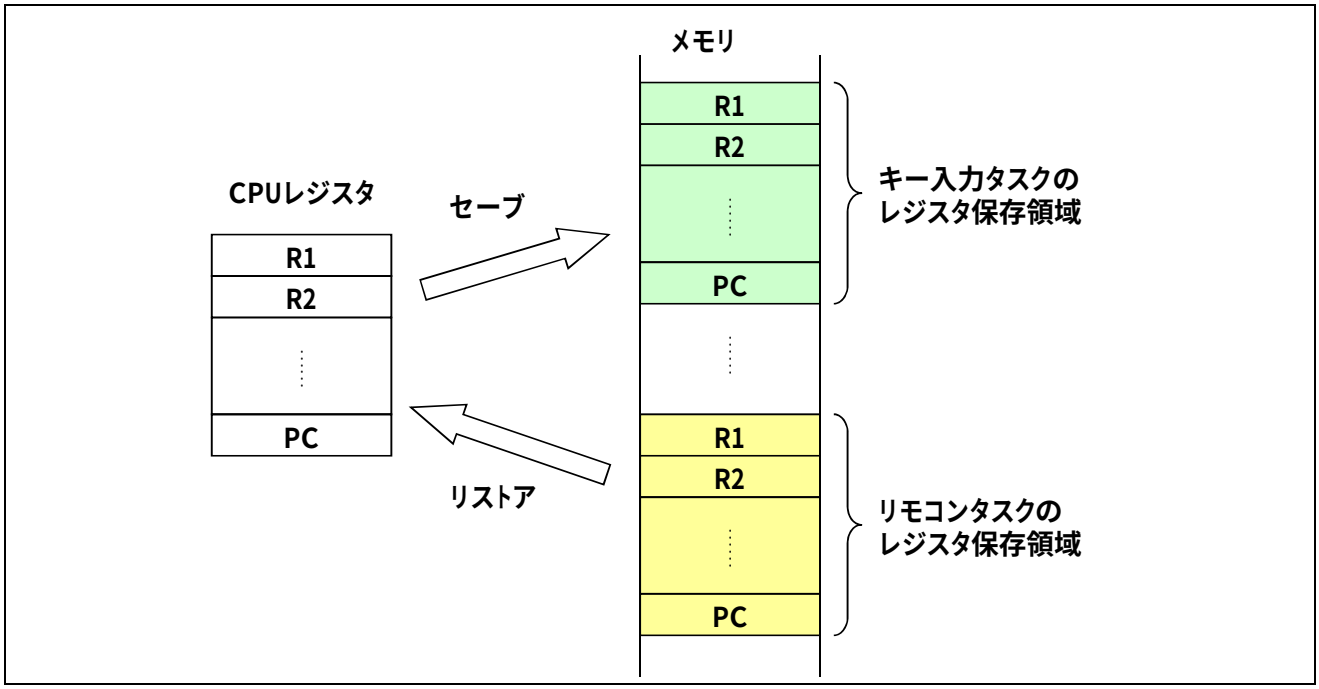

図2.3 タスクの切り替え

また、タスクが実行するには、CPU レジスタだけでなく、スタック領域も必要です。スタック領域も各タスク毎に別々の領域を使用します。

2.2 サービスコール

プログラマは、プログラム中でどのようにカーネルの機能を使用するのでしょうか？これには、カーネルの機能をプログラムから何らかの形で呼び出すことが必要です。このカーネルの機能を呼び出すことを、サービスコールといいます。すなわちサービスコールにより、タスクの起動などの処理を行うことができます(図 2.4 参照)。

図2.4 サービスコール

サービスコールは、以下のように C 言語の関数呼び出しで実現します。

```
act_tsk(ID_TASK1);
```

2.3 オブジェクト

タスクやセマフォなど、サービスコールによって操作する対象を「オブジェクト」と呼びます。

オブジェクトをカーネルに認識させる操作を「生成」と呼びます。オブジェクトの生成方法には、静的に cfg ファイルに定義する方法と、動的にサービスコール(cre_???, acre_???)で生成する方法があります。

オブジェクトは ID 番号によって識別されます。しかし、プログラム中にタスクの番号を直接書き込むと非常に可読性の低いプログラムになってしまいます。たとえば、

```
act_tsk(1);
```

とプログラム中に記述するとプログラマは絶えず ID 番号が 1 番のタスクは何かを知っている必要があります。また、他人がこのプログラムを見たときに ID 番号の 1 番のタスクが何かが一目では分かりません。

そこで RI600/PX では、cfg ファイルで静的に生成するタスクについては、そのタスクの名前 (ID 名称)を指定し、その ID 名称からタスクの ID 番号への変換を RI600/PX に付属しているプログラム"コンフィギュレータ cfg600px"が自動的に行うようになっています。具体的には、コンフィギュレータは、各タスクと ID 番号が対応づけられるように、以下のように定義されたヘッダファイル(kernel_id.h)を出力します。

```
#define ID_TASK1 1
```

図 2.5は、タスクを識別する様子を示したものです。

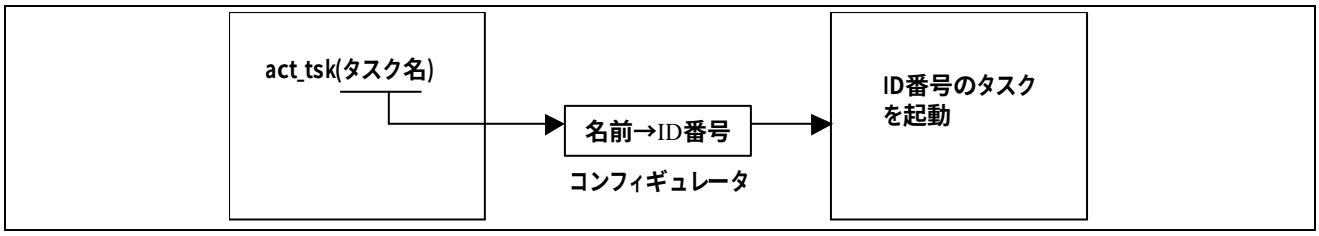

図2.5 タスクの識別

この定義を用いると、先の例は以下のように記述できます。

```
act_tsk(ID_TASK1);  /* タスク ID が"ID_TASK1"のタスクを起動 */
```

この例では、"ID_TASK1"に対応するタスクを起動するように指定しています。タスクの ID 名称から ID 番号への変換は、プログラムを生成するときにコンパイラの機能を使用することによって行うため、この機能による処理速度の低下はありません。

ここでは、タスクを例に説明しましたが、他の ID 番号で識別されるオブジェクトにも同様に ID 名称を付与することができます。

2.4 タスク

2.4.1 タスクの状態

カーネルでは、タスクを実行するべきか否かをタスクの状態を管理することにより制御しています。例えば、図 2.6にキー入力タスクの実行制御と状態の関係を示します。キー入力が発生した場合は、そのタスクを実行しなければなりません。すなわち、キー入力タスクが実行状態となります。またキー入力を待っているときはタスクを実行する必要はありません。すなわち、キー入力タスクは待ち状態になっています。

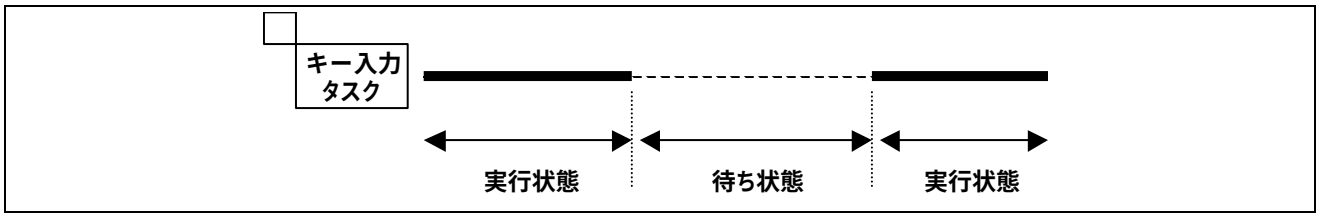

図2.6 タスクの状態

カーネルは、実行状態、待ち状態を含め、タスクを以下の 6 つの状態で管理しています。図 2.7に、タスクの状態遷移図を示します。

(1) 未登録状態(NON-EXISTENT)

カーネルに登録されていない仮想的な状態です。

(2) 休止状態(DORMANT)

カーネルに登録(生成)された後、まだ起動されていない状態、または終了後の状態です。

(3) 実行可能状態(READY)

実行するための準備がすべて整った状態ですが、他の高い優先度のタスクが実行中のため実行はできない状態です。

(4) 実行状態(RUNNING)

現在、CPU 上で実行している状態です。
カーネルは実行可能状態のタスクの中で最も高い優先度のタスクを実行状態にします。

(5) 待ち状態(WAITING)

tslp_tsk サービスコールなどを呼び出した場合、条件が満たされない場合は待ち状態になります。待ち状態は、wup_tsk サービスコールなど、待ち状態になった要因に対応するサービスコールが呼び出されると解除され、実行可能状態に遷移します。

(6) 強制待ち状態(SUSPENDED)

タスクの実行が(sus_tsk サービスコール)により強制的に中断させられた状態です。

(7) 二重待ち状態(WAITING-SUSPENDED)

待ち状態と強制待ち状態が重なった状態です。

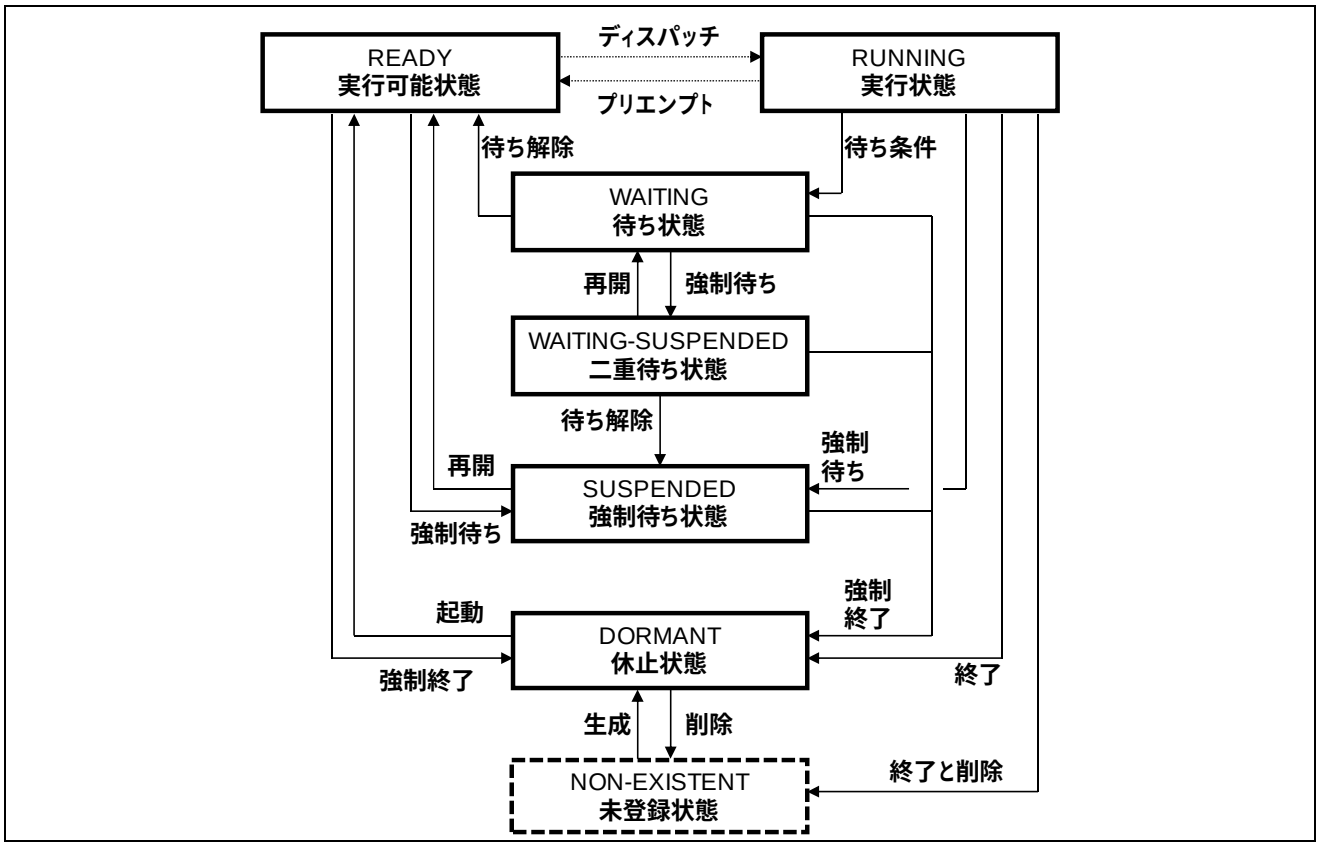

図2.7　**タスクの状態遷移図**

2.4.2 タスクのスケジューリング(優先度とレディキュー)

各タスクには、処理の優先順位を意味する「タスク優先度」が付与されます。タスク優先度は、値が小さいほど高い優先順位となり、1 が最高の優先順位です。利用可能な優先度の範囲は、1 から cfg ファイルに指定する system.priority となります。

カーネルは、実行可能状態にあるタスクの中で最も高い優先度のタスクを実行状態にします。

複数のタスクに同じ優先度を与えることもできます。

カーネルは、実行可能状態にあるタスクの中で最も高い優先度のタスクが複数存在する場合には、最も先に実行可能状態になったタスクを実行状態にします。カーネルはこれを実現するために、レディキューと呼ぶ実行可能状態のタスクの実行待ち行列を持っています。

図 2.8にレディキューの構造を示します。レディキューは優先度ごとに管理され、カーネルはタスクが接続されている最も優先度の高い待ち行列の先頭タスクを実行状態にします。

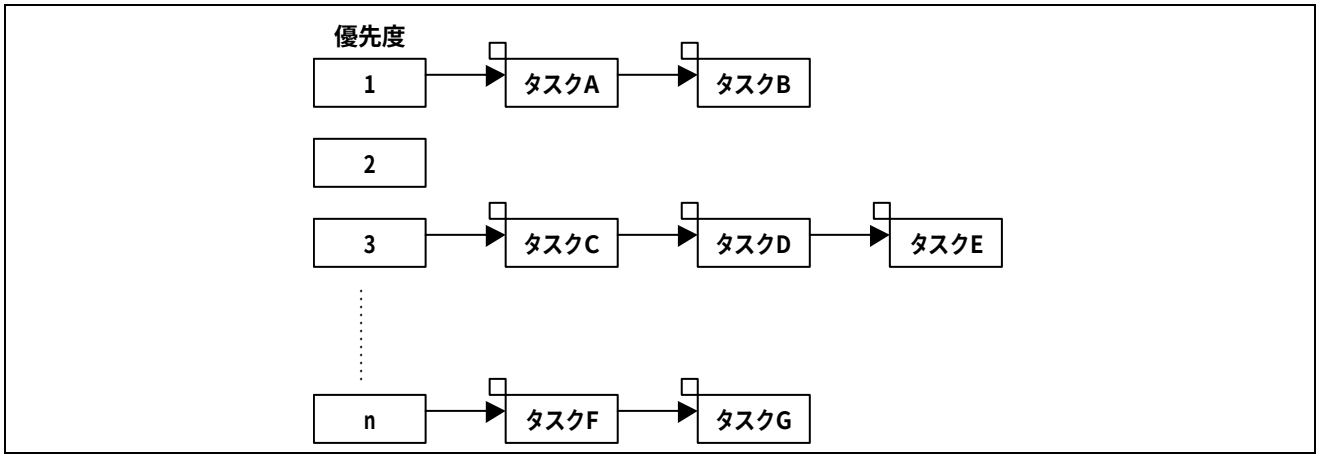

図2.8 レディキュー(実行待ち状態)

2.4.3 タスクの待ち行列

タスクは、セマフォやイベントフラグといったオブジェクトに対して、条件が満たされるまで待つ(待ち状態に遷移する)ように要求する(サービスコールを呼び出す)ことができます。

オブジェクトによっては、複数のタスクが待つ状況もあります。この場合、どういう順序で待っているタスクを管理するかを、オブジェクトを生成するときに属性によって指定することができます。具体的には、FIFO順で管理する(TA_TFIFO 属性)か、タスクの優先度順で管理する(TA_TPRI 属性)を選択することができます。

オブジェクトで待っているタスクの待ち解除は、この待ち行列の順序で行われます。

図 2.9および図 2.10に、あるオブジェクトに対して、タスク D(優先度 9)、タスク C(優先度 6)、タスク A(優先度 1)、タスク B(優先度 5)、という時間順で待ち行列に繋がれたときの様子を示します。

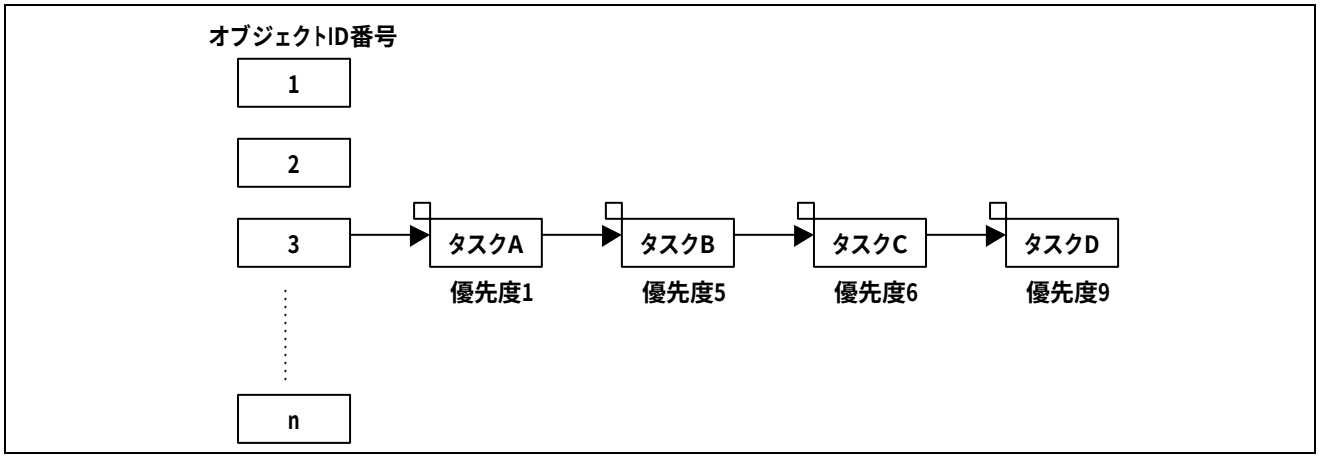

図2.9 TA_TPRI **属性の待ち行列**

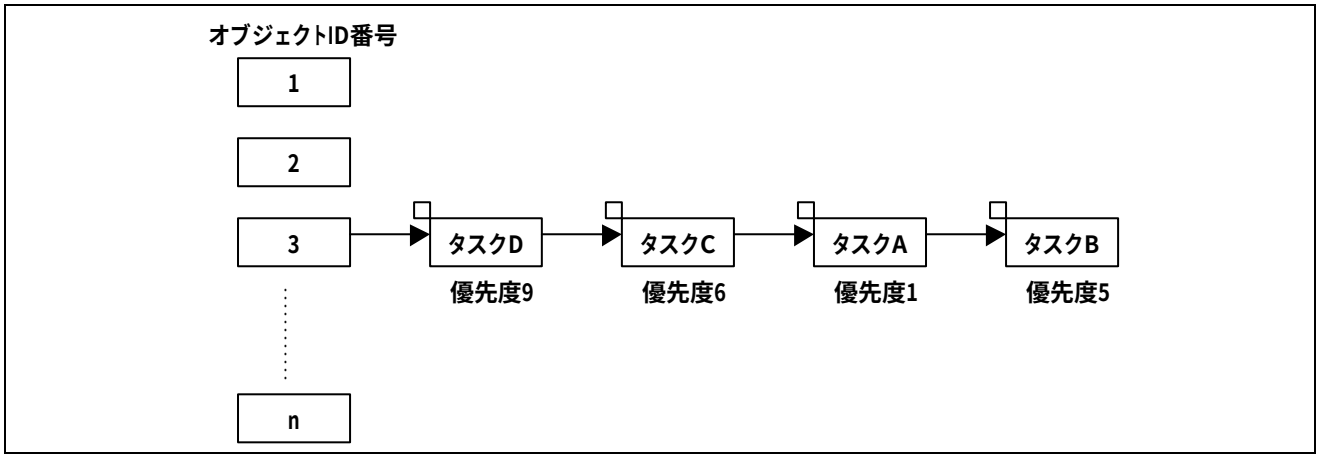

図2.10 TA_TFIFO **属性の待ち行列**

2.5 システムの状態

システムの状態は、以下の３つの直行する状態によって区別されます。

(1)タスクコンテキスト/非タスクコンテキスト

(2)ディスパッチ禁止/許可状態

(3)CPU ロック/ロック解除状態

システムの挙動や呼び出し可能なサービスコールは、これらの状態によって決まります。

2.5.1 タスクコンテキストと非タスクコンテキスト

システムは、「タスクコンテキスト」か「非タスクコンテキスト」のいずれかのコンテキスト状態で実行します。タスクコンテキストと非タスクコンテキストの違いを、表 2.1に示します。

表2.1 タスクコンテキストと非タスクコンテキスト

	タスクコンテキスト	非タスクコンテキスト
呼び出し可能なサービスコール	タスクコンテキストから呼び出しできるもの	非タスクコンテキストから呼び出しできるもの
タスクスケジューリング	2.5.2, 2.5.3項参照	発生しない

非タスクコンテキストで実行される処理には、以下があります。

- 割り込みハンドラ
- タイムイベントハンドラ（周期ハンドラ、アラームハンドラ）
- アクセス例外ハンドラ

2.5.2 ディスパッチ禁止/許可状態

システムは、ディスパッチ禁止状態か許可状態かのいずれかの状態をとります。ディスパッチ禁止状態では、タスクスケジューリングは行われません。また、待ち状態に遷移するサービスコールは呼び出しできません。

以下のサービスコールを呼び出すと、ディスパッチ禁止状態へ遷移します。

- dis_dsp
- chg_ims で割り込みマスクを 0 以外に変更

以下のサービスコールを呼び出すと、ディスパッチ許可状態に遷移します。

- ena_dsp
- ext_tsk
- exd_tsk
- chg_ims で割り込みマスクを 0 に変更

sns_dsp サービスコールでディスパッチ禁止状態かどうかを知ることができます。

ディスパッチ禁止状態は、他タスクとの間での排他制御の目的で利用することができます。しかし、ディスパッチ禁止状態の間は他のタスクは一切動作しなくなるため、システム全体の振る舞いに影響を与えます。タスク間の排他制御は、できるだけセマフォやミューテックスを利用することを推奨します。ディスパッチ禁止状態を使用する場合も、その期間はごく短時間になるようにすべきです。

2.5.3 CPU ロック/ロック解除状態

システムは、CPU ロック状態か CPU ロック解除状態かのいずれかの状態をとります。CPU ロック状態では、割り込みの受け付けが禁止され、タスクスケジューリングも行われません。ただし、カーネル管理外の割り込みは受け付けられます。また、待ち状態に遷移するサービスコールは呼び出しできません。

CPU ロック状態へは loc_cpu, iloc_cpu、解除状態へは unl_cpu, iunl_cpu, ext_tsk, exd_tsk サービスコールによって遷移します。また、sns_loc サービスコールで CPU ロック状態かどうかを知ることができます。

また、CPU ロック状態から呼び出し可能なサービスコールは、表 2.2のように制限されます。

表2.2 CPU **ロック状態で使用可能なサービスコール**

loc_cpu, iloc_cpu	unl_cpu, iunl_cpu	sns_ctx	sns_loc	sns_dsp
sns_dpn	vsta_knl, ivsta_knl	vsys_dwn, ivsys_dwn	ext_tsk, * exd_tsk *	sns_tex

【注】 CPU **ロック状態は解除されます。**

CPU ロック状態は、カーネル管理割り込み処理間、あるいはカーネル管理割り込み処理とタスクの間での排他制御の目的で利用することができます。しかし、CPU ロック状態の間は他のタスクは一切動作しなくなるだけでなく、カーネル管理割り込みも禁止されるため、システム全体の振る舞いに影響を与えます。したがって、CPU ロック状態を使用する場合、その期間はごく短時間になるようにすべきです。

2.5.4 ディスパッチ保留状態

タスクディスパッチが保留される状態のことを、ディスパッチ保留状態と呼びます。具体的には、以下のいずれかに該当する場合がディスパッチ保留状態です。

(1) 非タスクコンテキスト実行中
(2) ディスパッチ禁止状態
(3) CPUロック状態

sns_dpn サービスコールで、ディスパッチ保留状態かどうかを知ることができます。

2.6 処理の単位と優先順位

アプリケーションは、以下の処理単位によって実行制御されます。

1. タスク
   マルチタスクの制御対象となる単位です。
2. 割り込みハンドラ
   割り込みが発生したときに実行されます。
3. タイムイベントハンドラ（周期ハンドラ、アラームハンドラ）
   指定した周期や時刻になったときに実行されます。
4. アクセス例外ハンドラ
   タスクまたはタスク例外処理ルーチンが、許可されていないメモリ領域にアクセスしたときに実行されます。

また、各処理単位は以下の優先順位で処理されます。

(1) 割り込みハンドラ、タイムイベントハンドラ
(2)アクセス例外ハンドラ
(3) ディスパッチャ (カーネルの処理の一部)
(4) タスク

なお、ディスパッチャとは、実行するタスクを切り換えるカーネルの処理のことです。割り込みハンドラ、タイムイベントハンドラはディスパッチャよりも優先順位が高いため、これらが実行している間は、タスクは実行されません。

割り込みハンドラは、割り込みレベルが高いほど優先順位が高くなります。
タイムイベントハンドラの優先順位は、タイマ割り込みレベル(clock.IPL)と同じとなります。
タスク間の優先順位は、タスクに付与された優先度に従います。

2.7 割り込み

割り込みが発生すると、cfg ファイルで定義された割り込みハンドラが起動されます。

2.7.1 割り込みの種類

割り込みは、カーネル管理割り込みとカーネル管理外割り込みに分類されます。

- カーネル管理割り込み
  カーネル割り込マスクレベル (system.system_IPL)より割り込み優先レベルが低い割り込みをカーネル管理割り込みといいます。
  カーネル管理割り込みハンドラでは、サービスコールを呼び出すことができます。
  サービスコール処理中にカーネル管理割り込みが発生した場合、カーネル管理割り込みを受け付け可能となるまで割り込み受理が遅延されます。

- カーネル管理外割り込み
  カーネル割り込マスクレベル (system.system_IPL)より割り込み優先レベルが高い割り込みをカーネル管理外割り込みといいます。ノンマスカブル割り込み(NMI)は、カーネル管理外割り込みの扱いとなります。
  カーネル管理外割り込みハンドラでは、サービスコールを呼び出してはなりません。
  サービスコール処理中にカーネル管理外割り込みが発生した場合でも、直ちに割り込みが受理されるため、カーネル処理に依存しない高速な割り込み応答が可能です。

図 2.11に、カーネル割り込みマスクレベルを 10 に設定した場合の、カーネル管理割り込みとカーネル管理外割り込みの関係を示します。

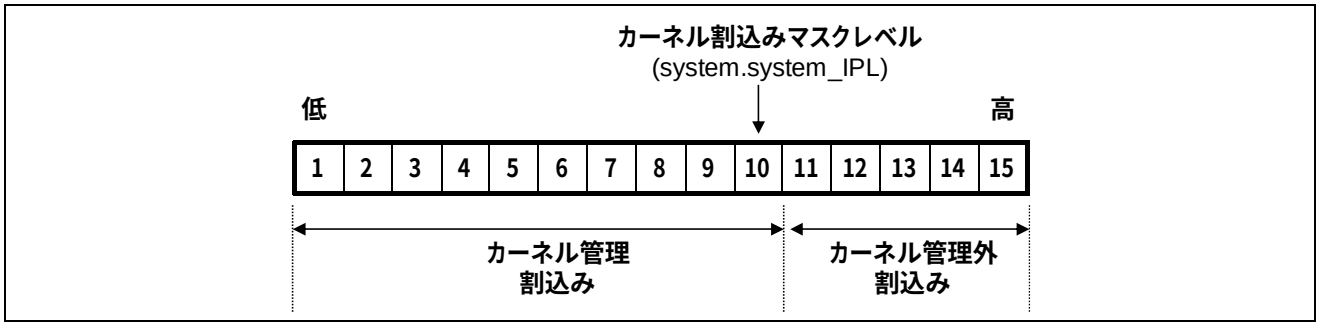

図2.11 割り込み優先レベルと割り込みの種類

2.7.2 CPU 例外の扱い

以下の CPU 例外事象は、カーネル管理外割り込みの扱いとなります。

1. 無条件トラップ(INT命令, BRK命令)(INT #1～#8はカーネル予約です)
2. 未定義命令例外
3. 特権命令例外
4. 浮動小数点例外

一方、アクセス例外ハンドラはカーネル管理割り込みの扱いとなります。

2.7.3 PSW レジスタの I ビットおよび IPL ビットの扱い

CPU 仕様では、I ビットが 0 の時はすべての割り込みがマスクされ、I ビットが 1 の時は IPL ビット値以下のレベルの割り込みがマスクされます。

アプリケーションでの I ビットおよび IPL ビットの変更については、次節も参照してください。

(1) タスク・タスク例外処理ルーチン

I ビットの初期値は 1、IPL ビットの初期値は 0 です。つまり、全割り込みが許可された状態です。

IPL ビットを変更するには、loc_cpu, chg_ims サービスコールによる方法があります。

タスクでは、I ビットを 0 にしてはなりません。なお、RI600/PX では I ビットを 0 にする方法は提供されていません。

(2) 割り込みハンドラ

I ビットの初期値は、cfg ファイルで割り込みハンドラを定義する際に多重割り込みを許可する指定(pragma_switch=E)を行った場合は 1、そうでない場合は 0 となります。

ハンドラ起動時の IPL ビットは以下となります。

- 割り込みの場合：当該割り込み優先レベル
- CPU 例外の場合：例外発生前と同じ

IPL ビットを変更するには、iloc_cpu, ichg_ims サービスコールによる方法があります。

また、I ビットと IPL ビットは割り込みハンドラが直接変更することもできます。

(3) タイムイベントハンドラ

タイムイベントハンドラ起動時の I ビットは 1 です。

タイムイベントハンドラ起動時の IPL ビットは、タイマ割り込み優先レベル(cfg ファイルの clock.IPL)です。IPL ビットを変更するには、iloc_cpu, ichg_ims サービスコールによる方法があります。

また、I ビットと IPL ビットはタイムイベントハンドラが直接変更することもできます。

(4) サービスコール内の制御

(a) I ビット
タスクコンテキストから呼び出した場合は、I ビットはサービスコール内で 0 と 1 を切り替えながら実行されます。
非タスクコンテキストから呼び出した場合は、サービスコール内では呼び出し前の I ビットの状態が維持されます。

(b) IPL ビット
サービスコール内では、IPL ビットは呼び出し前の値とカーネル割り込みマスクレベル(system.system_IPL)を切り替えながら実行されます。

(c) サービスコール終了後の状態
PSW レジスタは呼び出し前の値に戻ります。ただし、chg_ims, ichg_ims, loc_cpu, iloc_cpu, unl_cpu, iunl_cpu サービスコールでは IPL ビットが変更されます。

2.7.4 割り込みの禁止

割り込みを禁止したい場合は、以下のいずれかの方法で実現してください。

(1) 不特定の割り込みを禁止する(PSW レジスタの I, IPL ビットを変更する)

(a) CPUロック状態にする
CPUロック状態では、PSW.IPL=カーネル割り込みマスクレベル(system.system_IPL)となります。つまり、CPUロック状態で禁止される割り込みは、カーネル管理割り込みのみです。カーネル管理外割り込みを禁止したい場合は、2, 3の方法で行ってください。
また、CPUロック状態では使用可能なサービスコールに制限があります。「2.5.3 CPUロック/ロック解除状態」を参照してください。

(b) chg_ims, ichg_imsを用いて、IPLを変更する
ただし、非タスクコンテキストの場合は、起動時よりもIPLを下げてはなりません。
また、タスクコンテキストでIPLを0以外に変更するとディスパッチ禁止状態に、IPLを0にするとディスパッチ許可状態に遷移します。
IPLを0以外に変更している間は、通常はena_dspを呼び出さないようにしてください。ena_dspを行うと、その時点でディスパッチ許可状態に遷移します。ディスパッチによって、PSWはディスパッチ先のタスクの状態に更新されるので、意図せずにIPL値が下がってしまうことがあります。

(c) ハンドラ(非タスクコンテキスト)の場合は、IビットとIPLビットを直接変更することができます。ただし、起動時よりIPLを下げてはなりません。
なお、コンパイラではPSWレジスタを操作するための以下の組み込み関数が用意されています。
・set_ipl() ：PSWレジスタのIPLビットの変更
・get_ipl() ：PSWレジスタのIPLビットの参照
・set_psw() ：PSWレジスタの設定
・get_psw() ：PSWレジスタの参照

なお、タスクコンテキスト実行中に PSW.I を 0 にしてはなりません。

(2) 特定の割り込み要因を禁止する

特定の割り込み要因を禁止するには、割り込みコントローラ(ICU)の割り込み要求許可レジスタや、当該 I/O の制御レジスタなどを変更してください。

2.7.5 使用できるサービスコール

割り込みハンドラは非タスクコンテキストに分類されるため、タスクコンテキスト専用のサービスコールは使用できません。

カーネル管理外割り込みハンドラなど、PSW.IPL>カーネル割り込みマスクレベル(system.system_IPL)状態では、一部[1]を除いてサービスコールを呼び出してはなりません。呼び出した場合、エラーE_CTX を返します。ただし、この場合、サービスコール処理内で一時的に PSW.IPL がカーネル割り込みマスクレベルに下がるため、このエラーはデバッグ目的でのみ利用してください。

[1] chg_ims, ichg_ims, get_ims, iget_ims, vsta_knl, ivsta_knl, vsys_dwn, ivsys_dwn

2.7.6 RX マイコンの高速割り込み機能

RX マイコンは「高速割り込み」機能をサポートしています。ひとつの割り込み要因だけを高速割り込みとすることができます。高速割り込みは、割り込み優先レベル 15 として扱われます。高速割り込みを使用する場合は、割り込み優先レベル 15 の割り込み要因をひとつに限定する必要があります。

本カーネルで高速割り込みを使用する場合は、その割り込みはカーネル管理外割り込みとして扱う必要があります。つまり、カーネル割り込みマスクレベル(system.system_IPL)は、14 以下に設定する必要があります。

cfg ファイルで高速割り込みを定義する際には、os_int に NO を指定し、さらに pragma_switch に F を指定する必要があります。

また、アプリケーションで CPU の FINTV レジスタをそのハンドラの開始アドレスに初期化する必要があります。

2.8 スタック

スタックには、システムスタックとユーザスタックがあります。

- ユーザスタック
  タスクごとに 1 つずつ存在するスタックです。
  cfg ファイルでタスクを静的に生成する場合、スタックサイズとセクション名を指定します。
  cre_tsk または acre_tsk でタスクを動的に生成する際には、スタックサイズと先頭アドレスを指定します。

- システムスタック
  非タスクコンテキストとカーネルが使用するスタックで、システムで一つだけ存在します。システムスタックは、cfg ファイルの system.stack_size にサイズを指定することで生成されます。

2.9 メモリ保護

2.9.1 概要

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

(1) タスク・タスク例外処理ルーチンによる不正メモリアクセスの検出

タスク・タスク例外処理ルーチンは、アクセス許可された領域(メモリオブジェクト)のみにアクセスできます。許可されていない領域をアクセスすると、アクセス例外ハンドラが起動されます。

(2) ユーザスタック保護

各タスクのユーザスタックは、他のタスクからはアクセスできません。ユーザスタックがオーバフローしたり、タスクが他のタスクのユーザスタックをアクセスすると、アクセス例外ハンドラが起動されます。

(3) カーネルによる不正メモリアクセスの検出

いくつかのサービスコールは、引数としてポインタを受け取ります。カーネルは、サービスコールを呼び出したタスクがポインタで指定された領域をアクセス可能かを検査します。アクセス許可がない場合には、サービスコールはエラーE_MACV を返します。

また、いくつかのサービスコールはユーザスタックにタスクのコンテキストレジスタを保存します。このときにユーザスタックがオーバフローする場合には、サービスコールはエラーE_MACV を返します。なお、タスク・タスク例外処理ルーチン実行中のユーザスタックのオーバフローは検出されません。

この機能は、タスクコンテキストから呼び出されたサービスコールでのみ行われます。

2.9.2 ドメイン、メモリオブジェクト、アクセス許可ベクタ

メモリ保護機能は、以下を制御することで実現しています。

- だれが
- どこに対して
- どのアクセスが許可されるか

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

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

メモリオブジェクトは、通常は cfg ファイルで静的に登録しますが、動的に登録(ata_mem サービスコール)、登録解除(det_mem サービスコール)することもできます。メモリオブジェクトは、先頭アドレスが 16 バイト境界で、かつサイズが 16 バイトの整数倍でなければなりません。

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

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

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

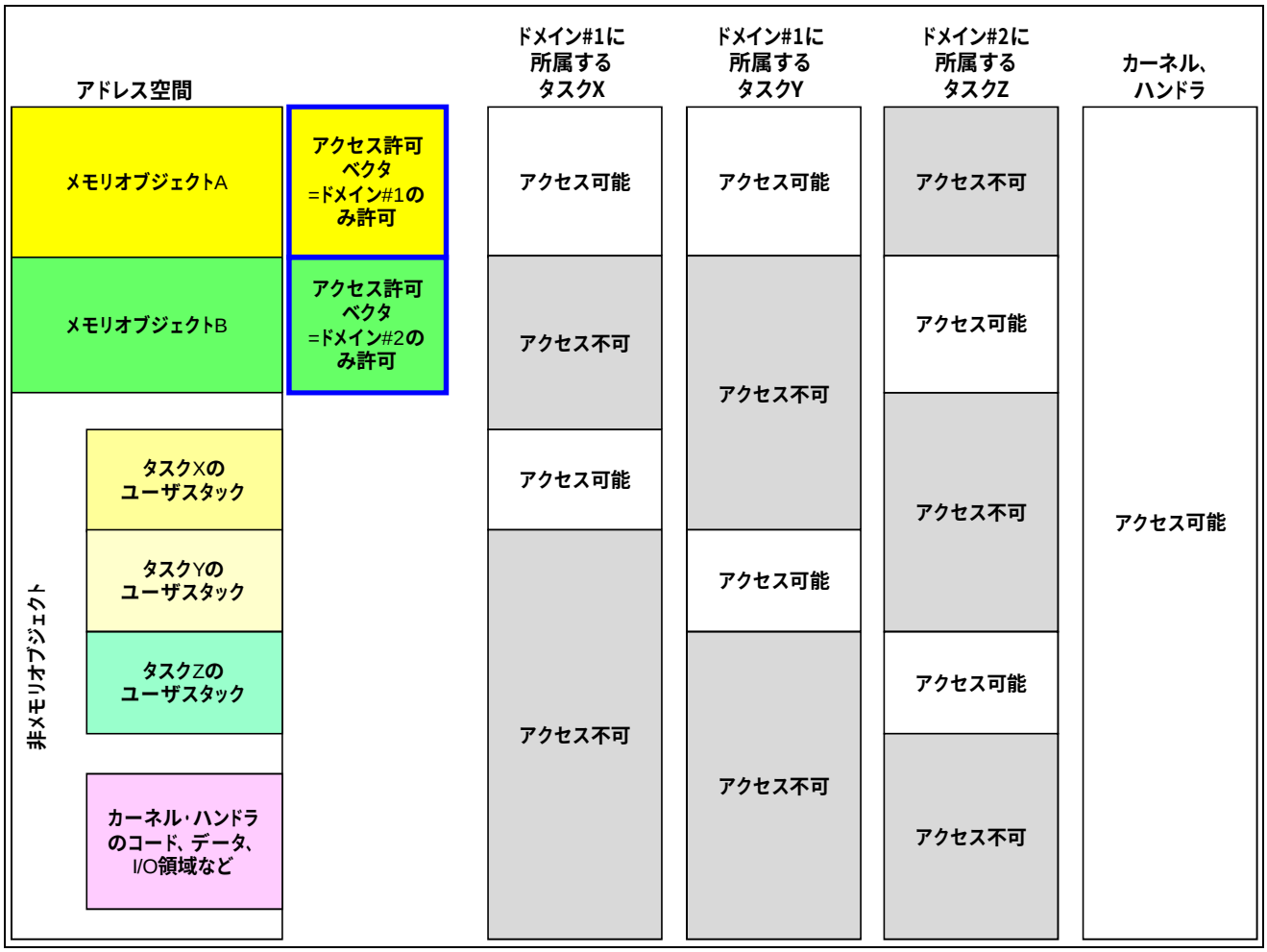

図2.12　メモリ保護の概要

表 2.3にメモリオブジェクトに関する操作をまとめます。

表2.3　メモリオブジェクト操作

操作	静的(cfg ファイル)	動的(サービスコール)
登録	memory_object[]	ata_mem
登録解除	－	det_mem
アクセス許可変更	－	sac_mem
アクセス権チェック(メモリオブジェクト以外もチェック可能)	－	vprb_mem
状態参照	－	ref_mem

2.9.3　メモリオブジェクトの数の制約

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

2.9.4 メモリオブジェクトの静的登録の留意事項

cfg ファイルの memory_object[]によってメモリオブジェクトを静的に登録するとき、メモリオブジェクトのアドレスを絶対アドレス値またはセクション名で指定することができます。

絶対アドレス指定は、I/O レジスタ領域などの指定するときに使用します。

セクション名指定では、メモリオブジェクトを構成する先頭のセクションと最後のセクション名を指定します。

セクション名を指定した場合は、リンク時に必ず想定通りのセクション配置となるように注意してください。たとえば、メモリオブジェクトの先頭は 16 バイト境界でなければならないので、メモリオブジェクトの先頭(memory_object[].start_address)に指定したセクションに対し、リンカの"aligned_section"オプションを指定してください。

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

■例

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

2.9.5 信頼されたドメイン

RI600/PX では、メモリアクセス以外の保護機能をサポートしていません。これが悪用されると、例えば以下のような不正アクセスが検出されなくなります。

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

このような悪意による不正メモリアクセスを防ぐため、RI600/PX は「信頼されたドメイン」と呼ぶ機能をサポートしています。以下のようなソフトウェア構成に変更を与えるサービスコールは、信頼されたドメインに所属するタスクからのみ呼出し可能となっています。これらのサービスコールを信頼されていないドメインから呼び出した場合は、E_OACV エラーが検出されます。

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

2.9.6 アクセス許可の変更

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

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

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

2.9.7 ユーザスタック保護

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

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

2.9.8 アクセス許可のチェック

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

2.9.9 アクセス例外ハンドラ

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

2.9.10 プロセッサモード

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

RI600/PX では、タスクコンテキストはユーザモード、非タスクコンテキストはスーパバイザモードで動作する仕様となっています。

2.9.11 MPU の有効化

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

2.9.12 メモリオブジェクト内にすべき領域

(1) タスクがアクセスする領域

タスクがアクセスできるのは、そのタスク自身のユーザスタックを除くと、アクセス許可が適切に設定されたメモリオブジェクトのみです。したがって、タスクがアクセスするプログラムセクション、定数セクション、初期化データセクション、未初期化データセクションは、メモリオブジェクト内に配置する必要があります。また、タスクで I/O 領域をアクセスする場合には、その領域もメモリオブジェクトとする必要があります。

(2) メールボックスで扱うメッセージ

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

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

(3) 固定長・可変長メモリプール領域

メモリプール領域は、メモリブロックを使用するタスクからアクセス可能なメモリオブジェクト内とする必要があります。

ただし、カーネルはメモリプール領域内に管理テーブルを生成します。カーネル管理テーブルが破壊されると、システムの正常な動作は保証されません。

- cfg ファイルで固定長メモリプールを静的に生成する場合
  固定長メモリプール領域は memorypool[].section で指定したセクションに生成されます。
  memorypool[].section を省略した場合、固定長メモリプール領域は BURI_HEAP セクションに生成されます。
- cre_mpf, acre_mpf で固定長メモリプールを動的に生成する場合
  固定長メモリプール領域はアプリケーション側で確保し、これらのサービスコールでその先頭アドレスを指定します。
- cfg ファイルで可変長メモリプールを静的に生成する場合
  可変長メモリプール領域は variable_memorypool[].mpl_section で指定したセクションに生成されます。
  variable_memorypool[].mpl_section を省略した場合、可変長メモリプール領域は BURI_HEAP セクションに生成されます。
- cre_mpl, acre_mpl で可変長メモリプールを動的に生成する場合
  可変長メモリプール領域はアプリケーション側で確保し、これらのサービスコールで先頭そのアドレスを指定します。

2.9.13 メモリオブジェクト外にすべき領域

(1) BURI_HEAP 以外の RI600/PX のセクション

BURI_HEAP 以外の RI600/PX のセクションはカーネルのみがアクセスするため、メモリオブジェクト外でなければなりません。「7.4.2 RI600/PX のセクション一覧」を参照してください。

(2) タスクのユーザスタック領域

タスクのユーザスタック領域はメモリオブジェク外でなければなりません。他のユーザスタックおよびメモリオブジェクトと重なっていた場合、システムの正常な動作は保証されません。

- cfg ファイルでタスクを静的に生成する場合
  ユーザスタック領域は task[].stack_section で指定したセクションに生成されます。task[].stack_section を省略した場合、ユーザスタック領域は SURI_STACK セクションに生成されます。
- cre_tsk, acre_tsk でタスクを動的に生成する場合
  スタック領域はアプリケーション側で確保し、これらのサービスコールでそのアドレスを指定します。

(3) データキュー領域

通常は、データキュー領域はメモリオブジェクト・ユーザスタック以外の領域としてください。メモリオブジェクト内にデータキュー領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤ってデータキュー領域を書き換えてしまう危険があります。

- cfg ファイルでデータキューを静的に生成する場合
  データキュー領域は RI600/PX の BRI_RAM セクションに生成されます。
- cre_dtq, acre_dtq でデータキューを動的に生成する場合
  データキュー領域はアプリケーション側で確保し、これらのサービスコールでそのアドレスを指定します。

(4) メッセージバッファ領域

通常は、メッセージバッファ領域はメモリオブジェクト・ユーザスタック以外の領域としてください。メモリオブジェクト内にメッセージバッファ領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤ってメッセージバッファ領域を書き換えてしまう危険があります。

- cfg ファイルでメッセージバッファを静的に生成する場合
  メッセージバッファ領域は message_buffer[].mbf_section で指定したセクションに生成されます。
  message_buffer[].mbf_section を省略した場合、メッセージバッファ領域は BRI_RAM セクションに生成されます。
- cre_mbf, acre_mbf でメッセージバッファを動的に生成する場合
  メッセージバッファ領域はアプリケーション側で確保し、これらのサービスコールでそのアドレスを指定します。

(5) 固定長メモリプール管理領域

通常は、固定長メモリプール管理領域はメモリオブジェクト・ユーザスタック以外の領域を指定してください。メモリオブジェクト内に固定長メモリプール管理領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤って固定長メモリプール管理領域を書き換えてしまう危険があります。

- cfg ファイルで固定長メモリプールを静的に生成する場合
  固定長メモリプール管理領域は、RI600/PX の BRI_RAM セクションに生成されます。
- cre_mpf, acre_mpf で固定長メモリプールを動的に生成する場合
  固定長メモリプール管理領域はアプリケーション側で確保し、これらのサービスコールでそのアドレスを指定します。

2.10 カーネルのアイドリング

実行可能状態のタスクが存在しなくなると、カーネル内部で無限ループとなり、割り込みが発生するのを待ちます。

2.11 タスクをスーパバイザモードで動作させるには

タスクをスーパバイザモードで動作させることはできません。
スーパバイザモードで実行させたい処理は、INT 命令の割り込みハンドラとして実装してください。
例えば、WAIT 命令は特権命令なので、スーパバイザモードで実行する必要があります。
なお、INT #1~8 は、RI600/PX で予約されているので、使用しないでください。

3. カーネルの機能

本章では、主にカーネルサービスコールの機能やその使い方について解説します。

3.1 モジュール構成

カーネルは、図 3.1に示すモジュールから構成されています。これらの個々のモジュールはそれぞれのモジュールの機能を実現する関数群より構成されています。カーネルはライブラリ形式で提供され、システム生成時に必要な機能のみがリンクされます。すなわち、これらのモジュールを構成する関数群の中で使用している関数のみをリンケージエディタの機能によりリンクします。ただし、スケジューラとタスク管理の一部および時間管理の一部は必須機能関数ですので常時リンクされます。

アプリケーションプログラムはユーザが作成するプログラムで、タスク、割り込みハンドラ、周期ハンドラ、アラームハンドラ、およびアクセス例外ハンドラから構成されます。

図3.1 カーネルの構成

3.2 モジュール概要

(1) スケジューラ

タスクの持つ優先度に基づいて、タスクの実行待ち行列(レディキュー)を形成し、その待ち行列の先頭にある優先度の高い (優先度の値の小さい)タスクの処理を実行するよう制御を行います。

(2) タスク管理機能

タスクの生成、削除、起動、終了、優先度の変更等のタスク操作を行う機能です。

(3) タスク付属同期機能

タスク間の同期をとるために、タスクを待ち状態 (もしくは強制待ち状態・二重待ち状態)にしたり、待ち状態になったタスクを起床させたりする機能です。

(4) タスク例外処理機能

タスク例外処理ルーチンの定義、タスク例外の発生等を行う機能です。

(5) 同期・通信機能

タスクとは独立したオブジェクトを用いて、タスク間の同期をとったり通信を行ったりするための機能です。以下の 6 つの機能モジュールが用意されています。

- セマフォ
  セマフォは、複数のタスクで共有する装置や共有変数といった資源の競合を防ぐためのオブジェクトです。
- イベントフラグ
  ビットパターンの AND/OR 条件に応じてタスクの実行制御を行うオブジェクトです。
- データキュー
  1 ワード(32 ビット)のデータ通信を行うオブジェクトです。
- メールボックス
  任意長のメッセージを、ポインタを受け渡すことで通信するオブジェクトです。
- ミューテックス
  ミューテックスは、排他制御を行うためのオブジェクトです。優先度逆転現象を回避する機能をサポートしています。
- メッセージバッファ
  任意長のメッセージを、コピーによって通信するオブジェクトです。

(6) 固定長メモリプール

あらかじめ定めた固定サイズのメモリの動的な獲得・解放を行うオブジェクトです。

(7) 可変長メモリプール

任意サイズのメモリの動的な獲得・解放を行うオブジェクトです。

(8) 割り込み管理機能

割り込みハンドラからの復帰処理を行います。

(9) 時間管理機能

カーネルが使用するタイマの設定、タイムアウト処理、タイムイベントハンドラの起動を行う機能です。

(10) システム状態管理機能

システム状態を変更・参照する機能です。

(11) システム構成管理機能

カーネルの構成情報を取得する機能です。

(12) オブジェクトリセット機能

データキュー、メールボックス、メッセージバッファ、固定長メモリプール、および可変長メモリプールを初期状態に戻す機能です。

本機能は、μITRON4.0 仕様外の機能です。

(13) メモリオブジェクト管理機能

メモリオブジェクトおよびユーザスタックを保護する機能です。

3.3 タスク管理機能

タスク管理機能とは、タスクの生成、削除、起動、終了、優先度の変更等のタスク操作を行う機能です。
タスクを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに task[]を記述します。
- 動的な生成
  cre_tsk または acre_tsk サービスコールを使用します。

生成時には、以下の情報を指定します。

- タスクの開始アドレス
- ユーザスタックサイズ
- ユーザスタック領域の先頭アドレス(動的生成)、またはユーザスタック領域を配置するセクション(静的生成)
- 初期優先度
- 初期起動(TA_ACT 属性)の指定
- 拡張情報
- タスクが所属するドメイン ID

タスク管理機能のサービスコールには、次のものがあります。

(1) タスクを生成する(cre_tsk, acre_tsk)

cre_tsk は指定した ID で、acre_tsk は空いている ID をカーネルが割り当てて、タスクを生成します。生成されたタスクは休止状態となりますが、TA_ACT 属性を指定した場合は、生成と同時に起動され、実行可能状態となります。TA_ACT 属性を指定して起動されたタスクには、生成時に指定した拡張情報が渡されます。

これらのサービスコールでは、ユーザスタック領域の先頭アドレスとサイズを指定します。ユーザスタック領域はアプリケーション側で、16 バイト境界アドレスから 16 の整数倍のサイズを確保する必要があります。

タスクのユーザスタック領域はメモリオブジェク外でなければなりません。他のユーザスタックおよびメモリオブジェクトと重なっていた場合、システムの正常な動作は保証されません。

cre_tsk, acre_tsk は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) タスクを削除する(del_tsk)

指定された ID のタスクを削除します。削除された ID は、以降 cre_tsk または acre_tsk で生成する際に再利用できるようになります。

del_tsk は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) タスクを起動する(act_tsk, iact_tsk)

指定された ID のタスクを起動します。本サービスコールは sta_tsk, ista_tsk とは異なり、起動要求は蓄積されますが、対象タスクに渡る起動コードを指定することはできません。対象タスクには、そのタスクを生成するときに指定した拡張情報が渡されます。

(4) 蓄積していたタスク起動要求を無効にする(can_act, ican_act)

指定された ID のタスクに蓄積されていた起動要求を無効にします。

(5) タスクを起動する(起動コード指定)(sta_tsk, ista_tsk)

指定された ID のタスクを起動します。本サービスコールは act_tsk, iact_tsk とは異なり、起動要求は蓄積されませんが、対象タスクに渡る起動コードを指定することができます。

(6) 自タスクを終了する(ext_tsk)

自タスクを終了させ、休止状態に移行させます。起動要求が蓄積されている場合は、再度タスクの起動処理を行います。その際、自タスクはリセットされたように振る舞います。

タスクの開始関数からリターンした場合は、本サービスコールと同じ振る舞いになります。

(7) 自タスクを終了・削除する(exd_tsk)

自タスクを終了させ、さらに削除します。削除された ID は、以降 cre_tsk または acre_tsk で生成する際に再利用できるようになります。

(8) 他タスクを強制終了させる(ter_tsk)

休止状態以外の他タスクを終了させ、休止状態に移行させます。起動要求が蓄積されている場合は、再度タスクの起動処理を行います。

(9) タスクの優先度を変更する(chg_pri, ichg_pri)

指定された ID のタスクの優先度を変更します。

タスクの優先度を変更すると、そのタスクが実行可能状態または実行状態であるときは、レディキューも更新されます(図 3.2参照)。

また、対象タスクが TA_TPRI 属性を持つオブジェクトの待ち行列につながれている場合は、待ち行列も更新されます(図 3.3参照)。

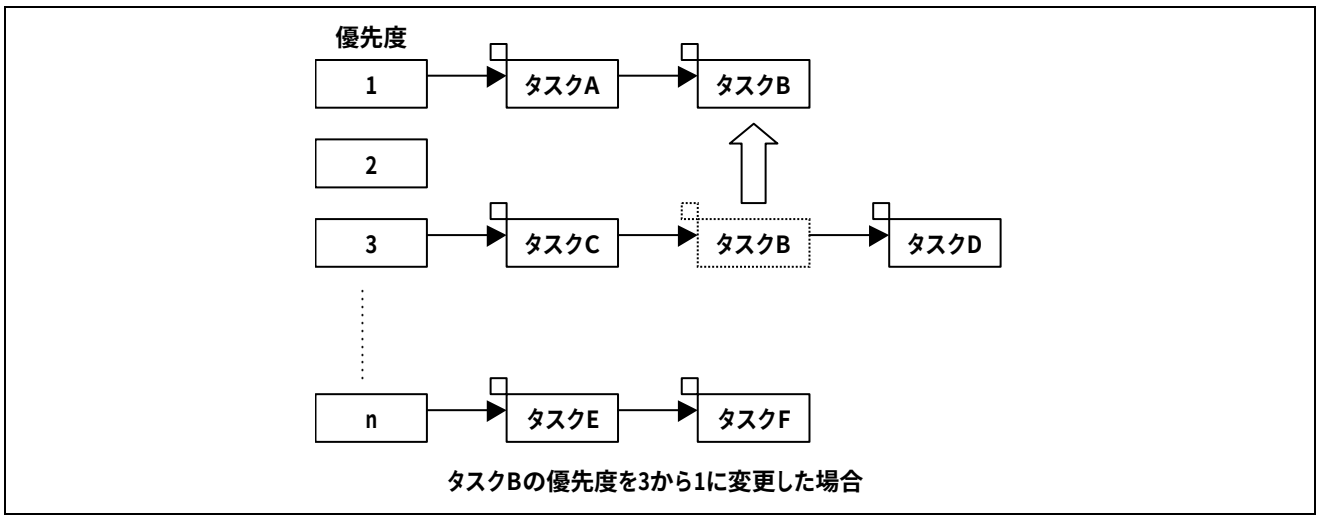

図3.2 実行可能状態または実行状態のタスクの優先度を変更した時のレディキュー

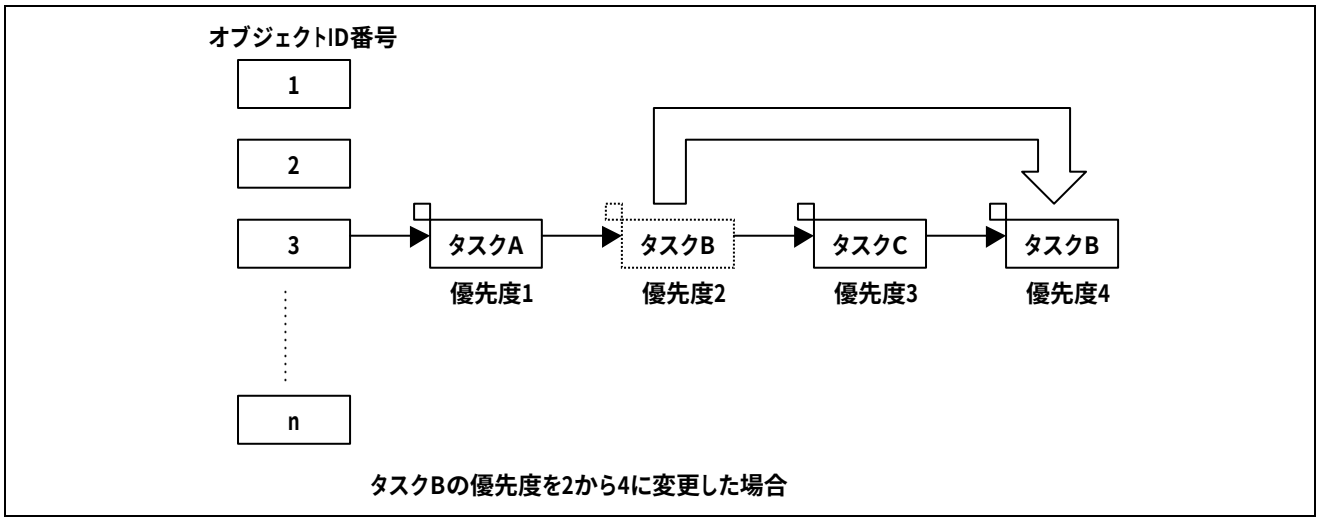

図3.3　TA_TPRI 属性のオブジェクト待ち状態のタスクの優先度を変更した時の待ち行列

一般には、優先度の変更はシステム全体の振る舞いに影響を与えるため、本サービスコールを使用しないでシステム設計を行うことが推奨されます。

なお、タスクの優先度には「ベース優先度」と「現在優先度」の 2 つがあります。通常は、この 2 つは同じですが、ミューテックスをロックしている間だけ異なります。詳細は、「3.10 ミューテックス」を参照してください。

(10) タスクの優先度を取得する(get_pri, iget_pri)

指定された ID のタスクの優先度を取得します。

(11) タスクの状態を参照する(ref_tsk, iref_tsk)

指定された ID のタスクの状態を参照します。

(12) タスクの状態を参照する(簡易版)(ref_tst, iref_tst)

指定された ID のタスクの状態を参照します。ref_tsk, iref_tsk と比べ、参照する情報が少ないためにオーバヘッドが小さいのが特徴です。

3.4 タスク付属同期機能

タスク付属同期機能とは、タスク間の同期をとるためにタスクを待ち状態 (もしくは強制待ち状態・二重待ち状態)にしたり、待ち状態になったタスクを起床させたりする機能です。

タスク付属同期機能のサービスコールには、次のものがあります。

(1) タスクのスリープ(slp_tsk, tslp_tsk)と、その起床(wup_tsk, iwup_tsk)

slp_tsk は、自タスクをスリープさせます。スリープしたタスクは WAITING 状態になります。

tslp_tsk は、slp_tsk に対してタイムアウトを指定できるようにしたサービスコールです。

wup_tsk および iwup_tsk は、スリープしているタスクを起床します。起床されたタスクは、WAITING 状態が解除されます。指定したタスクがスリープしていない場合は、起床要求が蓄積されます。起床要求が蓄積されているタスクが slp_tsk または tslp_tsk を呼び出すと、起床要求が減算(-1)されるだけで待ち状態には移行しません(図 3.4参照)。

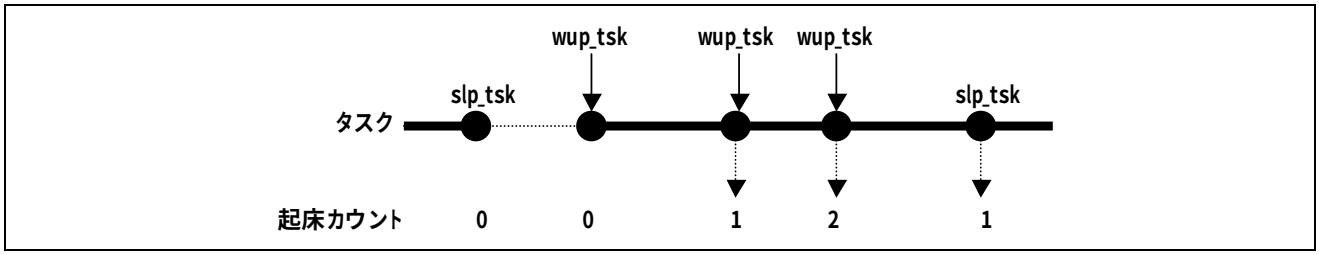

図3.4 起床要求の蓄積

(2) タスクの起床要求を無効にする(can_wup, ican_wup)

指定された ID のタスクに蓄積されていた起床要求をクリアします(図 3.5参照)。

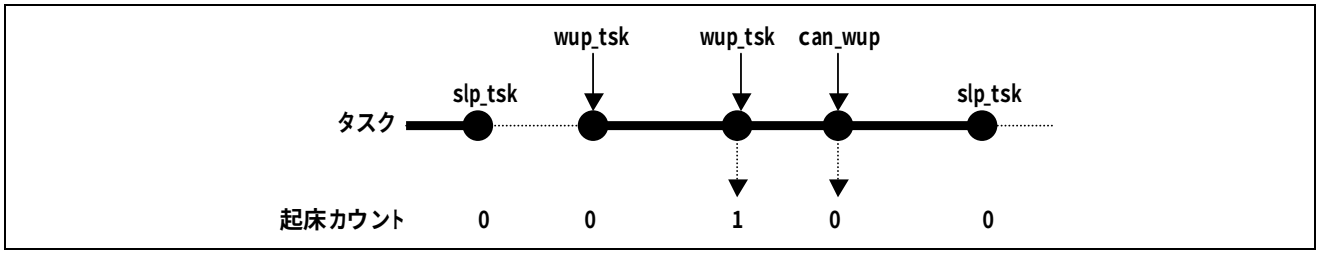

図3.5 起床要求のキャンセル

(3) 強制待ち要求(sus_tsk, isus_tsk)と、その再開(rsm_tsk, irsm_tsk, frsm_tsk, ifrsm_tsk)

sus_tsk, isus_tsk は、指定された ID のタスクの実行を強制的に中断させます。対象タスクは、実行状態または実行可能状態の場合は強制待ち状態に、待ち状態の場合は二重待ち状態に移行します。

強制待ち要求は 1 回だけ記憶されます。強制待ち状態のタスクに sus_tsk, isus_tsk を行うと、エラーE_QOVR となります。

rsm_tsk, irsm_tsk は、指定された ID のタスクの強制待ち要求ネストを減算(-1)し、対象タスクは元の状態に戻ります(図 3.6参照)。

frsm_tsk, ifrsm_tsk は、指定された ID のタスクの強制待ち要求ネストを 0 にし、対象タスクは元の状態に戻ります。

本カーネルでは前述のように強制待ち要求は 1 回だけ記憶される仕様なので、frsm_tsk, ifrsm_tsk は rsm_tsk, irsm_tsk と同じ振る舞いとなります。

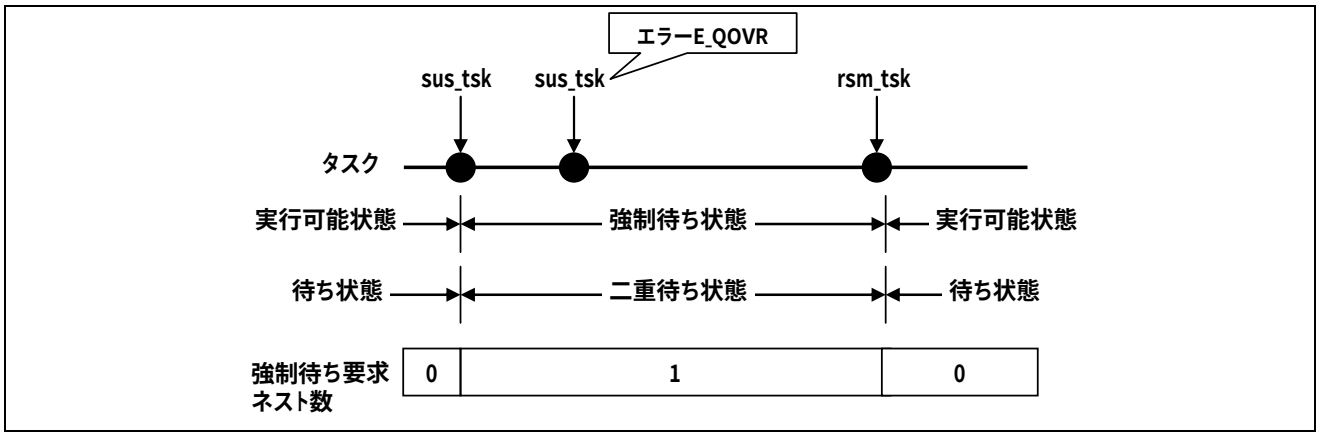

図3.6 タスクの強制待ちと再開

(4) タスクの待ち状態を強制解除する(rel_wai, irel_wai)

本サービスコールは、指定された ID のタスクの待ち状態を強制的に解除します。なお、本サービスコールで強制待ちを解除することはできません。

(5) 自タスクの一定時間待ち状態に移行する(dly_tsk)

自タスクを一定時間待たせます。自タスクは待ち状態に移行します。

3.5 タスク例外処理機能

タスクに対してタスク例外を要求すると、タスク例外処理ルーチンが起動されます。

タスク例外処理ルーチンは、タスク毎に定義できます。タスク例外処理ルーチンを定義するには以下の方法があります。

- 静的な定義
  cfg ファイルの task[].texrtn を記述します。
- 動的な定義
  def_tex サービスコールを使用します。

定義時には、以下の情報を指定します。

- タスク例外処理ルーチンの開始アドレス

以下のすべての条件が満たされたときに、該当タスクの処理を一時中断して、代わりにタスク例外処理ルーチンが起動されます。

1. 当該タスクはタスク例外許可状態である。
2. 該当タスクの保留例外要因が0でない。
3. 非タスクコンテキストが実行されていない。
4. 該当タスクは実行状態である。

例外要因はビットパターンで表現されます。タスク例外を要求(ras_tex, iras_tex)すると、当該タスクの保留例外パターンは指定された例外要因パターンとの論理和に更新されます。

タスク例外処理ルーチン起動時には、タスク例外禁止状態に変更され、保留例外要因は 0 クリアされます。タスク例外処理ルーチンには、引数として 0 クリア前の保留例外要因と、タスクの拡張情報が渡されます。

タスク例外処理ルーチンが終了すると、タスク例外許可状態に変更され、そのタスクは中断していた地点から実行を再開します。

タスク起動直後は、タスク例外禁止状態です。タスク例外を許可するには、該当タスクが明示的に ena_tex サービスコールを呼び出す必要があります。

- タスク例外禁止状態にする操作
  (1) タスクの起動
  (2) dis_tex サービスコール
  (3) タスク例外処理ルーチンの起動
  (4) def_tex サービスコールによるタスク例外処理ルーチンの定義解除
- タスク例外処理許可状態にする操作
  (1) ena_tex サービスコール
  (2) タスク例外処理ルーチンの終了

前述のタスク例外処理ルーチンの 4 つの起動条件のうち、「4. 該当タスクは実行状態である。」は、そのタスクにスケジューリングされることを意味します。これ以外の条件が満たされる可能性のある操作を、表 3.1 にまとめます。

表3.1 タスク例外処理ルーチン起動条件

起動条件を満たす可能性のある操作	起動条件		
	1. 該当タスクはタスク例外許可状態である。	2, 該当タスクの保留例外要因が 0 でない。	3. 非タスクコンテキストが実行されていない。
ras_tex, iras_tex		○	
ena_tex	○		
タスク例外処理ルーチンの終了	○		
割り込みハンドラの終了			○

タスク例外処理機能のサービスコールには、次のものがあります。

(1) タスク例外処理ルーチンを定義・定義解除する(def_tex)

指定したタスク ID のタスク例外処理ルーチンを定義します。定義を解除することもできます。
def_tex は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) タスク例外を要求する(ras_tex, iras_tex)

指定したタスクに対し、指定した例外要因パターンでタスク例外を要求します。

(3) タスク例外を禁止する(dis_tex)

自タスクのタスク例外を禁止します。

(4) タスク例外を許可する(ena_tex)

自タスクのタスク例外を許可します。

(5) タスク例外処理禁止状態を調べる(sns_tex)

自タスクのタスク例外が禁止されているかどうかを調べます。

(6) タスク例外処理状態を参照する(ref_tex, iref_tex)

指定されたタスクのタスク例外処理状態を参照します。

3.6 セマフォ

3.6.1 機能解説

セマフォは、複数のタスクで共有する装置や共有変数といった資源の競合を防ぐためのオブジェクトです。例えば、ある共有変数に対してタスク A が更新中にタスクスイッチが発生し、タスク B が共有変数を参照すると、タスク B は更新途中の不正な状態の共有変数を読み出してしまう可能性があります。セマフォを使用することで、このような競合を回避することができます。

セマフォは、このような資源の有無や数をカウンタで表現することで排他制御や同期機能を提供するオブジェクトです。

セマフォと実際に排他制御したい資源を対応付けるのは、アプリケーション側の責任です。

セマフォを使用した排他制御を行うために重要なことは、以下のルールを守ることです。

(1) 資源を使用する前にセマフォを獲得する
(2) 資源の使用を終えたらセマフォを解放する

セマフォを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに semaphore[]を記述します。
- 動的な生成
  cre_sem または acre_sem サービスコールを使用します。

生成時には、以下の情報を指定します。

- 待ち行列の並び方(TA_TFIFO(FIFO 順)または TA_TPRI(タスク優先度順))
- セマフォの資源数初期値
- セマフォの資源数最大値

図 3.7にセマフォの動作例を示します。

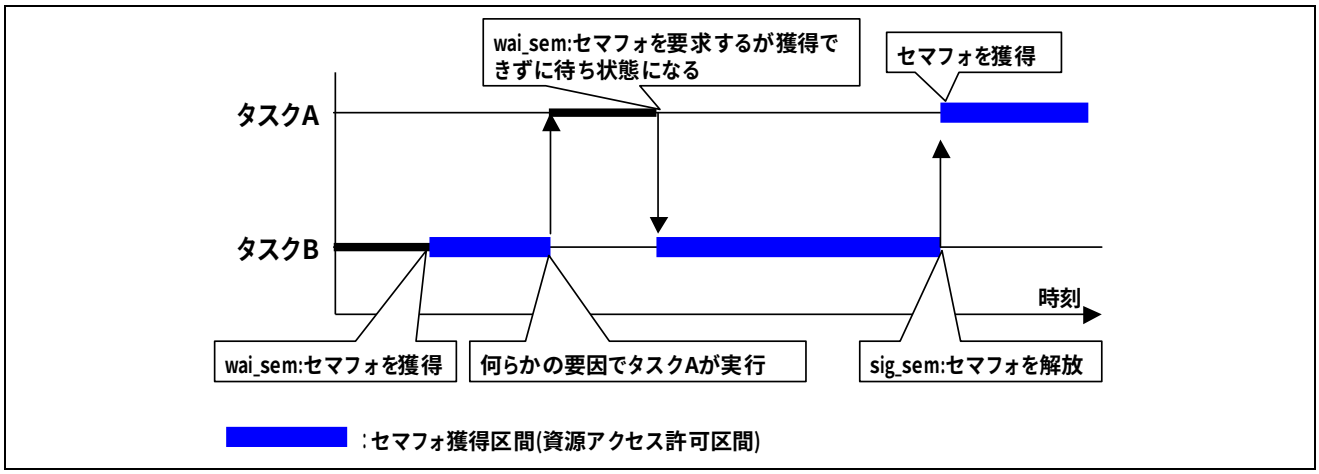

図3.7 セマフォの動作例

セマフォ機能のサービスコールには、次のものがあります。

(1) セマフォを生成する(cre_sem, acre_sem)

cre_sem は指定した ID で、acre_sem は空いている ID をカーネルが割り当てて、セマフォを生成します。
cre_sem, acre_sem は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) セマフォを削除する(del_sem)

指定された ID のセマフォを削除します。削除された ID は、以降 cre_sem または acre_sem で生成する際に再利用できるようになります。
del_sem は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) セマフォを獲得する(wai_sem, twai_sem)

セマフォを獲得します。セマフォの資源数が正なら、セマフォの資源数を 1 減算します。セマフォの資源数が 0 の場合は、セマフォを得ることができないので、本サービスコールは呼び出しタスクを待ち状態に移行させます。

(4) セマフォを獲得する(ポーリング)(pol_sem, ipol_sem)

セマフォを獲得します。wai_sem, twai_sem と異なるのは、セマフォの資源数が 0 の場合に、待ち状態にはならずに直ちにエラーリターンする点です。

(5) セマフォを返却する(sig_sem, isig_sem)

セマフォを返却します。本サービスコールは、セマフォを待っているタスクがあればそのタスクの待ち状態を解除し、なければセマフォの資源数を 1 加算します。

(6) セマフォの状態を参照する(ref_sem, iref_sem)

セマフォの資源数や待ちタスク ID を参照します。

3.6.2 優先度逆転問題

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

この様子を図 3.8に示します。この図では、タスク A とタスク C は同じ資源を使用し、タスク B はその資源を使用しない例になっています。タスク A は資源を使用するためにセマフォを獲得しようとしますが、既にタスク C がセマフォを獲得しているので待ち状態になります。ところが、タスク C がセマフォを解放する前に、優先度がタスク C よりも高くてタスク A より低く、かつ資源とは関係のないタスク B が実行すると、タスク C によるセマフォの解放はタスク B の実行によって遅れ、その結果タスク A がセマフォを獲得するのも遅れます。タスク A の立場では、資源競合しておらず、かつ自分より優先度の低いタスク B が優先的に実行されてしまうことになります。

この問題を回避するには、セマフォではなく、ミューテックスを使用してください。

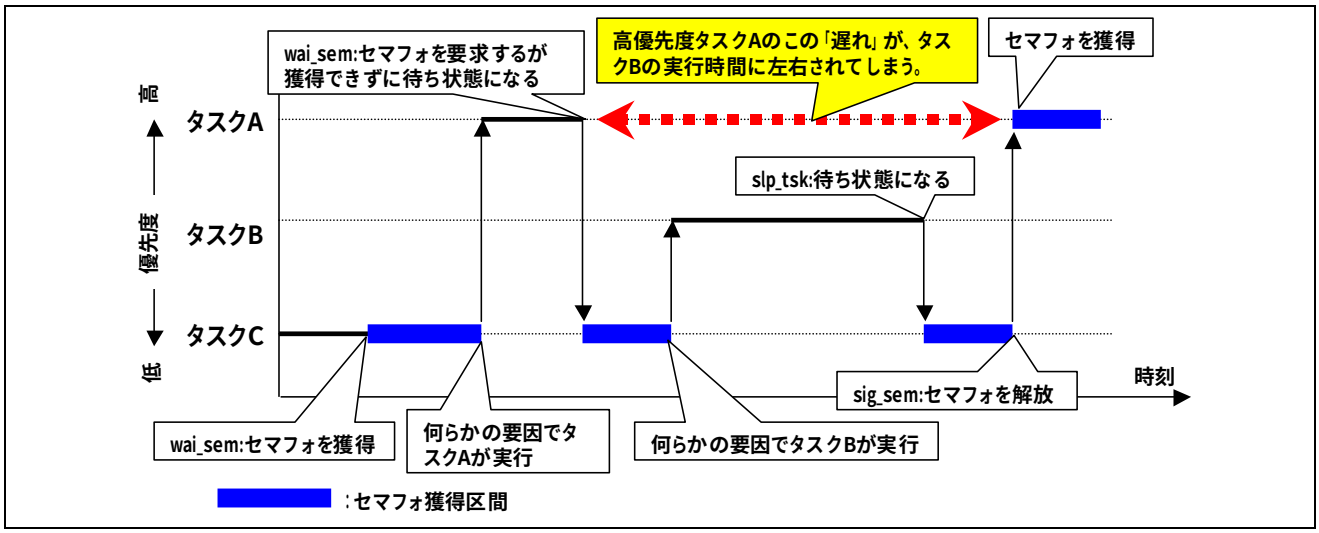

図3.8 優先度逆転現象

3.7 イベントフラグ

イベントフラグは、事象に対応したビットの集合で、1 つの事象が 1 ビットで表わされます。タスクは、イベントフラグの指定したビット郡のすべて(AND 条件)またはいずれか(OR 条件)がセットされるのを待つことができます。

また、複数のタスクが事象の発生を待つことができるかどうかを選択することができます。

- TA_WMUL 属性(複数タスクの待ち可能)
- TA_WSGL 属性(複数タスクの待ち禁止)

また、TA_CLR 属性を指定することにより、タスクがイベントフラグ待ち条件を満たしたときにイベントフラグを 0 クリアすることができます。

イベントフラグには、複数のタスクを同時に待ち解除できるという特徴があります。複数タスクの同時待ち解除を行うには、TA_WMUL 属性を指定し、かつ TA_CLR 属性を指定しないようにします。この場合の動作例を、図 3.9に示します。

図 3.9では、タスク A からタスク F までの 6 個のタスクが待ち行列につながれています。そして、set_flg によって、フラグパターンを 0x0F にすると、待ち条件を満たすタスクが待ち行列の前から順に外されていきます。この図では、タスク A、タスク C、タスク E が該当します。

もし、このイベントフラグに TA_CLR 属性が指定されていれば、タスク A の待ちが解除された時点でイベントフラグは 0 クリアされるため、タスク C、タスク E は待ち解除されません。

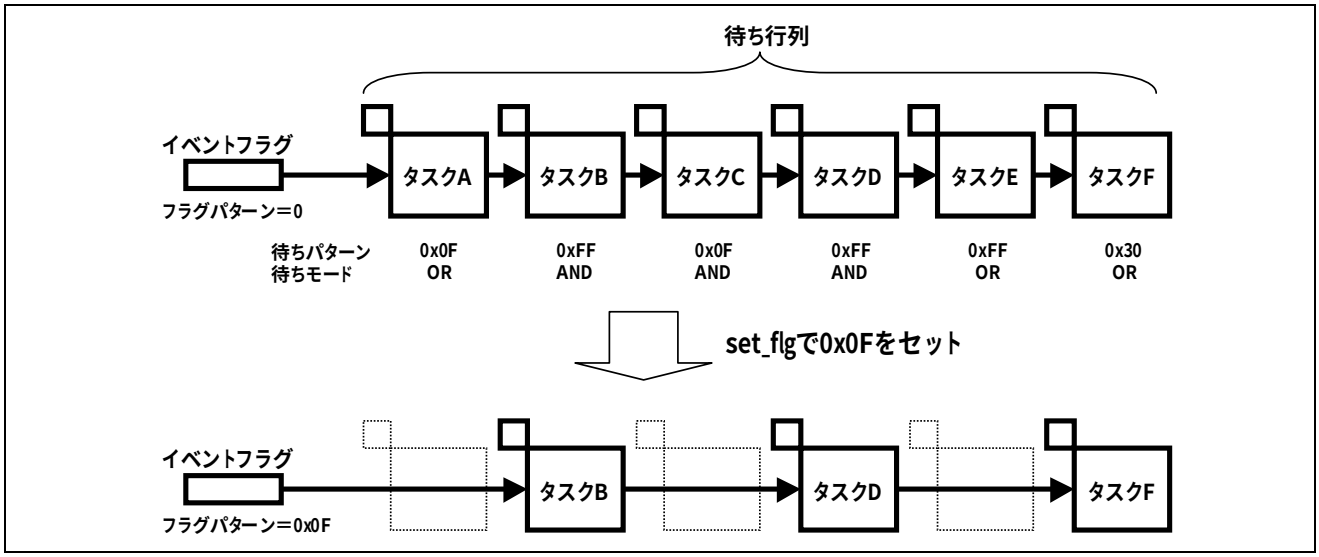

図3.9 イベントフラグの動作例

イベントフラグを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに flag[]を記述します。
- 動的な生成
  cre_flg または acre_flg サービスコールを使用します。

生成時には、以下の情報を指定します。

- 待ち行列の並び方(TA_TFIFO(FIFO 順)または TA_TPRI(タスク優先度順))
- イベントフラグ初期値
- 複数タスクの待ちを許す(TA_WMUL)か否か(TA_WSGL)
- 待ち解除時にイベントフラグをクリア(TA_CLR)

イベントフラグ機能のサービスコールには、次のものがあります。

(1) イベントフラグを生成する(cre_flg, acre_flg)

cre_flg は指定した ID で、acre_flg は空いている ID をカーネルが割り当てて、イベントフラグを生成します。
cre_flg, acre_flg は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) イベントフラグを削除する(del_flg)

指定された ID のイベントフラグを削除します。削除された ID は、以降 cre_flg または acre_flg で生成する際に再利用できるようになります。
del_flg は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) イベントフラグを待つ(wai_flg, twai_flg)

指定されたビットがイベントフラグにセットされるのを待ちます。以下のいずれかの待ちモードを指定します。

- AND 待ち：指定されたビットが全てセットされるのを待ちます。
- OR 待ち：指定されたビットのいずれかがセットされるのを待ちます。

待ち解除時には、待ち解除直前のイベントフラグ値が呼び出し元タスクに返されます。対象イベントフラグに TA_CLR 属性が指定されていた場合は、この時イベントフラグが 0 クリアされます。この場合、返される値はクリア直前のイベントフラグ値です。

(4) イベントフラグを得る(ポーリング) (pol_flg, ipol_flg)

指定されたビットがイベントフラグにセットされているかどうかを調べます。wai_flg, twai_flg と異なるのは、待ち条件を満たさない場合に、待ち状態にはならずに直ちにエラーリターンする点です。

(5) イベントフラグをセットする(set_flg, iset_flg)

イベントフラグに対し、指定されたビットをセットします。これにより、このイベントフラグで待っていたタスクの待ち状態が解除される場合があります。

(6) イベントフラグをクリアする(clr_flg, iclr_flg)

イベントフラグに対し、指定されたビットをクリアします。

(7) イベントフラグの状態を参照する(ref_flg, iref_flg)

イベントフラグのビットパターンや待ちタスク ID を参照します。

3.8 データキュー

データキューとは、1 ワード(32 ビット)のデータ通信を行うオブジェクトです。図 3.10に、データキューの概要を示します。

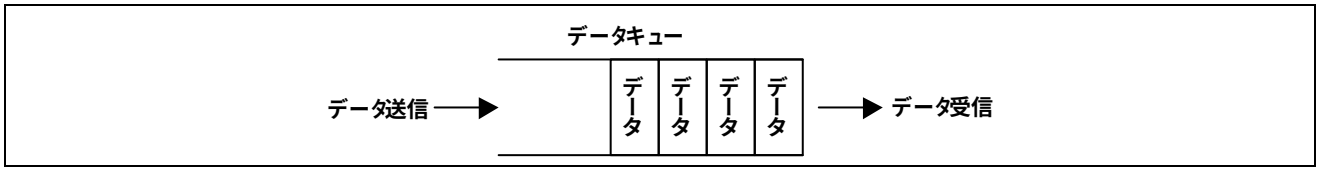

図3.10 データキュー

データキューに送信されたデータは蓄積されます。データキューからの受信では、最も古いデータから順に取り出されます(FIFO)。特殊な使い方として、データ数 0 のデータキューを使用することもできます。

データキューを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに dataqueue[]を記述します。
- 動的な生成
  cre_dtq または acre_dtq サービスコールを使用します。

生成時には、以下の情報を指定します。

- 送信待ち行列の並び方(TA_TFIFO(FIFO 順)または TA_TPRI(タスク優先度順))
- データ数
- データキュー領域の先頭アドレス(動的生成)

データキュー機能のサービスコールには、次のものがあります。

(1) データキューを生成する(cre_dtq, acre_dtq)

cre_dtq は指定した ID で、acre_dtq は空いている ID をカーネルが割り当てて、データキューを生成します。
通常は、データキュー領域はメモリオブジェクト・ユーザスタック以外の領域としてください。メモリオブジェクト内にデータキュー領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤ってデータキュー領域を書き換えてしまう危険があります。
cre_dtq, acre_dtq は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) データキューを削除する(del_dtq)

指定された ID のデータキューを削除します。削除された ID は、以降 cre_dtq または acre_dtq で生成する際に再利用できるようになります。
del_dtq は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) データを送信する(snd_dtq, tsnd_dtq)

データキューにデータを送信します。データキューがデータでいっぱいの場合は、本サービスコールは呼び出し元タスクをデータ送信待ち状態に移行させます。

(4) データを送信する(ポーリング) (psnd_dtq, ipsnd_dtq)

データキューにデータを送信します。snd_dtq, tsnd_dtq と異なるのは、データキューがデータでいっぱいの場合に、データ送信待ち状態にはならずに直ちにエラーリターンする点です。

(5) データを強制的に送信する(fsnd_dtq, ifsnd_dtq)

データキューにデータを送信します。データキューがデータでいっぱいの場合は、最も古いデータが破棄されます。

(6) データを受信する(rcv_dtq, trcv_dtq)

データキューからデータを受信します。データキューにデータがない場合は、本サービスコールは呼び出し元タスクをデータ受信待ち状態に移行させます。データキューがデータでいっぱいで送信待ちタスクが存在する場合は、データ送信待ち行列先頭のタスクの待ち状態が解除されます。

(7) データを受信する(ポーリング) (prcv_dtq, iprcv_dtq)

データキューからデータを受信します。データキューにデータがない場合は、本サービスコールは直ちにエラーリターンします。データキューがデータでいっぱいで送信待ちタスクが存在する場合は、データ送信待ち行列先頭のタスクの待ち状態が解除されます。

(8) データキューの状態を参照する(ref_dtq, iref_dtq)

データキューに蓄積されているデータ数や送信・受信待ちタスク ID を参照します。

3.9 メールボックス

3.9.1 機能解説

メールボックスとは、メッセージと呼ぶ任意のサイズのデータ通信を行うオブジェクトです。図 3.11に、メールボックスの概要を示します。

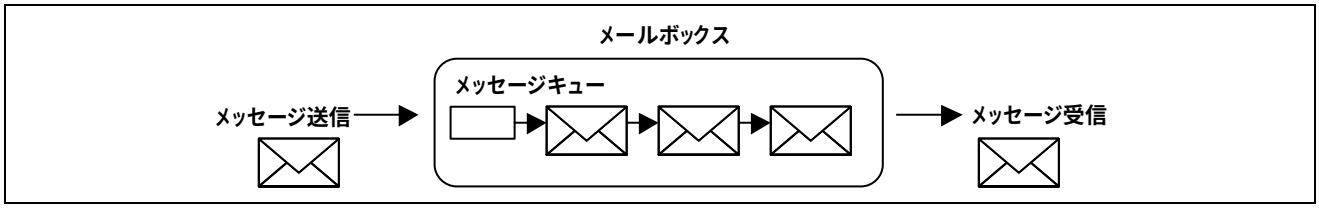

図3.11 メールボックス

メールボックスでは、メッセージアドレスの受け渡しによってデータ通信を行うため、メッセージサイズに依存しない高速な通信が行われます。

メッセージ領域は送信・受信側双方がアクセス可能なメモリにアプリケーションが作成しなければなりません（ローカル変数領域にメッセージを作成してはなりません）。また、メッセージ送信するタスクでは、メッセージ送信後にそのメッセージ領域をアクセスしてはなりません。

メールボックスに送信されたメッセージはメッセージキューによって管理されます。メッセージは、メッセージキューの順に受信されます。

メッセージキューの並び方として、以下のいずれかを選択できます。

- TA_MPRI 属性：メッセージに設定された優先度順
- TA_MFIFO 属性：FIFO 順

メールボックスを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに mailbox[]を記述します。
- 動的な生成
  cre_mbx または acre_mbx サービスコールを使用します。

生成時には、以下の情報を指定します。

- 受信待ち行列の並び方(TA_TFIFO(FIFO 順)または TA_TPRI(タスク優先度順))
- メッセージキューの並び方(TA_MFIFO(FIFO 順)または TA_MPRI(メッセージ優先度順))
- 最大メッセージ優先度

メールボックス機能のサービスコールには、次のものがあります。

(1) メールボックスを生成する(cre_mbx, acre_mbx)

cre_mbx は指定した ID で、acre_mbx は空いている ID をカーネルが割り当てて、メールボックスを生成します。

cre_mbx, acre_mbx は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) メールボックスを削除する(del_mbx)

指定された ID のメールボックスを削除します。削除された ID は、以降 cre_mbx または acre_mbx で生成する際に再利用できるようになります。

del_mbx は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) メッセージを送信する(snd_mbx, isnd_mbx)

メールボックスにメッセージを送信します。

(4) メッセージを受信する(rcv_mbx, trcv_mbx)

メールボックスからメッセージを受信します。メールボックスにメッセージがない場合は、本サービスコールはメッセージが送信されるまで呼び出し元タスクを待ち状態に移行させます。

(5) メッセージを受信する(ポーリング) (prcv_mbx, iprcv_mbx)

メールボックスからメッセージを受信します。rcv_mbx, trcv_mbx と異なるのは、メールボックスにメッセージがない場合に、待ち状態にはならずに直ちにエラーリターンする点です。

(6) メールボックスの状態を参照する(ref_mbx, iref_mbx)

メールボックスに入っている先頭のメッセージアドレスと待ちタスク ID を参照します。

3.9.2 注意事項

(1) メールボックス機能の使用は推奨しません

メッセージ先頭にはカーネル管理テーブルがあります。この管理テーブルは保護されないため、アプリケーションによって破壊される危険性があります。メッセージ通信にはメールボックスの代わりにデータキューまたはメッセージバッファの使用を推奨します。

(2) メッセージの作成場所

メールボックスでは、メッセージのポインタだけが送受信され、メッセージ内容はコピーされません。したがって、メッセージは送信タスクと受信タスクの双方がアクセス可能な領域、つまり送受信タスク双方がアクセス可能なメモリオブジェクト内に作成するようにしてください。

以下に、悪い例を示します。

- 悪い例：自動変数としてメッセージを作成
  自動変数は、通常はその関数のスタックフレームに配置されます。メッセージを送信した関数が終了するとスタックフレームも解放され、以降その領域は別の目的で使用(上書き)されることにになります。つまり、受信時点ではメッセージ内容はすでに上書きされている可能性があります。
  さらにメモリ保護の観点では、タスクのスタックは別のタスクからはアクセスできない仕様のため、受信タスクは受信したメッセージにアクセスすることができません。

3.10 ミューテックス

3.10.1 機能解説

ミューテックスは排他制御を行うためのオブジェクトです。セマフォとの主な相違は以下の通りです。

(1) 優先度逆転現象を回避するための優先度上限プロトコルをサポートしています。
(2) 単一資源の排他制御にのみ使用できます。

アプリケーションでは、共有資源を使用する前にミューテックスをロックし、資源を使い終えたらミューテックスをアンロックします。ミューテックスをロックしている期間は、そのタスクの優先度をが自動的に引き上げられるので、優先度逆転現象を回避できます。

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

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

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

図 3.12に、ミューテックスの動作例を示します。

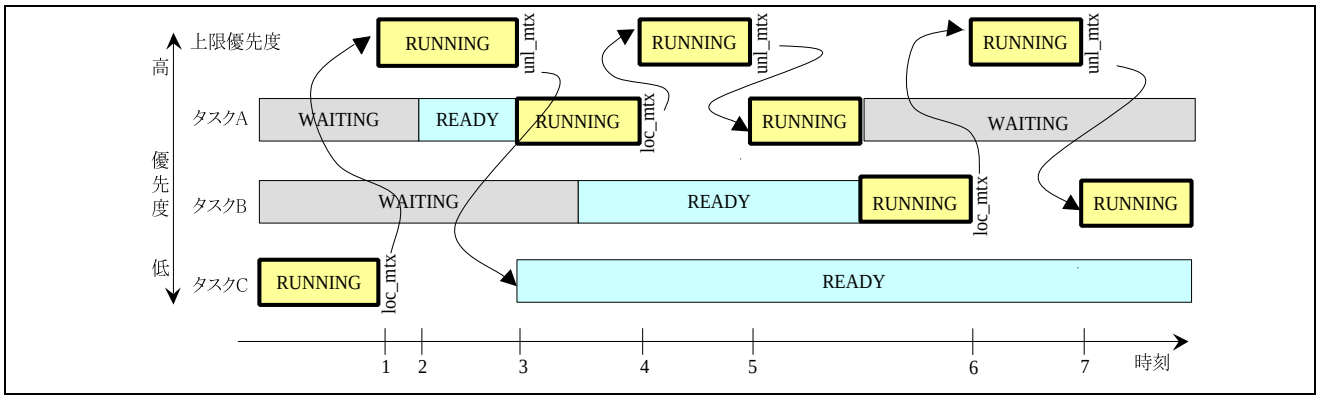

図3.12 ミューテックスの動作例

図の解説

時刻1：タスクCがloc_mtxでミューテックスをロックすると、優先度がミューテックスの上限優先度に引き上げられます。

時刻2：タスクCが上限優先度で実行中にタスクAがREADY状態になりました。タスクAの優先度は本来はタスクCよりも高いですが、タスクCはタスクAよりも高い上限優先度で実行しているため、タスクAはRUNNING状態にはなりません。すなわちタスクCは、ミューテックスをロックしている間は、本来の優先度がより高いタスクAが実行可能になっても、タスクAに邪魔されずに処理を継続することができます。

時刻3：タスクCがunl_mtxでミューテックスのロックを解除すると、タスクCは元の優先度に戻ります。この結果、優先度の高いタスクAがRUNNING状態になります。

時刻4：タスクAがloc_mtxを発行すると、タスクAの優先度は上限優先度に引き上げられます。
時刻5：タスクAがunl_mtxを発行すると、タスクAの優先度は元に戻ります。
時刻6：タスクBがloc_mtxを発行すると、タスクBの優先度は上限優先度に引き上げられます。
時刻7：タスクBがunl_mtxを発行すると、タスクBの優先度は元に戻ります。

ミューテックスを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに mutex[]を記述します。
- 動的な生成
  cre_mtx または acre_mtx サービスコールを使用します。

生成時には、以下の情報を指定します。

- 上限優先度

ミューテックス機能のサービスコールには、次のものがあります。

(1) ミューテックスを生成する(cre_mtx, acre_mtx)

cre_mtx は指定した ID で、acre_mtx は空いている ID をカーネルが割り当てて、ミューテックスを生成します。
cre_mtx, acre_mtx は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) ミューテックスを削除する(del_mtx)

指定された ID のミューテックスを削除します。削除された ID は、以降 cre_mtx または acre_mtx で生成する際に再利用できるようになります。
del_mtx は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) ミューテックスをロックする(loc_mtx, tloc_mtx)

ミューテックスをロックし、現在優先度をミューテックスの上限優先度に引き上げます。他のタスクがロック中の場合は、ロックが解放されるまで本サービスコールは呼び出しタスクを待ち状態に移行させます。

(4) ミューテックスをロックする(ポーリング) (ploc_mtx)

ミューテックスをロックし、優先度を上限優先度に引き上げます。loc_mtx, tloc_mtx と異なるのは、他のタスクがロック中の場合に、待ち状態にはならずに直ちにエラーリターンする点です。

(5) ミューテックスのロックを解除する(unl_mtx)

ミューテックスのロックを解除します。呼び出しタスクが他にミューテックスをロックしていない場合は、現在優先度をベース優先度に戻します。本ミューテックスのロックを待っているタスクがあればそのタスクの待ち状態を解除します。

(6) ミューテックスの状態を参照する(ref_mtx)

ミューテックスをロックしているタスク ID や、待ちタスク ID を参照します。

3.10.2 ベース優先度と現在優先度

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

ミューテックスをロックしていない時は、両者は常に同じです。

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

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

なお、get_pri, iget_pri を用いると、現在優先度を参照することができます。

3.11 メッセージバッファ

メッセージバッファは、メールボックスと同様に任意のサイズのデータ通信を行うオブジェクトです。メールボックスと異なる点は、データ内容そのものがコピーされる点です。このため、メッセージ送信後は相手が受信したかどうかに関わらず、直ちに送信したメッセージ領域を再利用することができます。図 3.13に、メッセージバッファの概要を示します。

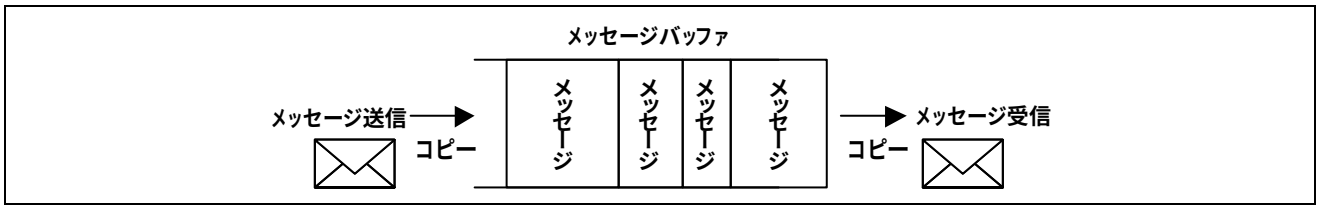

図3.13 **メッセージバッファ**

メッセージバッファには、メールボックスと同様にメッセージが蓄積されます。蓄積されたメッセージは、FIFO 順に取り出されます。

メッセージバッファを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに message_buffer[]を記述します。
- 動的な生成
  cre_mbf または acre_mbf サービスコールを使用します。

生成時には、以下の情報を指定します。

- バッファサイズ
- バッファ領域の先頭アドレス(動的生成)またはバッファを配置するセクション(静的生成)
- 最大メッセージサイズ

メッセージバッファ機能のサービスコールには、次のものがあります。

(1) メッセージバッファを生成する(cre_mbf, acre_mbf)

cre_mbf は指定した ID で、acre_mbf は空いている ID をカーネルが割り当てて、メッセージバッファを生成します。

通常は、メッセージバッファ領域はメモリオブジェクト・ユーザスタック以外の領域としてください。メモリオブジェクト内にメッセージバッファ領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤ってメッセージバッファ領域を書き換えてしまう危険があります。

cre_mbf, acre_mbf は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) メッセージバッファを削除する(del_mbf)

指定された ID のメッセージバッファを削除します。削除された ID は、以降 cre_mbf または acre_mbf で生成する際に再利用できるようになります。

del_mbf は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) メッセージを送信する(snd_mbf, tsnd_mbf)

メッセージバッファにメッセージを送信します。
メッセージバッファメッセージを送信するには、メッセージバッファに以下の空きサイズが必要です。

(送信メッセージサイズを 4 の倍数に切り上げた値) + VTSZ_MBFTBL

メッセージバッファの空きサイズがこれに満たない場合は、空きサイズができるまで本サービスコールは呼び出し元タスクをメッセージ送信待ち状態に移行させます。この待ち行列は FIFO で管理されます。

(4) メッセージを送信する(ポーリング) (psnd_mbf, ipsnd_mbf)

メッセージバッファにメッセージを送信します。snd_mbf, tsnd_mbf と異なるのは、メッセージバッファの空きサイズが足りない場合に、メッセージ送信待ち状態にはならずに直ちにエラーリターンする点です。

(5) メッセージを受信する(rcv_mbf, trcv_mbf)

メッセージバッファからメッセージを受信します。メッセージバッファにメッセージがない場合は、メッセージバッファにメッセージが送信されるまで、本サービスコールは呼び出し元タスクをメッセージ受信待ち状態に移行させます。この待ち行列は FIFO で管理されます。
メッセージバッファからメッセージを受信すると、メッセージバッファの空きサイズは以下のサイズだけ増加します。

(受信メッセージサイズを 4 の倍数に切り上げた値) + VTSZ_MBFTBL

この結果、メッセージ送信待ちタスクが送信しようとしていたメッセージのサイズよりもメッセージバッファの空きサイズが大きくなると、そのメッセージがメッセージバッファに送信され、そのタスクの待ち状態が解除されます。

(6) メッセージを受信する(ポーリング) (prcv_mbf)

メッセージバッファからメッセージを受信します。rcv_mbf, trcv_mbf と異なるのは、メッセージバッファにメッセージがない場合に、メッセージ受信待ち状態にはならずに直ちにエラーリターンする点です。

(7) メッセージバッファの状態を参照する(ref_mbf, iref_mbf)

メッセージバッファに蓄積されているメッセージ数やメッセージバッファの空きサイズ、送信・受信待ちタスク ID を参照します。

3.12 固定長メモリプール

3.12.1 機能解説

固定長メモリプールは、決められたサイズのメモリブロックを動的に獲得・解放するオブジェクトです。

固定長メモリプールは、可変長メモリプールに比べて、任意サイズのメモリブロックを獲得できない短所がある反面、獲得・返却のオーバヘッドが小さいというメリットがあります。

図 3.14に固定長メモリプールの概要を示します。

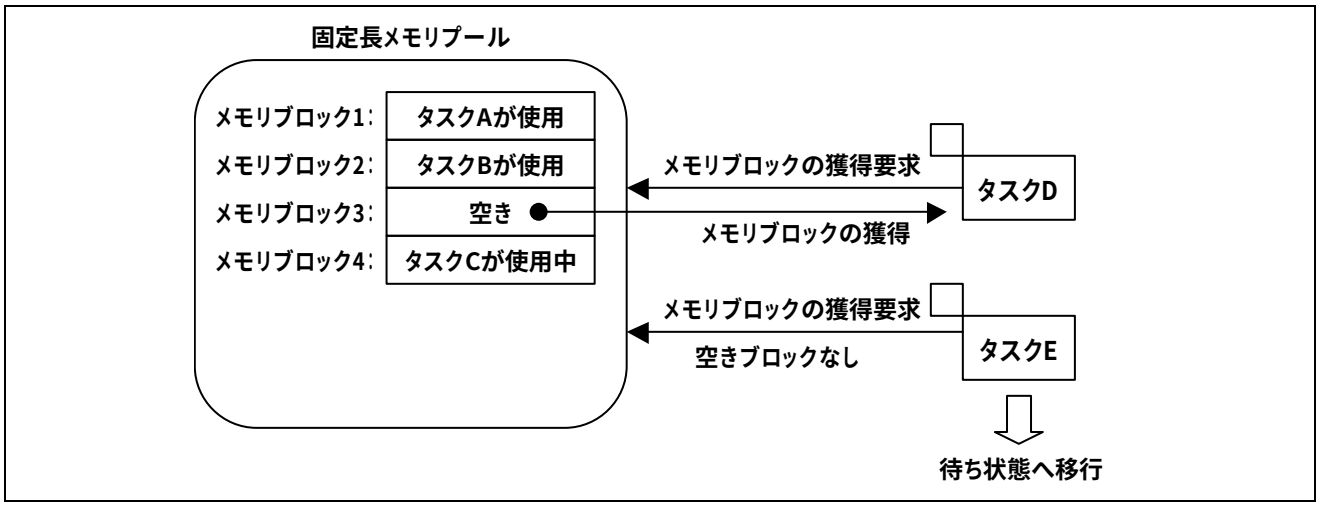

図3.14 固定長メモリプール

固定長メモリプールを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに memorypool[]を記述します。
- 動的な生成
  cre_mpf または acre_mpf サービスコールを使用します。

生成時には、以下の情報を指定します。

- メモリ獲得待ち行列の並び方(TA_TFIFO(FIFO 順)または TA_TPRI(タスク優先度順))
- ブロックサイズ
- ブロック数
- メモリプール領域の先頭アドレス(動的生成)、またはメモリプール領域を配置するセクション(静的生成)
- 固定長メモリプール管理領域の先頭アドレス(動的生成)

固定長メモリプール機能のサービスコールには、次のものがあります。

(1) 固定長メモリプールを生成する(cre_mpf, acre_mpf)

cre_mpf は指定した ID で、acre_mpf は空いている ID をカーネルが割り当てて、固定長メモリプールを生成します。

固定長メモリプール領域は、メモリブロックを使用するタスクからアクセス可能なメモリオブジェクト内とする必要があります。複数の固定長メモリプールや可変長メモリプール領域をひとつのメモリオブジェクトとしても構いません。なお、メモリブロックごとに、異なるアクセス許可を設定することはできません。

また、通常は、固定長メモリプール管理領域はメモリオブジェクト・ユーザスタック以外の領域を指定してください。メモリオブジェクト内に固定長メモリプール管理領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤って固定長メモリプール管理領域を書き換えてしまう危険があります。

cre_mpf, acre_mpf は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) 固定長メモリプールを削除する(del_mpf)

指定された ID の固定長メモリプールを削除します。削除された ID は、以降 cre_mpf または acre_mpf で生成する際に再利用できるようになります。

del_mpf は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) メモリブロックを獲得する(get_mpf, tget_mpf)

固定長メモリプールからメモリブロックを獲得します。固定長メモリプールに空きメモリブロックがない場合は、メモリブロックが解放されるまで本サービスコールは呼び出し元タスクを待ち状態に移行させます。

(4) メモリブロックを獲得する(ポーリング) (pget_mpf, ipget_mpf)

固定長メモリプールからメモリブロックを獲得します。get_mpf, tget_mpf と異なるのは、固定長メモリプールに空きメモリブロックがない場合に、待ち状態にはならずに直ちにエラーリターンする点です。

(5) メモリブロックを解放する(rel_mpf, irel_mpf)

メモリブロックを解放します。メモリブロックの獲得を待っているタスクがある場合は、そのタスクの待ち状態を解除します。

(6) 固定長メモリプールの状態を参照する(ref_mpf, iref_mpf)

固定長メモリプールの空きメモリブロック数や待ちタスク ID を参照します。

3.12.2 注意事項

カーネルは、固定長メモリプール領域内に管理テーブルを生成します。この管理テーブルが誤って書き換えられた場合、システムの正常な動作は保証されません。

3.13 可変長メモリプール

3.13.1 機能解説

可変長メモリプールは、任意のサイズのメモリブロックを動的に獲得・解放するオブジェクトです。

可変長メモリプールは、固定長メモリプールに比べて、任意サイズのメモリブロックを獲得できる長所がある反面、獲得・返却のオーバヘッドが大きいというデメリットがあります。また、後述する断片化問題にも注意する必要があります。

cfg ファイルで可変長メモリプールを生成する時には、プール領域と獲得可能な最大サイズを指定します。

可変長メモリプールを生成するには、以下の方法があります。なお、メモリ獲得待ち行列は、FIFO 順で管理されます。

- 静的な生成
  cfg ファイルに variable_memorypool[]を記述します。
- 動的な生成
  cre_mpl または acre_mpl サービスコールを使用します。

生成時には、以下の情報を指定します。

- メモリプールサイズ
- 獲得可能なメモリブロックサイズの最大値
- メモリプール領域の先頭アドレス(動的生成)、またはメモリプール領域を配置するセクション(静的生成)

可変長メモリプール機能のサービスコールには、次のものがあります。

(1) 可変長メモリプールを生成する(cre_mpl, acre_mpl)

cre_mpl は指定した ID で、acre_mpl は空いている ID をカーネルが割り当てて、可変長メモリプールを生成します。

可変長メモリプール領域は、メモリブロックを使用するタスクからアクセス可能なメモリオブジェクト内とする必要があります。複数の可変長メモリプールや固定長メモリプール領域をひとつのメモリオブジェクトとしても構いません。なお、メモリブロックごとに、異なるアクセス許可を設定することはできません。

cre_mpl, acre_mpl は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) 可変長メモリプールを削除する(del_mpl)

指定された ID の可変長メモリプールを削除します。削除された ID は、以降 cre_mpl または acre_mpl で生成する際に再利用できるようになります。

del_mpl は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) メモリブロックを獲得する(get_mpl, tget_mpl)

可変長メモリプールから指定したサイズのメモリブロックを獲得します。可変長メモリプールの空きサイズが不足している場合は、空きサイズができるまで本サービスコールは呼び出し元タスクを待ち状態に移行させます。

(4) メモリブロックを獲得する(ポーリング) (pget_mpl, ipget_mpl)

可変長メモリプールから指定したサイズのメモリブロックを獲得します。get_mpl tget_mpl と異なるのは、可変長メモリプールの空きサイズが足りない場合に、待ち状態にはならずに直ちにエラーリターンする点です。

(5) メモリブロックを解放する(rel_mpl)

メモリブロックを解放します。

メモリブロックの解放により、可変長メモリプールの空きサイズは増加します。この結果、メモリブロックの獲得待ちタスクの要求メモリブロックサイズよりも可変長メモリプールの空きサイズが大きくなると、そのタスクはメモリブロックを獲得して待ち状態が解除されます。

(6) 可変長メモリプールの状態を参照する(ref_mpl, iref_mpl)

空きサイズの合計や最大連続空きメモリブロックサイズ、待ちタスク ID を参照します。

3.13.2 空き領域の断片化について

可変長メモリプールからメモリブロックの獲得と解放(返却)を繰り返していると、空き領域の断片化が発生し、空き領域のトータルサイズは十分でも、連続した空き領域が存在しない、つまり大きなメモリブロックを獲得できない状況になることがあります(図 3.15)。

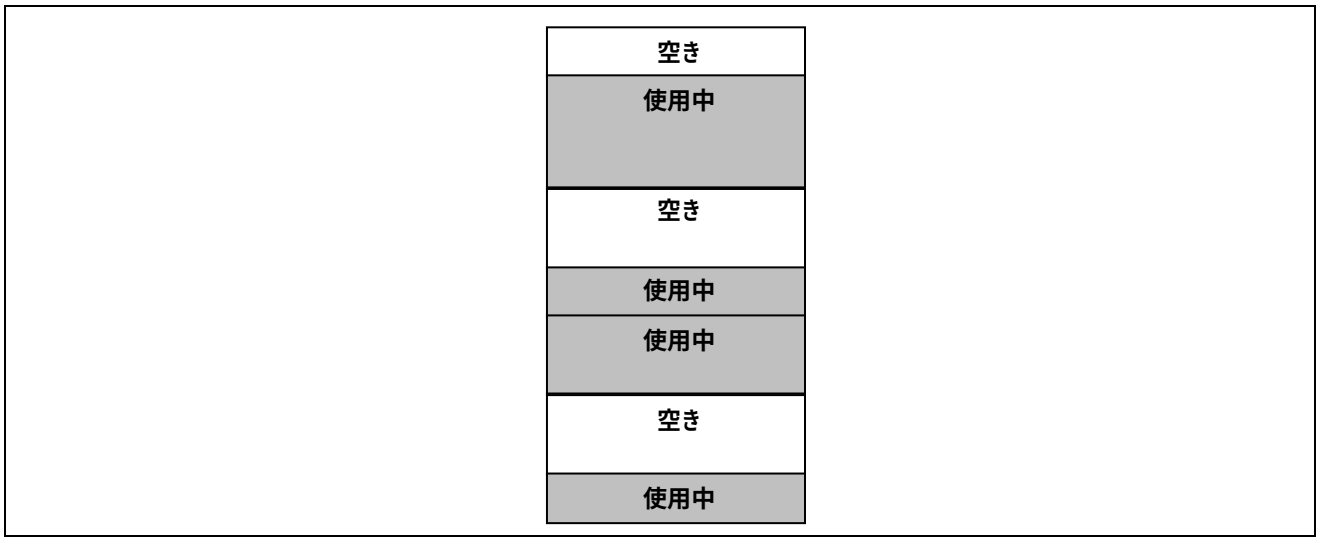

図3.15 空き領域の断片化

本カーネルでは、メモリブロックサイズのバリエーションを制限することで、断片化が発生しにくいようになっています。

メモリブロックサイズのバリエーションは、cfg ファイルの variable_memorypool[].max_memsize を元に決定されます。詳細は、「8.4.15 可変長メモリプール定義(variable_memorypool[])」を参照してください。

3.13.3 注意事項

カーネルは可変長メモリプール領域内に管理テーブルを生成します。この管理テーブルが誤って書き換えられた場合、システムの正常な動作は保証されません。

3.14 時間管理機能

カーネルは、時間に関する以下のような機能をサポートしています。

- システム時刻の参照・設定
- タイムイベントハンドラ（周期ハンドラ、アラームハンドラ）の実行制御
- タイムアウトなどの時間によるタスクの実行制御

サービスコールで使用する時間パラメータの単位時間はミリ秒です。

3.14.1 タスクのタイムアウト

tslp_tsk や twai_sem のように、'tで始まるサービスコールではタイムアウトを指定できます。

指定したタイムアウト時間が経過しても待ち条件が満たされない場合、待ち状態が解除されます。そして、サービスコールのリターン値としてエラーE_TMOUT が返されます。

タイムアウトは、本来発生するはずのイベントが発生しなかったときの異常検出に利用できます。

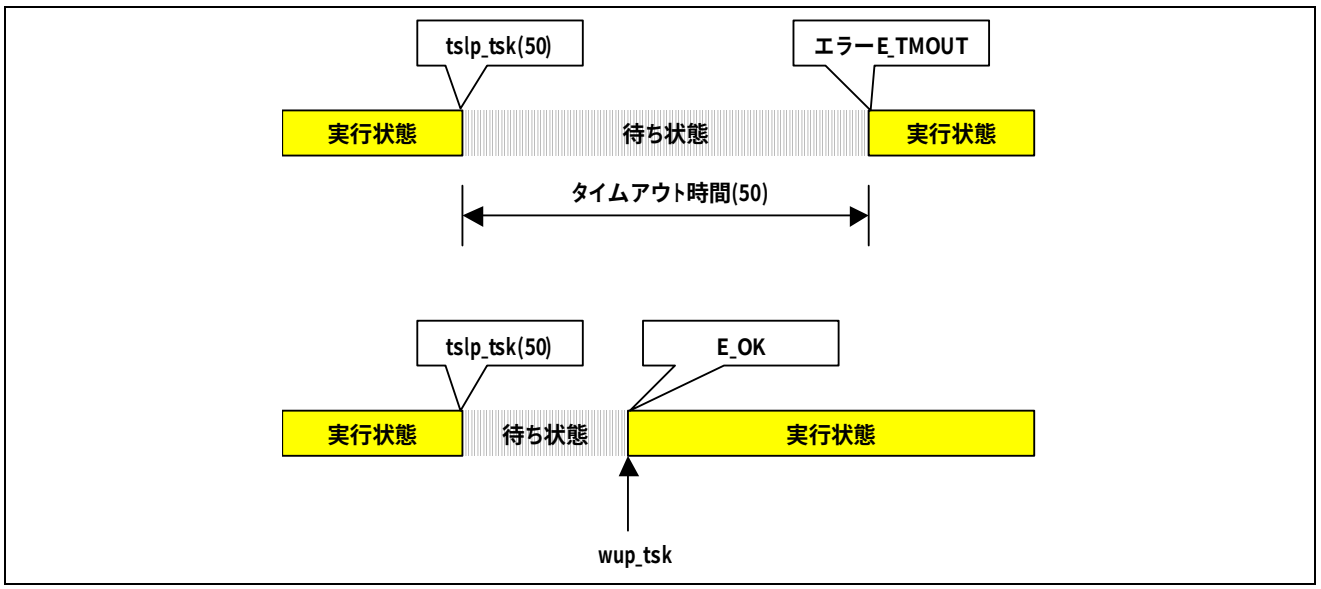

図3.16 タイムアウト

3.14.2 タスクの遅延

dly_tsk を用いて、タスクを指定した時間だけ待ち状態に移行させることができます。指定した時間が経過すると、待ち状態が解除され、リターン値として E_OK が返されます。

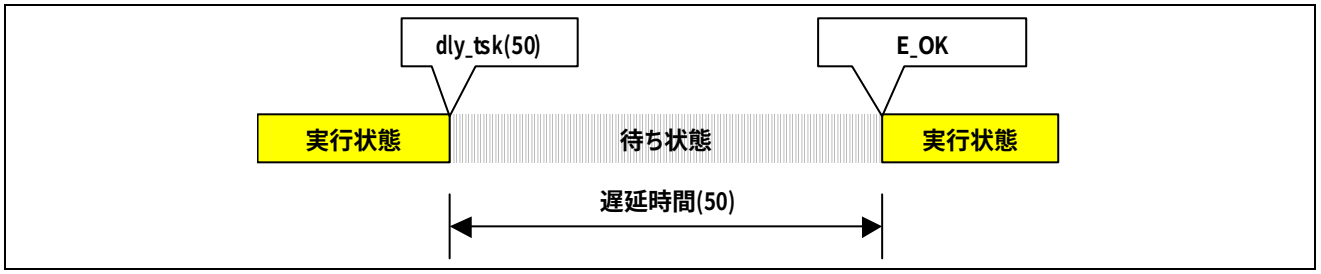

図3.17 タスクの遅延

3.14.3 周期ハンドラ

周期ハンドラは、指定した起動位相経過後、起動周期ごとに起動されるタイムイベントハンドラです。周期ハンドラの起動には、起動位相を保存する方法と起動位相を保存しない方法があります。起動位相を保存する場合は、周期ハンドラの生成時点(システム起動時点)を基準に周期ハンドラの起動時刻が決定されます。起動位相を保存しない場合は、周期ハンドラの動作開始時点を基準に周期ハンドラの起動時刻が決定されます。

周期ハンドラには、生成時に指定された拡張情報が渡されます。

周期ハンドラを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに cyclic_hand[]を記述します。
- 動的な生成
  cre_cyc または acre_cyc サービスコールを使用します。

生成時には、以下の情報を指定します。

- 周期ハンドラの開始アドレス
- 起動周期
- 起動位相
- 動作を開始する(TA_STA)か否か
- 位相を保存する(TA_PHS)か否か
- 拡張情報

図 3.18および図 3.19に、周期ハンドラの動作例を示します。

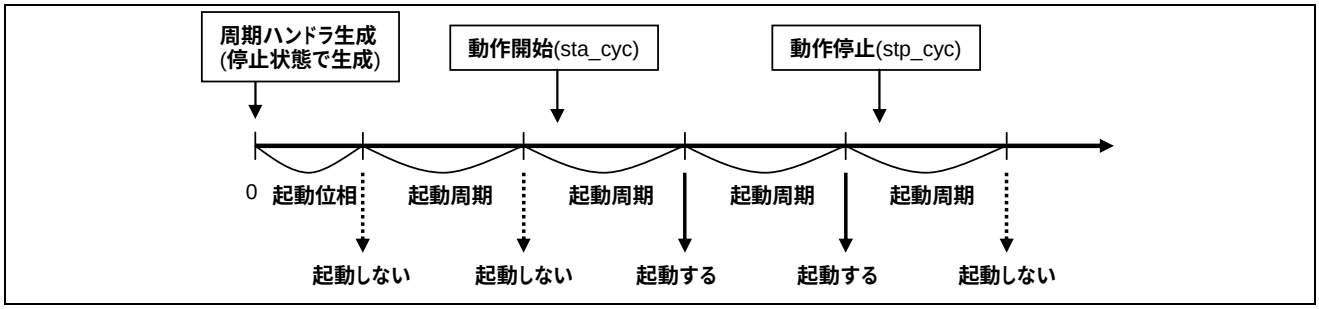

図3.18 起動位相を保存する場合の動作

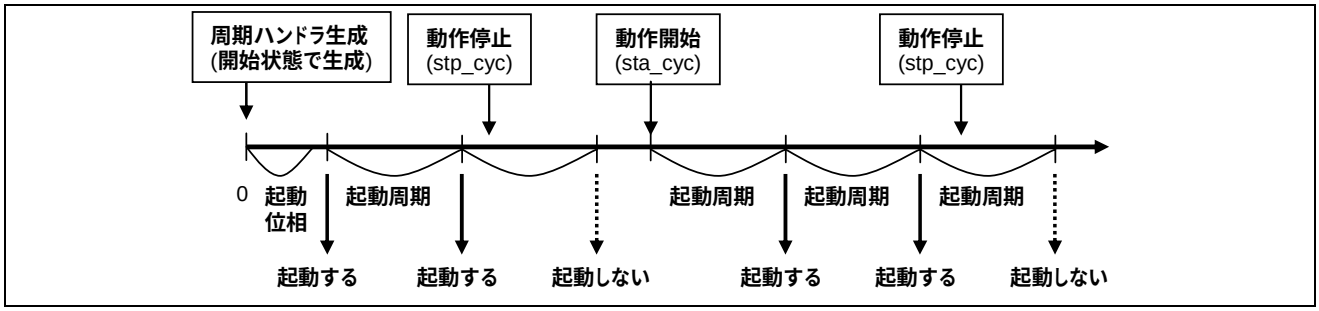

図3.19 起動位相を保存しない場合の動作

周期ハンドラ機能のサービスコールには、次のものがあります。

(1) 周期ハンドラを生成する(cre_cyc, acre_cyc)

cre_cyc は指定した ID で、acre_cyc は空いている ID をカーネルが割り当てて、周期ハンドラを生成します。
cre_cyc, acre_cyc は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) 周期ハンドラを削除する(del_cyc)

指定された ID の周期ハンドラを削除します。削除された ID は、以降 cre_cyc または acre_cyc で生成する際に再利用できるようになります。
del_cyc は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) 周期ハンドラの動作を開始する(sta_cyc, ista_cyc)

周期ハンドラの動作を開始します。

(4) 周期ハンドラの動作を停止する(stp_cyc, istp_cyc)

周期ハンドラの動作を停止します。

(5) 周期ハンドラの状態を参照する(ref_cyc, iref_cyc)

周期ハンドラの動作状態や、次の起動までの残り時間を参照します。

3.14.4 アラームハンドラ

アラームハンドラは、指定した時刻になると 1 度だけ起動されるタイムイベントハンドラです。アラームハンドラを用いることにより、時刻に依存した処理を行うことができます。

図 3.20にアラームハンドラの動作例を示します。

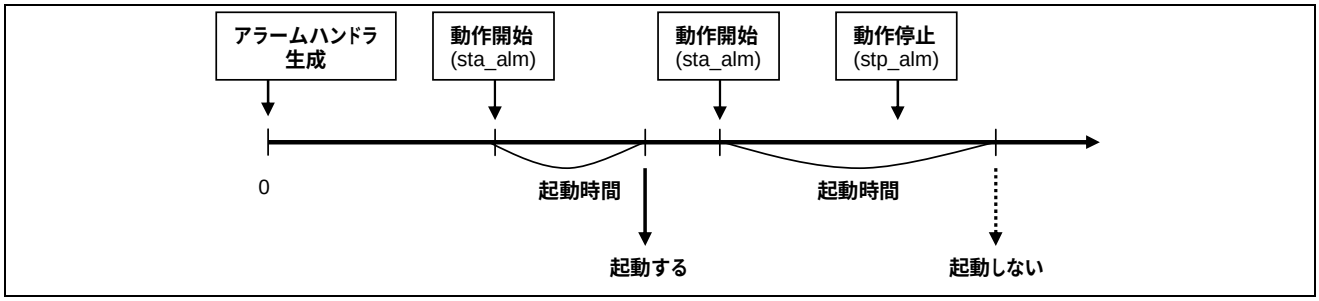

図3.20 アラームハンドラの動作例

アラームハンドラには、生成時に指定された拡張情報が渡されます。

アラームハンドラを生成するには、以下の方法があります。

- 静的な生成
  cfg ファイルに alarm_hand[]を記述します。
- 動的な生成
  cre_alm または acre_alm サービスコールを使用します。

生成時には、以下の情報を指定します。

- アラームハンドラの開始アドレス
- 拡張情報

アラームハンドラ機能のサービスコールには、次のものがあります。

(1) アラームハンドラを生成する(cre_alm, acre_alm)

cre_alm は指定した ID で、acre_alm は空いている ID をカーネルが割り当てて、アラームハンドラを生成します。

cre_alm, acre_alm は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) アラームハンドラを削除する(del_alm)

指定された ID のアラームハンドラを削除します。削除された ID は、以降 cre_alm または acre_alm で生成する際に再利用できるようになります。

del_alm は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) アラームハンドラの動作を開始する(sta_alm, ista_alm)

指定された時間後に起動するように、アラームハンドラの動作を開始します。

(4) アラームハンドラの動作を停止する(stp_alm, istp_alm)

アラームハンドラの動作を停止します。

(5) アラームハンドラの状態を参照する(ref_alm, iref_alm)

アラームハンドラの動作状態や、起動までの残り時間を参照します。

3.14.5 時間の精度

タイムアウトなどの時間パラメータの単位はすべてミリ秒です。

その精度は TIC_NUME／TIC_DENO[ms]となります。この精度で、システム時刻の更新や時間管理が行われます。TIC_NUME および TIC_DENO は、それぞれ cfg ファイルの system.tic_nume および system.tic_deno に定義します。

時間イベント(タイムアウト発生や周期ハンドラ起動など)は、指定した相対時間以上が経過してから発生するようになっています。

図 3.21に、実時刻が 9.2[ms]の時点で tslp_tsk(5)を実行した場合の例を示します。

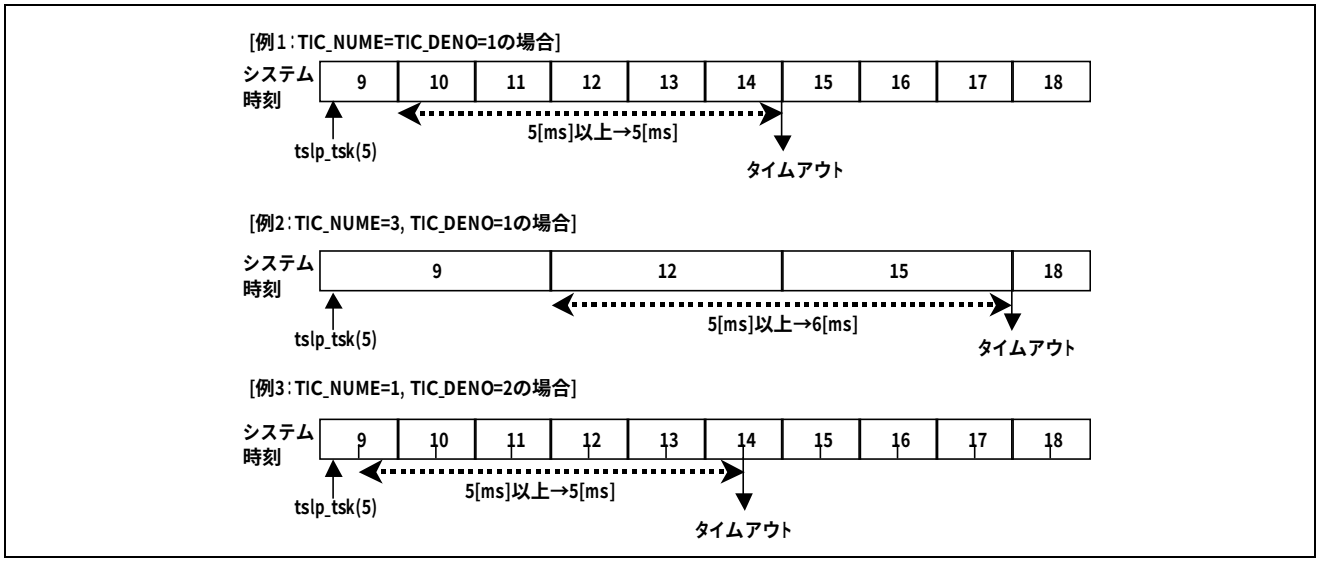

図3.21 **時間の精度**(tslp_tsk)

周期ハンドラの場合は、各回の相対時間を以下のように扱います。

1. TA_PHS属性が指定されていない周期ハンドラ
   (a) sta_cyc, ista_cycで動作開始した場合
   sta_cyc, ista_cyc時点を基準として、n回目の相対時間は次式の値として扱います。
   (起動周期) × n
   (b) cfgファイルで生成時にTA_STA属性を指定して動作開始した場合
   システム起動時点(生成時点)を基準として、n回目の相対時間は次式の値として扱います。
   (起動位相) + (起動周期) × (n – 1)
2. TA_PHS属性が指定された周期ハンドラ
   1 の(b)と同じ扱いとなります。ただし、実際にハンドラが起動されるかどうかは、ハンドラの動作状態で決まります。

3.14.6 注意事項

タイマ割り込み発生時には、カーネルは以下の処理を行います。

(a) システム時刻の更新
(b) アラームハンドラの起動と実行
(c) 周期ハンドラの起動と実行
(d) タイムアウト付きサービスコールおよびdly_tskによるタスクのタイムアウト処理

これらの処理は全て、タイマ割り込みレベル(clock.IPL)以下の割り込みをマスクした状態で行われます。

上記のうち、(b),(c),(d)は、複数のタスクやハンドラに対する処理が重複する可能性があるため、このような場合カーネルの処理時間が極端に長くなります。これは、以下のような弊害をもたらします。

- 割り込みに対するレスポンスの悪化
- システム時刻の遅れ

これを避けるために、以下を遵守してください。

- タイムイベントハンドラの処理は、可能な限り短くしてください。
- タイムイベントハンドラの周期、タイムアウト付きサービスコールで指定するタイムアウト値は、なるべく大きな値にしてください。極端な例としては、ある周期ハンドラの周期時間が 1ms で、そのハンドラ処理時間が 1ms 以上かかるような場合、永久にその周期ハンドラだけが実行されることになり、事実上ハングアップします。

3.15 システム状態管理機能

(1) タスクの実行待ち行列を回転する(rot_rdq, irot_rdq)

本サービスコールにより、TSS（タイムシェアリングシステム）を実現することができます。すなわち、一定周期でレディキューを回転すれば、TSS で必要なラウンドロビンスケジューリングを実現することができます。

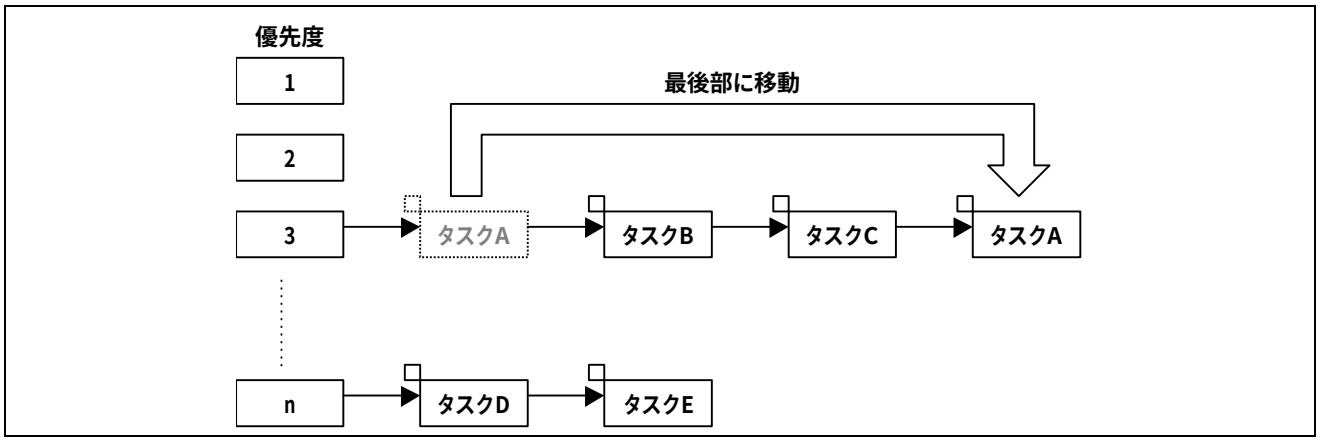

図3.22 rot_rdq によるレディキューの操作

(2) 実行状態のタスク ID を得る(get_tid, iget_tid)

get_tid は、自タスクのタスク ID を取得します。非タスクコンテキストから iget_tid を呼び出した場合は、その時点で実行していたタスク ID を取得します。

(3) CPU ロック状態への移行(loc_cpu, iloc_cpu)とその解除(unl_cpu, iunl_cpu)

loc_cpu, iloc_cpu により CPU ロック状態に移行することができます。CPU ロック状態を解除するには、unl_cpu, iunl_cpu を使います。

(4) ディスパッチ禁止状態への移行(dis_dsp)とその解除(ena_dsp)

dis_dsp によりディスパッチ禁止状態に移行することができます。ディスパッチ禁止状態を解除するには、ena_dsp を使います。

(5) コンテキスト種別を確認する(sns_ctx)

現在実行中のコンテキストが、タスクコンテキストか非タスクコンテキストかを確認します。

(6) CPU ロック状態かどうかを調べる(sns_loc)

現在が CPU ロック状態かどうかを調べます。

(7) ディスパッチ禁止状態かどうかを調べる(sns_dsp)

現在がディスパッチ禁止状態かどうかを調べます。

(8) ディスパッチ保留状態かどうかを調べる(sns_dpn)

現在がディスパッチ保留状態かどうかを調べます。

ディスパッチ保留状態とは、ディスパッチャより優先順位の高い処理を実行中であることを意味し、他のタスクは実行されません。具体的には、以下のいずれかのケースに該当する場合がディスパッチ保留状態です。

- CPU ロック状態
- ディスパッチ禁止状態
- 非タスクコンテキスト

ディスパッチ保留状態でないときには、待ち状態に移行するサービスコールを利用可能です。ソフトウェア部品など、どういった状態で呼び出されるか分からないソフトウェアでは、本サービスコールを用いて待ち状態に移行するサービスコールを呼び出すかエラーとするか、といった判定を行うことができます。

(9) カーネルを起動する(vsta_knl, ivsta_knl)

コンフィギュレーション結果に従って、カーネルを起動します。

(10) システムダウン(vsys_dwn, ivsys_dwn)

システムダウンさせてシステムダウンルーチンを起動します。

3.16 割り込み管理機能

割り込みが発生すると、割り込みハンドラが起動されます。割り込みハンドラは、cfg ファイルの interrupt_vector[](可変ベクタ)、または interrupt_fvector[](固定ベクタ)で定義します。

また、「2.7 割り込み」も参照してください。

(1) 割り込みマスクを変更する(chg_ims, ichg_ims)

割り込みマスク(PSW レジスタの IPL ビット)を指定された値に変更します。

タスクコンテキストで IPL を 0 以外に変更するとディスパッチ禁止状態に、IPL を 0 にするとディスパッチ許可状態に遷移します。

(2) 割り込みマスクを参照する(get_ims, iget_ims)

現在の割り込みマスク(PSW レジスタの IPL ビット)を参照します。

(3) カーネル管理割り込みハンドラから復帰する(ret_int)

カーネル管理割り込みハンドラから復帰します。

ret_int サービスコールは、割り込みハンドラの最後で呼び出されるように自動的にコンフィギュレーションされるので、ユーザが明示的に ret_int サービスコールの呼び出しを記述する必要はありません。

3.17 システム構成管理機能

(1) バージョン情報を参照する(ref_ver, iref_ver)

本カーネルのμITRON 仕様バージョンや、カーネルのバージョンを参照します。なお、これらのサービスコールで取得される情報と同じ情報を、カーネル構成マクロ(「4.2.1 定数マクロ」参照)から取得することもできます。

3.18 オブジェクトリセット機能

オブジェクトリセット機能は、各種オブジェクトを初期状態に戻す機能で、μITRON4.0 仕様外の機能です。

(1) データキューをリセットする(vrst_dtq)

データキューを初期化します。送信待ちのタスクは、待ち状態が解除され、エラーEV_RST が返ります。また、データキューに送信されていたデータは破棄されます。

(2) メールボックスをリセットする(vrst_mbx)

メールボックスを初期化します。メールボックスのメッセージキューにつながれていたメッセージは、カーネルの管理から外れます。

(3) メッセージバッファをリセットする(vrst_mbf)

メッセージバッファを初期化します。送信待ちのタスクは、待ち状態が解除され、エラーEV_RST が返ります。また、バッファに送信されていたメッセージは破棄されます。

(4) 固定長メモリプールをリセットする(vrst_mpf)

固定長メモリプールを初期化します。メモリ獲得待ちのタスクは、待ち状態が解除され、エラーEV_RST が返ります。また、既に獲得されていたメモリブロックは、空き状態として扱います。このため、本サービスコール以降は獲得していたメモリブロックを使用してはなりません。

(5) 可変長メモリプールをリセットする(vrst_mpl)

可変長メモリプールを初期化します。メモリ獲得待ちのタスクは、待ち状態が解除され、エラーEV_RST が返ります。また、既に獲得されていたメモリブロックは、空き状態として扱います。このため、本サービスコール以降は獲得していたメモリブロックを使用してはなりません。

3.19 メモリオブジェクト管理機能

メモリオブジェクト管理機能は、メモリオブジェクトの登録・登録解除・アクセス許可の変更、アクセス許可チェックなどの機能です。

メモリオブジェクトを登録するには、以下の方法があります。

- 静的な登録
  cfg ファイルに memory_object[]を記述します。
- 動的な登録
  ata_mem サービスコールを使用します。

登録時には、以下の情報を指定します。

- メモリオブジェクトの先頭アドレス
- メモリオブジェクトのサイズ(動的登録)、または終端アドレス(静的登録)
- アクセス許可ベクタ

メモリオブジェクト管理機能のサービスコールには、次のものがあります。

(1) メモリオブジェクトを登録する(ata_mem)

指定した領域を、指定したアクセス許可状態でメモリオブジェクトとして登録します。

ただし、メモリオブジェクト数制約によって、E_OACV エラーが返る場合があります。「2.9.3 メモリオブジェクトの数の制約」を参照してください。

ata_mem は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(2) メモリオブジェクトの登録を解除する(det_mem)

指定したメモリオブジェクトの登録を解除します。

det_mem は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(3) メモリオブジェクトのアクセス許可を変更する(sac_mem)

指定したメモリオブジェクトのアクセス許可を変更します。

ただし、メモリオブジェクト数制約によって、E_OACV エラーが返る場合があります。「2.9.3 メモリオブジェクトの数の制約」を参照してください。

sac_mem は、信頼されたドメインに所属するタスクからのみ呼び出せます。

(4) メモリ領域に対するアクセスチェック(vprb_mem)

指定したタスクから指定した領域に対し、指定したアクセスが可能かどうかをチェックします。

(5) メモリオブジェクト状態を参照する(ref_mem)

メモリオブジェクトのアクセス許可ベクタを参照します。

4. データタイプとマクロ

本章では、RI600/PX が提供するサービスコールを発行する際に使用するデータタイプ、マクロについて解説しています。

4.1 データタイプ

以下に、サービスコールを発行する際に指定する各種パラメータのデータタイプ一覧を示します。

なお、データタイプのマクロ定義は、inc600¥itron.h、または itron.h からインクルードされる lib600¥kernel.h で行われています。

表4.1 データタイプ

定義ファイル	マクロ	データタイプ	意味
itron.h	B	signed char	符号付き 8 ビット整数
	H	signed short	符号付き 16 ビット整数
	W	signed long	符号付き 32 ビット整数
	D	signed long long	符号付き 64 ビット整数
	UB	unsigned char	符号なし 8 ビット整数
	UH	unsigned short	符号なし 16 ビット整数
	UW	unsigned long	符号なし 32 ビット整数
	UD	unsigned long long	符号なし 64 ビット整数
	VB	unsigned char	データタイプが一定しない値(8 ビット)
	VH	unsigned short	データタイプが一定しない値(16 ビット)
	VW	unsigned long	データタイプが一定しない値(32 ビット)
	VD	unsigned long long	データタイプが一定しない値(64 ビット)
	VP	void *	データタイプが一定しない値(ポインタ)
	FP	void (*)	処理プログラムの起動アドレス(ポインタ)
	INT	signed long	符号付き 32 ビット整数
	UINT	unsigned long	符号なし 32 ビット整数
	BOOL	signed long	真偽値(TRUE または FALSE)
	ER	signed long	エラーコード
	ID	signed short	オブジェクト ID
	ATR	unsigned short	オブジェクト属性
	STAT	unsigned short	オブジェクト状態
	MODE	unsigned short	サービスコールの動作モード
	PRI	signed short	タスクまたはメッセージの優先度
	SIZE	unsigned long	領域のサイズ(単位：バイト)
	TMO	signed long	タイムアウト(単位：ミリ秒)
	RELTIM	unsigned long	相対時間(単位：ミリ秒)
	VP_INT	signed long	データタイプが一定しない値(ポインタ)、または符号付き 32 ビット整数
	ER_ID	signed long	エラーコード、またはオブジェクト ID
	ER_UINT	signed long	エラーコード、または符号なし 32 ビット整数
	ER_BOOL	signed long	エラーコード、または真偽値(TRUE または FALSE)
kernel.h	ACPTN	unsigned short	アクセス許可パターン
	FLGPTN	unsigned long	イベントフラグのビットパターン
	IMASK	unsigned short	割り込みマスク
	TEXPTN	unsigned long	タスク例外要因

4.2 マクロ

4.2.1 定数マクロ

表4.2 定数マクロ

分類	マクロ	定義内容	定義場所	説明
一般	NULL	0	itron.h	無効ポインタ
	TRUE	1	itron.h	真
	FALSE	0	itron.h	偽
	E_OK	0	itron.h	正常終了
属性	TA_NULL	0	itron.h	オブジェクト属性を指定しない
	TA_HLNG	0x0000	kernel.h	高級言語用インタフェース
	TA_ASM	0x0001	kernel.h	アセンブリ言語用インタフェース
	TA_TFIFO	0x0000	kernel.h	タスクの待ち行列は FIFO 順
	TA_TPRI	0x0001	kernel.h	タスクの待ち行列はタスク優先度順
	TA_MFIFO	0x0000	kernel.h	メッセージキューは FIFO 順
	TA_MPRI	0x0002	kernel.h	メッセージキューはメッセージ優先度順
	TA_ACT	0x0002	kernel.h	タスクを生成と同時に起動
	TA_WSGL	0x0000	kernel.h	イベントフラグに複数タスクの待ちを許さない
	TA_WMUL	0x0002	kernel.h	イベントフラグに複数タスクの待ちを許す
	TA_CLR	0x0004	kernel.h	待ち解除時にイベントフラグをクリア
	TA_CEILING	0x0003	kernel.h	優先度上限プロトコル
	TA_STA	0x0002	kernel.h	周期ハンドラを動作状態で生成
	TA_PHS	0x0004	kernel.h	周期ハンドラ位相を保存
タイムアウト	TMO_POL	0	itron.h	ポーリング
	TMO_FEVR	-1	itron.h	永久待ち
	TMO_NBLK	-2	itron.h	ノンブロッキング
動作モード	TWF_ANDW	0x0000	kernel.h	イベントフラグの AND 待ち
	TWF_ORW	0x0001	kernel.h	イベントフラグの OR 待ち
タスク例外	TTEX_ENA	0x0000	kernel.h	タスク例外許可状態
	TTEX_DIS	0x0001	kernel.h	タスク例外禁止状態
状態	TTS_RUN	0x0001	kernel.h	実行状態
	TTS_RDY	0x0002	kernel.h	実行可能状態
	TTS_WAI	0x0004	kernel.h	待ち状態
	TTS_SUS	0x0008	kernel.h	強制待ち状態
	TTS_WAS	0x000C	kernel.h	二重待ち状態
	TTS_DMT	0x0010	kernel.h	休止状態
	TTW_SLP	0x0001	kernel.h	起床待ち状態
	TTW_DLY	0x0002	kernel.h	時間経過待ち状態
	TTW_SEM	0x0004	kernel.h	セマフォ資源獲得待ち状態
	TTW_FLG	0x0008	kernel.h	イベントフラグ待ち状態
	TTW_SDTQ	0x0010	kernel.h	データキューへの送信待ち状態
	TTW_RDTQ	0x0020	kernel.h	データキューからの受信待ち状態
	TTW_MBX	0x0040	kernel.h	メールボックスからの受信待ち状態
	TTW_MTX	0x0080	kernel.h	ミューテックス待ち状態
	TTW_SMBF	0x0100	kernel.h	メッセージバッファへの送信待ち状態
	TTW_RMBF	0x0200	kernel.h	メッセージバッファからの受信待ち状態

分類	マクロ	定義内容	定義場所	説明
	TTW_MPF	0x2000	kernel.h	固定長メモリブロック獲得待ち待ち状態
	TTW_MPL	0x4000	kernel.h	可変長メモリブロック獲得待ち待ち状態
	TCYC_STP	0x0000	kernel.h	周期ハンドラ非動作状態
	TCYC_STA	0x0001	kernel.h	周期ハンドラ動作状態
	TALM_STP	0x0000	kernel.h	アラームハンドラ非動作状態
	TALM_STA	0x0001	kernel.h	アラームハンドラ動作状態
その他の定数	TSK_SELF	0	kernel.h	自タスク指定
	TSK_NONE	0	kernel.h	該当するタスクがない
	TPRI_SELF	0	kernel.h	自タスクのベース優先度の指定
	TPRI_INI	0	kernel.h	タスクの起動時優先度の指定
カーネル構成	TMIN_TPRI	1	kernel.h	タスク優先度の最小値
	TMAX_TPRI	system.priority	kernel_id.h	タスク優先度の最大値
	TMIN_MPRI	1	kernel.h	メッセージ優先度の最小値
	TMAX_MPRI	system.message_pri	kernel_id.h	メッセージ優先度の最大値
	TKERNEL_MAKER	0x011B	kernel.h	カーネルのメーカーコード
	TKERNEL_PRID	0x0004	kernel.h	カーネルの識別番号
	TKERNEL_SPVER	0x5403	kernel.h	ITRON 仕様のバージョン番号
	TKERNEL_PRVER	0x0100	kernel.h	カーネルのバージョン番号
	TMAX_ACTCNT	255	kernel.h	タスク起動要求キューイング数の最大値
	TMAX_WUPCNT	255	kernel.h	タスク起床要求キューイング数の最大値
	TMAX_SUSCNT	1	kernel.h	タスク強制待ち要求ネスト数の最大値
	TBIT_FLGPTN	32	kernel.h	イベントフラグのビット数
	TBIT_TEXPTN	32	kernel.h	タスク例外要因のビット数
	TIC_NUME	system.tic_nume	kernel_id.h	タイムティックの周期の分子
	TIC_DENO	system.tic_deno	kernel_id.h	タイムティックの周期の分母
	TMAX_MAXSEM	65535	kernel.h	セマフォの最大資源数の最大値
	VTMAX_DOMAIN	*1	kernel_id.h	最大ドメイン ID
	VTMAX_TSK	*1	kernel_id.h	最大タスク ID
	VTMAX_SEM	*1	kernel_id.h	最大セマフォ ID
	VTMAX_FLG	*1	kernel_id.h	最大イベントフラグ ID
	VTMAX_DTQ	*1	kernel_id.h	最大データキューID
	VTMAX_MBX	*1	kernel_id.h	最大メールボックス ID
	VTMAX_MTX	*1	kernel_id.h	最大ミューテックス ID
	VTMAX_MBF	*1	kernel_id.h	最大メッセージバッファ ID
	VTMAX_MPF	*1	kernel_id.h	最大固定長メモリプール ID
	VTMAX_MPL	*1	kernel_id.h	最大可変長メモリプール ID
	VTMAX_CYH	*1	kernel_id.h	最大周期ハンドラ ID
	VTMAX_ALH	*1	kernel_id.h	最大アラームハンドラ ID
	VTSZ_MBFTBL	4	kernel.h	メッセージバッファのメッセージ管理テーブルのサイズ(単位：バイト)
	VTMAX_AREASIZE	0x10000000	kernel.h	各種領域サイズの最大値(単位：バイト)
	VTKNL_LVL	system.system_IPL	kernel_id.h	カーネル割り込みマスクレベル
	VTIM_LVL	clock.IPL	kernel_id.h	タイマ割り込み優先レベル
エラー	E_SYS	-5	itron.h	システムエラー

分類	マクロ	定義内容	定義場所	説明
コード	E_NOSPT	-9	itron.h	未サポート機能
	E_RSFN	-10	itron.h	予約機能コード
	E_RSATR	-11	itron.h	予約属性
	E_PAR	-17	itron.h	パラメータエラー
	E_ID	-18	itron.h	不正 ID 番号
	E_CTX	-25	itron.h	コンテキストエラー
	E_MACV	-26	itron.h	メモリアクセス違反
	E_OACV	-27	itron.h	オブジェクトアクセス違反
	E_ILUSE	-28	itron.h	サービスコール不正使用
	E_NOMEM	-33	itron.h	メモリ不足
	E_NOID	-34	itron.h	ID 番号不足
	E_OBJ	-41	itron.h	オブジェクト状態エラー
	E_NOEXS	-42	itron.h	オブジェクト未生成
	E_QOVR	-43	itron.h	キューイングオーバフロー
	E_RLWAI	-49	itron.h	待ち状態の強制解除
	E_TMOUT	-50	itron.h	ポーリング失敗またはタイムアウト
	E_DLT	-51	itron.h	待ちオブジェクトの削除
	E_CLS	-52	itron.h	待ちオブジェクトの状態変化
	EV_RST	-127	itron.h	オブジェクトリセットによる待ち解除
保護拡張機能	TDOM_SELF	0	kernel.h	自タスクが所属するドメイン
	TACP_SHARED	((1u << (VTMAX_DOMAIN)) -1)	kernel.h	すべてのドメインがアクセスできることを示すアクセス許可パターン
	TACT_SRW *2	{TACP_SHARED, TACP_SHARED, TACP_SHARED}	kernel.h	すべてのアクセスがすべてのドメインに許可されることを示すアクセス許可パターン
	TACT_SRO *2	{TACP_SHARED, 0, TACP_SHARED}	kernel.h	書き込みアクセスはすべてのドメインに禁止され、読み出し・実行アクセスはすべてのドメインに許可されていることを示すアクセス許可パターン
	TPM_READ	1	kernel.h	オペランドリードアクセス
	TPM_WRITE	2	kernel.h	オペランドライトアクセス
	TPM_EXEC	4	kernel.h	実行アクセス

【注】 *1 コンフィギュレーション結果に依存

*2 初期代入文の右辺にのみ記述できます。

4.2.2 関数マクロ

(1) ER MERCD(ER ercd)

説明	ercd のメインエラーコードを返します。	
ヘッダ	itron.h	
引数	ercd	エラーコード
リターン値	ercd のメインエラーコード	

(2) ER SERCD(ER ercd)

説明	ercd のサブエラーコードを返します。	
ヘッダ	itron.h	
引数	ercd	エラーコード
リターン値	ercd のサブエラーコード	
備考	カーネルが返すエラーコードのサブエラーコードは全て-1 です。	

(3) ER ERCD(ER mercd, ER sercd)

説明	mercd のメインエラーコードと sercd のサブエラーコードからなるエラーコードを返します。	
ヘッダ	itron.h	
引数	mercd	メインエラーコード
	sercd	サブエラーコード
リターン値	エラーコード	

(4) SIZE TSZ_DTQ(UINT dtqcnt)

説明	データ数が dtqcnt のデータキュー領域のサイズを返します。(単位：バイト)	
ヘッダ	kernel.h	
引数	dtqcnt	データ数
リターン値	データキュー領域のサイズ	

(5) SIZE TSZ_MPF(UINT blkcnt, UINT blksz)

説明	blksz バイトのメモリブロックを blkcnt 個獲得可能な格納することができる固定長メモリプール領域のサイズを返します。(単位：バイト)	
ヘッダ	kernel.h	
引数	blkcnt	メモリブロック数
	blksz	メモリブロックサイズ
リターン値	固定長メモリプール領域のサイズ	

(6) SIZE TSZ_MPFMB(UINT blkcnt, UINT blksz)

説明	blksz バイトのメモリブロックを blkcnt 個獲得可能な格納することができる固定長メモリプールの管理領域のサイズを返します。(単位：バイト)	
ヘッダ	kernel.h	
引数	blkcnt	メモリブロック数
	blksz	メモリブロックサイズ
リターン値	固定長メモリプール管理領域のサイズ	

(7) ATR TA_DOM(ID domid)

説明	domid で指定されたドメイン ID に所属することを示す属性値を返します。 タスク生成時に指定する tskatr の bit7-4 の設定用マクロです。	
ヘッダ	kernel.h	
引数	domid	ドメイン ID(TDOM_SELF も指定可能)
リターン値	tskatr の bit7-4	

(8) ACPTN TACP(ID domid)

説明	domid で指定されたドメインからのみアクセスできるアクセス許可パターンを返します。	
ヘッダ	kernel.h	
引数	domid	ドメイン ID(TDOM_SELF は指定できません)
リターン値	アクセス許可パターン	

(9) ACVCT TACT_PRW (ID domid)

説明	すべてのアクセス(読み出し、書き込み、実行)が、domid で指定されたドメインにのみ許可されていることを示すアクセス許可ベクタを返します。	
ヘッダ	kernel.h	
引数	domid	ドメイン ID(TDOM_SELF は指定できません)
リターン値	アクセス許可ベクタ	
備考	初期代入文の右辺にのみ記述できます。	

(10) ACVCT TACT_PRO (ID domid)

説明	読み出し・実行アクセスが、domid で指定されたドメインにのみ許可され、書き込みアクセスがすべてのドメインに禁止されていることを示すアクセス許可ベクタを返します。	
ヘッダ	kernel.h	
引数	domid	ドメイン ID(TDOM_SELF は指定できません)
リターン値	アクセス許可ベクタ	
備考	初期代入文の右辺にのみ記述できます。	

(11) ACVCT TACT_SRPW (ID domid)

説明	読み出し・実行アクセスがすべてのドメインに許可され、書き込みアクセスが domid で指定されたドメインにのみ許可されていることを示すアクセス許可ベクタ	
ヘッダ	kernel.h	
引数	domid	ドメイン ID(TDOM_SELF は指定できません)
リターン値	アクセス許可ベクタ	
備考	初期代入文の右辺にのみ記述できます。	

5. サービスコールリファレンス

5.1 ヘッダファイル

アプリケーションのソースでは、RI600/PX が提供する kernel.h と、cfg600px が出力する kernel_id.h をインクルードしてください。

5.2 サービスコールのリターン値とエラーコード

5.2.1 概要

リターン値を持つサービスコールでは、正の値または 0(E_OK)が正常終了、負の値がエラーコードを意味します。正常終了時のリターン値の意味はサービスコール毎に異なりますが、多くのサービスコールの正常終了時は E_OK のみが返ります。

ただし、BOOL 型のリターン値を持つサービスコールはこの限りではありません。

5.2.2 メインエラーコードとサブエラーコード

エラーコードは、下位 8 ビットのメインエラーコードとそれを除いた上位ビットのサブエラーコードから構成されています。本カーネルが返す全てのエラーコードのサブエラーコードは-1 です。

なお、標準ヘッダ itron.h には、以下のマクロが定義されています。

- ER MERCD (ER ercd); エラーコードからメインエラーコードを取り出す
- ER SERCD (ER ercd); エラーコードからサブエラーコードを取り出す
- ER ERCD (ER mercd, ER sercd); メインエラーコードとサブエラーコードからエラーコードを生成する。

5.3 システム状態とサービスコール

サービスコールを呼び出せるかどうかは、システムの状態に依存します。

5.3.1 タスクコンテキストと非タスクコンテキスト

(1) sns で始まるサービスコール

sns で始まる名称のサービスコールは、タスクコンテキストと非タスクコンテキストのどちらからも呼び出せます。

(2) (1)以外のサービスコール

i で始まるサービスコールは非タスクコンテキスト専用、その他はタスクコンテキスト専用です。

許可されない状態から呼び出した場合、サービスコールによってエラー(E_CTX エラー)を検出するものとそうでないものがあるので、注意してください。詳細は、各サービスコールのエラーコード欄で確認してください。

5.3.2 CPU ロック状態

CPU ロック状態から呼び出し可能なサービスコールは以下のものに限定されています。これら以外のサービスコールを CPU ロック状態から呼び出した場合は、E_CTX エラーを検出します。

- ext_tsk(CPU ロック状態は解除されます)
- exd_tsk(CPU ロック状態は解除されます)
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

5.3.3 ディスパッチ禁止状態

待ち状態に遷移するサービスコールを呼び出すと、E_CTX エラーを返します。

5.3.4 カーネル管理外の割り込みハンドラなど

カーネル管理外割り込みハンドラなど、PSW.IPL>カーネル割り込みマスクレベル(system.system_IPL)状態では、一部[2]を除いてサービスコールを呼び出してはなりません。呼び出した場合、エラーE_CTX を返します。ただし、この場合、サービスコール処理内で PSW.IPL がカーネル割り込みマスクレベルに一時的に下がるため、このエラーはデバッグ目的でのみ利用してください。

5.4 µITRON 仕様外の仕様

vrst_dtq サービスコールなどのように"v", "iv", "V"で始まる名称は、µITRON4.0 仕様外の本カーネル独自の仕様です。

また、以下の"ixxx_yyy"('i'で始まる名称)のサービスコールは、µITRON4.0 仕様でタスクコンテキスト専用として用意されている"xxx_yyy"のサービスコールを非タスクコンテキストから呼び出せるようにしたもので、µITRON4.0 仕様外です。

ista_tsk, ichg_pri, iget_pri, iref_tsk, iref_tst, isus_tsk, irsm_tsk, ifrsm_tsk, ipol_sem, iref_sem, iclr_flg, ipol_flg, iref_flg, iprcv_dtq, iref_dtq, isnd_mbx, iprcv_mbx, iref_mbx, ipsnd_mbf, iref_mbf, ipget_mpf, irel_mpf, iref_mpf, ipget_mpl, iref_mpl, iset_tim, iget_tim, ista_cyc, istp_cyc, iref_cyc, ista_alm, istp_alm, iref_alm, ichg_ims, iget_ims, iref_ver, vprb_mem

[2] chg_ims, ichg_ims, get_ims, iget_ims, vsta_knl, ivsta_knl, vsys_dwn, ivsys_dwn。

5.5 タスク管理機能

表 5.1に、タスク管理機能の仕様を示します。

表5.1 タスク管理機能の仕様

No.	項目	内容
1	タスク ID	1～VTMAX_TSK *1
2	タスク優先度	1～TMAX_TPRI *2
3	タスク起動要求キューイング数の最大値	255
4	拡張情報(タスクに渡すパラメータ)	32 ビット
5	タスク属性	TA_HLNG：高級言語記述 TA_ACT：起動属性

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大タスク ID を意味します。
*2 cfg600px が kernel_id.h に出力するマクロで、system.priority に指定した値です。

表5.2 タスク管理機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_tsk		タスクの生成	T		E	D	U	
2	acre_tsk		タスクの生成(ID 番号自動割付け)	T		E	D	U	
3	del_tsk		タスクの削除	T		E	D	U	
4	act_tsk	[S]	タスクの起動	T		E	D	U	
5	iact_tsk	[S]			N	E	D	U	
6	can_act	[S]	タスク起動要求のキャンセル	T		E	D	U	
7	ican_act				N	E	D	U	
8	sta_tsk	[B]	タスクの起動(起動コード指定)	T		E	D	U	
9	ista_tsk				N	E	D	U	
10	ext_tsk	[S][B]	自タスクの終了	T		E	D	U	L
11	exd_tsk		自タスクの終了と削除	T		E	D	U	L
12	ter_tsk	[S][B]	タスクの強制終了	T		E	D	U	
13	chg_pri	[S][B]	タスク優先度の変更	T		E	D	U	
14	ichg_pri				N	E	D	U	
15	get_pri	[S]	タスク優先度の参照	T		E	D	U	
16	iget_pri				N	E	D	U	
17	ref_tsk		タスクの状態参照	T		E	D	U	
18	iref_tsk				N	E	D	U	
19	ref_tst		タスクの状態参照(簡易版)	T		E	D	U	
20	iref_tst				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.5.1 タスクの生成(cre_tsk, acre_tsk)

■C 言語 API

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

■パラメータ

tskid	タスク ID
pk_ctsk	タスク生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct t_ctsk {
          ATR     tskatr;     タスク属性
          VP_INT  exinf;      拡張情報
          FP      task;       タスクの実行開始アドレス
          PRI     itskpri;    タスク起動時優先度
          SIZE    stksz;      ユーザスタックサイズ(バイト数)
          VP      stk;        ユーザスタック領域の先頭アドレス
} T_CTSK;
```

■リターン値

・cre_tsk の場合
正常終了（E_OK）、またはエラーコード
・acre_tsk の場合
生成したタスクの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 (1)tskatr の bit0,bit2,bit3,bit8～bit15 のいずれかが 1 (2)VTMAX_DOMAIN < (tskatr の bit4～bit7 の値)
E_PAR	パラメータエラー (1)pk_ctsk == NULL (2)task == NULL (3)itskpri < 0, TMAX_TPRI < itskpri (4)stk が 16 バイト境界でない (5)stksz が 16 の倍数でない (6)stksz < system.context で定まる最小値未満[3], VTMAX_AREASIZE < stksz (7)stk+stksz＞0x100000000
E_ID	不正 ID 番号(cre_tsk のみ) tskid≦0, VTMAX_TSK < tskid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_ctsk が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOMEM	メモリ不足 stk == NULL
E_NOID	空き ID なし(acre_tsk のみ)
E_OBJ	オブジェクト状態不正(cre_tsk のみ) tskid のタスクが存在

[3] 表11.2参照

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_tsk は、指定された tskid のタスクを pk_ctsk で指定された内容で生成します。acre_tsk は pk_ctsk で指定された内容でタスクを生成し、生成したタスク ID を返します。

タスク生成時に行われる処理は、表 5.3の通りです。

表5.3 タスク生成時に行われる処理

No.	処理内容
1	タスク起動要求キューイング数をクリアする。
2	タスク例外処理ルーチンを定義されていない状態にする。

(1) タスク ID(tskid)

cre_tsk は、tskid で指定した ID のタスクを生成します。

(2) タスク属性(tskatr)

tskatr には以下を指定できます。

tskatr := (TA_HLNG | [TA_ACT] | [TA_DOM(domid)])

以下に、tskatr のビット位置を示します。

bit15～bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	ドメイン ID (TA_DOM(domid))				0	0	TA_ACT (=2)	TA_HLNG (=0)

- TA_HLNG(0x0000)
  タスクの記述言語は C 言語のみをサポートしています。
- TA_ACT(0x0002)
  TA_ACT を指定すると、生成されたタスクは起動され、READY 状態になります。タスク起動時に行われる処理は、表 5.4の通りです。TA_ACT を指定しない場合は、生成されたタスクは DORMANT 状態になります。

表5.4 タスク起動時に行われる処理

No.	処理内容
1	タスクのベース優先度と現在優先度を初期化する。
2	起床要求キューイング数をクリアする。
3	強制待ち要求ネスト数をクリアする。
4	保留例外要因をクリアする。
5	例外処理禁止状態にする。

- TA_DOM(domid) (bit4～7)
  生成するタスクが所属するドメイン ID を指定します。0 の場合は、本サービスコールを呼び出したタスクと同じドメインとします。
  TA_DOM()の domid には、所属するドメイン ID を指定します。domid に TDOM_SELF を指定した場合は、本サービスコールを呼び出したタスクと同じドメインとします。

(3) 拡張情報(exinf)

TA_ACT 属性、または act_tsk, iact_tsk によってタスクが起動された場合、exinf がタスクに引数として渡されます。また、exinf はタスク例外処理ルーチンにも引数として渡されます。exinf は、ユーザが生成するタスクに関する情報を設定するなどの目的で自由に使用できます。

(4) タスクの実行開始アドレス(task)

task には、タスクの実行開始アドレスを指定します。

(5) タスク起動時優先度(itskpri)

itskpri には、タスク起動時の優先度を指定します。指定可能な範囲は、1～TMAX_TPRI です。

(6) ユーザスタックサイズ(stksz), ユーザスタック領域の先頭アドレス(stk)

アプリケーション側でユーザスタック領域を確保し、その先頭アドレスを stk に、サイズを stksz に指定します。

なお、μITRON4.0 仕様には、stk に NULL を指定することでカーネルがユーザスタック領域を割り当てる機能がありますが、RI600/PX はこの機能をサポートしていません。

ユーザスタック領域は、以下を満たす必要があります。

1. 先頭アドレスは16バイト境界であること。そうでない場合はE_PARエラーを返します。
2. サイズは16の整数倍であること。そうでない場合はE_PARエラーを返します。
3. ユーザスタック領域は、他のユーザスタックおよびメモリオブジェクトと重なっていないこと。そうでない場合、エラーは検出されず、システムの正常な動作は保証されません。

5.5.2 タスクの削除(del_tsk)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	tskid≦0, VTMAX_TSK < tskid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_OBJ	オブジェクト状態不正
	tskid のタスクが休止状態ではない
E_NOEXS	オブジェクト未生成
	tskid のタスクが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
tskid で示されたタスクを削除します。

5.5.3 タスクの起動(act_tsk, iact_tsk)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(act_tsk のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない
E_QOVR	キューイングのオーバフロー(既に起動要求キューイング数が最大値に達している)

■機能説明

tskid で示されたタスクを起動します。起動したタスクは休止状態から実行可能状態へ移行します。

タスク起動時に行われる処理は、表 5.4の通りです。

act_tsk では、tskid=TSK_SELF(=0)の指定により、自タスクの指定になります。

対象タスクには、タスク生成時に指定したタスクの拡張情報がパラメータとして渡ります。

対象タスクが休止状態でない場合には、本サービスコールによるタスクの起動要求は、最大 255 回まで記憶されます。

5.5.4 タスクの起動要求のキャンセル(can_act, ican_act)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

キューイングされていた起動要求の回数(正の値または 0)、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの can_act の呼び出し (2)タスクコンテキストからの ican_act の呼び出し
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

tskid で示されたタスクにキューイングされていた起動要求キューイング数を求め、その結果をリターンパラメータとして返し、同時にその起動要求を全て無効にします。

can_act では、tskid=TSK_SELF(=0)の指定により、自タスクの指定になります。

休止状態のタスクを対象として呼び出すこともできます。その場合のリターン値は 0 となります。

5.5.5 タスクの起動(起動コード指定)(sta_tsk, ista_tsk)

■C 言語 API

```
ER      sta_tsk(ID tskid, VP_INT stacd);
ER      ista_tsk(ID tskid, VP_INT stacd);
```

■パラメータ

tskid	タスク ID
stacd	タスク起動コード

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	tskid≦0, VTMAX_TSK < tskid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(sta_tsk のみ)
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OBJ	オブジェクト状態エラー
	tskid のタスクが休止状態でない
E_NOEXS	オブジェクト未生成
	tskid のタスクが存在しない

■機能説明

tskid で示されたタスクを起動します。起動したタスクは休止状態から実行可能状態へ移行します。
タスク起動時に行われる処理は、表 5.4の通りです。
起動したタスクには、パラメータとして stacd で示されたタスク起動コードが渡されます。

5.5.6 自タスクの終了(ext_tsk)

■C 言語 API

```
void        ext_tsk(void);
```

■パラメータ

なし

■リターン値

サービスコールの呼び出し元には戻りません。

以下のエラーが発生するとシステムダウンとなります。

E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能説明

ext_tsk サービスコールは、自タスクを正常終了します。タスクの状態は、実行状態から休止状態へ移行します。起動要求がキューイングされている場合は、自タスクをいったん終了させた後に再起動します。再起動時に行われる処理は、表 5.4の通りです。

タスク終了時に行われる処理は、表 5.5の通りです。

表5.5 タスク終了時に行われる処理

項番	処理内容
1	タスクがロックしていたミューテックスをロック解除する。

本サービスコールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

本サービスコールは、ディスパッチ禁止状態および CPU ロック状態からも呼び出せます。この場合、ディスパッチ禁止状態および CPU ロック状態は解除されます。

なお、タスク開始関数からリターンした場合は、ext_tsk サービスコールと同じ動作となります。

非タスクコンテキストまたはカーネル管理外割り込みハンドラから本サービスコールを呼び出した場合、回復不可能なエラーとして、システムダウンルーチンにジャンプします。

5.5.7 自タスクの終了と削除(exd_tsk)

■C 言語 API

```
void        exd_tsk(void);
```

■パラメータ

なし

■リターン値

サービスコールの呼び出し元には戻りません。
以下のエラーが発生するとシステムダウンとなります。

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）

■機能説明

exd_tsk サービスコールは、自タスクを正常終了後、削除します。
タスク終了時に行われる処理は、表 5.5の通りです。

本サービスコールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

本サービスコールは、ディスパッチ禁止状態および CPU ロック状態からも呼び出せます。この場合、ディスパッチ禁止状態および CPU ロック状態は解除されます。

非タスクコンテキストまたはカーネル管理外割り込みハンドラから本サービスコールを呼び出した場合、回復不可能なエラーとして、システムダウンルーチンにジャンプします。

5.5.8 タスクの強制終了(ter_tsk)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	tskid≦0, VTMAX_TSK < tskid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_ILUSE	サービスコール不正使用
	tskid のタスクが自タスク
E_OBJ	オブジェクト状態エラー
	tskid のタスクが休止状態
E_NOEXS	オブジェクト未生成
	tskid のタスクが存在しない

■機能説明

tskid で示された他タスクを強制的に終了させます。終了させた他タスクは休止状態へ移行します。この時、表 5.5に示す処理が行われます。

起動要求がキューイングされている場合には、表 5.4のとおりタスクを起動する際に行うべき処理を行い、対象タスクを実行可能状態に移行します。

メッセージバッファ送信待ち、可変長メモリプールのメモリ獲得待ち行列の先頭タスクの待ちを強制終了させることによって、他のタスクの待ち(メッセージバッファ送信待ち,可変長メモリプールメモリ獲得待ち)が解除されることがあります。

本サービスコールは、タスクが占有していたミューテックス以外の資源（セマフォやメモリブロックなど）を自動的に解放する機能はありません。タスクは、必ず終了する前に資源の解放を行ってください。

5.5.9 タスク優先度の変更(chg_pri, ichg_pri)

■C 言語 API

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

■パラメータ

tskid	タスク ID
tskpri	タスクのベース優先度

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー tskpri < 0, TMAX_TPRI< tskpri
E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(chg_pri のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_ILUSE	サービスコール不正使用 上限優先度の違反
E_OBJ	オブジェクト状態エラー tskid のタスクが休止状態
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

tskid で示されたタスクのベース優先度を、tskpri で示された値に変更します。
chg_pri では、tskid=TSK_SELF(=0)の指定により、自タスク指定となります。
tskpri=TPRI_INI(=0)の指定により、タスク生成時に指定した初期タスク優先度に戻します。
変更したタスクのベース優先度は、タスクが終了、または本サービスコールを呼び出すまで有効です。タスクが休止状態になると終了前のタスクのベース優先度は無効になり、次に起動されたときにはタスク生成時に指定した初期タスク優先度になります。

tskid で示されたタスクの現在優先度も、tskpri で示された値に変更します。ただし、対象タスクが TA_CEILING 属性のミューテックスをロックしている場合は、現在優先度は変更しません。

対象タスクが TA_CEILING 属性のミューテックスをロックしているかロックを待っている場合で、tskpri に指定されたベース優先度が、それらのミューテックスのいずれかの上限優先度よりも高い場合には、E_ILUSE を返します。

5.5.10 タスク優先度の参照(get_pri, iget_pri)

■C 言語 API

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

■パラメータ

tskid	タスク ID
p_tskpri	現在優先度を返す記憶域へのポインタ

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー p_tskpri == NULL
E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの get_pri の呼び出し (2)タスクコンテキストからの iget_pri の呼び出し
E_MACV	メモリアクセス違反(get_pri のみ) 呼び出しタスクからの p_tskpri が示す領域に対するオペランドライトアクセス許可がない
E_OBJ	オブジェクト状態エラー tskid のタスクが休止状態
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

tskid で示されたタスクの現在優先度を参照し、p_tskpri の指す領域に返します。
get_pri では、tskid=TSK_SELF(=0)の指定により、自タスク指定となります。

5.5.11 タスクの状態参照(ref_tsk, iref_tsk)

■C 言語 API

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

■パラメータ

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

■パケットの構造

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rtsk == NULL
E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_tsk の呼び出し (2)タスクコンテキストからの iref_tsk の呼び出し
E_MACV	メモリアクセス違反(ref_tsk のみ) 呼び出しタスクからの pk_rtsk が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

tskid で示されたタスクの状態を参照し、pk_rtsk が指す領域に返します。
ref_tsk では、tskid=TSK_SELF(=0)の指定により自タスクの指定になります。
pk_rtsk の指す領域には、以下の値を返します。なお、*のデータはタスクが休止状態の場合は不定です。

(1) tskstat

現在のタスクの状態です。tskstat には、次の値を返します。
- TTS_RUN（0x0001）実行状態
- TTS_RDY（0x0002）実行可能状態
- TTS_WAI（0x0004）待ち状態
- TTS_SUS（0x0008）強制待ち状態
- TTS_WAS（0x000c）二重待ち状態
- TTS_DMT（0x0010）休止状態

(2) tskpri *

タスクの現在優先度です。

(3) tskbpri *

タスクのベース優先度です。

(4) tskwait *

tskstat が TTS_WAI、TTS_WAS のときに有効で、次の値を返します。
- TTW_SLP（0x0001）slp_tsk、tslp_tsk サービスコールによる待ち
- TTW_DLY（0x0002）dly_tsk サービスコールによる待ち
- TTW_SEM（0x0004）wai_sem、twai_sem サービスコールによる待ち
- TTW_FLG（0x0008）wai_flg、twai_flg サービスコールによる待ち
- TTW_SDTQ（0x0010）snd_dtq、tsnd_dtq サービスコールによる待ち
- TTW_RDTQ（0x0020）rcv_dtq、trcv_dtq サービスコールによる待ち
- TTW_MBX（0x0040）rcv_mbx、trcv_mbx サービスコールによる待ち
- TTW_MTX（0x0080）loc_mtx、tloc_mtx サービスコールによる待ち
- TTW_SMBF（0x0100）snd_mbf、tsnd_mbf サービスコールによる待ち
- TTW_RMBF（0x0200）rcv_mbf、trcv_mbf サービスコールによる待ち
- TTW_MPF（0x2000）get_mpf、tget_mpf サービスコールによる待ち
- TTW_MPL（0x4000）get_mpl、tget_mpl サービスコールによる待ち

(5) wobjid *

tskstat が TTS_WAI、TTS_WAS のときに有効で、待ち対象のオブジェクト ID を返します。

(6) lefttmo *

tskstat が TTS_WAI または TTS_WAS の場合で、かつ tskwait が TTW_DLY 以外の場合の、対象タスクの残り待ち時間を返します。次のタイムティックでタイムアウトする場合は 0 が返ります。

永久待ちの場合は TMO_FEVR が返ります。

TTW_DLY(dly_tsk による待ち状態)の場合は、この値は不定値となります。

(7) actcnt

現在の起動要求キューイング数を返します。

(8) wupcnt *

現在の起床要求キューイング数を返します。

(9) suscnt *

現在の強制待ち要求ネスト数を返します。

5.5.12 タスクの状態参照(簡易版)(ref_tst, iref_tst)

■C 言語 API

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

■パラメータ

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

■パケットの構造

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rtst == NULL
E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_tst の呼び出し (2)タスクコンテキストからの iref_tst の呼び出し
E_MACV	メモリアクセス違反(ref_tst のみ) 呼び出しタスクからの pk_rtst が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

本サービスコールの仕様は、ref_tsk, iref_tsk の簡易版です。tskstat と tskwait には、ref_tsk, iref_tsk と同じ値を返します。

ref_tst では、tskid=TSK_SELF(=0)の指定により自タスクの指定になります。

5.6 タスク付属同期機能

表 5.6に、タスク付属同期機能の仕様を示します。

表5.6 タスク付属同期機能の仕様

No.	項目	内容
1	タスク起床要求キューイング数の最大値	255
2	タスク強制待ちネスト数の最大値	1

表5.7 タスク付属同期機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	slp_tsk	[S][B]	起床待ち	T		E		U	
2	tslp_tsk	[S]	同上(タイムアウト有)	T		E		U	
3	wup_tsk	[S][B]	タスクの起床	T		E	D	U	
4	iwup_tsk	[S][B]			D	E	D	U	
5	can_wup	[S][B]	タスク起床要求のキャンセル	T		E	D	U	
6	ican_wup				D	E	D	U	
7	rel_wai	[S][B]	待ち状態の強制解除	T		E	D	U	
8	irel_wai	[S][B]			D	E	D	U	
9	sus_tsk	[S][B]	強制待ち状態への移行	T		E	D	U	
10	isus_tsk				D	E	D	U	
11	rsm_tsk	[S][B]	強制待ち状態からの再開	T		E	D	U	
12	irsm_tsk				D	E	D	U	
13	frsm_tsk	[S]	強制待ち状態からの強制再開	T		E	D	U	
14	ifrsm_tsk				D	E	D	U	
15	dly_tsk	[S][B]	自タスクの遅延	T		E		U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.6.1 起床待ち(slp_tsk, tslp_tsk)

■C 言語 API

```
ER          slp_tsk(void);
ER          tslp_tsk(TMO tmout);
```

■パラメータ

《tslp_tsk》

tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー(tslp_tsk のみ) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_RLWAI	待ち状態強制解除 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗 (tslp_tsk のみ)

■機能説明

自タスクを起床待ち状態に移行させます。ただし、自タスクに対する起床要求がキューイングされている場合は、起床要求キューイング数を 1 減らしてそのまま実行を継続します。

起床待ち状態は、wup_tsk, iwup_tsk サービスコールによって解除されます。この場合、本サービスコールは正常終了します。

tslp_tsk サービスコールでは、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち状態のまま tmout 時間が経過すると、待ち状態は解除され、エラーコードとして E_TMOUT が返ります。

tmout=TMO_POL(=0)を指定した場合､起床要求キューイング数が正なら起床要求キューイング数を 1 減らして実行を継続し、0 ならエラーコードとして E_TMOUT を返します。

tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。この場合、slp_tsk サービスコールと同じ動作となります。

5.6.2 タスクの起床(wup_tsk, iwup_tsk)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(wup_tsk のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OBJ	オブジェクト状態エラー tskid のタスクが休止状態
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない
E_QOVR	キューイングのオーバフロー 既に起床要求キューイング数が最大値に達している

■機能説明

slp_tsk、または tslp_tsk サービスコールの呼び出しにより待ち状態になっているタスクの待ち状態を解除します。

wup_tsk では、tskid=TSK_SELF(=0)の指定により、自タスクの指定になります。

対象タスクが slp_tsk、または tslp_tsk サービスコールによる待ち状態でない場合には、本サービスコールによる起床要求は、最大 255 回まで記憶されます。

5.6.3 タスク起床要求のキャンセル(can_wup, ican_wup)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

キューイングされていた起床要求の回数(正の値または 0)、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの can_wup の呼び出し (2)タスクコンテキストからの ican_wup の呼び出し
E_OBJ	オブジェクト状態エラー tskid のタスクが休止状態
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

tskid で示されたタスクにキューイングされていた起床要求回数を求め、その結果をリターンパラメータとして返し、同時にその起床要求を全て無効にします。

can_wup では、tskid=TSK_SELF(=0)の指定により、自タスクの指定になります。

5.6.4 待ち状態の強制解除(rel_wai, irel_wai)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	tskid≦0, VTMAX_TSK < tskid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(rel_wai のみ)
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OBJ	オブジェクト状態エラー
	tskid のタスクが待ち状態でない
E_NOEXS	オブジェクト未生成
	tskid のタスクが存在しない

■機能説明

tskid で示されるタスクが何らかの待ち状態(強制待ち状態は含まれません)の場合、それを強制的に解除します。本サービスコールにより待ち状態を解除したタスクには、エラーコードとして E_RLWAI が返ります。

二重待ち状態のタスクに対して本サービスコールを呼び出すと、対象タスクは強制待ち状態へ移行します。その後 rsm_tsk, irsm_tsk、または frsm_tsk, ifrsm_tsk サービスコールが呼び出され、強制待ち状態が解除されると、対象タスクにはエラーコードとして E_RLWAI が返されます。

メッセージバッファ送信待ち,可変長メモリプールのメモリ獲得待ち行列の先頭タスクの待ちを解除させることによって,他のタスクの待ち(メッセージバッファ送信待ち,可変長メモリプールメモリ獲得待ち)が解除されることがあります。

なお、強制待ち状態を解除するには、rsm_tsk, irsm_tsk, frsm_tsk, ifrsm_tsk を使用してください。

5.6.5 強制待ち状態への移行(sus_tsk, isus_tsk)

■C 言語 API

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

■パラメータ

tskid	タスク ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(sus_tsk のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OBJ	オブジェクト状態エラー (1)tskid のタスクが休止状態 (2)ディスパッチ禁止状態から呼び出した isus_tsk で、tskid に実行状態のタスクを指定
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない
E_QOVR	キューイングのオーバフロー 既に強制待ち要求ネスト数が最大値に達している

■機能説明

tskid で示されたタスクの実行を中断させ、強制待ち状態へ移行します。tskid で示されたタスクが待ち状態にある場合は、二重待ち状態へ移行します。このとき強制待ち要求ネスト数は 0 から 1 に変化します。 対象タスクが既に強制待ち状態もしくは二重待ち状態の場合は、エラーE_QOVR を返します。このとき、強制待ち要求ネスト数は 1 のままとなります。すなわち、本サービスコールによる強制待ち要求ネスト数の最大値は 1 です。

sus_tsk では、tskid=TSK_SELF(=0)の指定により自タスクの指定になります。ただし、ディスパッチ禁止状態において tskid に TSK_SELF または自タスク ID を指定して sus_tsk サービスコールを呼び出した場合、エラーE_CTX を返します。

強制待ち状態は、rsm_tsk, irsm_tsk、または frsm_tsk, ifrsm_tsk サービスコールの呼び出しにより解除されます。

5.6.6 強制待ち状態からの再開(rsm_tsk, irsm_tsk), 強制待ち状態からの強制再開 (frsm_tsk, ifrsm_tsk)

■C 言語 API

```
ER          rsm_tsk(ID tskid);
ER          irsm_tsk(ID tskid);
ER          frsm_tsk(ID tskid);
EE          ifrsm_tsk(ID tskid);
```

■パラメータ

tskid	タスク ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 tskid≦0, VTMAX_TSK < tskid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(rsm_tsk, frsm_tsk のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OBJ	オブジェクト状態エラー tskid のタスクが強制待ち状態でない
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

tskid で示されたタスクの強制待ち状態を解除します。

具体的には、rsm_tsk, irsm_tsk サービスコールは、tskid で示されたタスクが強制待ち状態の場合は、強制待ち要求ネスト数を 1 減算します。本カーネルでは、強制待ち要求ネスト数の最大値は 1 なので、これによって強制待ち用要求ネスト数は 0 になり、強制待ち状態が解除されます。

frsm_tsk, ifrsm_tsk サービスコールは、強制待ち要求ネスト数を 0 にします。本カーネルでは、強制待ち要求ネスト数の最大値は 1 なので、rsm_tsk, irsm_tsk と同じ振る舞いとなります。

5.6.7 タスク遅延(dly_tsk)

■C 言語 API

```
ER          dly_tsk(RELTIM dlytim);
```

■パラメータ

dlytim	遅延時間(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (0x7FFFFFFF - TIC_NUME)/TIC_DENO < dlytim
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_RLWAI	待ち状態強制解除 待ちの間に rel_wai, irel_wai サービスコールが呼び出された

■機能説明

自タスクの状態を実行状態から時間経過待ち状態へ移行し、dlytim で指定された時間が経過するのを待ちます。dlytim で指定された時間が経過した時点で、自タスクの状態を実行可能状態に移行します。dlytim＝0 を指定した場合にも、自タスクを待ち状態に移行させます。

本サービスコールは、tslp_tsk サービスコールとは異なり、dlytim 時間だけ実行を遅延して終了した場合に正常終了します。また、遅延時間中に wup_tsk, iwup_tsk サービスコールが実行されても、待ち状態は解除されません。遅延時間が経過する前に待ち状態を解除するのは、rel_wai, irel_wai または ter_tsk サービスコールが呼び出された場合に限られます。

5.7 タスク例外処理機能

表 5.8に、タスク例外処理機能の仕様を示します。

表5.8 タスク付属同期機能の仕様

No.	項目	内容
1	例外要因	32 ビット
2	タスク例外処理ルーチン起動条件	以下のすべてを満たしたとき 1. 当該タスクはタスク例外許可状態である。 2. 該当タスクの保留例外要因が 0 でない。 3. 非タスクコンテキストが実行されていない。 4. 該当タスクは実行状態である。

表5.9 タスク例外処理機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	def_tex		タスク例外処理ルーチンの定義	T		E	D	U	
2	ras_tex	[S]	タスク例外処理の要求	T		E	D	U	
3	iras_tex	[S]			N	E	D	U	
4	dis_tex	[S]	タスク例外処理の禁止	T		E	D	U	
5	ena_tex	[S]	タスク例外処理の許可	T		E	D	U	
6	sns_tex	[S]	タスク例外処理禁止状態の参照	T	N	E	D	U	L
7	ref_tex		タスク例外処理の状態参照	T		E	D	U	
8	iref_tex				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.7.1 タスク例外処理ルーチンの定義(def_tex)

■C 言語 API

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

■パラメータ

tskid	タスク ID
pk_dtex	タスク例外処理ルーチン定義情報を入れたパケットへのポインタ

■パケットの構造

```
typedef    struct t_dtex {
           ATR     texatr;      タスク例外処理ルーチン属性
           FP      texrtn;      タスク例外処理ルーチンの実行開始アドレス
} T_DTEX;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_RSATR	予約属性 texatr != TA_HLNG
E_PAR	パラメータエラー texrtn == NULL
E_ID	不正 ID 番号 tskid < 0, VTMAX_TSK < tskid,
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)pk_dtex!=NULL で、呼出しタスクからの pk_dtex が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

tskid で指定されたタスクに対して、pk_dtex で指定された内容でタスク例外処理ルーチンを定義します。

既にタスク例外処理ルーチンが定義されていた場合は、本サービスコールによって定義内容が更新されます。

pk_dtex に NULL(=0)を指定すると、タスク例外処理ルーチンの定義を解除するとともに、保留例外要因を 0 にクリアし、タスク例外を禁止します。

(1) タスク ID(タスク ID)

タスク例外処理ルーチンを定義するタスク ID を指定します。
tskid=TSK_SELF(=0)の指定により自タスクの指定になります。

(2) タスク例外処理ルーチン属性(texatr)

texatr には TA_HLNG のみを指定できます。

- TA_HLNG(0x0000)
  タスク例外処理ルーチンの記述言語は C 言語のみをサポートしています。

(3) タスク例外処理ルーチンの実行開始アドレス(texrtn)

texrtn には、タスク例外処理ルーチンの実行開始アドレスを指定します。

5.7.2 タスク例外処理の要求(ras_tex, iras_tex)

■C 言語 API

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

■パラメータ

tskid	タスク ID
rasptn	要求するタスク例外要因

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー rasptn == 0
E_ID	不正 ID 番号 (1)tskid<0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(ras_tex のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OBJ	オブジェクト状態エラー (1)tskid のタスクが休止状態 (2)tskid のタスクにタスク例外処理ルーチンが定義されていない
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

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

ras_tex では、tskid=TSK_SELF(=0)の指定により自タスクの指定になります。

本サービスコールにより、タスク例外処理ルーチンを起動する条件が揃った場合には、タスク例外処理ルーチンを起動する処理を行います。起動する条件は、「3.5 タスク例外処理機能」を参照してください。

タスク例外処理ルーチン起動時には、保留例外要因は 0 クリアされ、タスク例外は禁止されます。タスク例外処理ルーチンには、引数として 0 クリア前の保留例外要因と、タスクの拡張情報が渡されます。

タスク例外処理ルーチンが終了すると、タスク例外許可状態に変更され、そのタスクはタスク例外処理ルーチン起動直前の地点から実行を再開します。

タスク例外処理ルーチンのリターン時には、カーネルはタスクのユーザスタックにタスクのコンテキストレジスタを保存します。この処理においてユーザスタックのオーバフローを検出した場合は、システムダウンとなります。

タスク例外処理ルーチンで CPU ロック状態に移行した場合、タスク例外処理ルーチンからリターンするまでに CPU ロック状態を解除しなければなりません。CPU ロック状態のままタスク例外処理ルーチンからリターンした場合、システムダウンとなります。

割り込み優先レベル(PSW.IPL)は、タスク例外処理ルーチンの起動前後、タスク例外処理ルーチンからのリターン前後で変化しません。割り込み優先順位がカーネル割り込みマスクレベルより高い状態で、タスク例外処理ルーチンからリターンした場合は、システムダウンとなります。

5.7.3 タスク例外処理の禁止(dis_tex)

■C 言語 API

```
ER          dis_tex(void);
```

■パラメータ

なし

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_OBJ	オブジェクト状態エラー 自タスクにタスク例外処理ルーチンが定義されていない

■機能説明

自タスクのタスク例外を禁止します。

5.7.4 タスク例外処理の許可(ena_tex)

■C 言語 API

```
ER          ena_tex(void);
```

■パラメータ

なし

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OBJ	オブジェクト状態エラー 自タスクにタスク例外処理ルーチンが定義されていない

■機能説明

自タスクのタスク例外を許可します。

5.7.5 タスク例外処理禁止状態の参照(sns_tex)

■C 言語 API

```
BOOL        sns_tex(void);
```

■パラメータ

なし

■リターン値

TRUE または FALSE またはエラーコード

■エラーコード

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）

■機能説明

実行状態のタスクがタスク例外許可状態なら FALSE、そうでなければ TRUE を返します。

表5.10 sns_tex のリターン値

実行状態のタスク	実行状態のタスクのタスク例外処理ルーチン	実行状態のタスクのタスク例外禁止状態	リターン値	備考
存在する	定義済み	許可状態	FALSE	
		禁止状態	TRUE	
	未定義	禁止状態	TRUE	タスク例外処理ルーチンが未定義の時は、タスク例外禁止状態です。
存在しない	－	－	TRUE	

5.7.6 タスク例外処理の状態参照(ref_tex,iref_tex)

■C 言語 API

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

■パラメータ

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

■パケットの構造

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rtex == NULL
E_ID	不正 ID 番号 (1)tskid <0, VTMAX_TSK < tskid (2)非タスクコンテキストからの呼び出しで、tskid == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_tex の呼び出し (2)タスクコンテキストからの iref_tex の呼び出し
E_MACV	メモリアクセス違反(ref_tex のみ) 呼び出しタスクからの pk_rtex が示す領域に対するオペランドライトアクセス許可がない
E_OBJ	オブジェクト状態エラー (1)tskid のタスクが休止状態 (2)tskid のタスクにタスク例外処理ルーチンが定義されていない
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

tskid で示されたタスクのタスク例外処理ルーチンの状態を参照し、pk_rtex が指す領域に返します。
ref_tex では、tskid=TSK_SELF(=0)の指定により自タスクの指定になります。
pk_rtex の指す領域には、以下の値を返します。

(1) タスク例外処理の状態(texstat)

次のいずれかの値を返します。

- TTEX_ENA(=0）タスク例外処理許可状態
- TTEX_DIS(=1）タスク例外処理禁止状態

(2) 保留例外要因(pndptn)

保留されている例外要因です。

5.8 同期・通信(セマフォ)機能

表 5.11に、セマフォ機能の仕様を示します。

表5.11 セマフォ機能の仕様

No.	項目	内容
1	セマフォ ID	1～VTMAX_SEM *1
2	最大セマフォカウント	1～TMAX_MAXSEM *2
3	セマフォ属性	TA_TFIFO :タスク待ち行列は FIFO 順 TA_TPRI : タスクの待ち行列はタスクの現在優先度順

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大セマフォ ID を意味します。
*2 kernel.h で定義されるマクロです。本カーネルでの定義値は 65535 です。

表5.12 セマフォ機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_sem		セマフォの生成	T		E	D	U	
2	acre_sem		セマフォの生成(ID 番号自動割付け)	T		E	D	U	
3	del_sem		セマフォの削除	T		E	D	U	
4	sig_sem	[S][B]	セマフォ資源の返却	T		E	D	U	
5	isig_sem	[S][B]			N	E	D	U	
6	wai_sem	[S][B]	セマフォ資源の獲得	T		E		U	
7	pol_sem	[S][B]	同上(ポーリング)	T		E	D	U	
8	ipol_sem				N	E	D	U	
9	twai_sem	[S]	同上(タイムアウト有)	T		E		U	
10	ref_sem		セマフォの状態参照	T		E	D	U	
11	iref_sem				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.8.1 セマフォの生成(cre_sem, acre_sem)

■C 言語 API

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

■パラメータ

semid	セマフォ ID
pk_csem	セマフォ生成情報を格納したパケットへのポインタ

■パケットの構造

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

■リターン値

・cre_sem の場合
正常終了（E_OK）、またはエラーコード
・acre_sem の場合
生成したセマフォの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 sematr の bit0 以外が 1
E_PAR	パラメータエラー (1)pk_csem == NULL (2)maxsem≦0, TMAX_MAXSEM < maxsem (3)maxsem < isemcnt
E_ID	不正 ID 番号(cre_sem のみ) semid≦0, VTMAX_SEM < semid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_csem が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOID	空き ID なし(acre_sem のみ)
E_OBJ	オブジェクト状態不正(cre_sem のみ) semid のセマフォが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_sem は、指定された semid のセマフォを pk_csem で指定された内容で生成します。acre_sem は pk_csem で指定された内容でセマフォを生成し、生成したセマフォ ID を返します。

(1) セマフォ ID(semid)

cre_sem は、semid で指定した ID のセマフォを生成します。

(2) セマフォ属性(sematr)

sematr には以下を指定できます。

sematr := (TA_TFIFO || TA_TPRI)

- TA_TFIFO(0x0000)
  待ち行列の順序を FIFO 順とします。
- TA_TPRI(0x0001)
  待ち行列の順序をタスクの現在優先度順とします。

(3) セマフォ資源数の初期値(isemcnt)

セマフォ資源数の初期値を、0～maxsem の範囲で指定します。

(4) セマフォ資源数の最大値(maxsem)

セマフォ資源数の最大値を指定します。

5.8.2 セマフォの削除(del_sem)

■C 言語 API

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

■パラメータ

semid	セマフォ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 semid≦0, VTMAX_SEM < semid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成 semid のセマフォが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

semid で示されたセマフォを削除します。

semid で示されたセマフォで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

5.8.3 セマフォ資源の返却(sig_sem, isig_sem)

■C 言語 API

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

■パラメータ

semid	セマフォ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	semid≦0, VTMAX_SEM < semid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(sig_sem のみ)
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	semid のセマフォが存在しない
E_QOVR	キューイングオーバフロー
	既にセマフォのカウント値が最大値に達している

■機能説明

semid で示されたセマフォに資源をひとつ返却します。対象セマフォで待っているタスクがあれば、セマフォの待ち行列先頭タスクに資源を割り付けて待ち状態を解除します。セマフォに対して待っているタスクがなければ、そのセマフォの資源数を 1 増やします。

なお、セマフォの資源数の最大値は、セマフォ生成時に指定します。

5.8.4 セマフォ資源の獲得(wai_sem, pol_sem, ipol_sem, twai_sem)

■C 言語 API

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

■パラメータ

semid	セマフォ ID
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 semid≦0, VTMAX_SEM < semid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの pol_sem の呼び出し (2)タスクコンテキストからの ipol_sem の呼び出し
E_MACV	メモリアクセス違反(wai_sem, twai_sem のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成 semid のセマフォが存在しない
E_RLWAI	待ち状態強制解除(wai_sem, twai_sem のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(wai_sem, twai_sem のみ) 待ちの間に semid のセマフォが削除された

■機能説明

semid で指定されるセマフォから、資源をひとつ獲得します。

対象セマフォの資源数が 1 以上の場合には、セマフォの資源数から 1 を減じ、実行を継続します。資源数が 0 の場合には、wai_sem, twai_sem サービスコールでは呼び出しタスクはそのセマフォの待ち行列につながれ、pol_sem, ipol_sem サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

twai_sem サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、pol_sem サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。この場合、wai_sem サービスコールと同じ動作となります。

5.8.5 セマフォの状態参照(ref_sem, iref_sem)

■C 言語 API

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

■パラメータ

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

■パケットの構造

```
typedef     struct t_rsem {
            ID      wtskid;     待ち行列先頭のタスク ID
            UINT    semcnt;     現在のセマフォカウント値
} T_RSEM;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rsem == NULL
E_ID	不正 ID 番号 semid≦0, VTMAX_SEM < semid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_sem の呼び出し (2)タスクコンテキストからの iref_sem の呼び出し
E_MACV	メモリアクセス違反(ref_sem のみ) 呼び出しタスクからの pk_rsem が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 semid のセマフォが存在しない

■機能説明

semid で示されたセマフォの状態を参照します。
pk_rsem が指す領域に、待ち行列の先頭タスク ID(wtskid)、現在のセマフォカウント値(semcnt)を返します。
対象セマフォの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE(=0)を返します。

5.9 同期・通信(イベントフラグ)機能

表 5.13に、イベントフラグ機能の仕様を示します。

表5.13 イベントフラグ機能の仕様

No.	項目	内容
1	イベントフラグ ID	1～VTMAX_FLG *1
2	イベントフラグのビット長	32 ビット
3	イベントフラグ属性	TA_TFIFO :タスク待ち行列は FIFO 順 TA_TPRI : タスクの待ち行列はタスクの現在優先度順 *2 TA_WSGL : 複数タスクの待ちを許さない TA_WMUL : 複数タスクの待ちを許す TA_CLR : 待ち解除時にイベントフラグを 0 クリア

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大イベントフラグ ID を意味します。
*2 TA_CLR 指定がない場合は、TA_TPRI を指定してもタスクの待ち行列は FIFO 順で管理されます。この振る舞いは、μITRON4.0 仕様の範囲外です。

表5.14 イベントフラグ機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_flg		イベントフラグの生成	T		E	D	U	
2	acre_flg		イベントフラグの生成(ID 番号自動割付け)	T		E	D	U	
3	del_flg		イベントフラグの削除	T		E	D	U	
4	set_flg	[S][B]	イベントフラグのセット	T		E	D	U	
5	iset_flg	[S][B]			N	E	D	U	
6	clr_flg	[S][B]	イベントフラグのクリア	T		E	D	U	
7	iclr_flg				N	E	D	U	
8	wai_flg	[S][B]	イベントフラグ待ち	T		E		U	
9	pol_flg	[S][B]	同上(ポーリング)	T		E	D	U	
10	ipol_flg				N	E	D	U	
11	twai_flg	[S]	同上(タイムアウト有)	T		E		U	
12	ref_flg		イベントフラグの状態参照	T		E	D	U	
13	iref_flg				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.9.1 イベントフラグの生成(cre_flg, acre_flg)

■C 言語 API

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

■パラメータ

flgid	イベントフラグ ID
pk_cflg	イベントフラグ生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct t_cflg {
           ATR     flgatr;      イベントフラグ属性
           FLGPTN  iflgptn;     イベントフラグの初期値
} T_CFLG;
```

■リターン値

・cre_flg の場合
正常終了（E_OK）、またはエラーコード
・acre_flg の場合
生成したイベントフラグの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 flgatr の bit0, bit1, bit2 以外が 1
E_PAR	パラメータエラー pk_cflg == NULL
E_ID	不正 ID 番号(cre_flg のみ) flgid≦0, VTMAX_FLG < flgid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_cflg が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOID	空き ID なし(acre_flg のみ)
E_OBJ	オブジェクト状態不正(cre_flg のみ) flgid のイベントフラグが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_flg は、指定された flgid のイベントフラグを pk_cflg で指定された内容で生成します。acre_flg は pk_cflg で指定された内容でイベントフラグを生成し、生成したイベントフラグ ID を返します。

(1) イベントフラグ ID(flgid)

cre_flg は、flgid で指定した ID のイベントフラグを生成します。

(2) イベントフラグ属性(flgatr)

flgatr には以下を指定できます。

flgatr := ((TA_TFIFO || TA_TPRI) | (TA_WSGL || TA_WMUL) | [TA_CLR])

- TA_TFIFO(0x0000)
  待ち行列の順序を FIFO 順とします。
- TA_TPRI(0x0001)
  待ち行列の順序をタスクの現在優先度順とします。
  ただし、TA_CLR の指定がない場合は、TA_TPRI を指定してもタスクの待ち行列は FIFO 順で管理されます。この振る舞いは、μITRON4.0 仕様の範囲外です。
- TA_WSGL(0x0000)
  イベントフラグに対し、複数のタスクが待つことを禁止します。
- TA_WMUL (0x0002)
  イベントフラグに対し、複数のタスクが待つことを許可します。
- TA_CLR(0x0004)
  待ち解除時に、イベントフラグのすべてのビットをクリアします。

(3) イベントフラグの初期値(iflgptn)

イベントフラグの初期値を指定します。

5.9.2 イベントフラグの削除(del_flg)

■C 言語 API

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

■パラメータ

flgid	イベントフラグ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	flgid ≦0, VTMAX_FLG < flgid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	flgid のイベントフラグが存在しない
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

flgid で示されたイベントフラグを削除します。

flgid で示されたイベントフラグで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

5.9.3 イベントフラグのセット(set_flg, iset_flg)

■C 言語 API

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

■パラメータ

flgid	イベントフラグ ID
setptn	セットするビットパターン

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 flgid≦0, VTMAX_FLG < flgid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(set_flg のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成 flgid のイベントフラグが存在しない

■機能説明

flgid で示されたイベントフラグを、setptn で示された値との論理和(OR)に更新します。

そして、待ち行列の順に待ち行列につながれているタスクの待ち解除条件を満たすかどうかを調べます。待ち解除条件を満たせば、該当タスクの待ち状態を解除します。このとき、対象のイベントフラグ属性に TA_CLR 属性が指定されている場合には、イベントフラグのビット・パターンを 0 クリアし、処理を終了します。

イベントフラグに TA_WMUL 属性が指定されており、かつ TA_CLR 属性が指定されていない場合、set_flg, iset_flg の 1 回の呼び出しで複数のタスクが待ち解除される可能性があります。待ち解除されるタスクが複数ある場合には、イベントフラグの待ち行列につながれていた順序で待ち解除されます。

5.9.4 イベントフラグのクリア(clr_flg, iclr_flg)

■C 言語 API

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

■パラメータ

flgid	イベントフラグ ID
clrptn	クリアするビットパターン

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 flgid≦0, VTMAX_FLG < flgid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの clr_flg の呼び出し (2)タスクコンテキストからの iclr_flg の呼び出し
E_NOEXS	オブジェクト未生成 flgid のイベントフラグが存在しない

■機能説明

flgid で示されたイベントフラグを、clrptn で示された値との論理積(AND)に更新します。

clrptn の全ビットを 1 とした場合、イベントフラグに対して何の操作も行なわないことになりますが、エラーにはなりません。

5.9.5 イベントフラグ待ち(wai_flg, pol_flg, ipol_flg, twai_flg)

■C 言語 API

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

■パラメータ

flgid	イベントフラグ ID
waiptn	待ちビットパターン
wfmode	待ちモード
p_flgptn	待ち解除時のビットパターンを返す記憶域へのポインタ
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)p_flgptn == NULL (2)waiptn == 0 (3)wfmode が不正 (4)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 flgid≦0, VTMAX_FLG < flgid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの pol_flg の呼び出し (2)タスクコンテキストからの ipol_flg の呼び出し
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (wai_flg, twai_flg のみ) (2)呼び出しタスクからの p_flgptn が示す領域に対するオペランドライトアクセス許可がない (wai_flg, pol_flg, twai_flg のみ)
E_ILUSE	サービスコール不正使用 TA_WSGL 属性のイベントフラグに待ちタスクが存在
E_NOEXS	オブジェクト未生成 flgid のイベントフラグが存在しない
E_RLWAI	待ち状態強制解除(wai_flg, twai_flg のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	ポーリング失敗、またはタイムアウト
E_DLT	待ちオブジェクトの削除(wai_flg, twai_flg のみ) 待ちの間に flgid のイベントフラグが削除された

■機能説明

flgid で指定されるイベントフラグのビットパターンが、waiptn と wfmode で指定される待ち解除条件を満たすのを待ちます。p_flgptn の指す領域には、待ち解除される時のイベントフラグのビットパターンを返します。

本サービスコール呼び出し時にすでに待ち解除条件が成立している場合は、本サービスコールは直ちに終了します。待ち解除条件が成立していない場合は、wai_flg, twai_flg サービスコールの場合はイベント待ち行列につながれ、pol_flg, ipol_flg サービスコールの場合は直ちにエラーE_TMOUT で終了します。

待ち行列は、生成時に TA_TFIFO 属性を指定した場合は FIFO で管理されます。

一方、生成時に TA_TPRI 属性を指定した場合は、タスクの現在優先度順で管理されます。同じ優先度のタスクの中では、FIFO で管理されます。

ただし、TA_CLR 属性が指定されていない場合は、TA_TPRI 属性を指定しても待ち行列の管理は TA_FIFO 属性と同じとなります。この振る舞いは、μITRON4.0 仕様の範囲外です。

なお、TA_WSGL 属性を指定した場合は複数のタスクが同時にイベントフラグを待つことはできないため、TA_TFIFO 属性と TA_TPRI 属性の違いはありません。

wfmode には、次のような指定を行います。

```
wfmode := ( (TWF_ANDW||TWF_ORW) )
```

- TWF_ANDW （0x00000000）：AND 待ち
- TWF_ORW （0x00000001）：OR 待ち

TWF_ANDW では、waiptn で指定したビットの全てがセットされるのを待ちます。TWF_ORW では、flgid で示されたイベントフラグのうち waiptn で指定したビットのいずれかがセットされるのを待ちます。

twai_flg サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、pol_flg サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。この場合、wai_flg サービスコールと同じ動作となります。

5.9.6 イベントフラグの状態参照(ref_flg, iref_flg)

■C 言語 API

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

■パラメータ

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

■リターン値

正常終了（E_OK）、またはエラーコード

■パケットの構造

```
typedef    struct t_rflg {
           ID      wtskid;      待ち行列先頭のタスク ID
           FLGPTN  flgptn;      イベントフラグのビットパターン
} T_RFLG;
```

■エラーコード

E_PAR	パラメータエラー pk_rflg == NULL
E_ID	不正 ID 番号 flgid≦0, VTMAX_FLG < flgid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_flg の呼び出し (2)タスクコンテキストからの iref_flg の呼び出し
E_MACV	メモリアクセス違反(ref_flg み) 呼び出しタスクからの pk_rflg が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 flgid のイベントフラグが存在しない

■機能説明

flgid で示されたイベントフラグの状態を参照します。

pk_rflg の指す領域に、待ち行列の先頭タスク ID(wtskid)、現在のイベントフラグのビットパターン(flgptn)を返します。

対象イベントフラグの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE(=0)を返します。

5.10 同期・通信(データキュー)機能

表 5.15に、データキュー機能の仕様を示します。

表5.15 データキュー機能の仕様

No.	項目	内容
1	データキューID	1～VTMAX_DTQ *1
2	データサイズ	4 バイト
3	データの個数	最大 65535
4	データキュー属性	TA_TFIFO :送信タスクの待ち行列は FIFO 順 TA_TPRI : 送信タスクの待ち行列はタスクの現在優先度順

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大データキューID を意味します。

表5.16 データキュー機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_dtq		データキューの生成	T		E	D	U	
2	acre_dtq		データキューの生成(ID 番号自動割付け)	T		E	D	U	
3	del_dtq		データキューの削除	T		E	D	U	
4	snd_dtq	[S]	データキューへの送信	T		E		U	
5	psnd_dtq	[S]	同上(ポーリング)	T		E	D	U	
6	ipsnd_dtq	[S]			N	E	D	U	
7	tsnd_dtq	[S]	同上(タイムアウト有)	T		E		U	
8	fsnd_dtq	[S]	データキューへの強制送信	T		E	D	U	
9	ifsnd_dtq	[S]			N	E	D	U	
10	rcv_dtq	[S]	データキューからの受信	T		E		U	
11	prcv_dtq	[S]	同上(ポーリング)	T		E	D	U	
12	iprcv_dtq				N	E	D	U	
13	trcv_dtq	[S]	同上(タイムアウト有)	T		E		U	
14	ref_dtq		データキューの状態参照	T		E	D	U	
15	iref_dtq				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.10.1 データキューの生成(cre_dtq, acre_dtq)

■C 言語 API

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

■パラメータ

dtqid	データキューID
pk_cdtq	データキュー生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct t_cdtq {
           ATR     dtqatr;      データキュー属性
           UINT    dtqcnt;      データキュー領域の容量(データの個数)
           VP      dtq;         データキュー領域の先頭アドレス
} T_CDTQ;
```

■リターン値

・cre_dtq の場合
正常終了（E_OK）、またはエラーコード
・acre_dtq の場合
生成したデータキューの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 dtqatr の bit0 以外が 1
E_PAR	パラメータエラー (1)pk_cdtq == NULL (2)dtqcnt > 65535 (3)dtqcnt!=0 で、dtq+TSZ_DTQ(dtqcnt)>0x100000000
E_ID	不正 ID 番号(cre_dtq のみ) dtqid≦0, VTMAX_DTQ < dtqid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_cdtq が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOMEM	メモリ不足 dtqcnt!=0 で、dtq == NULL
E_NOID	空き ID なし(acre_dtq のみ)
E_OBJ	オブジェクト状態不正(cre_dtq のみ) dtqid のデータキューが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_dtq は、指定された dtqid のデータキューを pk_cdtq で指定された内容で生成します。acre_dtq は pk_cdtq で指定された内容でデータキューを生成し、生成したデータキューID を返します。

(1) データキューID(dtqid)

cre_dtq は、dtqid で指定した ID のデータキューを生成します。

(2) データキュー属性(dtqatr)

dtqatr には以下を指定できます。

dtqatr := (TA_TFIFO || TA_TPRI)

- TA_TFIFO(0x0000)
  送信待ち行列の順序を FIFO 順とします。
- TA_TPRI(0x0001)
  送信待ち行列の順序をタスクの現在優先度順とします。

なお、受信待ち行列は常に FIFO 順につながれます。

(3) データキュー領域の容量(dtqcnt)、データキュー領域の先頭アドレス(dtq)

アプリケーション側で、TSZ_DTQ(dtqcnt)バイトのデータキュー領域を確保し、その先頭アドレスを dtq に指定します。

なお、μITRON4.0 仕様には、dtq に NULL を指定することでカーネルがデータキュー領域を割り当てる機能がありますが、RI600/PX はこの機能をサポートしていません。

また、dtqcnt に 0 を指定することもできます。この場合、データキューにデータを蓄えておくことはできないため、送信側と受信側の先に実行した方が待ち状態になり、他方が行われた時点で待ちが解除される、つまり送信側と受信側が完全に同期した動作となります。なお、dtqcnt が 0 の場合は、dtq は無視されます。

カーネルは、データキュー領域に関するアクセス権については何も関知しません。通常は、データキュー領域はメモリオブジェクト・ユーザスタック以外の領域に作成してください。メモリオブジェクト内にデータキュー領域を作成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤ってデータキュー領域を書き換えてしまう危険があります。

5.10.2 データキューの削除(del_dtq)

■C 言語 API

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

■パラメータ

dtqid	データキューID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	dtqid ≦0, VTMAX_DTQ < dtqid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	dtqid のデータキューが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

dtqid で示されたデータキューを削除します。

dtqid で示されたデータキューで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

5.10.3 データキューへの送信(snd_dtq,psnd_dtq,ipsnd_dtq,tsnd_dtq, fsnd_dtq, ifsnd_dtq)

■C 言語 API

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

■パラメータ

dtqid	データキューID
data	データキューへ送信するデータ
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー
	tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号
	dtqid≦0, VTMAX_DTQ < dtqid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_ILUSE	サービスコール不正使用
	dtqcnt が 0 のデータキューに対する fsnd_dtq, ifsnd_dtq の発行
E_MACV	メモリアクセス違反(snd_dtq,psnd_dtq,tsnd_dtq,fsnd_dtq のみ)
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	dtqid のデータキューが存在しない
E_RLWAI	待ち状態強制解除(snd_dtq, tsnd_dtq のみ)
	待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(snd_dtq, tsnd_dtq のみ)
	待ちの間に dtqid のデータキューが削除された
EV_RST	オブジェクトリセット(vrst_dtq)による待ち解除(snd_dtq, tsnd_dtq のみ)

■機能説明

dtqid で示されたデータキューに対して、data で示されたデータ(4 バイト)を送信します。

なお、fsnd_dtq, ifsnd_dtq は、dtqcnt=0 で生成したデータキューを指定した場合は常にエラーE_ILUSE となります。

(1) 対象データキューに受信待ちタスクが存在する場合

データキューには格納せずに受信待ち行列の先頭タスクにデータを渡し、そのタスクの待ち状態を解除します。

(2) 対象データキューに受信待ちタスクが存在しない場合

(a) データキューに空きがある場合

data をデータキューの末尾に格納します。データキューカウントは＋１されます。

(b) データキューに空きがない場合

1. snd_dtq, tsnd_dtqの場合
   呼び出しタスクはデータキューの空き領域を待つための待ち行列（送信待ち行列）につながれます。
   tsnd_dtqサービスコールの場合、tmoutには待ち時間を指定します。
   tmoutに正の値を指定した場合、待ち条件が満たされないままtmout時間が経過すると、エラーコードとしてE_TMOUTを返します。tmout=TMO_POL(=0)を指定した場合、psnd_dtqサービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。したがって、snd_dtqサービスコールと同じ処理を行います。
2. psnd_dtq, ipsnd_dtqの場合
   直ちにエラーE_TMOUTで終了します。
3. fsnd_dtq, ifsnd_dtqの場合
   送信待ちタスクが存在するかどうかに関わらず、データキュー内の最古のデータを削除してdataを格納します。

snd_dtq, tsnd_dtq によって待ち状態に遷移している間に、他のタスクから vrst_dtq が発行された場合、そのタスクの待ち状態が解除され、本サービスコールはエラーEV_RST で終了します。

5.10.4 データキューからの受信(rcv_dtq, prcv_dtq, iprcv_dtq, trcv_dtq)

■C 言語 API

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

■パラメータ

dtqid	データキューID
p_data	受信したデータを返す記憶域へのポインタ
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)p_data == NULL (2)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 dtqid≦0, VTMAX_DTQ < dtqid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(rcv_dtq,prcv_dtq,trcv_dtq のみ) (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの p_data が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 dtqid のデータキューが存在しない
E_RLWAI	待ち状態強制解除(rcv_dtq, trcv_dtq のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(rcv_dtq, trcv_dtq のみ) 待ちの間に dtqid のデータキューが削除された

■機能説明

dtqid で示されたデータキューからデータを受信し、p_data の指す領域に格納します。

データキューにデータがあれば、その先頭のデータ（最古のデータ）を受信します。データキュー内のデータを受信することで、データキューカウントは－1 されます。そして、送信待ちのタスクがあれば、送信待ち行列の先頭のタスクの送信データをデータキューに格納し、そのタスクの待ち状態を解除します。

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

データキューにデータがなく、データ送信待ちタスクも存在しない場合、rcv_dtq, trcv_dtq サービスコールでは、呼び出しタスクはデータ到着を待つ待ち行列（受信待ち行列）につながれ、prcv_dtq サービスコールでは直ちにエラーE_TMOUT で終了します。受信待ち行列は、FIFO で管理されます。

trcv_dtq サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、prcv_dtq サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。したがって、rcv_dtq サービスコールと同じ処理を行います。

5.10.5 データキューの状態参照(ref_dtq, iref_dtq)

■C 言語 API

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

■パラメータ

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

■パケットの構造

```
typedef     struct t_rdtq {
            ID      stskid;     送信待ち行列先頭のタスク ID
            ID      rtskid;     受信待ち行列先頭のタスク ID
            UINT    sdtqcnt;    データキューに入っているデータの数
} T_RDTQ;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rdtq == NULL
E_ID	不正 ID 番号 dtqid≦0, VTMAX_DTQ < dtqid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_dtq の呼び出し (2)タスクコンテキストからの iref_dtq の呼び出し
E_MACV	メモリアクセス違反(ref_dtq のみ) 呼び出しタスクからの pk_rdtq が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 dtqid のデータキューが存在しない

■機能説明

dtqid で示されたデータキューの状態を参照し、pk_rdtq が指す領域に送信待ちタスク ID(stskid)、受信待ちタスク ID(rtskid)、データキューに格納されているデータの数(sdtqcnt)を返します。

送信待ちタスク、受信待ちタスクが無い場合は、待ちタスク ID として TSK_NONE(=0)を返します。

5.11 同期・通信(メールボックス)機能

表 5.17に、メールボックス機能の仕様を示します。

表5.17 メールボックス機能の仕様

No.	項目	内容
1	メールボックス ID	1～VTMAX_MBX *1
2	メッセージ優先度	1～TMAX_MPRI *2
3	メールボックス属性	TA_TFIFO :タスクの待ち行列は FIFO 順 TA_TPRI : タスクの待ち行列はタスクの現在優先度順 TA_MFIFO : メッセージのキューイングは FIFO TA_MPRI : メッセージのキューイングは優先度順

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大メールボックス ID を意味します。
*2 cfg600px が kernel_id.h に出力するマクロで、system.message_pri に指定した値です。

表5.18 メールボックス機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_mbx		メールボックスの生成	T		E	D	U	
2	acre_mbx		メールボックスの生成(ID 番号自動割付け)	T		E	D	U	
3	del_mbx		メールボックスの削除	T		E	D	U	
4	snd_mbx	[S][B]	メールボックスへの送信	T		E	D	U	
5	isnd_mbx				N	E	D	U	
6	rcv_mbx	[S][B]	メールボックスからの受信	T		E		U	
7	prcv_mbx	[S][B]	同上(ポーリング)	T		E	D	U	
8	iprcv_mbx				N	E	D	U	
9	trcv_mbx	[S]	同上(タイムアウト有)	T		E		U	
10	ref_mbx		メールボックスの状態参照	T		E	D	U	
11	iref_mbx				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.11.1 メールボックスの生成(cre_mbx, acre_mbx)

■C 言語 API

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

■パラメータ

mbxid	メールボックス ID
pk_cmbx	メールボックス生成情報を格納したパケットへのポインタ

■パケットの構造

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

■リターン値

・cre_mbx の場合
正常終了（E_OK）、またはエラーコード

・acre_mbx の場合
生成したメールボックスの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 mbxatr の bit0, bit1 以外が 1
E_PAR	パラメータエラー (1)pk_cmbx == NULL (2)TA_MPRI 属性指定時で、maxmpri≦0 または TMAX_MPRI < maxmpri
E_ID	不正 ID 番号(cre_mbx のみ) mbxid≦0, VTMAX_MBX < mbxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_cmbx が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOID	空き ID なし(acre_mbx のみ)
E_OBJ	オブジェクト状態不正(cre_mbx のみ) mbxid のメールボックスが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_mbx は、指定された mbxid のメールボックスを pk_cmbx で指定された内容で生成します。acre_mbx は pk_cmbx で指定された内容でメールボックスを生成し、生成したメールボックス ID を返します。

(1) メールボックス ID(mbxid)

cre_mbx は、mbxid で指定した ID のメールボックスを生成します。

(2) メールボックス属性(mbxatr)

mbxatr には以下を指定できます。

mbxatr := ((TA_TFIFO || TA_TPRI) | (TA_MFIFO||TA_MPRI))

- TA_TFIFO(0x0000)
  待ち行列の順序を FIFO 順とします。
- TA_TPRI(0x0001)
  待ち行列の順序をタスクの現在優先度順とします。
- TA_MFIFO(0x0000)
  メッセージキューの順序を FIFO 順とします。
- TA_MPRI(0x0002)
  メッセージキューの順序をメッセージの現在優先度順とします。

(3) 優先度別メッセージキューヘッダの先頭アドレス(mprihd)

mprihd は単に無視されます。

5.11.2 メールボックスの削除(del_mbx)

■C 言語 API

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

■パラメータ

mbxid	メールボックス ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	mbxid ≦0, VTMAX_MBX < mbxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	mbxid のメールボックスが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

mbxid で示されたメールボックスを削除します。

mbxid で示されたメールボックスで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

5.11.3 メールボックスへの送信(snd_mbx, isnd_mbx)

■C 言語 API

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

■パラメータ

mbxid	メールボックス ID
pk_msg	送信メッセージの先頭アドレス

■パケットの構造

《メールボックスのメッセージヘッダ》

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

《メールボックスの優先度付きメッセージヘッダ》

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)pk_msg == NULL (2)mbxid のメールボックスが TA_MPRI 属性の場合で、msgpri≦0 または TMAX_MPRI< msgpri
E_ID	不正 ID 番号 mbxid≦0, VTMAX_MBX < mbxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(snd_mbx のみ) (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからのメッセージヘッダ領域に対するオペランドライトアクセス許可と オペランドリードアクセス許可がない。 メッセージヘッダ領域: ・TA_MFIFO 属性の場合 ： pk_msg から始まる T_MSG 構造体 ・TA_MPRI 属性の場合 ： pk_msg から始まるT_MSG_PRI 構造体
E_NOEXS	オブジェクト未生成 mbxid のメールボックスが存在しない

■機能説明

mbxid で示されたメールボックスに pk_msg で示されたメッセージを送信します。

すでに対象メールボックスにメッセージの受信を待つタスクが存在していれば、待ち行列先頭のタスクに送信したメッセージが渡され、そのタスクの待ち状態が解除されます。

メッセージの受信を待つタスクが存在しない場合は、メッセージをメッセージキューにつなぎます。メッセージキューは、生成時に指定した属性にしたがって管理されます。

TA_MFIFO 属性のメールボックスにメッセージを送る場合は、図 5.1に示すように先頭に T_MSG 構造体を付加した形式で、メッセージを作成してください。

TA_MPRI 属性のメールボックスにメッセージを送る場合は、図 5.2に示すように先頭に T_MSG_PRI 構造体を付加した形式で、メッセージを作成してください。

T_MSG, T_MSG_PRI の領域はカーネルが使用するため、送信後は書き換えてはなりません。メッセージ送信後、メッセージが受信される前にこの領域を書き換えた場合の動作は保証されません。

また、メッセージは受信側が読み出しアクセス可能な領域に作成する必要があります。

```
typedef struct {
    T_MSG   t_msg;      /* T_MSG 構造体                                */
    B       data[8];    /* ユーザメッセージデータ構造の例(任意の構造) */
} USER_MSG;
```

図5.1 **メッセージの形式例**(TA_MFIFO **属性の場合**)

```
typedef struct {
    T_MSG_PRI  t_msg;    /* T_MSG_PRI 構造体                           */
    B          data[8];  /* ユーザメッセージデータ構造の例(任意の構造) */
} USER_MSG;
```

図5.2 **メッセージの形式例**(TA_MPRI **属性の場合**)

5.11.4 メールボックスからの受信(rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx)

■C 言語 API

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

■パラメータ

mbxid	メールボックス ID
ppk_msg	受信メッセージ先頭アドレスを返す領域へのポインタ
tmout	タイムアウト指定(ミリ秒)

■パケットの構造

《メールボックスのメッセージヘッダ》

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

《メールボックスの優先度付きメッセージヘッダ》

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)ppk_msg == NULL (2)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 mbxid≦0, VTMAX_MBX < mbxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの prcv_mbx の呼び出し (2)タスクコンテキストからの iprcv_mbx の呼び出し
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (rcv_mbx, trcv_mbx のみ) (2)呼び出しタスクからの ppk_msg が示す領域に対するオペランドライトアクセス許可がない (rcv_mbx, prcv_mbx, trcv_mbx のみ)
E_NOEXS	オブジェクト未生成 mbxid のメールボックスが存在しない
E_RLWAI	待ち状態強制解除(rcv_mbx, trcv_mbx のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(rcv_mbx, trcv_mbx のみ) 待ちの間に mbxid のメールボックスが削除された

■機能説明

mbxid で示されたメールボックスからメッセージを受信し、受信したメッセージの先頭アドレスを ppk_msg が指す領域に返します。

メールボックスにメッセージが存在しない場合は、rcv_mbx, trcv_mbx サービスコールでは、呼び出しタスクはメッセージ到着を待つ待ち行列（受信待ち行列）につながれ、prcv_mbx, iprcv_mbx サービスコールでは直ちにエラーE_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

trcv_mbx サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、prcv_mbx サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。したがって、rcv_mbx サービスコールと同じ処理を行います。

5.11.5 メールボックスの状態参照(ref_mbx, iref_mbx)

■C 言語 API

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

■パラメータ

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

■パケットの構造

(1)T_RMBX

```
typedef    struct t_rmbx {
           ID      wtskid;      待ち行列先頭のタスク ID
           T_MSG   *pk_msg;     次に受信されるメッセージの先頭アドレス
} T_RMBX;
```

(2)T_MSG

《メールボックスのメッセージヘッダ》

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

《メールボックスの優先度付きメッセージヘッダ》

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rmbx == NULL
E_ID	不正 ID 番号 mbxid≦0, VTMAX_MBX < mbxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_mbx の呼び出し (2)タスクコンテキストからの iref_mbx の呼び出し
E_MACV	メモリアクセス違反(ref_mbx のみ) 呼び出しタスクからの pk_rmbx が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 mbxid のメールボックスが存在しない

■機能説明

mbxid で示されたメールボックスの状態を参照します。pk_rmbx が示す領域に待ちタスク ID(wtskid)、次に受信されるメッセージの先頭アドレス(pk_msg)を返します。

対象メールボックスの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE(=0)を返します。

次に受信されるメッセージが無い場合は、メッセージの先頭アドレスとして NULL(=0)を返します。

5.12 拡張同期・通信(ミューテックス)機能

表 5.19に、ミューテックス機能の仕様を示します。

表5.19 ミューテックス機能の仕様

No.	項目	内容
1	ミューテックス ID	1～VTMAX_MTX *1
2	ミューテックス属性	TA_CEILING :優先度上限プロトコル *2

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大ミューテックス ID を意味します。

*2 本カーネルにおける TA_CEILING 属性(優先度上限プロトコル)では、簡略化した優先度制御規則を採用しています。簡略化した優先度制御規則では、タスクの優先度を高くする制御はすべて行われますが、タスクの優先度を低くする制御は、タスクがロックしていたミューテックスが無くなったとき(複数のミューテックスをロックしていた場合は、それら全てを解放したとき)にのみ行われます。

表5.20 ミューテックス機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_mtx		ミューテックスの生成	T		E	D	U	
2	acre_mtx		ミューテックスの生成(ID 番号自動割付け)	T		E	D	U	
3	del_mtx		ミューテックスの削除	T		E	D	U	
4	loc_mtx		ミューテックスのロック	T		E		U	
5	ploc_mtx		同上(ポーリング)	T		E	D	U	
6	tloc_mtx		同上(タイムアウト有)	T		E		U	
7	unl_mtx		ミューテックスのロック解除	T		E	D	U	
8	ref_mtx		ミューテックスの状態参照	T		E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.12.1 ミューテックスの生成(cre_mtx, acre_mtx)

■C 言語 API

```
ER      cre_mtx(ID mtxid, T_CMTX *pk_cmtx);
ER_ID   acre_mtx(T_CMTX *pk_cmtx);
```

■パラメータ

mtxid	ミューテックス ID
pk_cmtx	ミューテックス生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct t_cmtx {
           ATR     mtxatr;      ミューテックス属性
           PRI     ceilpri;     上限優先度
} T_CMTX;
```

■リターン値

・cre_mtx の場合
正常終了（E_OK）、またはエラーコード
・acre_mtx の場合
生成したミューテックスの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 mtxatr!=TA_CEILING
E_PAR	パラメータエラー (1)pk_cmtx == NULL (2)ceilpri≦0, TMAX_TPRI < ceilpri
E_ID	不正 ID 番号(cre_mtx のみ) mtxid≦0, VTMAX_MTX < mtxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_cmtx が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOID	空き ID なし(acre_mtx のみ)
E_OBJ	オブジェクト状態不正(cre_mtx のみ) mtxid のミューテックスが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_mtx は、指定された mtxid のミューテックスを pk_cmtx で指定された内容で生成します。acre_mtx は pk_cmtx で指定された内容でミューテックスを生成し、生成したミューテックス ID を返します。

(1) ミューテックス ID(mtxid)

cre_mtx は、mtxid で指定した ID のミューテックスを生成します。

(2) ミューテックス属性(mtxatr)

mtxatr には TA_CEILING のみを指定できます。

- TA_CEILING(0x0003)
  優先度上限プロトコル

(3) 上限優先度(ceilpri)

ミューテックスをロックしたタスクの現在優先度は、ceilpri に上昇します。

5.12.2 ミューテックスの削除(del_mtx)

■C 言語 API

```
ER        del_mtx(ID mtxid);
```

■パラメータ

mtxid	ミューテックス ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	mtxid ≦0, VTMAX_MTX < mtxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	mtxid のミューテックスが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

mtxid で示されたミューテックスを削除します。

mtxid で示されたミューテックスで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

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

5.12.3 ミューテックスのロック(loc_mtx, ploc_mtx, tloc_mtx)

■C 言語 API

```
ER      loc_mtx(ID mtxid);
ER      ploc_mtx(ID mtxid);
ER      tloc_mtx(ID mtxid, TMO tmout);
```

■パラメータ

mtxid	ミューテックス ID
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 mtxid≦0, VTMAX_MTX < mtxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_ILUSE	サービスコール不正使用 (1)呼び出しタスクは既に mtxid のミューテックスをロック済み (2)上限優先度違反(呼び出しタスクのベース優先度 > 対象ミューテックスの上限優先度)
E_NOEXS	オブジェクト未生成 mtxid のミューテックスが存在しない
E_RLWAI	待ち状態強制解除(loc_mtx, tloc_mtx のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(loc_mtx, tloc_mtx のみ) 待ちの間に mtxid のミューテックスが削除された

■機能説明

mtxid で指定されるミューテックスをロックします。

対象ミューテックスがロックされていない場合は、呼び出しタスクがミューテックスをロックします。その際、呼び出しタスクの現在優先度はミューテックスの上限優先度まで引き上げられます。

対象ミューテックスがロックされている場合には、呼び出しタスクを待ち行列につなぎ、ミューテックスのロック待ち状態に移行させます。待ち行列は、優先度順に管理されます。

tloc_mtx サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、ploc_mtx サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。この場合、loc_mtx サービスコールと同じ動作となります。

5.12.4 ミューテックスのロック解除(unl_mtx)

■C 言語 API

```
ER          unl_mtx(ID mtxid);
```

■パラメータ

mtxid	ミューテックス ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	mtxid≦0, VTMAX_MTX < mtxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_ILUSE	サービスコール不正使用
	呼び出しタスクは mtxid のミューテックスをロックしていない
E_NOEXS	オブジェクト未生成
	mtxid のミューテックスが存在しない

■機能説明

mtxid で示されたミューテックスのロックを解除します。

対象ミューテックスに対してロックを待っているタスクがあれば、ミューテックスの待ち行列先頭タスクを待ち解除し、待ち解除されたタスクがミューテックスをロックします。その際、ロックするタスクの現在優先度はミューテックスの上限優先度まで引き上げられます。

ミューテックスに対して待っているタスクがなければ、そのミューテックスをロックされていない状態にします。

本カーネルの TA_CEILING 属性は、簡略化した優先度上限プロトコルを採用しています。つまり、本サービスコールによって呼び出しタスクがロックしているミューテックスが全て無くなったときのみ、現在優先度をベース優先度に戻します。呼び出しタスクがまだ他のミューテックスをロックしている場合、本サービスコールでは現在優先度は変化しません。

5.12.5 ミューテックスの状態参照(ref_mtx)

■C 言語 API

```
ER          ref_mtx(ID mtxid, T_RMTX *pk_rmtx);
```

■パラメータ

mtxid	ミューテックス ID
pk_rmtx	ミューテックス状態を返すパケットへのポインタ

■パケットの構造

```
typedef     struct t_rmtx {
            ID      htskid;     ミューテックスをロックしているタスク ID
            ID      wtskid;     待ち行列先頭のタスク ID
} T_RMTX;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rmtx == NULL
E_ID	不正 ID 番号 mtxid≦0, VTMAX_MTX < mtxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 呼び出しタスクからの pk_rmtx が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 mtxid のミューテックスが存在しない

■機能説明

mtxid で示されたミューテックスの状態を参照します。

pk_rmtx が指す領域に、ミューテックスをロックしているタスク ID(htskid)、ミューテックスの待ち行列の先頭タスク ID(wtskid)を返します。

対象ミューテックスをロックしているタスクが存在しない場合は、htskid には TSK_NONE(=0)が返ります。

対象ミューテックスに待ちタスクが無い場合は、wtskid には TSK_NONE(=0)が返ります。

5.13 同期・通信(メッセージバッファ)機能

表 5.21に、メッセージバッファ機能の仕様を示します。

表5.21 メッセージバッファ機能の仕様

No.	項目	内容
1	メッセージバッファ ID	1～VTMAX_MBF *1
2	バッファサイズ	0 または 8～65532 (バイト)
3	送信可能なメッセージサイズ	1～65528 (バイト)
4	メッセージバッファ属性	TA_TFIFO :送信タスクの待ち行列は FIFO 順

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大メッセージバッファ ID を意味します。

表5.22 メッセージバッファ機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_mbf		メッセージバッファの生成	T		E	D	U	
2	acre_mbf		メッセージバッファの生成(ID 番号自動割付け)	T		E	D	U	
3	del_mbf		メッセージバッファの削除	T		E	D	U	
4	snd_mbf		メッセージバッファへの送信	T		E		U	
5	psnd_mbf		同上(ポーリング)	T		E	D	U	
6	ipsnd_mbf				N	E	D	U	
7	tsnd_mbf		同上(タイムアウト有)	T		E		U	
8	rcv_mbf		メッセージバッファからの受信	T		E		U	
9	prcv_mbf		同上(ポーリング)	T		E	D	U	
10	trcv_mbf		同上(タイムアウト有)	T		E		U	
11	ref_mbf		メッセージバッファの状態参照	T		E	D	U	
12	iref_mbf				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.13.1 メッセージバッファの生成(cre_mbf, acre_mbf)

■C 言語 API

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

■パラメータ

mbfid	メッセージバッファ ID
pk_cmbf	メッセージバッファ生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct t_cmbf {
          ATR     mbfatr;     メッセージバッファ属性
          UINT    maxmsz;     メッセージの最大サイズ(バイト数)
          SIZE    mbfsz;      メッセージバッファ領域のサイズ(バイト数)
          VP      mbf;        メッセージバッファ領域の先頭アドレス
} T_CMBF;
```

■リターン値

・cre_mbf の場合
正常終了（E_OK）、またはエラーコード

・acre_mbf の場合
生成したメッセージバッファの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 mbfatr != TA_TFIFO
E_PAR	パラメータエラー (1)pk_cmbf == NULL (2)maxmsz == 0, 65528 < maxmsz (3)0 < mbfsz < 8, 65532 < mbfsz (4)mbfsz!=0 で、mbf+mbfsz > 0x100000000 (5)mbfsz!=0 で、mbfsz < maxmsz + VTSZ_MBFTBL
E_ID	不正 ID 番号(cre_mbf のみ) mbfid≦0, VTMAX_MBF < mbfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_cmbf が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOMEM	メモリ不足 mbfsz!=0 で、mbf == NULL
E_NOID	空き ID なし(acre_mbf のみ)
E_OBJ	オブジェクト状態不正(cre_mbf のみ) mbfid のメッセージバッファが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_mbf は、指定された mbfid のメッセージバッファを pk_cmbf で指定された内容で生成します。acre_mbf は pk_cmbf で指定された内容でメッセージバッファを生成し、生成したメッセージバッファ ID を返します。

(1) メッセージバッファ ID(mbfid)

cre_mbf は、mbfid で指定した ID のメッセージバッファを生成します。

(2) メッセージバッファ属性(mbfatr)

mbfatr には TA_TFIFO のみを指定できます。

- TA_TFIFO(0x0000)
  送信待ち行列の順序を FIFO 順とします。

なお、受信待ち行列は常に FIFO 順につながれます。

(3) メッセージバッファ領域のサイズ(mbfsz)、メッセージバッファ領域の先頭アドレス(mbf)

アプリケーション側で、mbfsz バイトのメッセージバッファ領域を確保し、その先頭アドレスを mbf に指定します。

なお、μITRON4.0 仕様には、mbf に NULL を指定することでカーネルがメッセージバッファ領域を割り当てる機能がありますが、RI600/PX はこの機能をサポートしていません。

また、mbfsz に 0 を指定することもできます。この場合、メッセージバッファにメッセージを蓄えておくことはできないため、送信側と受信側の先に実行した方が待ち状態になり、他方が行われた時点で待ちが解除される、つまり送信側と受信側が完全に同期した動作となります。なお、mbfsz が 0 の場合は、mbf は無視されます。

カーネルは、メッセージバッファ領域に関するアクセス権については何も関知しません。通常は、メッセージバッファ領域はメモリオブジェクト・ユーザスタック以外の領域を指定してください。メモリオブジェクト内にメッセージバッファ領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤ってメッセージバッファ領域を書き換えてしまう危険があります。

5.13.2 メッセージバッファの削除(del_mbf)

■C 言語 API

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

■パラメータ

mbfid	メッセージバッファ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	mbfid ≦0, VTMAX_MBF < mbfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	mbfid のメッセージバッファが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

mbfid で示されたメッセージバッファを削除します。

mbfid で示されたメッセージバッファで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

5.13.3 メッセージバッファへの送信(snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf)

■C 言語 API

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

■パラメータ

mbfid	メッセージバッファ ID
msg	送信メッセージの先頭アドレス
msgsz	送信メッセージのサイズ(バイト数)
《tsnd_mbf》	
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)msg == NULL (2)msgsz == 0 (3)(生成時に指定したメッセージの最大サイズ) < msgsz (4)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 mbfid≦0, VTMAX_MBF < mbfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(snd_mbf, psnd_mbf, tsnd_mbf のみ) (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの msg が示す領域(先頭：msg，サイズ：msgsz)に対するオペランドリード アクセス許可がない
E_NOEXS	オブジェクト未生成 mbfid のメッセージバッファが存在しない
E_RLWAI	待ち状態強制解除(snd_mbf, tsnd_mbf のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(snd_mbf, tsnd_mbf のみ) 待ちの間に mbfid のメッセージバッファが削除された
EV_RST	オブジェクトリセット(vrst_mbf)による待ち解除(snd_mbf, tsnd_mbf のみ)

■機能説明

mbfid で示されたメッセージバッファに対して、msg で示されたメッセージを送信します。送信するサイズは msgsz で示されたバイト数です。

対象メッセージバッファに受信待ちタスクが存在する場合には、メッセージバッファには格納せずに受信待ち行列の先頭タスクにメッセージを渡し、そのタスクの待ち状態を解除します。

対象メッセージバッファに既に送信待ちタスクが存在する場合、snd_mbf, tsnd_mbf サービスコールではメッセージバッファの空き領域を待つための待ち行列（送信待ち行列）につながれ、psnd_mbf, ipsnd_mbf サービスコールでは直ちにエラーE_TMOUT で終了します。送信待ち行列は、FIFO 順に並びます。

受信待ちタスクも送信待ちタスクも存在しない場合は、メッセージをメッセージバッファに格納します。この結果、メッセージバッファの空きサイズは、以下の式で算出されるサイズだけ減少します。

- 減少サイズ ＝ (msgsz を 4 の倍数に切り上げた値)＋VTSZ_MBFTBL

VTSZ_MBFTBL はカーネルがバッファ領域内に生成する管理テーブルのサイズ(4 バイト)です。

このサイズだけの空きがメッセージバッファに存在しない場合（バッファサイズが 0 の場合も含む）は、snd_mbf, tsnd_mbf では呼び出しタスクは送信待ち行列につながれ、psnd_mbf, ipsnd_mbf では直ちにエラーE_TMOUT で終了します。

tsnd_mbf サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、psnd_mbf サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。したがって、snd_mbf サービスコールと同じ処理を行います。

メッセージバッファ送信待ち行列の先頭タスクを rel_wai,irel_wai サービスコールによって待ち解除した場合、または ter_tsk サービスコールによって強制終了した場合、他のタスクのメッセージバッファ送信待ちが解除されることがあります。

snd_mbf, tsnd_mbf によって待ち状態に遷移している間に、他のタスクから vrst_mbf が発行された場合、そのタスクの待ち状態が解除され、本サービスコールはエラーEV_RST で終了します。

5.13.4 メッセージバッファからの受信(rcv_mbf, prcv_mbf, trcv_mbf)

■C 言語 API

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

■パラメータ

mbfid	メッセージバッファ ID
msg	受信メッセージを格納する記憶域へのポインタ
tmout	タイムアウト指定(ミリ秒)

■リターン値

受信メッセージのサイズ（バイト数、正の値）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)msg == NULL (2)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 mbfid≦0, VTMAX_MBF < mbfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの msg が示す領域(先頭：msg，サイズ：生成時に指定した最大メッセージサイズ) に対するオペランドリードアクセス許可がない
E_NOEXS	オブジェクト未生成 mbfid のメッセージバッファが存在しない
E_RLWAI	待ち状態強制解除(rcv_mbf, trcv_mbf のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(rcv_mbf, trcv_mbf のみ) 待ちの間に mbfid のメッセージバッファが削除された

■機能説明

mbfid で示されたメッセージバッファからメッセージを受信し、受信したメッセージを msg の指す領域に格納します。また、受信したメッセージサイズをリターンパラメータとして返します。

メッセージバッファにメッセージがあれば、メッセージ行列先頭のメッセージ（最古のメッセージ）を受信します。メッセージバッファ内のメッセージを受信することで、メッセージバッファの空きサイズは以下の式で算出されるサイズだけ増加します。

- 増加サイズ ＝ (msgsz を 4 の倍数に切り上げた値)＋VTSZ_MBFTBL

VTSZ_MBFTBL はカーネルがバッファ領域内に生成する管理テーブルのサイズ(4 バイト)です。

この結果、空きサイズがメッセージ送信待ち行列先頭のタスクが送信しようとしていたメッセージサイズよりも大きくなると、そのメッセージがメッセージバッファに格納され、そのタスクの待ち状態が解除されます。送信待ち行列の以降のタスクに対してもメッセージの格納が可能であれば、待ち行列の順に同様の処理を行います。

メッセージバッファのサイズが 0 の場合で送信待ちタスクが存在する場合、送信待ち行列先頭タスクのメッセージを受信します。この結果、そのメッセージ送信待ちタスクの待ち状態は解除されます。

メッセージバッファにメッセージがなく、メッセージ送信待ちタスクも存在しない場合、rcv_mbf, trcv_mbf サービスコールでは、呼び出しタスクはメッセージ到着を待つ待ち行列（受信待ち行列）につながれ、prcv_mbf サービスコールでは直ちにエラーE_TMOUT で終了します。受信待ち行列は、FIFO で管理されます。

trcv_mbf サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、prcv_mbf サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。したがって、rcv_mbf サービスコールと同じ処理を行います。

5.13.5 メッセージバッファの状態参照(ref_mbf, iref_mbf)

■C 言語 API

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

■パラメータ

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

■パケットの構造

```
typedef    struct t_rmbf {
           ID      stskid;      送信待ち行列の先頭タスク ID
           ID      rtskid;      受信待ち行列の先頭タスク ID
           UINT    smsgcnt;     メッセージバッファに入っているメッセージの数
           SIZE    fmbfsz;      空きバッファのサイズ（バイト数）
} T_RMBF;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rmbf == NULL
E_ID	不正 ID 番号 mbfid≦0, VTMAX_MBF < mbfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_mbf の呼び出し (2)タスクコンテキストからの iref_mbf 呼び出し
E_MACV	メモリアクセス違反(ref_mbf のみ) 呼び出しタスクからの pk_rmbf が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 mbfid のメッセージバッファが存在しない

■機能説明

mbfid で示されたメッセージバッファの状態を参照し、pk_rmbf が指す領域に送信待ちタスク ID (stskid)、受信待ちタスク ID(rtskid)、メッセージバッファに入っているメッセージの数(smsgcnt)、および空きバッファサイズ(fmbfsz)を返します。

受信待ちタスク、送信待ちタスクが無い場合は、待ちタスク ID として TSK_NONE(=0)を返します。

5.14 メモリプール管理(固定長メモリプール)機能

表 5.23に、固定長メモリプール機能の仕様を示します。

表5.23 固定長メモリプール機能の仕様

No.	項目	内容
1	固定長メモリプール ID	1～VTMAX_MPF *1
2	メモリブロック数の最大値	65535
3	メモリブロックサイズの最大値	65535(バイト)
4	プールサイズ(ブロックサイズ×ブロック数)の上限	VTMAX_AREASIZE(バイト) *2
5	固定長メモリプール属性	TA_TFIFO :タスク待ち行列は FIFO 順 TA_TPRI : タスクの待ち行列はタスクの現在優先度順

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大固定長メモリプール ID を意味します。
*2 kernel.h で定義されているマクロです。本カーネルでの定義値は、256MB です。

表5.24 固定長メモリプール機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_mpf		固定長メモリプールの生成	T		E	D	U	
2	acre_mpf		固定長メモリプールの生成(ID番号自動割付け)	T		E	D	U	
3	del_mpf		固定長メモリプールの削除	T		E	D	U	
4	get_mpf	[S][B]	固定長メモリブロックの獲得	T		E		U	
5	pget_mpf	[S][B]	同上(ポーリング)	T		E	D	U	
6	ipget_mpf				N	E	D	U	
7	tget_mpf	[S]	同上(タイムアウト有)	T		E		U	
8	rel_mpf	[S][B]	固定長メモリブロックの返却	T		E	D	U	
9	irel_mpf				N	E	D	U	
10	ref_mpf		固定長メモリプールの状態参照	T		E	D	U	
11	iref_mpf				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.14.1 固定長メモリプールの生成(cre_mpf, acre_mpf)

■C 言語 API

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

■パラメータ

mpfid	固定長メモリプール ID
pk_cmpf	固定長メモリプール生成情報を格納したパケットへのポインタ

■パケットの構造

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

■リターン値

・cre_mpf の場合
正常終了（E_OK）、またはエラーコード
・acre_mpf の場合
生成した固定長メモリプールの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 mpfatr の bit0 以外が 1
E_PAR	パラメータエラー (1)pk_cmpf == NULL (2)blksz == 0, 65535 < blksz (3)blkcnt == 0, 65535 < blkcnt (4)TSZ_MPF(blkcnt, blksz) > VTMAX_AREASIZE (5)mpf + TSZ_MPF(blkcnt, blksz) > 0x100000000
E_ID	不正 ID 番号(cre_mpf のみ) mpfid≦0, VTMAX_MPF < mpfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_cmpf が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOMEM	メモリ不足 (1)mpf == NULL (2)mpfmb == NULL
E_NOID	空き ID なし(acre_mpf のみ)
E_OBJ	オブジェクト状態不正(cre_mpf のみ) mpfid の固定長メモリプールが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_mpf は、指定された mpfid の固定長メモリプールを pk_cmpf で指定された内容で生成します。acre_mpf は pk_cmpf で指定された内容で固定長メモリプールを生成し、生成した固定長メモリプール ID を返します。

(1) 固定長メモリプール ID(mpfid)

cre_mpf は、mpfid で指定した ID の固定長メモリプールを生成します。

(2) 固定長メモリプール属性(mpfatr)

mpfatr には以下を指定できます。

mpfatr:= (TA_TFIFO || TA_TPRI)

- TA_TFIFO(0x0000)
  待ち行列の順序を FIFO 順とします。
- TA_TPRI(0x0001)
  待ち行列の順序をタスクの現在優先度順とします。

(3) 獲得可能なメモリブロック数(blkcnt), メモリブロックのサイズ(blksz), 固定長メモリプール領域の先頭アドレス(mpf)

アプリケーション側で、TSZ_MPF(blkcnt, blksz)バイトの固定長メモリプール領域を確保し、その先頭アドレスを mpf に指定します。

なお、μITRON4.0 仕様には、mpf に NULL を指定することでカーネルが固定長メモリプール領域を割り当てる機能がありますが、RI600/PX はこの機能をサポートしていません。

カーネルは、固定長メモリプール領域に関するアクセス権については何も関知しません。タスクが固定長メモリプールから獲得したメモリブロックにアクセスするには、固定長メモリプール領域は適切にアクセス許可が設定されたメモリオブジェクト内に確保する必要があります。

また、カーネルは固定長メモリプール領域内に管理テーブルを生成します。この管理テーブルがアプリケーションによって書き換えられた場合、システムの正常な動作は保証されません。

(4) 固定長メモリプールの管理領域の先頭アドレス(mpfmb)

アプリケーション側で、TSZ_MPFMB(blkcnt, blksz)バイトの固定長メモリプール管理領域を確保し、その先頭アドレスを mpfmb に指定します。

カーネルは、管理領域に関するアクセス権については何も関知しません。通常は、管理領域はメモリオブジェクト・ユーザスタック以外の領域を指定してください。メモリオブジェクト内に管理領域を生成した場合は、そのメモリオブジェクトへの書き込みアクセスが許可されたタスクが、誤って管理領域を書き換えてしまう危険があります。

5.14.2 固定長メモリプールの削除(del_mpf)

■C 言語 API

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

■パラメータ

mpfid	固定長メモリプール ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	mpfid ≦0, VTMAX_MPF < mpfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	mpfid の固定長メモリプールが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

mpfid で示された固定長メモリプールを削除します。

mpfid で示された固定長メモリプールで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

5.14.3 固定長メモリブロックの獲得(get_mpf, pget_mpf, ipget_mpf, tget_mpf)

■C 言語 API

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

■パラメータ

mpfid	固定長メモリプール ID
p_blk	メモリブロック先頭アドレスを返す記憶域へのポインタ
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)p_blk == NULL (2)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 mpfid≦0, VTMAX_MPF < mpfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの pget_mpf の呼び出し (2)タスクコンテキストからの ipget_mpf 呼び出し
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (get_mpf, tget_mpf のみ) (2)呼び出しタスクからの p_blk が示す領域に対するオペランドライトアクセス許可がない (get_mpf, pget_mpf, tget_mpf のみ)
E_NOEXS	オブジェクト未生成 mpfid の固定長メモリプールが存在しない
E_RLWAI	待ち状態強制解除(get_mpf, tget_mpf のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(get_mpf, tget_mpf のみ) 待ちの間に mpfid の固定長メモリプールが削除された
EV_RST	オブジェクトリセット(vrst_mpf)による待ち解除(get_mpf, tget_mpf のみ)

■機能説明

mpfid で示される固定長メモリプールからひとつのメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを p_blk の指す領域に返します。

既にメモリブロック獲得待ちタスクが存在する場合、または待ちタスクは存在しないが対象となる固定長メモリプールに空きブロックが存在しない場合は、get_mpf, tget_mpf サービスコールでは呼び出しタスクはそのメモリプールのメモリ獲得の待ち行列につながれ、pget_mpf, ipget_mpf サービスコールでは直ちにエラー E_TMOUT で終了します。待ち行列は、生成時に指定した属性にしたがって管理されます。

tget_mpf サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、pget_mpf サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合は、タイムアウト監視を行いません。したがって、get_mpf サービスコールと同じ処理を行います。

get_mpf, tget_mpf によって待ち状態に遷移している間に、他のタスクから vrst_mpf が発行された場合、そのタスクの待ち状態が解除され、本サービスコールはエラーEV_RST で終了します。

■補足

獲得するメモリブロックアドレスの境界調整数は 1 です。

これより大きい境界アドレスでメモリブロックを獲得したい場合は、メモリプール生成時に以下を守ってください。

(1) メモリブロックサイズを、目的の境界調整数の倍数とする。
(2) 固定長メモリプール領域の先頭アドレスを、目的の境界調整数のアドレスとする。

5.14.4 固定長メモリブロックの返却(rel_mpf, irel_mpf)

■C 言語 API

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

■パラメータ

mpfid	固定長メモリプール ID
blk	メモリブロックの先頭アドレス

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)blk == NULL (2)blk が獲得したメモリブロック以外のアドレス、または返却済みのメモリブロックのアドレス
E_ID	不正 ID 番号 mpfid≦0, VTMAX_MPF < mpfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(rel_mpf のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成 mpfid の固定長メモリプールが存在しない

■機能説明

mpfid で示された固定長メモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_mpf、pget_mpf, ipget_mpf または tget_mpf サービスコールで獲得したメモリブロックの先頭アドレスを指定してください。

対象固定長メモリプールでメモリブロックの獲得を待っているタスクがある場合、本サービスコールで返却したブロックを待ち行列先頭のタスクに割り付け、待ち状態を解除します。

5.14.5 固定長メモリプールの状態参照(ref_mpf, iref_mpf)

■C 言語 API

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

■パラメータ

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

■パケットの構造

```
typedef     struct t_rmpf {
            ID      wtskid;     待ち行列先頭のタスク ID
            UINT    fblkcnt;    空き領域のブロック数
} T_RMPF;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rmpf == NULL
E_ID	不正 ID 番号 mpfid≦0, VTMAX_MPF < mpfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_mpf の呼び出し (2)タスクコンテキストからの iref_mpf 呼び出し
E_MACV	メモリアクセス違反(ref_mpf のみ) 呼び出しタスクからの pk_rmpf が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 mpfid の固定長メモリプールが存在しない

■機能説明

mpfid で示された固定長メモリプールの状態を参照します。
pk_rmpf の指す領域に待ちタスク ID(wtskid)、空き領域のブロック数(fblkcnt)を返します。
対象メモリプールの待ちタスクが無い場合は、待ちタスク ID として TSK_NONE(=0)を返します。

5.15 メモリプール管理(可変長メモリプール)機能

表 5.25に、可変長メモリプール機能の仕様を示します。

表5.25 可変長メモリプール機能の仕様

No.	項目	内容
1	可変長メモリプール ID	1～VTMAX_MPL *1
2	プールサイズ	24～VTMAX_AREASIZE(バイト) *2
3	ブロックサイズ	1～0xBFFFFF4(バイト)
4	可変長メモリプール属性	TA_TFIFO :タスク待ち行列は FIFO 順

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大可変長メモリプール ID を意味します。
*2 kernel.h で定義されているマクロです。本カーネルでの定義値は、256MB です。

表5.26 可変長メモリプール機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_mpl		可変長メモリプールの生成	T		E	D	U	
2	acre_mpl		可変長メモリプールの生成(ID 番号自動割付け)	T		E	D	U	
3	del_mpl		可変長メモリプールの削除	T		E	D	U	
4	get_mpl		固定長メモリブロックの獲得	T		E		U	
5	pget_mpl		同上(ポーリング)	T		E	D	U	
6	ipget_mpl				N	E	D	U	
7	tget_mpl		同上(タイムアウト有)	T		E		U	
8	rel_mpl		可変長メモリブロックの返却	T		E	D	U	
9	ref_mpl		可変長メモリプールの状態参照	T		E	D	U	
10	iref_mpl				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。
*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.15.1 可変長メモリプールの生成(cre_mpl, acre_mpl)

■C 言語 API

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

■パラメータ

mplid	可変長メモリプール ID
pk_cmpl	可変長メモリプール生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct t_cmpl {
          ATR     mplatr;     可変長メモリプール属性
          SIZE    mplsz;      可変長メモリプール領域のサイズ
          VP      mpl;        可変長メモリプール領域の先頭アドレス
          UINT    maxblksz;   最大メモリブロックサイズ
} T_CMPL;
```

■リターン値

・cre_mpl の場合
正常終了（E_OK）、またはエラーコード

・acre_mpl の場合
生成した可変長メモリプールの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 mplatr != TA_TFIFO
E_PAR	パラメータエラー (1)pk_cmpl == NULL (2)mplsz < 24, VTMAX_AREASIZE < mplsz (3)maxblksz == 0, 0x0BFFFFF4 < maxblksz (4)maxblksz に対して mplsz が小さすぎる (5)mpl + mplsz > 0x100000000 (6)mpl が 4 バイト境界でない
E_ID	不正 ID 番号(cre_mpl のみ) mplid≦0, VTMAX_MPL < mplid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_cmpl が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOMEM	メモリ不足 mpl == NULL
E_NOID	空き ID なし(acre_mpl のみ)
E_OBJ	オブジェクト状態不正(cre_mpl のみ) mplid の可変長メモリプールが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_mpl は、指定された mplid の可変長メモリプールを pk_cmpl で指定された内容で生成します。acre_mpl は pk_cmpl で指定された内容で可変長メモリプールを生成し、生成した可変長メモリプール ID を返します。

(1) 可変長メモリプール ID(mplid)

cre_mpl は、mplid で指定した ID の可変長メモリプールを生成します。

(2) 可変長メモリプール属性(mplatr)

mplatr には TA_TFIFO のみを指定できます。

- TA_TFIFO(0x0000)
  待ち行列の順序を FIFO 順とします。

(3) 可変長メモリプール領域のサイズ(mplsz), 可変長メモリプール領域の先頭アドレス(mpl)

アプリケーション側で、mplsz バイトの可変長メモリプール領域を確保し、その先頭アドレスを mpl に指定します。

なお、μITRON4.0 仕様には、mpl に NULL を指定することでカーネルが可変長メモリプール領域を割り当てる機能がありますが、RI600/PX はこの機能をサポートしていません。

カーネルは、可変長メモリプール領域に関するアクセス権については何も関知しません。タスクが可変長メモリプールから獲得したメモリブロックにアクセスするには、可変長メモリプール領域は適切にアクセス許可が設定されたメモリオブジェクト内にある必要があります。

また、カーネルは可変長メモリプール領域内に管理テーブルを生成します。この管理テーブルが誤って書き換えられた場合、システムの正常な動作は保証されません。

(4) 最大メモリブロックサイズ(maxblksz)

メモリブロックの最大サイズを指定します。実際に獲得可能メモリブロックの最大サイズは、maxblksz より大きくなることがあります。

現在のカーネル実装では、実際に獲得されるメモリブロックのサイズは、最大で 12 種類のバリエーションから選択されます。このバリエーションは maxblksz を元に決定されます。表 5.27に、メモリブロックサイズのバリエーションを示します。ただし、この振る舞いは将来変更される可能性があります。

表5.27 メモリブロックサイズのバリエーション

項番	メモリブロックサイズ (16 進数)	例 1： maxblksz == 0x100	例 2： maxblksz == 0x20000
1	12 (0xC)	○	×
2	36 (0x24)	○	×
3	84 (0x54)	○	○
4	180 (0xB4)	○	○
5	372 (0x174)	×	○
6	756 (0x2F4)	×	○
7	1524 (0x5F4)	×	○
8	3060 (0xBF4)	×	○
9	6132 (0x17F4)	×	○
10	12276 (0x2FF4)	×	○
11	24564 (0x5FF4)	×	○
12	49140 (0xBFF4)	×	○
13	98292 (0x17FF4)	×	○
14	196596 (0x2FFF4)	×	○
15	393204 (0x5FFF4)	×	×
16	786420 (0xBFFF4)	×	×
17	1572852 (0x17FFF4)	×	×
18	3145716 (0x2FFFF4)	×	×
19	6291444 (0x5FFFF4)	×	×
20	12582900 (0xBFFFF4)	×	×
21	25165812 (0x17FFFF4)	×	×
22	50331636 (0x2FFFFF4)	×	×
23	100663284 (0x5FFFFF4)	×	×
24	201326580 (0xBFFFFF4)	×	×

5.15.2 可変長メモリプールの削除(del_mpl)

■C 言語 API

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

■パラメータ

mplid	可変長メモリプール ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	mplid ≦0, VTMAX_MPL < mplid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	mplid の可変長メモリプールが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

mplid で示された可変長メモリプールを削除します。

mplid で示された可変長メモリプールで待っているタスクがあった場合でもエラーにはなりませんが、待ち状態だったタスクは待ち状態が解除され、エラーコードとして E_DLT が返されます。

5.15.3 可変長メモリブロックの獲得(get_mpl, tget_mpl, pget_mpl, ipget_mpl)

■C 言語 API

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

■パラメータ

mplid	可変長メモリプール ID
blksz	メモリブロックサイズ（バイト数）
p_blk	メモリブロックの先頭アドレスを返す記憶域へのポインタ
tmout	タイムアウト指定(ミリ秒)

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)p_blk == NULL (2)blksz == 0 (3)blksz が獲得可能な最大サイズを超えている (4)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 mplid≦0, VTMAX_MPL < mplid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの pget_mpl の呼び出し (2)タスクコンテキストからの ipget_mpl 呼び出し
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (get_mpl, tget_mpl のみ) (2)呼び出しタスクからの p_blk が示す領域に対するオペランドライトアクセス許可がない (get_mpl, pget_mpl, tget_mpl のみ)
E_NOEXS	オブジェクト未生成 mplid の可変長メモリプールが存在しない
E_RLWAI	待ち状態強制解除(get_mpl, tget_mpl のみ) 待ちの間に rel_wai, irel_wai サービスコールが呼び出された
E_TMOUT	タイムアウト、またはポーリング失敗
E_DLT	待ちオブジェクトの削除(get_mpl, tget_mpl のみ) 待ちの間に mplid の可変長メモリプールが削除された
EV_RST	オブジェクトリセット(vrst_mpl)による待ち解除(get_mpl, tget_mpl のみ)

■機能説明

mplid で示される可変長メモリプールから、blksz で示される以上のサイズ（バイト数）のメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを p_blk の指す領域に返します。

既にメモリブロック獲得待ちタスクが存在する場合、get_mpl, tget_mpl サービスコールではメモリ獲得待ち行列につながれ、メモリ獲得待ち状態に移行します。pget_mpl, ipget_mpl サービスコールでは、直ちにエラーE_TMOUT で終了します。メモリ獲得待ち行列は、FIFO 順で管理されます。

tget_mpl サービスコールの場合、tmout には待ち時間を指定します。

tmout に正の値を指定した場合、待ち条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。tmout=TMO_POL(=0)を指定した場合、pget_mpl サービスコールと同じ処理を行います。tmout=TMO_FEVR(=-1)を指定した場合、タイムアウト監視を行いません。したがって、get_mpl サービスコールと同じ処理を行います。

可変長メモリプールメモリ獲得待ち行列の先頭タスクの待ちを rel_wai,irel_wai サービスコールによって待ち解除させた場合や,ter_tsk サービスコールによって強制終了した場合、他のタスクの可変長メモリプールメモリ獲得待ちが解除されることがあります。

get_mpl, tget_mpl によって待ち状態に遷移している間に、他のタスクから vrst_mpl が発行された場合、そのタスクの待ち状態が解除され、本サービスコールはエラーEV_RST で終了します。

■補足

獲得するメモリブロックアドレスの境界調整数は、可変長メモリプールの生成方法によって異なります。

- cre_mpl, acre_mpl によって生成した場合
  獲得するメモリブロックの境界調整数は 4 です。
- cfg ファイルで生成した場合
  獲得するメモリブロックの境界調整数は 1 です。ただし、可変長メモリプール毎に別セクションを指定し、リンク時にそのセクションを 4 バイト境界に配置した場合は、獲得するメモリブロックの境界調整数は 4 となります。

5.15.4 可変長メモリブロックの返却(rel_mpl)

■C 言語 API

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

■パラメータ

mplid	可変長メモリプール ID
blk	メモリブロックの先頭アドレス

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)blk == NULL (2)blk が獲得したメモリブロック以外のアドレス、または返却済みのメモリブロックのアドレス
E_ID	不正 ID 番号 mplid≦0, VTMAX_MPL < mplid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成 mplid の可変長メモリプールが存在しない

■機能説明

mplid で示された可変長メモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_mpl、pget_mpl, ipget_mpl または tget_mpl サービスコールで獲得したメモリブロックの先頭アドレスを指定してください。

メモリブロックの返却により、可変長メモリプールの空きサイズが増加します。この結果、対象可変長メモリプールでメモリブロックの獲得待ち行列の先頭タスクが要求するだけの連続空き領域ができると、そのタスクにメモリブロックを割り付けて待ち状態を解除します。待ち行列の以降のタスクに対してもメモリブロックを割り付け可能であれば、待ち行列の順に同様の処理を行います。

5.15.5 可変長メモリプールの状態参照(ref_mpl, iref_mpl)

■C 言語 API

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

■パラメータ

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

■パケットの構造

```
typedef     struct t_rmpl {
            ID      wtskid;     待ち行列先頭のタスク ID
            SIZE    fmplsz;     空き領域の合計サイズ（バイト数）
            UINT    fblksz;     獲得可能な最大メモリブロックサイズ（バイト数）
} T_RMPL;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rmpl == NULL
E_ID	不正 ID 番号 mplid≦0, VTMAX_MPL < mplid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_mpl の呼び出し (2)タスクコンテキストからの iref_mpl 呼び出し
E_MACV	メモリアクセス違反(ref_mpl のみ) 呼び出しタスクからの pk_rmpl が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 mplid の可変長メモリプールが存在しない

■機能説明

mplid で示された可変長メモリプールの状態を参照します。

pk_rmpl が指す領域に待ちタスク ID(wtskid)、現在の空き領域の合計サイズ(fmplsz)、獲得可能な最大メモリブロックのサイズ(fblksz)を返します。

通常空き領域は分断されており、fblksz には分断されている空き領域の中で最大の連続サイズが返ります。1 回の pget_mpl サービスコールで、fblksz までのブロックを即座に獲得できます。

5.16 時間管理(システム時刻管理)機能

表 5.28に、システム時刻管理の仕様を示します。

表5.28 システム時刻管理機能の仕様

No.	項目	内容
1	システム時刻値	符号なし 48 ビット
2	システム時刻の単位	1 [ms]
3	システム時刻の更新周期	TIC_NUME/TIC_DENO [ms] *1
4	システム時刻初期値	0x000000000000
5	システム時刻最大値	0xFFFFFFFFFFFF

【注】 *1 TIC_NUME および TIC_DENO は、cfg600px が kernel_id.h に出力するマクロで、それぞれ cfg ファイルに定義したタイムティック周期の分子(system.tic_nume)と分母(system.tic_deno)を意味します。

表5.29 システム時刻管理機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	set_tim	[S]	システム時刻の設定	T		E	D	U	
2	iset_tim				N	E	D	U	
3	get_tim	[S]	システム時刻の参照	T		E	D	U	
4	iget_tim				N	E	D	U	
5	isig_tim	[S]	タイムティックの供給	(cfg ファイルの clock.timer に CMT0、CMT1, CMT2, CMT3 のいずかを指定することで、自動的に実行されるようになります)					

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.16.1 システム時刻の設定(set_tim, iset_tim)

■C 言語 API

```
ER          get_tim(SYSTIM *p_systim);
ER          iset_tim(SYSTIM *p_systim);
```

■パラメータ

p_systim	設定するシステム時刻を示すパケットへのポインタ

■パケットの構造

```
typedef     struct systim {
            UH      utime;      システム時刻（上位）
            UW      ltime;      システム時刻（下位）
} SYSTIM;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー p_systim == NULL
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの set_tim の呼び出し (2)タスクコンテキストからの iset_tim の呼び出し
E_MACV	メモリアクセス違反(set_tim のみ) 呼び出しタスクからの p_systim が示す領域に対するオペランドリードアクセス許可がない

■機能説明

システムが保持しているシステム時刻の現在の値を、p_systim で示される値に設定します。

なお、システム時刻を変更しても、それ以前に行われた時間管理要求(タスクのタイムアウト、dly_tsk によるタスクの遅延、周期ハンドラ、およびアラームハンドラ)が発生する実時刻は変化しません。

5.16.2 システム時刻の参照(get_tim, iget_tim)

■C 言語 API

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

■パラメータ

p_systim	システム時刻を返すパケットへのポインタ

■パケットの構造

```
typedef     struct systim {
            UH      utime;      システム時刻（上位）
            UW      ltime;      システム時刻（下位）
} SYSTIM;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー p_systim == NULL
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの get_tim の呼び出し (2)タスクコンテキストからの iget_tim の呼び出し
E_MACV	メモリアクセス違反(get_tim のみ) 呼び出しタスクからの p_systim が示す領域に対するオペランドライトアクセス許可がない

■機能説明

システム時刻の現在値を読み出し、その結果を p_systim の指す領域に返します。

5.16.3 タイムティックの供給(isig_tim)

■機能説明

システム時刻を更新します。

cfg ファイルで clock.timer に CMT0, CMT1, CMT2, CMT3 のいずれかを指定すると、TIC_NUME/TIC_DENO[ms]で計算される周期で、自動的に isig_tim サービスコール相当の処理が実行されるようにコンフィギュレーションされます。つまり、本機能はサービスコールではありませんので、アプリケーションから呼び出す必要はありません。

タイムティックの供給時には、カーネルは時間に関する次のような処理を行います。

(1) システム時刻の更新

(2) タイムイベントハンドラの起動

(3) tslp_tskサービスコールなどのタイムアウト付きサービスコールで待ち状態になっているタスクのタイムアウト処理

5.17 時間管理(周期ハンドラ)機能

表 5.30に、周期ハンドラ機能の仕様を示します。

周期ハンドラは、cfg ファイルの cyclic_hand[]定義によって生成します。

表5.30 周期ハンドラ機能の仕様

No.	項目	内容
1	周期ハンドラ ID	1～VTMAX_CYC *1
2	起動周期	1～(0x7FFFFFFF-TIC_NUME)/TIC_DENO *2
3	起動位相	0～(0x7FFFFFFF-TIC_NUME)/TIC_DENO *2
4	拡張情報(ハンドラに渡すパラメータ)	32 ビット
5	周期ハンドラ属性	TA_HLNG：高級言語記述 TA_STA： 周期ハンドラの動作開始 TA_PHS： 起動位相の保存

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大周期ハンドラ ID を意味します。

*2 TIC_NUME および TIC_DENO は、cfg600px が kernel_id.h に出力するマクロで、それぞれ cfg ファイルに定義したタイムティック周期の分子(system.tic_nume)と分母(system.tic_deno)を意味します。

表5.31 周期ハンドラ機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_cyc		周期ハンドラの生成	T		E	D	U	
2	acre_cyc		周期ハンドラの生成(ID 番号自動割付け)	T		E	D	U	
3	del_cyc		周期ハンドラの削除	T		E	D	U	
4	sta_cyc	[S][B]	周期ハンドラの動作開始	T		E	D	U	
5	ista_cyc				N	E	D	U	
6	stp_cyc	[S][B]	周期ハンドラの動作停止	T		E	D	U	
7	istp_cyc				N	E	D	U	
8	ref_cyc		周期ハンドラの状態参照	T		E	D	U	
9	iref_cyc				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。

"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。

"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。

"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.17.1 周期ハンドラの生成(cre_cyc, acre_cyc)

■C 言語 API

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

■パラメータ

cycid	周期ハンドラ ID
pk_ccyc	周期ハンドラ生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef   struct t_ccyc {
          ATR     cycatr;     周期ハンドラ属性
          VP_INT  exinf;      拡張情報
          FP      cychdr;     周期ハンドラの実行開始アドレス
          RELTIM  cyctim;     起動周期
          RELTIM  cycphs;     起動位相
} T_CCYC;
```

■リターン値

・cre_cyc の場合
正常終了（E_OK）、またはエラーコード
・acre_cyc の場合
生成した周期ハンドラの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 cycatr の bit1,2 以外が 1
E_PAR	パラメータエラー (1)pk_ccyc == NULL (2)cychdr == NULL (3)cyctim == 0, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < cyctim (4)cyctim < cycphs
E_ID	不正 ID 番号(cre_cyc のみ) cycid≦0, VTMAX_CYC < cycid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_ccyc が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOID	空き ID なし(acre_cyc のみ)
E_OBJ	オブジェクト状態不正(cre_cyc のみ) cycid の周期ハンドラが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_cyc は、指定された cycid の周期ハンドラを pk_ccyc で指定された内容で生成します。acre_cyc は pk_ccyc で指定された内容で周期ハンドラを生成し、生成した周期ハンドラ ID を返します。

(1) 周期ハンドラ ID(cycid)

cre_cyc は、cycid で指定した ID の周期ハンドラを生成します。

(2) 周期ハンドラ属性(cycatr)

cycatr には以下を指定できます。

cycatr := (TA_HLNG | [TA_STA] | [TA_PHS])

- TA_HLNG(0x0000)
  周期ハンドラの記述言語は C 言語のみをサポートしています。
- TA_STA(0x0002)
  周期ハンドラは動作状態となります。
- TA_PHS(0x0004)
  TA_PHS を指定した場合、周期ハンドラの動作を開始する時に周期ハンドラの起動位相を保存して、次に起動すべき時刻を決定します。TA_PHS が指定されていない場合は、次に起動する時刻は sta_cyc,ista_cyc サービスコールが呼び出された時刻から cyctim 後となります。

(3) 拡張情報(exinf)

exinf は、引数として周期ハンドラに渡されます。exinf は、ユーザが生成する周期ハンドラに関する情報を設定するなどの目的で自由に使用できます。

(4) 周期ハンドラの実行開始アドレス(cychdr)

cychdr には、周期ハンドラの実行開始アドレスを指定します。

(5) 起動周期(cyctim)、起動位相(cycphs)

cyctim には起動周期(ミリ秒)を指定します。

cycphs には、本サービスコール呼出し時点からの起動位相(ミリ秒)を指定します。cycphs は、TA_STA, TA_PHS の少なくとも一方の指定がある場合のみ有効です。

5.17.2 周期ハンドラの削除(del_cyc)

■C 言語 API

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

■パラメータ

cycid	周期ハンドラ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	cycid≦0, VTMAX_CYC < cycid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	cycid の周期ハンドラが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
cycid で示された周期ハンドラを削除します。

5.17.3 周期ハンドラの動作開始(sta_cyc, ista_cyc)

■C 言語 API

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

■パラメータ

cycid	周期ハンドラ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 cycid≦0, VTMAX_CYC < cycid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの sta_cyc の呼び出し (2)タスクコンテキストからの ista_cyc の呼び出し
E_NOEXS	オブジェクト未生成 cycid の周期ハンドラが存在しない

■機能説明

cycid で示された周期ハンドラを、動作している状態に移行させます。

周期ハンドラ属性に TA_PHS が指定されていない場合には、このサービスコールが呼び出された時刻を基準として、その時刻から起動周期が経過する毎に、周期ハンドラが起動されます。

TA_PHS が指定されていない動作している状態の周期ハンドラが指定された場合は、周期ハンドラを次に起動する時刻の再設定のみを行います。

TA_PHS が指定されている場合は、周期ハンドラ生成時点の時刻を基準に起動するため、時刻の設定は行いません。

5.17.4 周期ハンドラの動作停止(stp_cyc, istp_cyc)

■C 言語 API

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

■パラメータ

cycid	周期ハンドラ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 cycid≦0, VTMAX_CYC < cycid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの stp_cyc の呼び出し (2)タスクコンテキストからの istp_cyc の呼び出し
E_NOEXS	オブジェクト未生成 cycid の周期ハンドラが存在しない

■機能説明

cycid で示された周期ハンドラを、動作していない状態に移行させます。

5.17.5 周期ハンドラの状態参照(ref_cyc, iref_cyc)

■C 言語 API

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

■パラメータ

cycid	周期ハンドラ ID
pk_rcyc	周期ハンドラの状態を返すパケットへのポインタ

■パケットの構造

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rcyc == NULL
E_ID	不正 ID 番号 cycid≦0, VTMAX_CYC < cycid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_cyc の呼び出し (2)タスクコンテキストからの iref_cyc の呼び出し
E_MACV	メモリアクセス違反(ref_cyc のみ) 呼び出しタスクからの pk_rcyc が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 cycid の周期ハンドラが存在しない

■機能説明

cycid で示された周期ハンドラの状態を参照し、pk_rcyc が指す領域に周期ハンドラの動作状態(cycstat)、周期ハンドラ起動までの残り時間(lefttim)を返します。

cycstat には、対象周期ハンドラの動作状態を返します。

- TCYC_STP（0x00000000）周期ハンドラが動作していない
- TCYC_STA（0x00000001）周期ハンドラが動作している

lefttim には、対象周期ハンドラを次に起動する時刻までの相対時間を返します。次のタイムティックで起動する場合は 0 が返ります。対象周期ハンドラが動作していない場合、lefttim は不定値となります。

5.18 時間管理(アラームハンドラ)機能

表 5.32に、アラームハンドラ機能の仕様を示します。

アラームハンドラは、cfg ファイルの alarm_hand[]定義によって生成します。

表5.32 アラームハンドラ機能の仕様

No.	項目	内容
1	アラームハンドラ ID	1～VTMAX_ALM *1
2	起動時間	0～(0x7FFFFFFF-TIC_NUME)/TIC_DENO *2
3	拡張情報(ハンドラに渡すパラメータ)	32 ビット
4	アラームハンドラ属性	TA_HLNG：高級言語記述

【注】 *1 cfg600px が kernel_id.h に出力するマクロで、最大周期ハンドラ ID を意味します。

*2 TIC_NUME および TIC_DENO は、cfg600px が kernel_id.h に出力するマクロで、それぞれ cfg ファイルに定義したタイムティック周期の分子(system.tic_nume)と分母(system.tic_deno)を意味します。

表5.33 アラームハンドラ機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	cre_alm		アラームハンドラの生成	T		E	D	U	
2	acre_alm		アラームハンドラの生成(ID 番号自動割付け)	T		E	D	U	
3	del_alm		アラームハンドラの削除	T		E	D	U	
4	sta_alm		アラームハンドラの動作開始	T		E	D	U	
5	ista_alm				N	E	D	U	
6	stp_alm		アラームハンドラの動作停止	T		E	D	U	
7	istp_alm				N	E	D	U	
8	ref_alm		アラームハンドラの状態参照	T		E	D	U	
9	iref_alm				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。

"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。

"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。

"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能

"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能

"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.18.1 アラームハンドラの生成(cre_alm, acre_alm)

■C 言語 API

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

■パラメータ

almid	アラームハンドラ ID
pk_calm	アラームハンドラ生成情報を格納したパケットへのポインタ

■パケットの構造

```
typedef    struct t_calm {
           ATR     almatr;      アラームハンドラ属性
           VP_INT  exinf;       拡張情報
           FP      almhdr;      アラームハンドラの実行開始アドレス
} T_CALM;
```

■リターン値

・cre_alm の場合
正常終了（E_OK）、またはエラーコード

・acre_alm の場合
生成したアラームハンドラの ID 番号（正の値）、またはエラーコード

■エラーコード

E_RSATR	予約属性 almatr != TA_HLNG
E_PAR	パラメータエラー (1)pk_calm == NULL (2)almhdr == NULL
E_ID	不正 ID 番号(cre_alm のみ) almid≦0, VTMAX_ALM < almid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_calm が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOID	空き ID なし(acre_alm のみ)
E_OBJ	オブジェクト状態不正(cre_alm のみ) almid のアラームハンドラが存在

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

cre_alm は、指定された almid のアラームハンドラを pk_calm で指定された内容で生成します。acre_alm は pk_calm で指定された内容でアラームハンドラを生成し、生成したアラームハンドラ ID を返します。

(1) アラームハンドラ ID(almid)

cre_alm は、almid で指定した ID のアラームハンドラを生成します。

(2) アラームハンドラ属性(almatr)

almatr には TA_HLNG のみを指定できます。

- TA_HLNG(0x0000)
  アラームハンドラの記述言語は C 言語のみをサポートしています。

(3) 拡張情報(exinf)

exinf には、アラームハンドラの拡張情報を指定します。exinf は、引数としてアラームハンドラに渡されます。exinf は、ユーザが生成するアラームハンドラに関する情報を設定するなどの目的で自由に使用できます。

(4) アラームハンドラの実行開始アドレス(almhdr)

almhdr には、アラームハンドラの実行開始アドレスを指定します。

5.18.2 アラームハンドラの削除(del_alm)

■C 言語 API

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

■パラメータ

almid	アラームハンドラ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	almid≦0, VTMAX_ALM < almid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反
	呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成
	almid のアラームハンドラが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
almid で示されたアラームハンドラを削除します。

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

■C 言語 API

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

■パラメータ

almid	アラームハンドラ ID
almtim	アラームハンドラの起動時間

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (0x7FFFFFFF - TIC_NUME)/TIC_DENO < almtim
E_ID	不正 ID 番号 almid≦0, VTMAX_ALM < almid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの sta_alm の呼び出し (2)タスクコンテキストからの ista_alm の呼び出し
E_NOEXS	オブジェクト未生成 almid のアラームハンドラが存在しない

■機能説明

almid で示されたアラームハンドラの起動時刻を、サービスコールが呼び出された時刻から almtim で指定された相対時間後に設定し、アラームハンドラの動作を開始します。

すでに動作しているアラームハンドラが指定された場合は、以前の起動時刻の設定を解除し、新しい起動時刻を設定します。

almtim に 0 が指定された場合は、次のタイムティックでアラームハンドラが起動されます。

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

■C 言語 API

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

■パラメータ

almid	アラームハンドラ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号 almid≦0, VTMAX_ALM < almid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの stp_alm の呼び出し (2)タスクコンテキストからの istp_alm の呼び出し
E_NOEXS	オブジェクト未生成 almid のアラームハンドラが存在しない

■機能説明

almid で示されたアラームハンドラの起動時刻の設定を解除し、アラームハンドラの動作を停止します。

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

■C 言語 API

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

■パラメータ

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

■パケットの構造

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

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_ralm == NULL
E_ID	不正 ID 番号 almid≦0, VTMAX_ALM < almid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_alm の呼び出し (2)タスクコンテキストからの iref_alm の呼び出し
E_MACV	メモリアクセス違反(ref_alm のみ) 呼び出しタスクからの pk_ralm が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 almid のアラームハンドラが存在しない

■機能説明

almid で示されたアラームハンドラの状態を参照し、pk_ralm が指す領域にアラームハンドラの動作状態(almstat)、アラームハンドラ起動までの残り時間(lefttim)を返します。

almstat には、対象アラームハンドラの動作状態を返します。

- TALM_STP（0x00000000）アラームハンドラが動作していない
- TALM_STA（0x00000001）アラームハンドラが動作している

lefttim には、対象アラームハンドラ起動までの相対時間を返します。次のタイムティックで起動する場合は 0 が返ります。対象アラームハンドラが動作していない場合、lefttim は不定値となります。

5.19 システム状態管理機能

表5.34 システム状態管理機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	rot_rdq	[S][B]	タスクの優先順位の回転	T		E	D	U	
2	irot_rdq	[S][B]			N	E	D	U	
3	get_tid	[S][B]	実行状態のタスク ID の参照	T		E	D	U	
4	iget_tid	[S]			N	E	D	U	
5	loc_cpu	[S][B]	CPU ロック状態への移行	T		E	D	U	L
6	iloc_cpu	[S]			N	E	D	U	L
7	unl_cpu	[S][B]	CPU ロック状態の解除	T		E	D	U	L
8	iunl_cpu	[S]			N	E	D	U	L
9	dis_dsp	[S][B]	ディスパッチの禁止	T		E	D	U	
10	ena_dsp	[S][B]	ディスパッチの許可	T		E	D	U	
11	sns_ctx	[S]	コンテキストの参照	T	N	E	D	U	L
12	sns_loc	[S]	CPU ロック状態の参照	T	N	E	D	U	L
13	sns_dsp	[S]	ディスパッチ禁止状態の参照	T	N	E	D	U	L
14	sns_dpn	[S]	ディスパッチ保留状態の参照	T	N	E	D	U	L
15	vsta_knl	[V]	カーネルの起動	T	N	E	D	U	L
16	ivsta_knl	[V]		T	N	E	D	U	L
17	vsys_dwn	[V]	システムダウン	T	N	E	D	U	L
18	ivsys_dwn	[V]		T	N	E	D	U	L

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.19.1 タスクの優先順位の回転(rot_rdq, irot_rdq)

■C 言語 API

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

■パラメータ

tskpri	タスク優先度

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)tskpri＜0、TMAX_MPRI< tskpri (2)非タスクコンテキストからの呼び出しで、tskpri==0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(rot_rdq のみ) 呼び出しタスクからの pk_ralm が示す領域に対するオペランドライトアクセス許可がない

■機能説明

tskpri で示された優先度のレディキューにつながれている先頭タスクをレディキューの最後尾につなぎかえます（レディキューを回転）。

rot_rdq では、tskpri=TPRI_SELF(=0)を指定すると、自タスクのベース優先度のレディキューを回転します。

なお、ミューテックス機能を使用しない場合はベース優先度と現在優先度は同じですが、ミューテックスをロック中は一般には現在優先度とベース優先度は一致しません。したがって、ミューテックスロック中は、rot_rdq で TPRI_SELF を指定しても自タスクが属する優先度のレディキューを回転することはできません。

5.19.2 実行状態のタスク ID の参照(get_tid, iget_tid)

■C 言語 API

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

■パラメータ

p_tskid	タスク ID を返す記憶域へのポインタ

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー p_tskid == NULL
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの get_tid の呼び出し (2)タスクコンテキストからの iget_tid の呼び出し
E_MACV	メモリアクセス違反(get_tid のみ) 呼び出しタスクからの p_tskid が示す領域に対するオペランドライトアクセス許可がない

■機能説明

実行状態のタスクの ID を求め、その結果を p_tskid の指す領域に返します。

具体的には、タスクコンテキストから呼び出された場合は自タスクの ID を返し、非タスクコンテキストから呼び出された場合はその時実行していたタスクの ID を返します。実行状態のタスクがない場合は、TSK_NONE(=0)を返します。

5.19.3 CPU ロック状態への移行(loc_cpu, iloc_cpu)

■C 言語 API

```
ER          loc_cpu(void);
ER          iloc_cpu(void);
```

■パラメータ

なし

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの loc_cpu の呼び出し (2)タスクコンテキストからの iloc_cpu の呼び出し
E_ILUSE	サービスコール不正使用 chg_ims で割り込みマスクを 0 以外に変更している状態で loc_cpu を呼び出した

■機能説明

システムを CPU ロック状態とします。

CPU ロック状態の特長を以下に示します。

1. CPUロック状態の間は、タスクのディスパッチが禁止されます。(補足参照)
2. cfgファイルで指定したsystem.system_IPL(カーネル割り込みマスクレベル)以下のレベルの割り込みが禁止されます。(PSWレジスタのIPLビットが、system.system_IPLに変更されます)
3. CPUロック状態から呼び出し可能なサービスコールは、以下のサービスコールのみです。
   - •ext_tsk
   - •exd_tsk
   - •loc_cpu, iloc_cpu
   - •unl_cpu, iunl_cpu
   - •sns_ctx
   - •sns_loc
   - •sns_dsp
   - •sns_dpn
   - •vsys_dwn, ivsys_dwn

CPU ロック状態は、以下の操作で解除されます。

(a) unl_cpu, iunl_cpuサービスコールの呼び出し

(b) ext_tskサービスコールの呼び出し(タスク開始関数からのリターンも含む)

(c) exd_tskサービスコールの呼び出し

カーネル管理割り込みハンドラ、およびタイムイベントハンドラの終了時には、必ず CPU ロック解除状態でなければなりません。CPU ロック状態の場合は、ret_int 時にシステムダウンとなります。なお、これらのハンドラ開始時は、常に CPU ロック解除状態です。

chg_ims サービスコールで割り込みマスクを 0 以外に変更している間に loc_cpu サービスコールを呼び出すと、エラーE_ILUSE が返ります。

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

■補足

CPU ロック状態とディスパッチ禁止状態は、独立して管理されます。

5.19.4 CPU ロック状態の解除(unl_cpu, iunl_cpu)

■C 言語 API

```
ER          unl_cpu(void);
ER          iunl_cpu(void);
```

■パラメータ

なし

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(unl_cpu のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している

■機能説明

CPU ロック状態を解除します。

具体的には、unl_cpu は loc_cpu サービスコールによって禁止されていたディスパッチを許可し、CPU の PSW レジスタの IPL ビットを 0 に変更します。ただし、dis_dsp サービスコールによるディスパッチ禁止状態から loc_cpu サービスコールが呼び出されていた場合は、loc_cpu サービスコール後もディスパッチ禁止状態が継続します。(補足参照)

iunl_cpu サービスコールは、CPU の PSW レジスタの IPL ビットを、iunl_cpu サービスコール直前の値に戻します。

ハンドラで iloc_cpu を使用する場合、ハンドラ終了前に必ず CPU ロック状態を解除してください。

CPU ロック解除状態から本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

■補足

CPU ロック状態とディスパッチ禁止状態は、独立して管理されます。そのため、unl_cpu, iunl_cpu サービスコールでは、ディスパッチ禁止状態は解除されません。

5.19.5 ディスパッチの禁止(dis_dsp)

■C 言語 API

```
ER          dis_dsp(void);
```

■パラメータ

なし

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）

■機能説明

システム状態をディスパッチ禁止状態にします。ディスパッチ禁止状態の特長を、以下に示します。

(1) タスクのスケジューリングが行われなくなるため、自タスク以外のタスクがRUNNING状態に移行することはなくなります。
(2) 割り込みは受け付けられます。
(3) 待ち状態になるサービスコールを呼び出せません。

以下の操作により、ディスパッチ禁止状態に移行します。

(1) dis_dspサービスコールの呼び出し
(2) chg_imsサービスコールで割り込みマスク(PSWレジスタのIPLビット)を0以外に変更

ディスパッチ禁止状態は、以下の操作で解除されます。

(1) ena_dspサービスコールの呼び出し
(2) ext_tskサービスコールの呼び出し(タスク開始関数からのリターンも含む)
(3) exd_tskサービスコールの呼び出し
(4) chg_imsサービスコールで割り込みマスク(PSWレジスタのIPLビット)を0に変更

ディスパッチ禁止状態の間は、ref_tsk, iref_tsk, ref_tst, iref_tst サービスコールで自タスクの状態を参照しても実行状態とは見えない場合があるので、注意してください。

すでにディスパッチ禁止状態のときに再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

5.19.6 ディスパッチの許可(ena_dsp)

■C 言語 API

```
ER          ena_dsp(void);
```

■パラメータ

なし

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している

■機能説明

dis_dsp サービスコール、または chg_ims サービスコールによって設定されていたディスパッチ禁止状態を解除し、タスクのスケジューリングを行います。

ディスパッチ許可状態から本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。

5.19.7 コンテキストの参照(sns_ctx)

■C 言語 API

```
BOOL        sns_ctx(void);
```

■パラメータ

なし

■リターン値

TRUE または FALSE またはエラーコード

■エラーコード

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）

■機能説明

現在のコンテキスト種別が非タスクコンテキストなら TRUE、タスクコンテキストなら FALSE を返します。
本サービスコールは、CPU ロック状態からも呼び出せます。

5.19.8 CPU ロック状態の参照(sns_loc)

■C 言語 API

```
BOOL        sns_loc(void);
```

■パラメータ

なし

■リターン値

TRUE または FALSE またはエラーコード

■エラーコード

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）

■機能説明

CPU ロック状態なら TRUE、そうでなければ FALSE を返します。
本サービスコールは、CPU ロック状態からも呼び出せます。

5.19.9 ディスパッチ禁止状態の参照(sns_dsp)

■C 言語 API

```
BOOL        sns_dsp(void);
```

■パラメータ

なし

■リターン値

TRUE または FALSE またはエラーコード

■エラーコード

E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)

■機能説明

ディスパッチ禁止状態なら TRUE、そうでなければ FALSE を返します。
本サービスコールは、CPU ロック状態からも呼び出せます。

5.19.10 ディスパッチ保留状態の参照(sns_dpn)

■C 言語 API

```
BOOL        sns_dpn(void);
```

■パラメータ

なし

■リターン値

TRUE または FALSE またはエラーコード

■エラーコード

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）

■機能説明

ディスパッチ保留状態なら TRUE、そうでなければ FALSE を返します。
ディスパッチ保留状態とは、以下のいずれかの条件を満たすときです。

(1) ディスパッチ禁止状態である
(2) CPUロック状態である
(3) 非タスクコンテキスト実行中である

本サービスコールは、CPU ロック状態からも呼び出せます。

5.19.11 カーネルの起動(vsta_knl, ivsta_knl)

■C 言語 API

```
void vsta_knl(void);
void ivsta_knl(void);
```

■パラメータ

なし

■機能説明

カーネルを起動します。本サービスコールからリターンすることはありません。

本サービスコールの処理概要を、以下に示します。

1. MPU(Memory Protection Unit)を全アクセス禁止に設定する。
2. INTBレジスタを、cfg600pxによって生成された可変ベクタテーブルの先頭アドレスに初期化
3. カーネル内部テーブルを初期化
4. cfgファイルで指定された各種オブジェクトの生成
5. システムタイマを初期化(__RI_init_cmt_knl()の呼出し)
   「7.2(2) タイマ初期化コールバック関数(_RI_init_cmt_knl())」を参照
6. マルチタスク環境へ移行

これらの処理でエラーを検出した場合は、システムダウンとなります。

本サービスコールは、必ず以下を満たす状態から呼び出すようにしてください。

1. CPUが割り込みを受理しないこと(例えば、PSW.I == 0)
2. スーパバイザモード(PSW.PM == 0)であること

なお、本サービスコールでは、E_CTX エラーは検出されません。

本サービスコールは、割り込みマスク(PSW.IPL)がカーネル割り込みマスクレベル(system.system_IPL)より高いときにも呼び出せます。

本サービスコールはμITRON4.0 仕様外の機能です。

5.19.12 システムダウン(vsys_dwn, ivsys_dwn)

■C 言語 API

```
void   vsys_dwn(W type, VW inf1, VW inf2, VW inf3);
void   ivsys_dwn(W type, VW inf1, VW inf2, VW inf3);
```

■パラメータ

type	エラー種別
inf1	システム異常情報 1
inf2	システム異常情報 2
inf3	システム異常情報 3

■機能説明

システムダウンルーチンに制御を渡します。

type には、エラー種別として発生したエラーに対応した値(1～0x7FFFFFFF)を設定してください。なお、0 以下の値はシステム用に予約されています。

カーネル内で異常を検出した場合にも、システムダウンルーチンが呼び出されます。

本サービスコールは、すべての状態から呼び出せます。

本サービスコールからリターンすることはありません。

また、本サービスコールでは、E_CTX エラーは検出されません。

引数仕様の詳細は、「6.8 システムダウンルーチン」を参照してください。

本サービスコールは、割り込みマスク(PSW.IPL)がカーネル割り込みマスクレベル(system.system_IPL)より高いときにも呼び出せます。

本サービスコールはμITRON4.0 仕様外の機能です。

5.20 割り込み管理機能

表5.35 割り込み管理機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	chg_ims		割り込みマスクの変更	T		E	D	U	
2	ichg_ims				N	E	D	U	
3	get_ims		割り込みマスクの参照	T		E	D	U	
4	iget_ims				N	E	D	U	
5	ret_int	[S][B]	カーネル管理割り込みハンドラからの復帰		N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.20.1 割り込みマスクの変更(chg_ims, ichg_ims)

■C 言語 API

```
ER      chg_ims(IMASK imask);
ER      ichg_ims(IMASK imask);
```

■パラメータ

imask	割り込みマスク値

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー imask > 15
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反(chg_ims のみ) スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している

■機能説明

CPU の割り込みマスク(PSW.IPL)を imask で指定した値に変更します。

imask には、0～15 の指定ができます。

chg_ims サービスコールでは、imask に 0 以外を指定するとシステムはディスパッチ禁止状態に移行(dis_dsp と等価)し、imask に 0 を指定するとシステムはディスパッチ許可状態に移行(ena_dsp と等価)します。一方、ichg_ims では、ディスパッチ禁止/許可状態の遷移はありません。

なお、PSW レジスタはタスクに従属するコンテキストとして扱われます。(補足 2 参照)

本サービスコールは、割り込みマスク(PSW.IPL)がカーネル割り込みマスクレベル(system.system_IPL)より高いときにも呼び出せます。

割り込みマスクを変更した場合は、そのコンテキスト(タスクやハンドラ)が終了する前に、必ず割り込みマスクを元に戻さなければなりません。

■補足

1. 非タスクコンテキストでは、起動時よりも割り込みマスク値を下げてはなりません。
2. 割り込みマスク(PSW.IPL)を0以外に変更後にena_dspを行うと、その時点でディスパッチ許可状態となります。PSWはタスクに従属するコンテキストであるため、別のタスクにディスパッチすると、割り込みマスクもディスパッチ先のタスクの状態に変更されることになります。
3. 割り込みマスクを0以外に変更している間は、loc_cpuはE_ILUSEエラーを返します。
4. 割り込みマスクがカーネル割り込みマスクレベルよりも高い時に呼び出せるサービスコールは､以下に制限されます。
   chg_ims, ichg_ims, get_ims, iget_ims, vsta_knl, ivsta_knl, vsys_dwn, ivsys_dwn

5.20.2 割り込みマスクの参照(get_ims, iget_ims)

■C 言語 API

```
ER          get_ims(IMASK *p_imask);
ER          iget_ims(IMASK *p_imask);
```

■パラメータ

p_imask	割り込みマスクレベルを返す記憶域へのポインタ

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー p_imask == NULL
E_MACV	メモリアクセス違反(get_ims のみ) 呼び出しタスクからの p_imask が示す領域に対するオペランドライトアクセス許可がない

■機能説明

現在の割り込みマスクレベル(PSW.IPL)を p_imask の指す領域に返します。
なお、本サービスコールでは、E_CTX エラーは検出されません。

本サービスコールは、割り込みマスク(PSW.IPL)がカーネル割り込みマスクレベル(system.system_IPL)より高いときにも呼び出せます。

5.20.3 カーネル管理割り込みハンドラからの復帰(ret_int)

■C 言語 API

なし（cfg ファイルでの割り込みハンドラ定義により、ハンドラ終了時に自動的に本サービスコールが呼び出されるようになります）

サービスコールの呼び出し元には戻りません。

以下のエラーが発生するとシステムダウンとなります。

E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）
E_MACV	メモリアクセス違反 割り込まれたタスクのスタックポインタがユーザスタック領域の範囲外を指している

■機能説明

カーネル管理割り込みハンドラからの復帰処理を行います。

タスクコンテキスト実行中に起動したカーネル管理割り込みハンドラから復帰する場合は、スケジューラを動作させ、タスクの切り替えを行います。

その他の場合は、割り込みを受理したプログラムに復帰します。

カーネル管理割り込みハンドラ終了時、すなわち本サービスコール呼び出し時点の割り込みマスク(PSW.IPL)は、カーネル割り込みマスクレベル(system.system_IPL)以下でなければなりません。そうでない場合、本サービスコールでコンテキストエラーが検出され、システムダウンとなります。このケースに該当するのは、以下のケースです。

(1) 誤って、カーネル管理外割り込みをカーネル管理割り込みハンドラとして定義した場合

(2) カーネル管理割り込みハンドラ内で、PSW.IPLをカーネル割り込みマスクレベル(system.system_IPL)以上の値に変更したまま終了した場合

また、CPU ロック状態で呼び出した場合、およびタスクコンテキストから呼び出した場合もシステムダウンとなります。

■補足

割り込みには、カーネル管理割り込みとカーネル管理外割り込みの 2 種類があります。これについては、「2.7.1 割り込みの種類」を参照してください。

カーネル管理割り込みハンドラは、cfg ファイルの interrupt_vector[].os_int に YES を指定して定義します。この場合、該当ハンドラ関数のコンパイル時に、ハンドラの最後で本サービスコールが呼び出されるオブジェクトコードが生成されます。

一方、カーネル管理外割り込みハンドラの場合は、可変ベクタ割り込みは cfg ファイルの interrupt_vector[].os_int に NO を指定して定義し、固定ベクタ割り込みは interrupt_fvector[]で定義します。この場合、該当ハンドラ関数のコンパイル時に、ハンドラの最後で RTE 命令でリターンするオブジェクトコードが生成されます。

5.21 システム構成管理機能

表5.36 システム構成管理機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	ref_ver		バージョン情報の参照	T		E	D	U	
2	iref_ver				N	E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.21.1 バージョン情報の参照(ref_ver, iref_ver)

■C 言語 API

```
ER          ref_ver(T_RVER *pk_rver);
ER          iref_ver(T_RVER *pk_rver);
```

■パラメータ

pk_rver	バージョン情報を返すパケットへのポインタ

■パケットの構造

```
typedef     struct t_rver {
            UH      maker;      カーネルのメーカーコード
            UH      prid;       カーネルの識別番号
            UH      spver;      ITRON 仕様のバージョン番号
            UH      prver;      カーネルのバージョン番号
            UH      prno[4];    カーネル製品の管理情報
} T_RVER;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rver == NULL
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し) 注：以下のケースでは、E_CTX エラーは検出されません。 (1)非タスクコンテキストからの ref_ver の呼び出し (2)タスクコンテキストからの iref_ver の呼び出し
E_MACV	メモリアクセス違反(ref_ver のみ) 呼び出しタスクからの pk_rver が示す領域に対するオペランドライトアクセス許可がない

■機能説明

現在実行中のカーネルのバージョンに関する情報を読み出し、その結果を pk_rver の指す領域に返します。
pk_rver の指すパケットには、次の情報を返します。

(1) カーネルのメーカーコード(maker)

maker は、このカーネルを作ったメーカを表します。本カーネルでは、ルネサスエレクトロニクスを意味する 0x011B です。

(2) カーネルの識別番号(prid)

prid は、カーネルや VLSI の種類を区別する番号を表します。本カーネルでは、0x0004 が返ります。

(3) ITRON 仕様のバージョン番号(spver)

spver は、カーネルの準拠する仕様を表しており、ビット対応に意味を持っています。

- bit15～12：MAGIC（TRON 仕様のシリーズを区別する番号）
  本カーネルでは、0x5（μITRON 仕様）です。
- bit11～0：SpecVer（製品の元となった TRON 仕様書のバージョン番号）
  本カーネルでは、0x403（μITRON4.0 仕様 Ver.4.03）です。

(4) カーネルのバージョン番号(prver)

prver は、カーネルのバージョン番号を表します。
prver の値は、製品バージョンによって異なります。例えば、V.1.00 Release 00 の prver は、0x0100 となります。

(5) カーネル製品の管理情報(prno)

prno は、製品管理情報や製品番号などを表します。
本カーネルの prno[0]から prno[3]の値は 0x0000 です。

5.22 オブジェクトリセット機能

オブジェクトリセット機能は、各種オブジェクトを初期状態に戻す機能です。本機能は、μITRON4.0 仕様外です。

表5.37 オブジェクトリセット機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	vrst_dtq	[V]	データキューのリセット	T		E	D	U	
2	vrst_mbx	[V]	メールボックスのリセット	T		E	D	U	
3	vrst_mbf	[V]	メッセージバッファのリセット	T		E	D	U	
4	vrst_mpf	[V]	固定長メモリプールのリセット	T		E	D	U	
5	vrst_mpl	[V]	可変長メモリプールのリセット	T		E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.22.1 データキューのリセット(vrst_dtq)

■C 言語 API

```
ER        vrst_dtq( ID dtqid );
```

■パラメータ

dtqid	データキューID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	dtqid≦0, VTMAX_DTQ < dtqid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	dtqid のデータキューが存在しない

■機能説明

dtqid で示されたデータキューをリセットします。

具体的には、データキューに格納されているデータをクリアし、データキューへの送信待ちタスクがある場合は、それらの待ち状態を解除します。この場合、待ち解除されたタスクには、エラーコード EV_RST が返ります。

なお、データキューからの受信待ちタスクは待ち解除されません。

本サービスコールはμITRON4.0 仕様外の機能です。

5.22.2 メールボックスのリセット(vrst_mbx)

■C 言語 API

```
ER          vrst_mbx( ID mbxid );
```

■パラメータ

mbxid	メールボックス ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	mbxid≦0, VTMAX_MBX < mbxid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	mbxid のメールボックスが存在しない

■機能説明

mbxid で示されたメールボックスをリセットします。
具体的には、メッセージキューを空にします。

本サービスコールはμITRON4.0 仕様外の機能です。

5.22.3 メッセージバッファのリセット(vrst_mbf)

■C 言語 API

```
ER          vrst_mbf( ID mbfid );
```

■パラメータ

mbfid	メッセージバッファ ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	(1)mbfid≦0, VTMAX_MBF < mbfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	mbfid のメッセージバッファが存在しない

■機能説明

mbfid で示されたメッセージバッファをリセットします。

具体的には、メッセージバッファに格納されているメッセージをクリアし、メッセージバッファへの送信待ちタスクがある場合は、それらの待ち状態を解除します。この場合、待ち解除されたタスクには、エラーコード EV_RST が返ります。

なお、メッセージバッファからの受信待ちタスクは待ち解除されません。

本サービスコールはμITRON4.0 仕様外の機能です。

5.22.4 固定長メモリプールのリセット(vrst_mpf)

■C 言語 API

```
ER          vrst_mpf( ID mpfid );
```

■パラメータ

mpfid	固定長メモリプール ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	(1)mpfid≦0, VTMAX_MPF < mpfid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	mpfid の固定長メモリプールが存在しない

■機能説明

mpfid で示された固定長メモリプールをリセットします。

具体的には、すべてのメモリブロックを未使用状態に変更し、メモリブロック獲得待ちタスクがある場合は、それらの待ち状態を解除します。この場合、待ち解除されたタスクには、エラーコード EV_RST が返ります。

すべてのメモリブロックは未使用状態の扱いとなるため、本サービスコール以降はそれ以前に獲得していたメモリブロックを使用してはなりません。

本サービスコールはμITRON4.0 仕様外の機能です。

5.22.5 可変長メモリプールのリセット(vrst_mpl)

■C 言語 API

```
ER      vrst_mpl( ID mplid );
```

■パラメータ

mplid	可変長メモリプール ID

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_ID	不正 ID 番号
	(1)mplid≦0, VTMAX_MPL < mplid
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し)
E_MACV	メモリアクセス違反
	スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成
	mplid の可変長メモリプールが存在しない

■機能説明

mplid で示された可変長メモリプールをリセットします。

具体的には、可変長メモリプールの全メモリを未使用状態に変更し、メモリブロック獲得待ちタスクがある場合は、それらの待ち状態を解除します。この場合、待ち解除されたタスクには、エラーコード EV_RST が返ります。

すべてのメモリブロックは未使用状態の扱いとなるため、本サービスコール以降はそれ以前に獲得していたメモリブロックを使用してはなりません。

本サービスコールはμITRON4.0 仕様外の機能です。

5.23 メモリオブジェクト管理機能

表5.38 メモリオブジェクト管理機能サービスコール一覧

No.	サービスコール *1		機能	呼び出し可能なシステム状態 *2					
				T	N	E	D	U	L
1	ata_mem		メモリオブジェクトの登録	T		E	D	U	
2	det_mem		メモリオブジェクトの登録解除	T		E	D	U	
3	sac_mem		メモリオブジェクトのアクセス許可ベクタの変更	T		E	D	U	
4	vprb_mem	[V]	メモリ領域に対するアクセス権のチェック	T		E	D	U	
5	ref_mem		メモリオブジェクト状態の参照	T		E	D	U	

【注】 *1 "[S]"はμITRON4.0 仕様のスタンダードプロファイルのサービスコールです。
"[B]"はμITRON4.0 仕様のベーシックプロファイルのサービスコールです。
"[V]"はμITRON4.0 仕様外のサービスコールです。

*2 それぞれの記号は、以下の意味です。
"T"はタスクコンテキストから呼び出し可能、"N"は非タスクコンテキストから呼び出し可能
"E"はディスパッチ許可状態から呼び出し可能、"D"はディスパッチ禁止状態から呼び出し可能
"U"は CPU ロック解除状態から発行可能、"L"は CPU ロック状態から呼び出し可能

5.23.1 メモリオブジェクトの登録(ata_mem)

■C 言語 API

```
ER        ata_mem( T_AMEM *pk_amem, ACVCT *p_acvct);
```

■パラメータ

pk_amem	生成するメモリオブジェクト情報へのポインタ
p_acvct	メモリオブジェクトのアクセス許可ベクタへのポインタ

■パケットの構造

```
typedef    struct t_amem {
           ATR     mematr       メモリオブジェクト属性
           VP      base         メモリオブジェクトの先頭アドレス
           SIZE    size         メモリオブジェクトのサイズ(バイト数)
} T_AMEM;

typedef    struct acvct {
           ACPTN   acptn1       オペランドリードアクセス許可パターン
           ACPTN   acptn2       オペランドライトアクセス許可パターン
           ACPTN   acptn3       実行アクセス許可パターン
} ACVCT;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)pk_amem == NULL (2)base が 16 バイト境界でない (3)size が 16 の倍数でない (4)p_acvct == NULL (5)acptn1 == acptn2 == acptn3 == 0 (6)acptn1,acptn2,acptn3 の最大ドメイン ID より大きなドメイン ID に対応する bit が 1 (7)base+size＞0x100000000 (8)size == 0
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_amem が示す領域に対するオペランドリードアクセス許可がない (3)呼び出しタスクからの p_acvct が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 (1)呼出しタスクは、信頼されたドメインに所属していない (2)同一ドメインにアクセス許可されるメモリオブジェクトが 7 個を超える
E_OBJ	登録済み 既に base が同じメモリオブジェクトが登録されている

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。

ata_mem サービスコールは、base で示されたアドレスから size バイトの領域を、p_acvct で指定したアクセス許可ベクタを持つメモリオブジェクトとして登録します。

メモリオブジェクト領域は、以下を満たす必要があります。

1. 先頭アドレスは16バイト境界であること。そうでない場合はE_PARエラーを返します。
2. サイズは16の整数倍であること。そうでない場合はE_PARエラーを返します。
3. 他のユーザスタックおよびメモリオブジェクトと重なっていないこと。そうでない場合、エラーは検出されず、システムの正常な動作は保証されません。

なお、メモリオブジェクト属性(mematr)は単に無視されます。

5.23.2 メモリオブジェクトの登録解除(det_mem)

■C 言語 API

```
ER          det_mem(VP base);
```

■パラメータ

base	メモリオブジェクトの先頭アドレス

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_OACV	オブジェクトアクセス違反 呼出しタスクは、信頼されたドメインに所属していない
E_NOEXS	オブジェクト未生成 開始アドレスが base のメモリオブジェクトが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
先頭アドレスが base のメモリオブジェクトの登録を解除します。

5.23.3 メモリオブジェクトのアクセス許可ベクタの変更(sac_mem)

■C 言語 API

```
ER          sac_mem( VP base, ACVCT *p_acvct);
```

■パラメータ

base	対象メモリオブジェクト開始アドレス
p_acvct	対象メモリオブジェクトのアクセス許可ベクタへのポインタ

■パケットの構造

```
typedef    struct acvct {
           ACPTN   acptn1      オペランドリードアクセス許可パターン
           ACPTN   acptn2      オペランドライトアクセス許可パターン
           ACPTN   acptn3      実行アクセス許可パターン
} ACVCT;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー (1)p_acvct == NULL (2)acptn1 == acptn2 == acptn3 == 0 (3)acptn1,acptn2,acptn3 の最大ドメイン ID より大きなドメイン ID に対応する bit が 1
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの p_acvct が示す領域に対するオペランドリードアクセス許可がない
E_OACV	オブジェクトアクセス違反 (1)呼出しタスクは、信頼されたドメインに所属していない (2)同一ドメインにアクセス許可されるメモリオブジェクトが 7 個を超える
E_NOEXS	オブジェクト未生成 開始アドレスが base のメモリオブジェクトが存在しない

■機能説明

本サービスコールは、信頼されたドメインに所属するタスクからのみ呼び出せます。
先頭アドレスがbaseのメモリオブジェクトのアクセス許可ベクタを、p_acvctで指定した内容に変更します。

5.23.4 メモリ領域に対するアクセス権のチェック(vprb_mem)

■C 言語 API

```
ER__BOOL   vprb_mem( VP base , SIZE size, ID tskid, MODE pmmode);
```

■パラメータ

base	チェック対象アドレス
size	チェック対象サイズ（バイト数）
tskid	タスク ID
pmmode	アクセスモード

■リターン値

TRUE または FALSE またはエラーコード

■エラーコード

E_ID	不正 ID 番号 tskid<0, VTMAX_TSK < tskid
E_PAR	パラメータエラー (1)size == 0 (2)pmmode == 0, pmmode の bit0～2 以外のいずれの bit が 1
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し
E_MACV	メモリアクセス違反 スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している
E_NOEXS	オブジェクト未生成 tskid のタスクが存在しない

■機能説明

base で指定されたアドレスから size バイトの領域について、tskid で指定されたタスクが pmmode で指定されたアクセスが許可されているかをチェックします。許可されている場合は TRUE、許可されていない場合は FALSE を返します。

pmmode には以下を指定できます。

pmmode := (TPM_READ | TPM_WRITE | TPM_EXEC)

- TPM_READ(0x0001)　オペランドリードアクセス権をチェック
- TPM_WRITE(0x0002)　オペランドライトアクセス権をチェック
- TPM_EXEC(0x0004)　実行権をチェック

tskid=TSK_SELF(=0)の指定により、自タスクの指定になります。

本サービスコールはμITRON4.0 仕様外の機能です。

5.23.5 メモリオブジェクトの状態参照(ref_mem)

■C 言語 API

```
ER        ref_mem(VP base, T_RMEM *pk_rmem);
```

■パラメータ

base	メモリオブジェクト先頭アドレス
pk_rmem	メモリオブジェクトの状態を返すパケットへのポインタ

■パケットの構造

```
typedef    struct acvct {
           ACPTN   acptn1      オペランドリードアクセス
           ACPTN   acptn2      オペランドライトアクセス
           ACPTN   acptn3      実行 アクセス
} ACVCT;

typedef    struct t_rmem {
           ACVCT   acvct;      アクセス許可ベクタ
} T_RMEM;
```

■リターン値

正常終了（E_OK）、またはエラーコード

■エラーコード

E_PAR	パラメータエラー pk_rmem == NULL
E_CTX	コンテキストエラー(許可されていないシステム状態からの呼び出し
E_MACV	メモリアクセス違反 (1)スタックポインタが呼出しタスクのユーザスタック領域の範囲外を指している (2)呼び出しタスクからの pk_rmem が示す領域に対するオペランドライトアクセス許可がない
E_NOEXS	オブジェクト未生成 開始アドレスが base のメモリオブジェクトが存在しない

■機能説明

先頭アドレスが base のメモリオブジェクトの状態を参照し、pk_rmem が指す領域に返します。

6. アプリケーション記述方法

6.1 ヘッダファイル

以下のヘッダファイルをインクルードしてください。

- kernel.h
  カーネルのヘッダファイルです。kernel.h は"インストールディレクトリ¥inc600"ディレクトリに格納されています。
- kernel_id.h
  kernel_id.h は、cfg600px によって出力されます。kernel_id.h には、cfg ファイルに指定した各種オブジェクト名称(xxx[].name)、カーネル構成マクロ、タスクやハンドラのプロトタイプ宣言などが定義されます。
  ID 名称は、以下のようにサービスコールに渡す ID 番号に利用できます。

```
ercd = act_tsk(ID_TASK1);
```

6.2 変数の扱い

C 言語における変数の記憶クラスと、タスクやハンドラといったカーネル仕様上のプログラム種別の間には、何ら関連性はありません。

表 6.1に、C 言語における変数の扱いを示します。例えば、グローバル変数を複数のタスクから同時にアクセスする可能性がある場合は、それらのタスクの間で排他制御が必要です。

表6.1 C 言語における変数の扱い

No.	記憶クラス	扱い	記憶域の割り当て
1	グローバル変数	全プログラム(タスク、ハンドラ)の共有変数	静的
2	関数外の static 変数	同一ファイル内の関数の共有変数	静的
3	オート変数、レジスタ変数、関数内の static 変数	当該関数内の変数	動的(スタック)

6.3 タスク

6.3.1 コーディング

図 6.1に、タスク開始関数のコーディング例を示します。

```
#include "kernel.h"
#include "kernel_id.h"
#pragma task Task1              /* (1) */
void  Task1(VP_INT exinf);      /* (2) */
void  Task1(VP_INT exinf)       /* (3) */
{
    /* 処理 */                  /* (4) */
    ext_tsk();                  /* (5) */
}
```

図6.1 タスク開始関数のコーディング例

(1) タスク開始関数に対して、#pragma taskディレクティブを記述します。ただし、cfgファイルのtask[]で生成したタスクの場合は、cfg600pxがkernel_id.hに本ディレクティブ出力するため、不要です。

(2) タスク開始関数のプロトタイプ宣言を行います。ただし、cfgファイルで生成したタスクの場合は、cfg600pxがkernel_id.hに本宣言を出力するため、不要です。

(3) タスクの開始関数のAPIは、ここに記載の通りです。
表6.2に、exinfに渡される値を示します。

表6.2 exinf に渡される値

起動方法	exinf に渡される値
cfg ファイルの task[]で initial_start に ON を指定	タスクの拡張情報
cre_tsk, acre_tsk で TA_ACT 属性を指定	
act_tsk, iact_tsk	
sta_tsk, ista_tsk	sta_tsk, ista_tsk で指定した起動コード(stacd)

(4) タスクでは、タスクコンテキストから呼び出し可能なサービスコールを使用することができます。

(5) タスクを終了させたい箇所で、ext_tsk()を呼び出してください。なお、タスク開始関数からのリターンでも、ext_tsk()と同じ動作を行います。

図 6.2のように、タスク開始関数を無限ループで記述することもできます。

```
#include "kernel.h"
#include "kernel_id.h"
#pragma task Task1
void  Task1(VP_INT exinf);
void  Task1(VP_INT exinf)
{
    for(;;) {
    /* 処理 */
     }
}
```

図6.2 無限ループするタスク開始関数のコーディング例

6.3.2 起動時の CPU 状態

タスクはユーザモードで実行されるので、特権命令は実行できません。なお、アセンブラでは、特権命令の使用を Warning とする機能(-chkpm オプション)があります。

また、タスク起動時は、全割り込みが許可された状態です。

表6.3 **タスク起動時の** PSW

No.	ビット	初期値	解説
1	IPL	0	**全割り込み受理可能**
2	I	1	
3	PM	1	**ユーザモード**
4	U	1	USP(**ユーザスタックポインタ**)
5	C,Z,S,O	**不定**	
6	**その他のビット**	0	

また、cfg ファイルで system.context に FPSW を含む設定をした場合、タスク起動時の FPSW は 0x00000100 となります。

6.4 タスク例外処理ルーチン

6.4.1 コーディング

図 6.3に、タスク例外処理ルーチン開始関数のコーディング例を示します。

```
#include "kernel.h"
#include "kernel_id.h"
#pragma taskexception Texrtn1                  /* (1) */
void  Texrtn1(TEXPTN texptn, VP_INT exinf);    /* (2) */
void  Texrtn1(TEXPTN texptn, VP_INT exinf)     /* (3) */
{
    /* 処理 */                                  /* (4) */
}
```

図6.3 タスク例外処理ルーチン開始関数のコーディング例

(1) タスク例外処理ルーチン開始関数に対して、#pragma taskexceptionディレクティブを記述します。ただし、cfgファイルのtask[]で定義したタスク例外処理ルーチンの場合は、cfg600pxがkernel_id.hに本ディレクティブを出力するため、不要です。

(2) タスク例外処理ルーチン開始関数のプロトタイプ宣言を行います。ただし、cfgファイルで定義したタスク例外処理ルーチンの場合は、cfg600pxがkernel_id.hに本宣言を出力するため、不要です。

(3) タスク例外処理ルーチンの開始関数のAPIは、ここに記載の通りです。
texptnには例外要因パターン、exinfにはタスクの拡張情報が渡されます。

(4) タスク例外処理ルーチンでは、タスクコンテキストから呼び出し可能なサービスコールを使用することができます。

6.4.2 起動時の CPU 状態

タスク例外処理ルーチンはユーザモードで実行されるので、特権命令は実行できません。なお、アセンブラでは、特権命令の使用を Warning とする機能(-chkpm オプション)があります。

表6.4 タスク例外処理ルーチン起動時の PSW

No.	ビット	初期値	解説
1	IPL	**タスク例外処理ルーチン起動直前のタスク本体と同じ**	
2	I	1	
3	PM	1	**ユーザモード**
4	U	1	USP(**ユーザスタックポインタ**)
5	C,Z,S,O	**不定**	
6	**その他のビット**	0	

また、cfg ファイルで system.context に FPSW を含む設定をした場合、タスク例外処理ルーチン起動時の FPSW は 0x00000100 となります。

6.5 割り込みハンドラ

6.5.1 コーディング

図 6.4に、割り込みハンドラ開始関数のコーディング例を示します。

```
#include "kernel.h"
#include "kernel_id.h"
void  int_handler(void)      /* (1) */
{
    /* 処理 */                /* (2) */
}                             /* (3) */
```

図6.4 割り込みハンドラ開始関数のコーディング例

(1) 割り込みハンドラの開始関数のAPIは、ここに記載の通りです。なお、割り込みハンドラ開始関数のプロトタイプ宣言は、cfg600pxがkernel_id.hに出力します。

(2) カーネル管理割り込みハンドラでは、非タスクコンテキストから呼び出し可能なサービスコールを使用することができます。一方、カーネル外割り込みハンドラでは、サービスコールを使用することはできません。

(3) ハンドラ開始関数のコンパイル時に、カーネル管理割り込みハンドラの場合は、ハンドラの最後でret_intサービスコールが呼び出されるオブジェクトコードが生成されます。
カーネル管理外割り込みハンドラの場合は、ハンドラの最後でRTE命令でリターンするオブジェクトコードが生成されます。
高速割り込みハンドラの場合は、ハンドラの最後でRTFI命令でリターンするオブジェクトコードが生成されます。

6.5.2 起動時の CPU 状態

割り込みハンドラは、スーパバイザモードで実行されます。

また、割り込み許可状態については、interrupt_vector[].pragma_switch に"E"を指定した場合は、当該割り込み優先レベルで割り込みをマスクした状態になります。

一方、pragma_switch に"E"を指定しない場合、および固定ベクタ割り込み(interrupt_fvector[])は、すべての割り込みを禁止した状態となります。

表6.5 割り込みハンドラ起動時の PSW

No.	ビット	初期値	解説
1	IPL	・割り込み：当該割り込み優先レベル ・CPU 例外：例外発生前と同じ	起動時より下げてはなりません。
2	I	•interrupt_vector[].pragma_switch に"E"を指定した場合：1 ・上記以外：0	
3	PM	0	スーパバイザモード
4	U	0	ISP(割り込みスタックポインタ)
5	C,Z,S,O	不定	
6	その他のビット	0	

6.6 タイムイベントハンドラ(周期ハンドラ、アラームハンドラ)

6.6.1 コーディング

周期ハンドラとアラームハンドラの開始関数のコーディング方法は、どちらも同じです。図 6.5に、タイムイベントハンドラ開始関数のコーディング例を示します。

```
#include "kernel.h"
#include "kernel_id.h"
#pragma cychandler Cychdr1      /* (1) */
void  Cychdr1(VP_INT exinf);    /* (2) */
void  Cychdr1(VP_INT exinf)     /* (3) */
{
    /* 処理 */                  /* (4) */
}                               /* (5) */
```

図6.5 タイムイベントハンドラ開始関数のコーディング例

(1) ハンドラの開始関数に対し、周期ハンドラの場合は#pragma cychandlerディレクティブを、アラームハンドラの場合は#pragma almhandlerディレクティブを記述します。ただし、cfgファイルのcyclic_hand[]およびalarm_hand[]で生成したハンドラの場合は、cfg600pxがkernel_id.hに本ディレクティブを出力するため、不要です。

(2) ハンドラ開始関数のプロトタイプ宣言を行います。ただし、cfgファイルで生成したハンドラの場合は、cfg600pxがkernel_id.hに本宣言を出力するため、不要です。

(3) タイムイベントハンドラの開始関数のAPIは、ここに記載の通りです。exinfにはタイムイベントハンドラの拡張情報が渡ります。

(4) タイムイベントハンドラでは、非タスクコンテキストから呼び出し可能なサービスコールを使用することができます。

(5) タイムイベントハンドラは、カーネル内のシステムクロック割り込みハンドラから関数コールされます。

6.6.2 起動時の CPU 状態

タイムイベントハンドラは、カーネル内のシステムクロック割り込みハンドラのコンテキストで実行されます。

タイムイベントハンドラは、スーパバイザモードで実行されます。

また、割り込み許可状態については、タイマ割り込みレベル(clock.IPL)で割り込みをマスクした状態になります。

表6.6 タイムイベントハンドラ起動時の PSW

No.	ビット	初期値	解説
1	IPL	**タイマ割り込みレベル**	cfg **ファイルの** clock.IPL **に指定した値** **起動時より下げてはなりません。**
2	I	1	
3	PM	0	**スーパバイザモード**
4	U	0	ISP(**割り込みスタックポインタ**)
5	C,Z,S,O	**不定**	
6	**その他のビット**	0	

6.7 アクセス例外ハンドラ

6.7.1 概要

アクセス例外ハンドラは、タスクコンテキストで許可されていないメモリアクセスが発生した時に呼び出されます。

アクセス例外ハンドラはシステムに一つだけ存在します。アクセス例外ハンドラは、必ずユーザが作成しなければなりません。

6.7.2 コーディング

図 6.6に、アクセス例外ハンドラ開始関数のコーディング例を示します。

```
#include "kernel.h"
#include "kernel_id.h"
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr);
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr)
{
    /* 処理 */
}
```

図6.6 アクセス例外ハンドラ開始関数のコーディング例

アクセス例外ハンドラの開始関数の API は、ここに記載の通りです。関数名は"_RI_sys_access_exception"に決められています。

渡される引数の仕様を、表 6.7に示します。

アクセス例外ハンドラでは、非タスクコンテキストから呼び出せるサービスコールを使用できます。

表6.7 アクセス例外ハンドラに渡される引数

No.	引数	レジスタ	解説
1	pc	R1	アクセス例外を発生させた命令アドレス
2	psw	R2	アクセス例外発生時の PSW
3	sts	R3	アクセス違反要因(= MPU(Memory Protection Unit)の MPESTS レジスタ値)
4	addr	R4	オペランドアクセスエラーの場合は、そのアクセスアドレス(=MPU の MPDEA レジスタの値)。 実行アクセスエラーの場合は不定。

6.7.3 起動時の CPU 状態

アクセス例外ハンドラは、スーパバイザモードで実行されます。

表6.8 割り込みハンドラ起動時の PSW

No.	ビット	初期値	解説
1	IPL	アクセス例外発生前と同じ	起動時より下げてはなりません。
2	I	0	全割り込み禁止
3	PM	0	スーパバイザモード
4	U	0	ISP(割り込みスタックポインタ)
5	C,Z,S,O	不定	
6	その他のビット	0	

6.8 システムダウンルーチン

6.8.1 概要

システムダウンルーチンは、システムダウン時に呼び出されるルーチンです。
以下のような事象が発生すると、システムダウンとなります。

- 未定義割り込みの発生
- vsys_dwn, ivsys_dwn サービスコールの発行
- 組み込まれていないサービスコールの発行[4]
- ext_tsk でのコンテキストエラー
- exd_tsk でのコンテキストエラー
- ret_int でのコンテキストエラー
- vsta_knl, ivsta_knl でのエラー

システムダウンルーチンは、必ずユーザが作成しなければなりません。

6.8.2 コーディング

図 6.7に、システムダウンルーチンの開始関数のコーディング例を示します。

```
#include "kernel.h"
#include "kernel_id.h"
void  _RI_sys_dwn__(W type, VW inf1, VW inf2, VW inf3);
void  _RI_sys_dwn__(W type, VW inf1, VW inf2, VW inf3)
{
    /* 処理 */
    while(1);
}
```

図6.7 **システムダウン開始関数のコーディング例**

システムダウンルーチンの開始関数の API は、ここに記載の通りです。関数名は"_RI_sys_dwn__"に決められています。
システムダウンルーチンでは、サービスコールを呼び出してはなりません。
システムダウンルーチンの開始関数からリターンしてはなりません。
渡される引数の仕様を、以下に示します。

(1) type == -1 (ret_int でのエラー)

表6.9 **システムダウンルーチンに渡される引数**(type == -1)

inf1	inf2	inf3	解説
E_CTX	2	**不定**	**タスクコンテキストからの** ret_int **の呼び出し**
	3	**不定**	PSW.IPL > **カーネル割り込みマスクレベルの状態からの** ret_int **の呼び出し**
	5	**不定**	CPU **ロック状態からの** ret_int **の呼出し**
E_MACV	12	**不定**	**割り込まれたタスクのスタックポインタがスタック領域の範囲外を指している**

[4] 「9.4 注意事項」参照

(2) type == -2 (ext_tsk でのエラー)

表6.10 システムダウンルーチンに渡される引数(type == -2)

inf1	inf2	inf3	解説
E_CTX	1	不定	非タスクコンテキストからの ext_tsk の呼出し
	4	不定	PSW.IPL > system.system_IPL の状態からの ext_tsk の呼び出し

(3) type == -3 (組み込まれていないサービスコールの呼出し)

表6.11 システムダウンルーチンに渡される引数(type == -3)

inf1	inf2	inf3	解説
E_NOSPT	不定	不定	組み込まれていないサービスコールの呼出し

(4) type == -4 (タスク例外処理ルーチンからリターン時のエラー)

表6.12 システムダウンルーチンに渡される引数(type == -4)

inf1	inf2	inf3	解説
E_CTX	7	不定	PSW.IPL > system.system_IPL の状態で、タスク例外処理ルーチンからリターン
	8	不定	CPU ロック状態で、タスク例外処理ルーチンからリターン
	9	不定	非タスクコンテキストで、タスク例外処理ルーチンからリターン

(5) type == -5 (exd_tsk でのエラー)

表6.13 システムダウンルーチンに渡される引数(type == -5)

inf1	inf2	inf3	解説
E_CTX	10	不定	PSW.IPL > system.system_IPL の状態からの exd_tsk の呼び出し
	11	不定	非タスクコンテキストからの exd_tsk の呼出し

(6) type == -6 (vsta_knl, ivsta_knl でのエラー)

表6.14 システムダウンルーチンに渡される引数(type == -6)

inf1	inf2	inf3	解説	
E_PAR	15	不定	メモリオブジェクト登録 (memory_object[]) に関するエラー	1. 先頭アドレスが 16 バイト境界でない。 2. acptn1,acptn2,acptn3 いずれかの bit15 に 1 がセットされている。 3.acptn1 == acptn2 == acptn3 == 0 4.acptn1,acptn2,acptn3 に最大ドメイン ID より大きな値のドメインに対応するビットがセットされている。 5.先頭アドレス > 終端アドレス
E_OBJ		不定		先頭アドレスが同じメモリオブジェクトが複数定義されている。
E_OACV		不定		ひとつのドメインにアクセス許可されているメモリオブジェクト数が 7 を超える。
E_PAR	16	不定	タスク生成(task[]) に関するエラー	ユーザスタックの終端アドレス+1 が 16 バイト境界でない。

(7) type == -16 (未定義の可変ベクタ割り込み)

表6.15 システムダウンルーチンに渡される引数(type == -16)

inf1	inf2	inf3
(a) cfg600px で-U オプションを指定しない場合 不定 (b) cfg600px で-U オプションを指定した場合 ベクタ番号	CPU の割り込み処理によって スタックに退避された PC	CPU の割り込み処理によって スタックに退避された PSW

(8) type == -17 (未定義の固定ベクタ割り込み)

表6.16 システムダウンルーチンに渡される引数(type == -17)

inf1	inf2	inf3
(a) cfg600px で-U オプションを指定しない場合 不定 (b) cfg600px で-U オプションを指定した場合 ベクタ番号	CPU の割り込み処理によって スタックに退避された PC	CPU の割り込み処理によって スタックに退避された PSW

(9) type > 0 (アプリケーションからの vsys_dwn, ivsys_dwn の呼出し)

0 または負の type 値はカーネル用に予約されています。アプリケーションから vsys_dwn, ivsys_dwn を呼び出す場合は、正の type 値を使用してください。

表6.17 システムダウンルーチンに渡される引数(type > 0)

inf1	inf2	inf3
vsys_dwn, ivsys_dwn に指定した引数		

6.8.3 起動時の CPU 状態

システムダウンルーチンは、スーパバイザモードで実行されます。

表6.18 システムダウンルーチン起動時の PSW

No.	ビット	初期値	解説
1	IPL	システムダウン発生時と同じ	
2	I	0	全割り込み禁止
3	PM	0	スーパバイザモード
4	U	0	ISP(割り込みスタックポインタ)
5	C,Z,S,O	不定	
6	その他のビット	0	

6.9 浮動小数点演算命令を使用する場合の注意

コンパイラが浮動小数点演算命令を出力するのは-fpu オプション指定時のみです。
アセンブラで-chkfpu オプションを指定すると、浮動小数点演算命令の記述を Warning として検出します。

(1) タスクで浮動小数点演算命令を使用する場合

cfg ファイルの system.context に、FPSW を含む設定を行ってください。
また、FPSW の初期値は 0x00000100 です。必要に応じて FPSW を初期化してください。

(2) ハンドラで浮動小数点演算命令を使用する場合

ハンドラが明示的に FPSW レジスタを保証する必要があります。
また、FPSW の初期値は不定です。図 6.8のようにして、FPSW レジスタの保証と初期化を行ってください。

```
#include <machine.h>    // コンパイラ提供の組み込み関数get_fpsw(), set_fpsw()を使用するために、
                        // machine.hをインクルード
#include "kernel.h"
#include "kernel_id.h"
void handler(void)
{
    unsigned long old_fpsw;    //  FPSWレジスタを退避するための変数を宣言
    old_fpsw = get_fpsw();      // FPSWレジスタを退避
    set_fpsw(0x00000100);       // 必要ならFPSWを初期化
    /* 浮動小数点演算処理 */
    set_fpsw(old_fpsw);         // FPSWレジスタを復帰
}
```

図6.8 浮動小数点演算命令を使用するハンドラの例

6.10 DSP 機能をサポートしたマイコンを使用する場合の注意

DSP 機能をサポートしたマイコンでは、ACC レジスタ(アキュムレータ)の扱いについて注意が必要です。
具体的には、ACC レジスタを更新する下記の命令を使用する場合は、以下に留意してください。

MACHI, MACLO, MULHI, MULLO, RACW, MVTACHI, MVTACLO

なお、コンパイラがこれらの命令を生成することはありません。また、アセンブラで-chkdsp オプションを指定すると、DSP 機能命令の記述を Warning として検出します。

(1) タスクで上記命令を使用する場合

cfg ファイルの system.context に ACC を含む設定を行ってください。

(2) 割り込みハンドラでの ACC レジスタの保証

アプリケーション内に前述の DSP 機能命令を使用するタスクおよび割り込みハンドラがひとつでもある場合は、すべての割り込みハンドラが ACC レジスタを保証する必要があります。その方法として、以下の 3 つがあります。

1. コンパイラのsave_accオプションを使用する。
2. cfgファイルでのすべての割り込みハンドラ定義時に、pragma_switchに"ACC"を指定する。
3. すべての割り込みハンドラが明示的にACCを保証する。
   図6.9のようにして、ACCレジスタを保証してください。

```
#include "kernel.h"
#include "kernel_id.h"

typedef struct {                                        // ACCレジスタ退避用構造体型
    unsigned long upp;   // bit63-32
    unsigned long mid;   // bit47-16
} ST_ACC;

#pragma inline_asm(get_acc)                             // ACCレジスタを退避する関数マクロ
void get_acc(ST_ACC *pk_acc)
{
    mvfachi r5
    mov.l   r5,[r1]
    mvfacmi r5
    mov.l   r5,4[r1]
}

#pragma inline_asm(set_acc)                             // ACCレジスタを復帰する関数マクロ
void set_acc(ST_ACC *pk_acc)
{
    mov.l   [r1],r5
    mvtachi r5
    mov.l   4[r1],r5
    shll    #16,r5
    mvtaclo r5
}
void handler(void)             // ハンドラ関数
{
    ST_ACC st_acc;             //  ACCレジスタを退避するための変数を宣言
    get_acc(&st_acc);          // ACCレジスタを退避
     /*  処理 */
    set_acc(&st_acc);          // ACCレジスタを復帰
}
```

図6.9　ACC レジスタを保証するハンドラの例

7. ロードモジュール生成手順

7.1 概要

RI600/PX のアプリケーションプログラムは、一般に以下に示す手順で開発します。

(1) プロジェクトの生成

High-performance Embedded Workshop を使用する場合は、High-performance Embedded Workshop 上で RI600/PX を使用したプロジェクトを新規に作成します。

(2) アプリケーションプログラムのコーディング

アプリケーションプログラムをコーディングします。必要に応じてサンプルのスタートアッププログラムを修正してください。

(3) コンフィギュレーションファイル(cfg ファイル)の作成

タスクのエントリーアドレスやスタックサイズなどを定義した cfg ファイルを、テキストエディタなどで作成します。

(4) コマンドラインコンフィギュレータ(cfg600px)の実行

cfg600px は、cfg ファイルを読み込み、システムデータ定義ファイル(kernel_rom.h,kernel_ram.h など)と、アプリケーション用インクルードファイル(kernel_id.h など)を生成します。

(5) ロードモジュール生成

make コマンド、もしくは High-performance Embedded Workshop 上でビルドを実行してロードモジュールを生成します。

図 7.1に、ロードモジュール生成フローを示します。

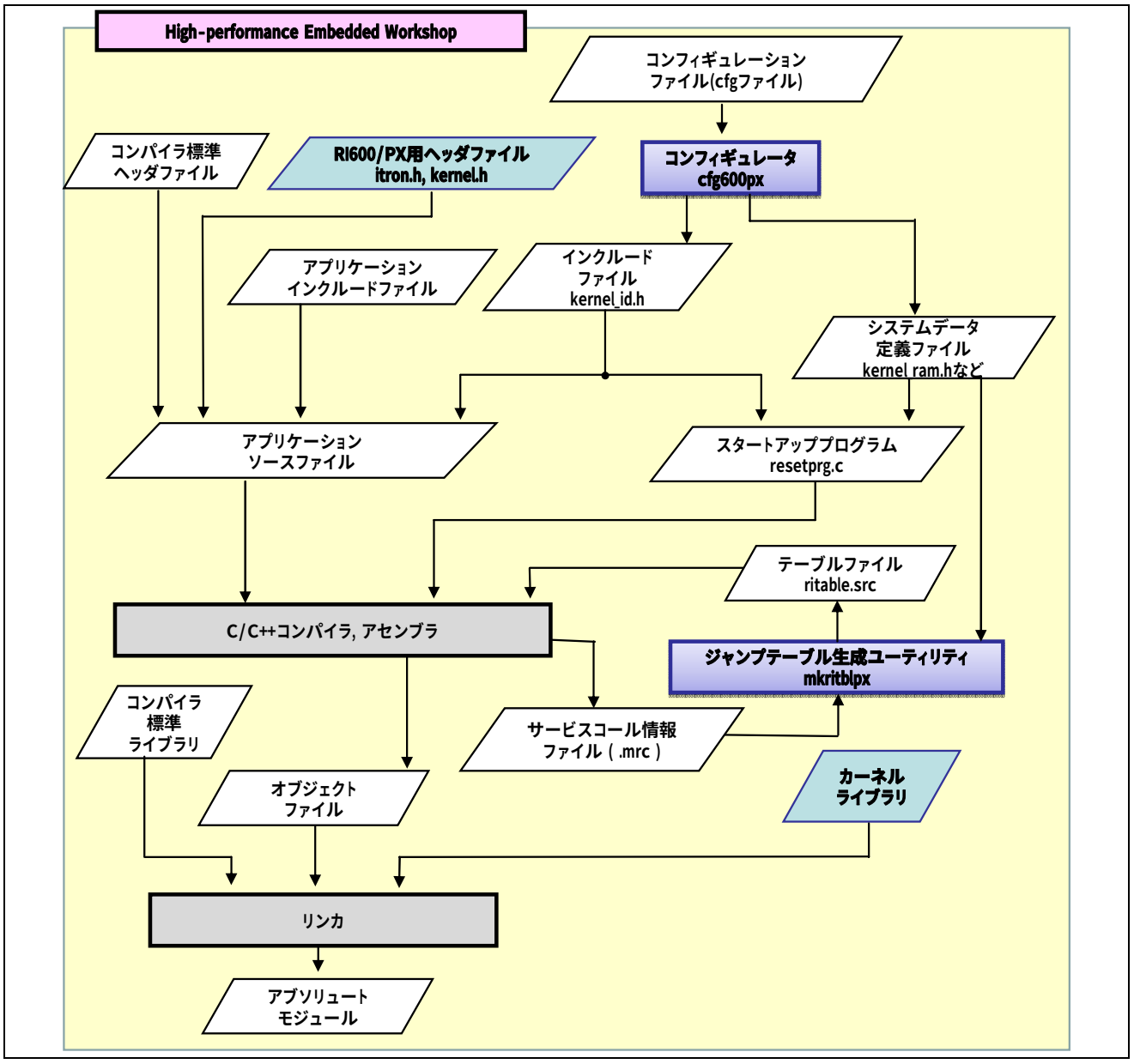

図7.1　**ロードモジュール生成フロー**

7.2 スタートアップファイル(resetprg.c)の作成

スタートアップファイル(ファイル名は任意ですが、通常は resetprg.c とします)には、以下を記述します。

(1) スタートアッププログラム(PowerON_Reset_PC())
(2) タイマ初期化コールバック関数(_RI_init_cmt_knl())
(3) アクセス例外ハンドラ(_RI_sys_acccess_exception())
(4) システムダウンルーチン(_RI_sys_dwn__())
(5) kernel_rom.hおよびkernel_ram.hの取り込み

(1) スタートアッププログラム(PowerON_Reset_PC())

スタートアッププログラムとは、CPU のリセットベクタに登録されるプログラムで、スーパバイザモードで実行します。通常は、以下のような処理を行います。

- プロセッサ・ハードウェアの初期化
  高速割り込みを使用する場合は、FINTV レジスタを高速割り込みハンドラの開始アドレスに初期化してください。
- C/C++言語ソフトウェアの実行環境の初期化(セクションの初期化など)
- カーネルを起動(vsta_knl, ivsta_knl の呼び出し)

リセットからカーネル起動までは、割り込みを受理しない状態を維持してください。リセット直後は、PSW.I=0 のため、PSW を変更しなければこの条件は満たされます。

通常、スタートアッププログラムの関数は"void PowerON_Reset_PC(void)"としてください。これ以外の関数名とする場合は、cfg ファイルで interrupt_fvector[31]にその関数名を定義する必要があります。

また、スタートアッププログラム関数には、#pragma entry ディレクティブが必要です。この宣言と、cfg600px が kernel_ram.h に出力する#pragma stacksize ディレクティブにより、コンパイラによって関数先頭でスタックポインタ(ISP)をシステムスタック(SI セクション)に初期化するコードが生成されます。

(2) タイマ初期化コールバック関数(_RI_init_cmt_knl())

本関数は、vsta_knl, ivsta_knl から呼び出されます。

cfg ファイルで clock.timer に CMT0, CMT1, CMT2, CMT3 のいずれかを指定した場合は、本関数から cfg600px が出力する ri_cmt.h に定義されているインライン関数"void _RI_init_cmt(void)"を呼び出すようにしてください。

cfg ファイルで clock.timer に NOTIMER を指定した場合は、ただちにリターンしてください。

なお、本関数は全割り込みを禁止した状態で呼び出されます。この状態を維持するようにしてください。

(3) アクセス例外ハンドラ(_RI_sys_acccess_exception())

「6.7 アクセス例外ハンドラ」を参照してください。

(4) システムダウンルーチン(_RI_sys_dwn__())

「6.8 システムダウンルーチン」を参照してください。

(5) kernel_rom.h, kernel_ram.h の取り込み

kernel_rom.h および kernel_ram.h は cfg600px が生成するシステム定義ファイルで、各種データ領域などの定義文が含まれます。

この 2 つのファイルは、必ず resetprg.c から以下の順序でインクルードしてください。

```
#include "kernel.h"        /* provided by RI600/PX */
#include "kernel_id.h"     /* generated by cfg600px */
#include "kernel_ram.h"    /* generated by cfg600px */
#include "kernel_rom.h"    /* generated by cfg600px */
```

(6) コンパイルオプション

コンパイルオプションとして-nostuff を指定する必要があります。

(7) スタートアップファイルの例

```
1.  #include    <machine.h>
2.  #include    <_h_c_lib.h>
3.  //#include  <stddef.h>                  // Remove the comment when you use errno
4.  //#include  <stdlib.h>                  // Remove the comment when you use rand()
5.  #include    "typedefine.h"              // Define Types
6.  #include    "kernel.h"                  // Provided by RI600/PX
7.  #include    "kernel_id.h"               // Generated by cfg600px
8.
9.  #if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
10. #include    "ri_cmt.h"                  // Generated by cfg600px
11.                                         // Do comment-out when clock.timer is either NOTIMER or OTHER.
12. #endif
13.
14. #ifdef __cplusplus
15. extern "C" {
16. #endif
17. void PowerON_Reset_PC(void);
18. void _RI_init_cmt_knl(void);
19. void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr);
20. void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 );
21. #ifdef __cplusplus
22. }
23. #endif
24.
25. // #ifdef __cplusplus               // Use SIM I/O
26. // extern "C" {
27. // #endif
28. // extern void _INIT_IOLIB(void);
29. // extern void _CLOSEALL(void);
30. // #ifdef __cplusplus
31. // }
32. // #endif
33.
34. #define FPSW_init 0x00000000  // FPSW bit base pattern
35.
36. // extern void srand(_UINT);     // Remove the comment when you use rand()
37. // extern _SBYTE *_s1ptr;        // Remove the comment when you use strtok()
38.
39. // #ifdef __cplusplus               // Use Hardware Setup
40. // extern "C" {
41. // #endif
42. // extern void HardwareSetup(void);
43. // #ifdef __cplusplus
44. // }
45. // #endif
46.
47. // #ifdef __cplusplus            // Remove the comment when you use global class object
48. // extern "C" {                  // Sections C$INIT and C$END will be generated
49. // #endif
50. // extern void _CALL_INIT(void);
51. // extern void _CALL_END(void);
52. // #ifdef __cplusplus
53. // }
54. // #endif
55.
56. ////////////////////////////////////////////////////////
```

```
57.   // Section definition
58.   ////////////////////////////////////////////////////////
59.   #pragma section P PS
60.   #pragma section B BS
61.   #pragma section C CS
62.   #pragma section D DS
63.
64.   #pragma entry PowerON_Reset_PC
65.
66.   ////////////////////////////////////////////////////////
67.   // Power-on Reset Program
68.   ////////////////////////////////////////////////////////
69.   void PowerON_Reset_PC(void)
70.   {
71.   #ifdef __ROZ                                // Initialize FPSW
72.   #define _ROUND 0x00000001                   // Let FPSW RMbits=01 (round to zero)
73.   #else
74.   #define _ROUND 0x00000000                   // Let FPSW RMbits=00 (round to nearest)
75.   #endif
76.   #ifdef __DOFF
77.   #define _DENOM 0x00000100                   // Let FPSW DNbit=1 (denormal as zero)
78.   #else
79.   #define _DENOM 0x00000000                   // Let FPSW DNbit=0 (denormal as is)
80.   #endif
81.       set_fpsw(FPSW_init | _ROUND | _DENOM);
82.   #endif
83.
84.       _INITSCT();                     // Initialize Sections
85.
86.   //  _INIT_IOLIB();                  // Use SIM I/O
87.
88.   //  errno=0;                        // Remove the comment when you use errno
89.   //  srand((_UINT)1);                // Remove the comment when you use rand()
90.   //  _s1ptr=NULL;                    // Remove the comment when you use strtok()
91.
92.   //  HardwareSetup();                // Use Hardware Setup
93.       nop();
94.
95.   //  set_fintv(<handler address>);   // Initialize FINTV register
96.
97.   //  _CALL_INIT();                   // Remove the comment when you use global class object
98.
99.       vsta_knl();                     // Start RI600/PX
100.                                      // Never return from vsta_knl
101.
102.  //  _CLOSEALL();                    // Use SIM I/O
103.
104.  //  _CALL_END();                    // Remove the comment when you use global class object
105.
106.      brk();
107.
108.  }
109.
110.  ////////////////////////////////////////////////////////
111.  // Timer initialize call-back
112.  ////////////////////////////////////////////////////////
113.  void _RI_init_cmt_knl(void)
114.  {
```

```
115.  #if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))  // Do comment-out when clock.timer is either NOTIMER or OTHER.
116.      _RI_init_cmt();
117.  #endif
118.  }
119.
120.  ////////////////////////////////////////////////////////
121.  // Access exception handler
122.  ////////////////////////////////////////////////////////
123.  void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr)
124.  {
125.      // Now PSW.I=0 (all interrupts are masked.)
126.
127.      ID  hTaskID;
128.
129.      iget_tid(&hTaskID);
130.      iras_tex(hTaskID, 1);
131.  }
132.
133.  ////////////////////////////////////////////////////////
134.  // System-down routine for RI600/PX
135.  ////////////////////////////////////////////////////////
136.  struct SYSDWN_INF{
137.      W   type;
138.      VW  inf1;
139.      VW  inf2;
140.      VW  inf3;
141.  };
142.
143.  volatile struct SYSDWN_INF _RI_sysdwn_inf;
144.
145.  void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 )
146.  {
147.      // Now PSW.I=0 (all interrupts are masked.)
148.
149.      _RI_sysdwn_inf.type = type;
150.      _RI_sysdwn_inf.inf1 = inf1;
151.      _RI_sysdwn_inf.inf2 = inf2;
152.      _RI_sysdwn_inf.inf3 = inf3;
153.
154.      while(1)
155.      ;
156.  }
157.
158.  ////////////////////////////////////////////////////////////////////////
159.  // RI600/PX system data
160.  ////////////////////////////////////////////////////////////////////////
161.  #include "kernel_ram.h"     // generated by cfg600px
162.  #include "kernel_rom.h"     // generated by cfg600px
```

7.3 カーネルライブラリ

カーネルライブラリには、リトルエンディアン用の ri600lit.lib とビッグエンディアン用の ri600big.lib があります。これらは、"インストールディレクトリ¥lib600"ディレクトリにあります。

マイコンのエンディアン種別に応じて、いずれかのライブラリを使用してください。

7.4 セクション

7.4.1 セクション名の命名規則

通常は、メモリオブジェクトごとにユニークなセクション名を付与します。これを容易にするため、以下のようなセクション命名規則を定めることを推奨します。

(1) 1 文字目：セクション種別

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

(2) 2 文字目以降

- RI*：RI600/PX 予約
  この領域は、ユーザモード(=タスクコンテキスト)からアクセスされません。
- U*：メモリオブジェクトまたはユーザスタック
  この領域は、ユーザモード(=タスクコンテキスト)からアクセスされます。
- S*：上記以外
  この領域は、ユーザモード(=タスクコンテキスト)からアクセスされません。

7.4.2 RI600/PX のセクション一覧

表7.1 RI600/PX のセクション一覧

セクション	説明	属性	アライメント数
PRI_KERNEL	カーネルプログラム	CODE	1
CRI_ROM	カーネル定数	ROMDATA	4
DRI_ROM	カーネル初期化データ	ROMDATA	4
RRI_RAM	カーネル初期化データ用変数	DATA	4
FIX_INTERRUPT_VECTOR	固定割り込みベクタテーブル 0xFFFFFF80 番地に配置する必要があります。	ROMDATA	4
INTERRUPT_VECTOR	可変割り込みベクタテーブル	ROMDATA	4
SI	システムスタック	DATA	4
SURI_STACK	タスクのユーザスタックに付与するセクション名は、cfg ファイルで指定することができます。これを省略した場合は、スタックのセクション名が SURI_STACK となります。	DATA	4
BRI_RAM	カーネルの変数セクションです。 また、メッセージバッファ領域に付与するセクション名は cfg ファイルで指定することができます。これを省略した場合はセクション名が BRI_RAM となります。	DATA	4
BURI_HEAP	可変長メモリプール、固定長メモリプール領域に付与するセクション名は、cfg ファイルで指定することができます。 これを省略した場合はその領域のセクション名が BURI_HEAP となります。] 本セクションは、必要に応じてメモリオブジェクトに含めても良いです。	DATA	4

7.4.3 リンク時の aligned_section オプション

以下のセクションは、リンク時に aligned_section オプションを指定する必要があります。

1. memory_object[].start_addressに指定したセクション
2. task[].stack_sectionに指定したセクション
3. SURI_STACKセクション

7.4.4 L, W セクションに関する注意

L セクションはリテラル領域、W セクションは swtich 文分岐テーブル領域です。これらはコンパイラによって生成されます。

これらのセクションは、#pragma section ディレクティブでセクション名を変更することはできません。

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

(1) ソースファイル中の全関数が、同一ドメインに所属するタスクとしてのみ実行する場合

特に留意事項はありません。L, W セクションを、そのドメインからリードアクセス可能なメモリオブジェクトとしてください。

(2) ソースファイル中の関数が、複数のドメインに所属するタスクとして実行する場合

関数ごとのリテラル・分岐テーブル領域に異なるセクション名を付与することはできないため、それらを異なるアクセス許可を持つ個別のメモリオブジェクトに分離することはできません。したがって、実行時のドメインごとに関数を別ファイル化して(1)を適用するか、または L, W セクションを全てのドメインからリードアクセス可能なメモリオブジェクトとしてください。

7.5 サービスコール情報ファイル(mrc ファイル)と必須コンパイラオプション

サービスコール情報ファイル(mrc ファイル)は、kernel.h をインクルードするファイルのコンパイルによって生成されます。

mrc ファイルには、ソース中で使用しているサービスコール名が出力されます。mrc ファイルは、mkritblpx への入力となります。

kernel.h をインクルードするファイルのコンパイル時には、"-ri600_preinit_mrc"オプションを指定してください。本オプションを指定しなくても動作に問題は生じませんが、アプリケーションで使用していないサービスコールモジュールがリンクされる場合があります。

アプリケーションをライブラリ化する場合は、コンパイル時に生成された mrc ファイルも mkritblpx に入力してください。これが難しい場合、カーネルとリンクしないアプリケーションプログラムで使用しているサービスコールを組み込むには、使用するサービスコール名を羅列した mrc ファイル(下記参照)を作成し、mkritblpx に入力してください。

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

```
sta_tsk
snd_mbx
rcv_mbx
prcv_mbx
```

7.6 注意事項

7.6.1 プロセッサモード

タスクとタスク例外処理ルーチンはユーザモード(PSW.PM=1)、ハンドラやカーネルはスーパバイザモード(PSW.PM=0)で実行します。

ユーザモードで特権命令を実行すると、CPU は特権命令例外を検出します。ユーザモードで実行するプログラムでは、特権命令を使用しないように注意してください。

なお、アセンブラには特権命令の記述をワーニングとする chkpm オプションがあるので、必要に応じて利用してください。

7.6.2 0 番地

メモリオブジェクトの先頭アドレスを除き、サービスコールに渡すポインタパラメータに 0 を指定するとエラーになります。

したがって、以下のセクションを 0 番地に配置してはなりません。

1. BURI_STACK
2. BURI_HEAP
3. task[].stack_section
4. message_buffer[].mbf_section
5. memorypool[].section
6. variable_memorypool[].mpl_section

8. コンフィギュレータ(cfg600px)

8.1 コンフィギュレーションファイル(cfg ファイル)の作成方法

アプリケーションプログラムで使用するOSの資源は、RI600/PXシステムに登録する必要があります。これを設定するのがコンフィギュレーションファイル(cfgファイル)であり、システムに登録するためのツールがコンフィギュレータ(cfg600px)です。

cfg600px は、コンフィギュレーションファイル(cfg ファイル)で定義した内容を元に、カーネル構築用のファイルを生成するツールです。

8.2 cfg ファイル内の表現形式

この節では cfg ファイル内における定義データの表現形式について説明します。

(1) コメント文

"//"から行の終わりまではコメント文とみなし、処理の対象になりません。

(2) 文の終わり

';'で文を終ります。

(3) 数値

数値は以下の形式で入力できます。

- 16 進数
  数値の先頭に"0x"か"0X"を付加します。または、数値の最後に'h'か'H'を付加します。後者の場合でかつ先頭が英文字 (A～F)で始まる場合は先頭に必ず'0'を付加してください。なおここで使用する数値表現で英文字 (A～F)は大文字・小文字を識別しません。[5]
- 10 進数
  23 のように整数のみで表します。ただし'0'で始めることはできません。
- 8 進数
  数値の先頭に'0'を付加するか数値の最後に'O'もしくは'o'を付加します。
- 2 進数
  数値の最後に'B'または'b'を付加します。ただし'0'で始めることはできません。

[5] 数値表現内の’A’～’F’,’a’～’f’を除いて全ての文字は、大文字・小文字の区別を行います。

表8.1　数値表現例

16 **進数**	0xf12 0Xf12 0a12h 0a12H 12h 12H
10 **進数**	32
8 **進数**	017 17o 17O
2 **進数**	101110b 101010B

また数値内に演算子を記述できます。使用できる演算子を表 8.2に示します。

表8.2　演算子

演算子	優先度	演算方向
()	**高**	**左から右**
－(**単項マイナス**)		**右から左**
* / %		**左から右**
+ －(**二項マイナス**)	**低**	**左から右**

数値の記述例を以下に示します。

- 123
- 123 + 0x23
- (23/4 + 3) * 2
- 100B + 0aH

0xFFFFFFFF を超える数値を指定してはなりません。

(4)　シンボル

シンボルは数字、英大文字、英小文字、'_'（アンダースコア）より構成される数字以外の文字で始まる文字列で表されます。

シンボルの記述例を以下に示します。

- _TASK1
- IDLE3

(5) 関数名

関数名は、数字、英大文字、英小文字、'_' (アンダースコア)、'$'(ドル記号)より構成される数字以外の文字で始まり、"()"で終わる文字列で表されます。

C 言語関数名の記述例を以下に示します。

- main()
- func()

アセンブリ言語で記述したモジュールを指定する場合は、その先頭ラベルを"_で始まるように命名し、"_"を除いたものを関数名として指定してください。

(6) 周波数

周波数は、数字と'.'(ピリオド) から構成され'MHz'で終わる文字列で表されます。小数点以下は 6 桁が有効です。なお周波数は 10 進数のみで記述可能です。

周波数の記述例を以下に示します。

- 16MHz
- 8.1234MHz

なお、周波数は'.'で始まってはなりません。

8.3 デフォルト cfg ファイル

多くの定義項目では、ユーザが記述を省略した場合にデフォルト cfg ファイルの内容が補われます。デフォルト cfg ファイルは、環境変数 LIB600 で指定されるディレクトリにあります。なお、このファイルを編集してはなりません。

8.4 cfg ファイルの定義項目

cfg ファイルでは以下の項目の定義をおこないます。

- システム定義(system)
- システムクロック定義(clock)
- 最大 ID 定義(maxdefine)
- ドメイン定義(domain[])
- メモリオブジェクト定義(memory_object[])
- タスク定義(task[])
- セマフォ定義(semaphore[])
- イベントフラグ定義(flag[])
- データキュー定義(dataqueue[])
- メールボックス定義(mailbox[])
- ミューテックス定義(mutex[])
- メッセージバッファ定義(message_buffer[])
- 固定長メモリプール定義(memorypool[])
- 可変長メモリプール定義(variable_memorypool[])
- 周期ハンドラ定義(cyclic_hand[])
- アラームハンドラ定義(alarm_hand[])
- 可変割り込みベクタ定義(interrupt_vector[])
- 固定割り込みベクタ定義(interrupt_fvector[])

8.4.1 システム定義(system)

ここでは、カーネルシステム全般に関連する情報を定義します。system は省略できません。

■形式

```
system {
        stack_size          = (1)システムスタックサイズ;
        priority            = (2)タスク優先度の最大値;
        system_IPL          = (3)カーネル割り込みマスクレベル;
        message_pri         = (4)メッセージ優先度の最大値;
        tic_deno            = (5)タイムティック分母;
        tic_nume            = (6)タイムティック分子;
        context             = (7)タスクコンテキストレジスタ;
};
```

■内容

(1) システムスタックサイズ(stack_size)

【説明】サービスコール処理および割り込み処理で使用するスタックサイズの合計を定義します。
【定義形式】数値
【定義範囲】8以上の4の倍数
【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0x800)を適用

(2) タスク優先度の最大値(priority)

【説明】アプリケーションで使用するタスク優先度の最大値を定義します。
【定義形式】数値
【定義範囲】1～255
【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は32)を適用

(3) カーネル割り込みマスクレベル(system_IPL)

【説明】カーネルのクリティカルセクション実行時の割り込みマスクレベル(PSWレジスタのIPL値)を定義します。これよりも高いレベルの割り込みは、「カーネル管理外割り込み」の扱いとなります。
【定義形式】数値
【定義範囲】1～15
【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は7)を適用

(4) メッセージ優先度の最大値(message_pri)

【説明】メールボックス機能で使用するメッセージの優先度の最大値を定義します。
なお、メールボックス機能を使用しない場合は、本項目は意味を持ちません。
【定義形式】数値
【定義範囲】1～255
【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は255)を適用

(5) タイムティックの分母(tic_deno)

【説明】タイムティックの分母を定義します。タイムティック分子と分母の少なくとも一方は１でなければなりません。
タイムティック時間(カーネルタイマの割り込み周期)は、次の式で算出されます。

タイムティック時間(単位：ミリ秒) = tic_nume / tic_deno

tic_nume, tic_deno の設定に関わらず、サービスコールで扱う時間の単位は常にミリ秒になります。tic_nume, tic_deno によって、カーネルが管理する時間の精度を規定することになります。

【定義形式】数値

【定義範囲】1～100

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

(6) タイムティックの分子(tic_nume)

【説明】タイムティックの分子を定義します。詳細は、前項を参照してください。

【定義形式】数値

【定義範囲】1～65535

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

(7) タスクコンテキストレジスタ(context)

【説明】タスクおよびタスク例外処理ルーチンが使用するレジスタセットを定義します。ここでの設定は、全タスクおよびタスク例外処理ルーチンに適用されます。

【定義形式】シンボル

【定義範囲】表8.3のいずれかから選択してください。

表8.3 system.context

設定値	タスクのコンテキストとして保証されるレジスタ			
	CPU		FPU	DSP
	PSW, PC, R0～R7, R14, R15	R8～R13	FPSW	ACC
NO	保証する	保証する	保証しない	保証しない
FPSW	保証する	保証する	保証する	保証しない
ACC	保証する	保証する	保証しない	保証する
FPSW,ACC	保証する	保証する	保証する	保証する
MIN	保証する	保証しない	保証しない	保証しない
MIN,FPSW	保証する	保証しない	保証する	保証しない
MIN,ACC	保証する	保証しない	保証しない	保証する
MIN,FPSW,ACC	保証する	保証しない	保証する	保証する

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はNO)を適用

【備考】system.contextについては、必ず「8.4.2 system.contextの注意事項」も参照してください。

8.4.2 system.context の注意事項

(1) FPU, ACC(アキュムレータ)に関する注意事項

system.context に設定すべき値は、FPU、DSP をどのように扱うかによって異なります。
以降の説明に従って、system.context を適切に設定してください。推奨以外の設定にした場合は、推奨設定に比べてわずかにカーネルの性能が悪化します。

【備考】コンパイラが浮動小数点演算命令を出力するのは、-cpu=rx600かつ-fpuオプション指定時のみです。
アセンブラで-chkfpuオプションを指定すると、浮動小数点演算命令の記述をWarningとして検出します。
また、コンパイラが DSP 機能命令を出力することはありません。アセンブラで-chkdsp オプションを指定すると、DSP 機能命令の記述を Warning として検出します。

(a) FPU および DSP(アキュムレータ)を搭載したマイコンを使用する場合

表8.4 FPU および DSP(アキュムレータ)を搭載したマイコンを使用する場合

タスク・タスク例外処理ルーチンの命令使用状況		system.context の推奨値
浮動小数点演算命令	DSP 機能命令	
あり	あり	FPSW と ACC を含む設定が必須
あり	なし	FPSW を含む設定が必須で、かつ ACC を含まない設定を推奨
なし	あり	ACC を含む設定が必須で、かつ FPSW を含まない設定を推奨
なし	なし	FPSW, ACC を含まない設定を推奨

(b) FPU を搭載し、かつ DSP(アキュムレータ)を搭載しないマイコンを使用する場合

表8.5 FPU を搭載し、かつ DSP(アキュムレータ)を搭載しないマイコンを使用する場合

タスク・タスク例外処理ルーチンの命令使用状況		system.context の推奨値
浮動小数点演算命令	DSP 機能命令	
あり	あり	(DSP を搭載しないマイコンのため、DSP 機能命令は使用できません)
あり	なし	FPSW を含み、かつ ACC を含まない設定が必須
なし	あり	(DSP を搭載しないマイコンのため、DSP 機能命令は使用できません)
なし	なし	ACC を含まない設定が必須で、かつ FPSW を含まない設定を推奨

(c) FPU を搭載せず、かつ DSP(アキュムレータ)を搭載するマイコンを使用する場合

表8.6 FPU を搭載せず、かつ DSP(アキュムレータ)を搭載するマイコンを使用する場合

タスク・タスク例外処理ルーチンの命令使用状況		system.context の推奨値
浮動小数点演算命令	DSP 機能命令	
あり	あり	(FPU を搭載しないマイコンのため、浮動小数点演算命令は使用できません)
あり	なし	
なし	あり	FPSW を含まず、かつ ACC を含む設定が必須
なし	なし	FPSW を含まない設定が必須で、かつ ACC を含まない設定を推奨

(d) FPU も DSP(アキュムレータ)も搭載しないマイコンを使用する場合

表8.7 FPU も DSP(アキュムレータ)も搭載しないマイコンを使用する場合

タスク・タスク例外処理ルーチンの命令使用状況		system.context の推奨値
浮動小数点演算命令	DSP 機能命令	
あり	あり	(FPU および DSP を搭載しないマイコンでは、浮動小数点演算命令および DSP 機能命令は使用できません)
あり	なし	
なし	あり	
なし	なし	FPSW と ACC を含まない設定が必須

(2) コンパイラの fint_register, base, pid オプションとの関係

system.context では、"MIN", "MIN,ACC", "MIN,FPSW", "MIN,ACC,FPSW"のいずれかを選択することで、カーネルは R8～R13 をタスクコンテキストとして保存しないようになり、より高速化が図れるようになっています。

system.context にこれらの設定をしても良いケースは、コンパイラの fint_register, base および pid オプションで R8～R13 をすべて使用する指定をした場合のみです。例えば、以下のようなオプション指定の場合です。

良い例：

例 1：-fint_register=4 -base=rom=R8 -base=ram=R9

例 2：-fint_register=3 -base=rom=R8 -base=ram=R9 -base=0x80000=R10

その他の場合で system.context に前述の設定をした場合、カーネルは正常に動作しません。例えば、以下のようなオプション指定の場合です。

悪い例：

例 3：fint_register, base オプションの指定なし

例 4：-fint_register=4

例 5：-base=rom=R8 -base=ram=R9

例 6：-fint_register=3 -base=rom=R8 -base=ram=R9

8.4.3 システムクロック定義(clock)

システムクロックに関する定義を行います。

■形式

```
clock {
        timer               = (1)システムタイマの選択;
        template            = (2)テンプレートファイル;
        timer_clock         = (3)システムタイマの周波数;
        IPL                 = (4)タイマ割り込み優先レベル;
};
```

■内容

(1) システムタイマの選択(timer)

【説明】システムクロックに使用するハードウェアタイマを定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。CMT0, CMT1, CMT2, CMT3のいずれかを指定した場合は、cfg600pxがタイマドライバソースコード(ri_cmt.h)を生成します。

- CMT0：マイコン内蔵の CMT チャネル 0 を使用する。
- CMT1：マイコン内蔵の CMT チャネル 1 を使用する。
- CMT2：マイコン内蔵の CMT チャネル 2 を使用する。
- CMT3：マイコン内蔵の CMT チャネル 3 を使用する。
- OTHER：上記以外のタイマを使用する。この場合、ユーザがタイマ初期化ルーチンを作成する必要があります。
- NOTIMER：システムクロックを使用しない。

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はCMT0)を適用

(2) テンプレート(template)

【説明】CMT(コンペアマッチタイマ)のハードウェア情報および初期化関数が定義されたテンプレートファイルを指定します。

timerにNOTIMERまたはOTHERを指定した場合、本指定は単に無視されます。テンプレートファイルは、RI600/PXによって提供されます。テンプレートファイルは、今後のリビジョンアップで追加される場合があります。提供されるテンプレートと対応マイコンの関係は、製品添付のリリースノートを参照してください。

テンプレートによっては、CMT1, CMT2, CMT3のいずれかがサポートされない場合があります。timerにそのテンプレートで未対応のCMTチャネルを指定した場合、cfg600pxはエラーを報告しませんが、cfg600pxが出力するri_cmt.hをインクルードするファイル(通常はresetprg.c)のコンパイル時にエラーになります。

【定義形式】シンボル

【定義範囲】―

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はrx630.tpl)を適用

(3) システムタイマの周波数(timer_clock)

【説明】システムタイマに供給されるクロックの周波数を定義します。timerにCMT0, CMT1, CMT2またはCMT3を選択した場合は、PCLK(周辺モジュールクロック)の周波数を指定してください。

【定義形式】周波数

【定義範囲】－

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は25MHz)を適用

(4) タイマ割り込み優先レベル(IPL)

【説明】システムタイマの割り込み優先レベルを定義します。
システムタイマの割り込みハンドラ処理中は、ここで定義した割り込みレベルより低いレベルの割り込みは受け付けられません。

【定義形式】数値

【定義範囲】1～system.system_IPL

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は4)を適用

(5) timer=OTHER 指定時の留意事項

以下の対処が必要です。

1. カーネル起動(vsta_knl)前に、タイマを初期化してください。
cfg600pxは、以下のマクロをkernel_id.hに出力します。この情報を元に、タイマを初期化してください。
    - _RI_CLOCK_IPL：タイマ割り込み優先レベル(clock.IPL)
    - TIC_DENO：タイムティック周期の分母(system.tic_deno)
    - TIC_NUME：タイムティック周期の分子(system.tic_nume)
2. 以下のように可変割り込みベクタを定義してください。

```
interrupt_vector[<ベクタ番号>] {
    entry_address = __RI_SYS_STMR_INH;
    os_int = YES;
};
```

8.4.4 最大 ID 定義(maxdefine)

maxdefine は、各オブジェクトの使用可能な最大 ID を定義します。また、オブジェクトを動的に生成するサービスコールを使用する場合は、対応するオブジェクトについて本定義を行う必要があります。

各オブジェクトの最大 ID を定義したマクロが kernel_id.h に出力されます(「4.2.1 定数マクロ参照」)。

■形式

```
maxdefine {
        max_task            = (1)最大タスク ID;
        max_sem             = (2)最大セマフォ ID;
        max_flag            = (3)最大イベントフラグ ID;
        max_dtq             = (4)最大データキューID;
        max_mbx             = (5)最大メールボックス ID;
        max_mtx             = (6)最大ミューテックス ID;
        max_mbf             = (7)最大メッセージバッファ ID;
        max_mpf             = (8)最大固定長メモリプール ID;
        max_mpl             = (9)最大可変長メモリプール ID;
        max_cyh             = (10)最大周期ハンドラ ID;
        max_alh             = (11)最大アラームハンドラ ID;
        max_domain          = (12)最大ドメイン ID;
};
```

■内容

(1) 最大タスク ID(max_task)

【説明】本定義により、cre_tsk, acre_tsk, del_tsk, exd_tsk, def_texを使用できます。使用できるタスクIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_task, task[]で指定された ID 番号, task[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_tsk, acre_tsk, del_tsk, def_texはE_NOSPTを返します。また、exd_tskはシステムダウンとなります。

使用できるタスクIDの範囲は以下の通りです。

- 最小値：1
- 最大値：task[]で指定された ID 番号, task[]定義数の中の最大値

(2) 最大セマフォ ID (max_sem)

【説明】本定義により、cre_sem, acre_sem, del_semを使用できます。使用できるセマフォIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_sem, semaphore[]で指定された ID 番号, semaphore[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_sem, acre_sem, del_semはE_NOSPTを返します
使用できるセマフォIDの範囲は以下の通りです。

- 最小値：1
- 最大値：semaphore[]で指定された ID 番号, semaphore []定義数の中の最大値

(3) 最大イベントフラグ ID (max_flag)

【説明】本定義により、cre_flg, acre_flg, del_flgを使用できます。使用できるイベントフラグIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_flag, flag[]で指定された ID 番号, flag[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_flg, acre_flg, del_flgはE_NOSPTを返します。
使用できるイベントフラグIDの範囲は以下の通りです。

- 最小値：1
- 最大値：flag[]で指定された ID 番号, flag []定義数の中の最大値

(4) 最大データキューID (max_dtq)

【説明】本定義により、cre_dtq, acre_dtq, del_dtqを使用できます。使用できるデータキューIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_dtq, data_queue[]で指定された ID 番号, data_queue[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_dtq, acre_dtq, del_dtqはE_NOSPTを返します。
使用できるデータキューIDの範囲は以下の通りです。

- 最小値：1
- 最大値：data_queue[]で指定された ID 番号, data_queue[]定義数の中の最大値

(5) 最大メールボックス ID (max_mbx)

【説明】本定義により、cre_mbx, acre_mbx, del_mbxを使用できます。使用できるメールボックスIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_mbx, mailbox[]で指定された ID 番号, mailbox[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_mbx, acre_mbx, del_mbxはE_NOSPTを返します。
使用できるメールボックスIDの範囲は以下の通りです。

- 最小値：1
- 最大値：mailbox[]で指定された ID 番号, mailbox[]定義数の中の最大値

(6) 最大ミューテックス ID (max_mtx)

【説明】本定義により、cre_mtx, acre_mtx, del_mtxを使用できます。使用できるミューテックスIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_mtx, mutex[]で指定された ID 番号, mutex[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_mtx, acre_mtx, del_mtxはE_NOSPTを返します。
使用できるミューテックスIDの範囲は以下の通りです。

- 最小値：1
- 最大値：mutex[]で指定された ID 番号, mutex[]定義数の中の最大値

(7) 最大メッセージバッファ ID (max_mbf)

【説明】定義により、cre_mbf, acre_mbf, del_mbfを使用できます。使用できるメッセージバッファIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_mbf, message_buffer[]で指定された ID 番号, message_buffer[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_mbf, acre_mbf, del_mbfはE_NOSPTを返します。
使用できるメッセージバッファIDの範囲は以下の通りです。

- 最小値：1
- 最大値：message_buffer[]で指定された ID 番号, message_buffer[]定義数の中の最大値

(8) 最大固定長メモリプール ID (max_mpf)

【説明】本定義により、cre_mpf, acre_mpf, del_mpfを使用できます。使用できる固定長メモリプールIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_mpf, memorypool[]で指定された ID 番号, memorypool[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_mpf, acre_mpf, del_mpfはE_NOSPTを返します。
使用できる固定長メモリプールIDの範囲は以下の通りです。

- 最小値：1
- 最大値：memorypool[]で指定された ID 番号, memorypool[]定義数の中の最大値

(9) 最大可変長メモリプール ID (max_mpl)

【説明】本定義により、cre_mpl, acre_mpl, del_mplを使用できます。使用できる可変長メモリプールIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_mpl, variable_memorypool[]で指定された ID 番号, variable_memorypool[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_mpl, acre_mpl, del_mplはE_NOSPTを返します。
使用できる可変長メモリプールIDの範囲は以下の通りです。

- 最小値：1
- 最大値：variable_memorypool[]で指定された ID 番号, variable_memorypool[]定義数の中の最大値

(10) 最大周期ハンドラ ID (max_cyh)

【説明】本定義により、cre_cyc, acre_cyc, del_cycを使用できます。使用できる周期ハンドラIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_cyh, cyclic_hand[]で指定された ID 番号, cyclic_hand[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_cyc, acre_cyc, del_cycはE_NOSPTを返します。
使用できる周期ハンドラIDの範囲は以下の通りです。

- 最小値：1
- 最大値：cyclic_hand[]で指定された ID 番号, cyclic_hand[]定義数の中の最大値

(11) 最大アラームハンドラ ID (max_alh)

【説明】本定義により、cre_alm, acre_alm, del_almを使用できます。使用できるアラームハンドラIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_alh, alarm_hand[]で指定された ID 番号, alarm_hand[]定義数の中の最大値

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】cre_alm, acre_alm, del_almはE_NOSPTを返します。
使用できる周期ハンドラIDの範囲は以下の通りです。

- 最小値：1
- 最大値：alarm_hand[]で指定された ID 番号, alarm_hand[]定義数の中の最大値

(12) 最大ドメイン ID (max_domain)

【説明】使用できるドメインIDの範囲は以下の通りです。

- 最小値：1
- 最大値：max_domain、domain[]で指定された ID 番号の中の最大値

【定義形式】数値

【定義範囲】1～15

【省略時の扱い】使用できるドメインIDの範囲は以下の通りです。

- 最小値：1
- 最大値：domain[]で指定された ID 番号の中の最大値

8.4.5 ドメイン定義(domain[])

domain[]は、ドメインを定義します。本定義を省略したドメイン ID のドメインは、trust = NO の扱いとなります。

■形式

```
domain[(1)ID 番号]{
        trust               = (2)信頼されたドメイン;
};
```

■内容

(1) ID 番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～15

【省略時の扱い】省略不可(エラー)

(2) 信頼されたドメイン(trust)

【説明】信頼されたドメインかどうかを定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- YES：信頼されたドメインとする
- NO：信頼されたドメインとしない。

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はNO)を適用

8.4.6 メモリオブジェクト定義(memory_object[])

memory_object[]は、メモリオブジェクトを定義(登録)する定義項目です。
なお、cfg ファイルにはメモリオブジェクト定義が少なくともひとつ必要です。

■形式

```
memory_object[]{
        start_address       = (1)メモリオブジェクトの先頭アドレス;
        end_address         = (2)メモリオブジェクトの終端アドレス;
        acptn1              = (3)オペランドリードアクセス許可パターン;
        acptn2              = (3)オペランドライトアクセス許可パターン;
        acptn3              = (3)実行アクセス許可パターン;
};
```

■内容

(1) メモリオブジェクトの先頭アドレス(start_address)

【説明】メモリオブジェクトの先頭アドレスを、数値またはセクション名で定義します。

セクション名を指定した場合、そのセクションはリンク時に必ず16バイト境界アドレスに配置する必要があります。リンク時、このセクションに"aligned_section"オプションを指定することで、この制約が満たされるようになります。

数値で指定する場合は、16の倍数でなければなりません。

【定義形式】シンボルまたは数値

【定義範囲】数値の場合は、0～0xFFFFFFF0で、かつ16の倍数

【省略時の扱い】省略不可(エラー)

(2) メモリオブジェクトの終端アドレス(end_address)

【説明】メモリオブジェクトの終端アドレスを、数値またはセクション名で定義します。

セクション名を指定した場合、そのセクションの終端アドレスを16の倍数+15に切り上げたアドレスをメモリオブジェクトの終端アドレスと扱います。リンク時にこのセクションの終端アドレスが16の倍数+15以外となる場合は、セクション終端アドレス+1～16の倍数+15に切り上げたアドレスまでの範囲に他のセクションを配置してはなりません。

数値で指定する場合は、16の倍数+15でなければなりません。

【定義形式】シンボルまたは数値

【定義範囲】数値の場合は、0x0000000F～0xFFFFFFFFで、かつの16の倍数+15

【省略時の扱い】省略不可(エラー)

(3) アクセス許可パターン(acptn1, acptn2, acptn3)

【説明】オペランドリードアクセス(acptn1), オペランドライトアクセス(acptn2), 実行アクセス(acptn3)のアクセス許可パターンを、シンボルまたは数値で定義します。

シンボルとしては、TACP_SHARED(全ドメインにアクセスを許可)のみを指定できます。

ドメインごとに許可/禁止を個別に指定するには、図8.1に従って数値で指定します。

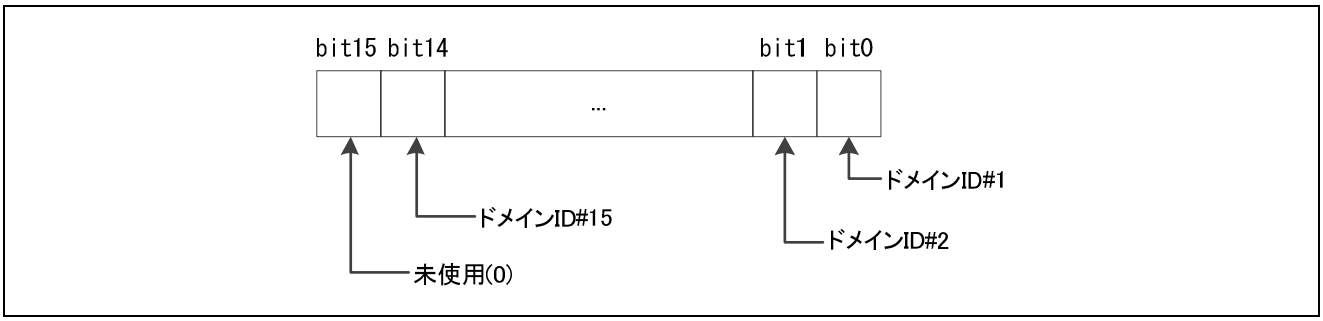

図8.1 アクセス許可パターン

【定義形式】シンボルまたは数値

【定義範囲】シンボル：TACP_SHARED(全ドメインにアクセスを許可)

数値：0～0x7FFF

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTACP_SHARED)を適用

8.4.7 タスク定義(task[])

task[]は、タスクを定義(生成)する定義項目です。

なお、本製品では特に初期起動タスクという仕様はありません。

cfg ファイルでタスクを定義する際に、task[].initial_start に ON を指定することで、そのタスクはシステム起動時に READY 状態となります。複数のタスクの initial_start を ON にした場合は、ID 番号の小さい順に READY 状態になります。

なお、cfg ファイルには initial_start を ON にしたタスク定義が少なくともひとつ必要です。

■形式

```
task[(1)ID 番号]{
        name                = (2)ID 名称;
        entry_address       = (3)タスクの開始アドレス;
        stack_size          = (4)タスクのユーザスタックサイズ;
        stack_section       = (5)スタック領域に付与するセクション名;
        priority            = (6)タスクの起動時優先度;
        initial_start       = (7)TA_ACT 属性(初期起動状態);
        exinf               = (8)拡張情報;
        texrtn              = (9)タスク例外処理ルーチンの開始アドレス;
        domain_num          = (10)所属ドメイン ID;
};
```

■内容

(1) ID 番号

【説明】タスクID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID 名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID 名称>  <ID 値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) タスクの開始アドレス(entry_address)

【説明】タスクの実行開始アドレスを定義します。

【定義形式】関数名

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(4) タスクのユーザスタックサイズ(stack_size)

【説明】タスクのユーザスタックサイズを定義します。ユーザースタックとは、各々のタスクが使用するスタック領域です。RI600/PXではタスクごとにユーザスタック割り当てる必要があります。
ここで指定したサイズのユーザスタック領域が、cfg600pxによって生成されます。

【定義形式】数値

【定義範囲】表8.8に示す値以上の16の倍数

表8.8 ユーザスタックサイズの下限値

system.context	ユーザスタックサイズの下限値
NO	68
FPSW	72
ACC	76
FPSW,ACC	80
MIN	44
MIN,FPSW	48
MIN,ACC	52
MIN,FPSW,ACC	56

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は256)を適用

(5) スタック領域に付与するセクション名(stack_section)

【説明】ユーザスタック領域に付与するセクション名を定義します。このセクションのセクション属性は"DATA"、アライメント数は4です。リンク時には、このセクションをRAM領域に配置してください。このセクションをメモリオブジェクトに含めてはなりません。また、0番地に配置してはなりません。

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はSURI_STACK)を適用

(6) タスクの起動時優先度(priority)

【説明】タスクの起動時優先度を定義します。

【定義形式】数値

【定義範囲】1～system.priority

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

(7) TA_ACT 属性(初期起動状態)(initial_start)

【説明】タスクの初期状態をREADY状態とするかDORMANT状態とするかを定義します。本項目は、タスクのTA_ACT属性に該当します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- ON：カーネル起動に READY 状態にする。
- OFF：カーネル起動に DORMANT 状態にする。

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はOFF)を適用

(8) 拡張情報(exinf)

【説明】タスクの拡張情報を定義します。

【定義形式】数値

【定義範囲】0～0xFFFFFFFF

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0)を適用

(9) タスク例外処理ルーチンの開始アドレス(texrtn)

【説明】タスク例外処理ルーチンの実行開始アドレスを定義します。タスク例外処理ルーチンを定義しない場合は、本定義を行わないでください。

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】タスク例外処理ルーチンは定義されません。

(10) 所属ドメイン ID(domain_num)

【説明】タスクが所属するドメインIDを定義します。

【定義形式】数値

【定義範囲】1～15

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

8.4.8 セマフォ定義(semaphore[])

semaphore[]は、セマフォを定義(生成)する定義項目です。

■形式

```
semaphore[(1)ID番号]{
        name                = (2)ID名称;
        max_count           = (3)セマフォカウンタ最大値;
        initial_count       = (4)セマフォカウンタ初期値;
        wait_queue          = (5)待ち行列属性;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。
【定義形式】数値
【定義範囲】1～255
【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル
【定義範囲】－
【省略時の扱い】省略不可(エラー)

(3) セマフォカウンタ最大値(max_count)

【説明】セマフォカウンタの最大値を定義します。
【定義形式】数値
【定義範囲】1～65535
【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

(4) セマフォカウンタ初期値(initial_count)

【説明】セマフォカウンタの初期値を定義します。
【定義形式】数値
【定義範囲】0～max_count
【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

(5) 待ち行列属性(wait_queue)

【説明】待ち行列属性を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- TA_TFIFO：待ち行列は FIFO 順
- TA_TPRI：待ち行列はタスク優先度順

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTA_TFIFO)を適用

8.4.9 イベントフラグ定義(flag[])

flag[]は、イベントフラグを定義(生成)する定義項目です。

■形式

```
flag[(1)ID番号]{
        name                = (2)ID名称;
        initial_pattern     = (3)イベントフラグの初期ビットパターン;
        wait_queue          = (4)待ち行列属性;
        wait_multi          = (5)複数待ちの許可属性;
        clear_attribute     = (6)クリア属性;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) イベントフラグの初期ビットパターン(initial_pattern)

【説明】イベントフラグの初期ビットパターンを定義します。

【定義形式】数値

【定義範囲】0～0xFFFFFFFF

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0)を適用

(4) 待ち行列属性(wait_queue)

【説明】待ち行列属性を定義します。

なお、wait_multiにTA_WSGLを指定した場合は、本項目は意味を持ちません。

また、wait_multi に TA_WMUL を指定した場合でも、clear_attribute に NO を指定した場合は、本項目の指定に関わらず、待ち行列は FIFO 順で管理されます。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- TA_TFIFO：待ち行列は FIFO 順
- TA_TPRI：待ち行列はタスク優先度順

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTA_TFIFO)を適用

(5) 複数待ちの許可属性(wait_multi)

【説明】複数タスク待ちの許可に関する属性を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- TA_WMUL：複数タスク待ちを許可する
- TA_WSGL：複数タスク待ちを許可しない

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTA_WSGL)を適用

(6) クリア属性(clear_attribute)

【説明】イベントフラグのクリア(TA_CLR)属性を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- YES：クリア属性を設定する
- NO：クリア属性を設定しない

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はNO)を適用

8.4.10 データキュー定義(dataqueue[])

dataqueue[]は、データキューを定義(生成)する定義項目です。

■形式

```
dataqueue[(1)ID番号]{
        name                = (2)ID名称;
        buffer_size         = (3)データキューの最大データ数;
        wait_queue          = (4)待ち行列属性;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) データキューの最大データ数(buffer_size)

【説明】データキューの最大データ数を定義します。データキュー領域のサイズは、buffser_size×4(バイト)となります。データ数0のデータキューを生成することもできます。

【定義形式】数値

【定義範囲】0～65535

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0)を適用

(4) 待ち行列属性(wait_queue)

【説明】送信待ち行列属性を定義します。なお、受信待ち行列は常にFIFO順で管理されます。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- TA_TFIFO：待ち行列は FIFO 順
- TA_TPRI：待ち行列はタスク優先度順

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTA_TFIFO)を適用

8.4.11 メールボックス定義(mailbox[])

mailbox[]は、メールボックスを定義(生成)する定義項目です。

■形式

```
mailbox[(1)ID番号]{
        name                = (2)ID名称;
        wait_queue          = (3)待ち行列属性;
        message_queue       = (4)メッセージキュー属性;
        max_pri             = (5)メッセージの最大優先度;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) 待ち行列属性(wait_queue)

【説明】待ち行列属性を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- TA_TFIFO：待ち行列はFIFO順
- TA_TPRI：待ち行列はタスク優先度順

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTA_TFIFO)を適用

(4) メッセージキュー属性(message_queue)

【説明】メッセージのメッセージキューへのつなぎ方を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- TA_MFIFO：メッセージキューは FIFO 順
- TA_MPRI：メッセージキューはメッセージ優先度順

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTA_MFIFO)を適用

(5) メッセージの最大優先度(max_pri)

【説明】message_queueにTA_MPRIを指定した場合は、ここにメッセージの最大優先度を定義します。

【定義範囲】1～system.message_pri

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

【備考】message_queueがTA_MFIFOの場合、本項目は意味を持ちません。

8.4.12 ミューテックス定義(mutex[])

mutex[]は、ミューテックスを定義(生成)する定義項目です。

■形式

```
mutex[(1)ID番号]{
        name                = (2)ID名称;
        ceilpri             = (3)上限優先度;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】―

【省略時の扱い】省略不可(エラー)

(3) 上限優先度(ceilpri)

【説明】RI600/PXのミューテックスは、優先度上限プロトコルで制御されます。ここには、その上限優先度を定義します。

【定義形式】数値

【定義範囲】1～system.priority

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

8.4.13 メッセージバッファ定義(message_buffer[])

message_buffer[]は、メッセージバッファを定義(生成)する定義項目です。

■形式

```
message_buffer[(1)ID番号]{
        name                = (2)ID名称;
        mbf_size            = (3)メッセージバッファのサイズ;
        mbf_section         = (4)メッセージバッファ領域に付与するセクション名;
        max_msgsz           = (5)最大メッセージサイズ;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) メッセージバッファのサイズ(mbf_size)

【説明】メッセージバッファのサイズをバイト単位で指定します。
サイズ0のメッセージバッファを生成することもできます。この場合、メッセージバッファの送受信側が完全に同期した通信を行うことになります。

【定義形式】数値

【定義範囲】0，または8～65532の4の倍数

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0)を適用

(4) メッセージバッファ領域に付与するセクション名(mbf_section)

【説明】メッセージバッファ領域に付与するセクション名を定義します。このセクションのセクション属性は"DATA"、アライメント数は4です。リンク時には、このセクションをRAM領域に配置してください。
メッセージバッファ領域には、カーネルのみがアクセスするため、通常はこのセクションをメモリオブジェクトに含めないでください。また、0番地に配置してはなりません。

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はBRI_RAM)を適用

【備考】mbf_sizeが0の場合、本項目は意味を持ちません。

(5) 最大メッセージサイズ(max_msgsz)

【説明】最大メッセージサイズをバイト単位で指定します。指定した値は4の倍数に切り上げて扱います。
mbf_size>0の場合は、max_msgszは(mbf_size - 4)以下でなければなりません。

【定義形式】数値

【定義範囲】1～65528

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は4)を適用

8.4.14 固定長メモリプール定義(memorypool[])

memorypool[]は、固定長メモリプールを定義(生成)する定義項目です。

■形式

```
memorypool[(1)ID番号]{
        name                = (2)ID名称;
        section             = (3)プール領域に付与するセクション名;
        num_block           = (4)メモリブロック数;
        siz_block           = (5)メモリブロックサイズ;
        wait_queue          = (6)待ち行列属性;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) プール領域に付与するセクション名(section)

【説明】プール領域に付与するセクション名を定義します。このセクションのセクション属性は"DATA"、アライメント数は4です。リンク時には、このセクションをRAM領域に配置してください。
タスクが獲得したメモリブロックにアクセスできるようにするためには、本セクションをメモリオブジェクトの一部とする必要があります。また、0番地に配置してはなりません。

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はBURI_HEAP)を適用

(4) メモリブロック数(num_block)

【説明】メモリブロック数を定義します。
なお、メモリプール領域のサイズは、num_block×siz_block(バイト)となります。プールサイズの上限は、VTMAX_AREASIZE(バイト)です。

【定義形式】数値

【定義範囲】1～65535

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

(5) メモリブロックサイズ(siz_block)

【説明】メモリブロックサイズをバイト単位で指定します。

【定義形式】数値

【定義範囲】4～65535

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は256)を適用

(6) 待ち行列属性(wait_queue)

【説明】待ち行列属性を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- TA_TFIFO：待ち行列は FIFO 順
- TA_TPRI：待ち行列はタスク優先度順

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はTA_TFIFO)を適用

8.4.15 可変長メモリプール定義(variable_memorypool[])

variable_memorypool[]は、可変長メモリプールを定義(生成)する定義項目です。

■形式

```
variable_memorypool[(1)ID番号]{
        name                = (2)ID名称;
        mpl_section         = (3)プール領域に付与するセクション名;
        heap_size           = (4)メモリプールのサイズ;
        max_memsize         = (5)メモリブロックサイズの上限;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) プール領域に付与するセクション名(mpl_section)

【説明】プール領域に付与するセクション名を定義します。このセクションのセクション属性は"DATA"、アライメント数は4です。リンク時には、このセクションをRAM領域に配置してください。
タスクが獲得したメモリブロックにアクセスできるようにするためには、本セクションをメモリオブジェクトの一部とする必要があります。また、0番地に配置してはなりません。

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はBURI_HEAP)を適用

(4) メモリプールのサイズ (heap_size)

【説明】メモリプールのサイズをバイト単位で定義します。指定した値は4の倍数に切り上げられます。

【定義形式】数値

【定義範囲】24～VTMAX_AREASIZE

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は(1024))を適用

(5) メモリブロックサイズの上限 (max_memsize)

【説明】獲得可能なメモリブロックサイズの上限をバイト単位で定義します。実際に獲得可能な最大サイズは、max_memsizeよりも大きくなる場合があります。

【定義形式】数値

【定義範囲】1～0xBFFFFF4 (192MB - 12)

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は36)を適用

【補足】現在のカーネル実装では、実際に獲得されるメモリブロックのサイズは、最大で12種類のバリエーションから選択されます。このバリエーションはmax_memsizeを元に決定されます。表8.9に、メモリブロックサイズのバリエーションを示します。ただし、この振る舞いは将来変更される可能性があります。

表8.9 メモリブロックサイズのバリエーション

項番	メモリブロックサイズ (16 進数)	例 1：max_memsize=0x100 の場合	例 2：max_memsize=0x20000 の場合
1	12 (0xC)	○	×
2	36 (0x24)	○	×
3	84 (0x54)	○	○
4	180 (0xB4)	○	○
5	372 (0x174)	×	○
6	756 (0x2F4)	×	○
7	1524 (0x5F4)	×	○
8	3060 (0xBF4)	×	○
9	6132 (0x17F4)	×	○
10	12276 (0x2FF4)	×	○
11	24564 (0x5FF4)	×	○
12	49140 (0xBFF4)	×	○
13	98292 (0x17FF4)	×	○
14	196596 (0x2FFF4)	×	○
15	393204 (0x5FFF4)	×	×
16	786420 (0xBFFF4)	×	×
17	1572852 (0x17FFF4)	×	×
18	3145716 (0x2FFFF4)	×	×
19	6291444 (0x5FFFF4)	×	×
20	12582900 (0xBFFFF4)	×	×
21	25165812 (0x17FFFF4)	×	×
22	50331636 (0x2FFFFF4)	×	×
23	100663284 (0x5FFFFF4)	×	×
24	201326580 (0xBFFFFF4)	×	×

8.4.16 周期ハンドラ定義(cyclic_hand[])

cyclic_hand[]は、周期ハンドラを定義(生成)する定義項目です。

■形式

```
cyclic_hand[(1)ID番号]{
        name                = (2)ID名称;
        entry_address       = (3)ハンドラ開始アドレス;
        interval_counter    = (4)起動周期;
        start               = (5)周期ハンドラの動作状態;
        phsatr              = (6)起動位相の保存;
        phs_counter         = (7)起動位相;
        exinf               = (8)拡張情報;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) ハンドラの開始アドレス(entry_address)

【説明】周期ハンドラの実行開始アドレスを定義します。

【定義形式】関数名

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(4) 起動周期(interval_counter)

【説明】起動周期をミリ秒単位で定義します。

【定義範囲】1 ～ (0x7FFFFFFF - system.tic_nume) / system.tic_deno

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は1)を適用

(5) 周期ハンドラの動作状態(start)

【説明】周期ハンドラの動作状態に関する属性を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- ON：周期ハンドラを動作状態にする(TA_STA 属性指定あり)
- OFF：周期ハンドラを動作状態にしない(TA_STA 属性指定なし)

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はOFF)を適用

(6) 起動位相の保存(phsatr)

【説明】起動位相の保存に関する属性を定義します。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- ON：起動位相を保存する(TA_PHS 属性指定あり)
- OFF：起動位相を保存しない(TA_PHS 属性指定なし)

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時はOFF)を適用

(7) 起動位相(phs_counter)

【説明】起動位相をミリ秒単位で定義します。起動位相は起動周期以下でなければなりません。

【定義形式】数値

【定義範囲】0～interval_counter

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0)を適用

(8) 拡張情報(exinf)

【説明】周期ハンドラの拡張情報を定義します。

【定義形式】数値

【定義範囲】0～0xFFFFFFFF

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0)を適用

8.4.17 アラームハンドラ定義(alarm_hand[])

alarmc_hand[]は、アラームハンドラを定義(生成)する定義項目です。

■形式

```
alarm_hand[(1)ID番号]{
        name                = (2)ID名称;
        entry_address       = (3)ハンドラ開始アドレス;
        exinf               = (4)拡張情報;
};
```

■内容

(1) ID番号

【説明】ID番号を定義します。

【定義形式】数値

【定義範囲】1～255

【省略時の扱い】自動的に割り当てられます。

(2) ID名称(name)

【説明】ID名称を定義します。指定されたID名称は、ID名称ヘッダファイル(kernel_id.h)に以下の形式で出力されます。

```
#define  <ID名称>  <ID値>
```

【定義形式】シンボル

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) ハンドラの開始アドレス(entry_address)

【説明】アラームハンドラの実行開始アドレスを定義します。

【定義形式】関数名

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(4) 拡張情報(exinf)

【説明】アラームハンドラの拡張情報を定義します。

【定義形式】数値

【定義範囲】0～0xFFFFFFFF

【省略時の扱い】デフォルトcfgファイルの設定値(出荷時は0)を適用

8.4.18 可変ベクタ割り込み定義(interrupt_vector[])

interrupt_vector[]は、可変ベクタに対する割り込みハンドラを定義する定義項目です。

本定義を行わないベクタ番号の割り込みが発生した場合は、システムダウンとなります。

なお、cfg600px はここで指定した割り込みに関する割り込み制御レジスタ(IPL 等)や、割り込み要因等の初期設定のコードは生成しません。初期設定はスタートアップファイル中もしくは、開発されるアプリケーションにあわせて作成していただく必要があります。

■注意

ベクタ番号 1～8 はカーネルが使用するため、定義しないでください。また、マイコン仕様で予約となっているベクタには定義しないでください。

■形式

```
interrupt_vector[(1)ベクタ番号]{
        entry_address       = (2)ハンドラの開始アドレス;
        os_int              = (3)カーネル管理割り込み指定;
        pragma_switch       = (4)PRAGMA 拡張機能に渡すスイッチ;
};
```

■内容

(1) ベクタ番号

【説明】ハンドラを定義する割り込みのベクタ番号を定義します。

【定義形式】数値

【定義範囲】0～255

【省略時の扱い】省略不可(エラー)

(2) ハンドラの開始アドレス(entry_address)

【説明】割り込みハンドラの実行開始アドレスを定義します。

【定義形式】関数名

【定義範囲】－

【省略時の扱い】省略不可(エラー)

(3) カーネル管理割り込み指定(os_int)

【説明】この割り込みがカーネル管理割り込みかどうかを定義します。

カーネル割り込みマスクレベル(system.system_IPL)以下の割り込みはカーネル管理割り込み、それ以外はカーネル管理外割り込みとして定義する必要があります。なお、system.system_IPLが15の場合は、すべての可変ベクタ割り込みはカーネル管理割り込みとする必要があります。

【定義形式】シンボル

【定義範囲】以下のいずれかから選択してください。

- YES：カーネル管理割り込み
- NO：カーネル管理外割り込み

【省略時の扱い】省略不可(エラー)

(4) PRAGMA 拡張機能に渡すスイッチ(pragma_switch))

【説明】cfg600pxは、entry_addressで指定された関数を、割り込み関数としてkenrel_id.hに#pragma interruptディレクティブを出力します。このpragmaディレクティブに渡すスイッチを指定します。
pragma仕様の詳細は、コンパイラのマニュアルを参照してください。

【定義形式】シンボル

【定義範囲】以下を指定できます。複数指定する場合は、カンマで区切ってください。なお、"ACC"と"NOACC"を同時に指定することはできません。

- E：多重割り込みを許可する"enable"スイッチを渡します。
- F：高速割り込みを指定する"fint"スイッチを渡します。なお、高速割り込みは必ずカーネル管理外割り込み(os_int=NO)としなければなりません。
- S：割り込みハンドラで使用するレジスタ数を制限する"save"スイッチを渡します。
- ACC：割り込みハンドラで ACC レジスタを保証する"acc"スイッチを渡します。
- NOACC：割り込みハンドラで ACC レジスタを保証しない"noacc"スイッチを渡します。

表8.10に、ACCレジスタの扱いを示します。

表8.10 ACC レジスタの扱い

pragma_switch の ACC, NOACC 指定	コンパイラの"save_acc"オプション	
	なし	あり
なし	"acc", noacc"いずれのスイッチも渡されません。ACC は保証されません。	"acc", noacc"いずれのスイッチも渡されません。ACC は保証されます。
"ACC"	acc スイッチが渡されます。ACC は保証されます。	
"NOACC"	noacc スイッチが渡されます。ACC は保証されません。	

【省略時の扱い】何もスイッチを渡しません。

8.4.19 固定ベクタ割り込み定義(interrupt_fvector[])

interrupt_fvector[]は、固定ベクタに対する割り込みハンドラを定義する定義項目です。固定ベクタの要因は、すべてカーネル管理外割り込みの扱いとなります。

固定割り込みベクタの各要因には、マイコン仕様ではベクタ番号が割り振られていませんが、RI600/PX では各ベクタアドレスに対して表 8.11に示すようにベクタ番号を割り当てています。

本定義を行わない場合の振る舞いも表 8.11に示します。

表8.11 固定割り込みベクタ

ベクタアドレス	ベクタ番号	要因 *1	定義省略時の扱い
0xFFFFFF80	0	**エンディアン選択レジスタ**	**コンパイラの** endian **オプションに応じて、以下が設定されます。** ・endian=little **の場合：**0xFFFFFFFF ・endian=big **の場合：**0xFFFFFFF8
0x FFFFFF84	1	**(予約領域)**	0xFFFFFFFF
0x FFFFFF88	2	**オプション機能選択レジスタ** 1	
0x FFFFFF8C	3	**オプション機能選択レジスタ** 0	
0x FFFFFF90	4	**(予約領域)**	
0x FFFFFF94	5	**(予約領域)**	
0x FFFFFF98	6	**(予約領域)**	
0x FFFFFF9C	7	ROM **コードプロテクト(フラッシュメモリ)**	
0x FFFFFFA0	8	**オンチップデバッガ** ID **コードプロテクト(フラッシュメモリ)**	
0x FFFFFFA4	9		
0x FFFFFFA8	10		
0x FFFFFFAC	11		
0x FFFFFFB0	12	**(予約領域)**	
0x FFFFFFB4	13	**(予約領域)**	
0x FFFFFFB8	14	**(予約領域)**	
0x FFFFFFBC	15	**(予約領域)**	
0x FFFFFFC0	16	**(予約領域)**	**システムダウン**
0x FFFFFFC4	17	**(予約領域)**	
0x FFFFFFC8	18	**(予約領域)**	
0x FFFFFFCC	19	**(予約領域)**	
0x FFFFFFD0	20	**特権命令例外**	
0x FFFFFFD4	21	**アクセス例外**	**アクセス例外ハンドラ**
0x FFFFFFD8	22	**(予約領域)**	**システムダウン**
0x FFFFFFDC	23	**未定義命令例外**	
0x FFFFFFE0	24	**(予約領域)**	
0x FFFFFFE4	25	**浮動小数点例外**	
0x FFFFFFE8	26	**(予約領域)**	
0x FFFFFFEC	27	**(予約領域)**	
0x FFFFFFF0	28	**(予約領域)**	
0x FFFFFFF4	29	**(予約領域)**	
0x FFFFFFF8	30	**ノンマスカブル割り込み**	
0x FFFFFFFC	31	**リセット**	PowerON_Reset_PC()

【注】 MCU **品種によって異なります。**

なお、cfg600px はここで指定した割り込みに関する割り込み制御レジスタ(IPL 等)や、割り込み要因等の初期設定のコードは生成しません。初期設定はスタートアップファイル中もしくは、開発されるアプリケーションにあわせて作成していただく必要があります。

■注意

マイコン仕様で予約となっているベクタには定義しないでください。

また、ベクタ 21(アクセス例外)にハンドラを定義しないでください。定義すると、アクセス例外ハンドラ(_RI_sys_access_exception())が起動されなくなります。

■形式

```
interrupt_fvector[(1)ベクタ番号]{
        entry_address       = (2)ハンドラの開始アドレス;
        pragma_switch       = (3)PRAGMA 拡張機能に渡すスイッチ;
};
```

■内容

(1) ベクタ番号

【説明】表8.11を参考にベクタ番号を定義します。

【定義形式】数値

【定義範囲】0～31

【省略時の扱い】省略不可(エラー)

(2) ハンドラの開始アドレス(entry_address)

【説明】割り込みハンドラの実行開始アドレス、または固定ベクタへの設定値を定義します。

【定義形式】関数名または数値

【定義範囲】数値の場合は0~0xFFFFFFFF

【省略時の扱い】省略不可(エラー)

(3) PRAGMA 拡張機能に渡すスイッチ(pragma_switch))

【説明】cfg600pxは、entry_addressで指定された関数を、割り込み関数としてkenrel_id.hに#pragma interruptディレクティブを出力します。このpragmaディレクティブに渡すスイッチを指定します。
pragma仕様の詳細は、コンパイラのマニュアルを参照してください。
なお、entry_addressに数値を指定した場合、およびベクタ番号31(リセット)については、#pragma interruptディレクティブは出力されません。

【定義形式】シンボル

【定義範囲】以下を指定できます。複数指定する場合は、カンマで区切ってください。なお、"ACC"と"NOACC"を同時に指定することはできません。

- S：割り込みハンドラで使用するレジスタ数を制限する"save"スイッチを渡します。
- ACC：割り込みハンドラで ACC レジスタを保証するコードを生成する"acc"スイッチを渡します。
- NOACC：割り込みハンドラで ACC レジスタを保証するコードを生成しない"noacc"スイッチを渡します。

ACCレジスタの扱いについては、表8.10を参照してください。

【省略時の扱い】何もスイッチを渡しません。

8.5 コンフィギュレータの実行

8.5.1 コンフィギュレータ概要

コンフィギュレータは、cfgファイルで定義した内容を元に、システム定義ファイルやアプリケーション用ヘッダファイルを出力するツールです。コンフィギュレータの実行によって以下のファイルが出力されます。

- ID 番号ヘッダファイル(kernel_id.h)
  カーネルオブジェクトの ID 番号を定義しているファイルです。
- サービスコール定義ファイル(kernel_sysint.h)
  サービスコールの INT 命令を用いて呼び出すための宣言ファイルです。kernel.h からインクルードされます。
- システム定義ファイル(kernel_rom.h, kernel_ram.h, ri600.inc)
  kernel_rom.h と kernel_ram.h は、スタートアップファイルからインクルードする必要があります。「7.2スタートアップファイル(resetprg.c)の作成」を参照してください。
  ri600.inc は、mkritblpx が生成する ritable.src からインクルードされます。
- ベクタテーブルテンプレートファイル(vector.tpl)
  vector.tpl は、mkritblpx に読み込まれます。
- CMT 定義ファイル(ri_cmt.h)
  clock.timer に CMT0, CMT1, CMT2, CMT3 のいずれかを指定した場合は、clock,template で指定されたテンプレートファイルが環境変数 LIB600 から検索され、ri_cmt.h にリネームされて出力されます。
  ri_cmt.h はスタートアップファイルからインクルードされます。

コンフィギュレータの動作概要を図 8.2に示します。

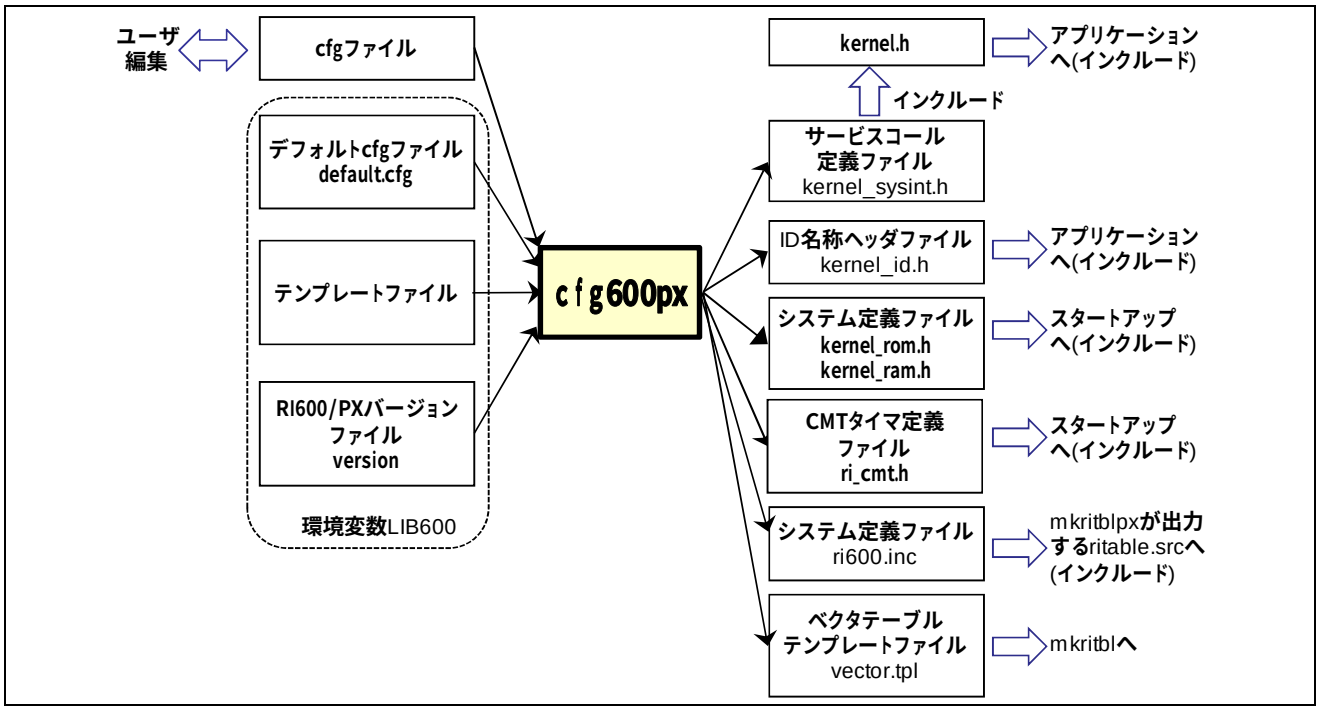

図8.2 コンフィギュレータ動作概要

8.5.2 環境設定

以下の環境変数の設定が必要です。

- LIB600
  "インストールディレクトリ¥lib600"

8.5.3 コンフィギュレータ起動方法

コンフィギュレータは、以下の形式で起動します。

```
cfg600px[△<オプション>…][△<cfg ファイル名>]
```

cfg ファイル名の拡張子を省略した場合は、拡張子".cfg"を補って解釈します。

8.5.4 コマンドオプション

(1) -U オプション

未定義割り込みが発生した時にはシステムダウンとなりますが、本オプションを指定すると、発生した割り込みのベクタ番号がシステムダウンルーチンに渡されるようになる(「6.8 システムダウンルーチン」を参照)ため、デバッグに役立ちます。ただし、カーネルのコードサイズが約 1.5kB 増加します。

(2) -v オプション

コマンドのオプションの説明と詳細なバージョンを表示します。

(3) -V オプション

コマンドが生成するファイルの作成状況を表示します。

8.6 エラーメッセージ

8.6.1 エラー形式とエラーレベル

本節では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ

エラーレベルは、表 8.12に示すように分類されます。

表8.12 エラーレベル

エラーレベル		動作
(W)	ワーニング	処理を継続します。
(E)	エラー	処理を中断します。

8.6.2 メッセージ一覧

(1) エラーメッセージ

O1001 (E) Syntax error.
文法エラー

O1002 (E) Illegal XXX --> <設定値>
(1) XXX の数値または ID 番号が間違っています。
(2) XXX の定義数が限界値を超えています。

O1003 (E) Unknown token --> <XXX>
XXX は定義名として認識できません。

O1005 (E) Task[1]'s priority is too large. --> <設定値>
task[].priority が、system.priority を超えています。

O1006 (E) clock.IPL is too large.--> <設定値>
clock.IPL が、system system_IPL 値を越えています。

O1007 (E) System timer’s vector <XXX> conflict
clock.timer で CMT0, CMT1, CMT2, CMT3 のいずれかを指定した場合で、そのタイマのベクタ番号に対して interrupt_vector[]が定義されています。

O1009 (E) XXX is already defined.
XXX はすでに定義済みです。

O1010 (E) XXX[YYY] is already defined.
ID 番号が YYY のオブジェクト XXX が多重定義されています。

O1013 (E) Zero divide error
演算式でゼロ除算が発生しました。

O1015 (E) Can't specify F switch when os_int=YES.
カーネル管理割り込みハンドラに対して"F"スイッチを指定することはできません。

O1016 (E) interrupt_vector[YYY].os_int must be YES.
カーネル割り込みマスクレベル(system.system_IPL)が 15 の場合、可変ベクタをカーネル管理外割り込みとして使用することはできません。

O1018 (E) mailbox[YYY].max_pri(設定値) is bigger than system.message_pri(設定値).
mailbox.max_pri は、system.message_pri 以下でなければなりません。

O1019 (E) Neither system.tic_nume nor system.tic_deno is 1.

system.tic_nume と system.tic_deno の少なくとも一方は 1 でなければなりません。

O1020 (E) Symbols other than NO and NO are defined simultaneously.

NO と NO 以外のシンボルを同時に指定することはできません。

O1022 (E) semaphore[YYY].initial_count is bigger than semaphore[YYY].max_count

semaphore.initial_count は max_count 以下でなければなりません。

O1023 (E) Size of memorypool[YYY] is larger than VTMAX_AREASIZE

固定長メモリプールのサイズ(memorypool.num_block×memorypool.siz_block)は、VTMAX_AREASIZE(256MB)以下でなければなりません。

O1024 (E) varilable_memorypool[YYY].max_memsize is larger than 192MB-12

variable_memorypool.max_memsize は、192MB-12 以下でなければなりません。

O1025 (E) mutex[YYY].ceilpri is bigger than system.priority.

mutex.ceilpri は、system.priority 以下でなければなりません。

O1026 (E) XXX is not a multiple of 4.

XXX は 4 の倍数でなければなりません。

O1027 (E) max_msgsz (設定値) is larger than mbf_size(設定値) - 4 .

message_buffer.max_msgsz は、message_buffer.mbf_size-4 以下でなければなりません。

O1028 (E) variable_memorypool[YYY].max_memsize(XXX) is too large. (Max.=ZZZ)

variable_memorypool.max_memsize に指定された値 XXX が大きすぎます。指定されたプールサイズ(variable_memorypool[].heap_size)に対して指定できる最大値は ZZZ です。

O1029 (E)Timer tick is too long.

system.tic_nume, system.tic_deno で定まるタイムティックが長すぎます。タイムティックを短くするか、clock.timer_clock を低くしてください。

O1030 (E)Timer tick is too short.

system.tic_nume, system.tic_deno で定まるタイムティックが短かすぎます。タイムティックを長くするか、clock.timer_clock を高くしてください。

O1033 (E) start_address must be smaller than end_address.

start_address は、end_address 未満でなければなりません。

O1034 (E) ACC and NOACC can’t be specified at the same time.

ACC と NOACC を同時に指定することはできません。

O1035 (E) task[XXX].domain_num(=YYY) is bigger than maxdefine.max_domain(=ZZZ).

task[XXX].domain_num の設定値 YYY が maxdefine.max_domain の設定値 ZZZ を超えています。

O2001 (E) Not enough memory

メモリが足りません。

O2003 (E) Illegal argument --> <XXX>

起動形式に誤りがあります。

O2004 (E) Can't write open <ファイル名>

ファイルを作成できません。

O2005 (E) Can't open <ファイル名>

環境変数"LIB600"の示すディレクトリの下のファイル XXX にアクセスできません。

O2006 (E) Can't open version file

カレントディレクトリまたは環境変数"LIB600"の示すディレクトリの下に、version ファイルがありません。

O2007 (E) Can't open default configuration file

カレントディレクトリまたは環境変数"LIB600"の示すディレクトリの下に、default.cfg ファイルがありません。

O2008 (E) Can't open configuration file <ファイル名>.

指定した cfg ファイルにアクセスできません。

```
O2009 (E) XXX is not defined
```

XXX が定義されていません。

```
O2010 (E) Initial start task is not defined.
```

task[].initial_start に ON を指定したタスク定義がひとつもありません。

```
O2011 (E) Environment variable "LIB600" not prepared.
```

環境変数"LIB600"が設定されていません。

```
O2013 (E) memory_object is not defined.
```

memory_object[]がひとつも定義されていません。

(2) ワーニングメッセージ

```
O3001 (W) XXX is already defined.
```

XXX は既に定義されています。定義内容は無視されます。

```
O3002 (W) The maximum ID of the object is larger than maxdefine.xxx.
```

maxdefine.xxx が、task[]などのオブジェクト定義で指定した ID 番号またはオブジェクト定義数より小さいため、オブジェクトの最大 ID を拡張しました。

```
O4001 (W) XXX is not defined.
```

XXX が省略されたたため、デフォルトコンフィギュレーションファイルの設定を適用しました。

```
O4005 (W) Timer counter value is less than your setting time
```

指定されたタイマ周期(system.tic_nume / system.tic_deno [ms])を誤差なく実現することができません。実際のタイマ周期は、指定された周期よりも短くなります。

9. テーブル生成ユーテリティ(mkritblpx)

9.1 概要

mkritblpx は、アプリケーションで使用しているサービスコール情報を収集して、サービスコールテーブルと割り込みベクタテーブルを生成するコマンドラインツールです。

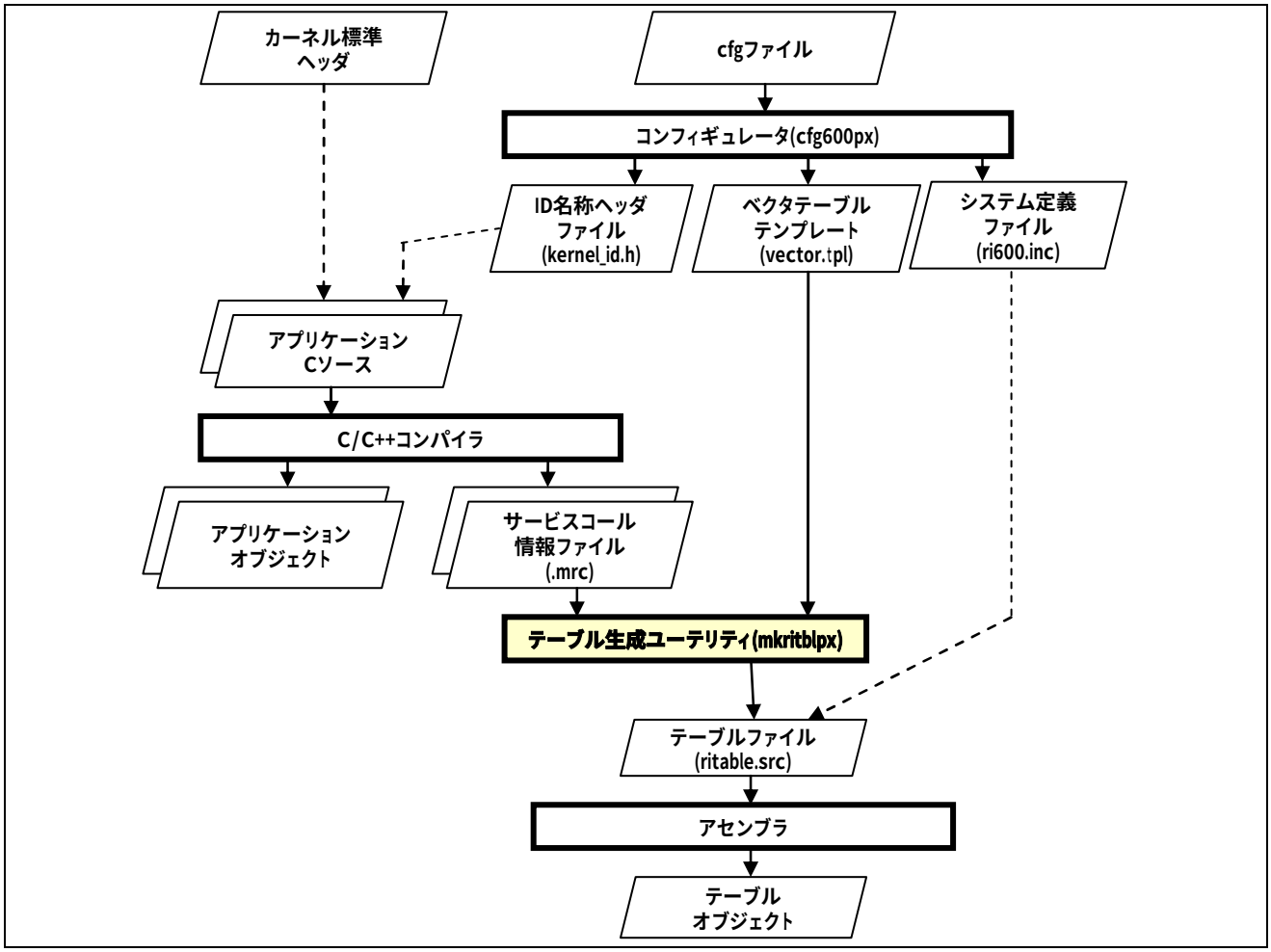

図9.1 mkritblpx **概要**

kernel.h からインクルードされる kernel_sysint.h では、サービスコール関数使用時に.assert 制御命令によって mrc ファイルにサービスコール情報を出力するように定義されています。mkritblpx は、これらのサービスコール情報ファイルを入力として、システムで使用するサービスコールだけがリンクされるようにサービスコールテーブルを生成します。

また、mkritblpx は cfg600px が出力したベクタテーブルテンプレートファイルと mrc ファイルを元に、割り込みベクタテーブルを生成します。

9.2 環境設定

以下の環境変数の設定が必要です。

- LIB600
  "インストールディレクトリ¥lib600"

9.3 テーブル生成ユーティリティ起動方法

テーブル生成ユーティリティは、以下の形式で起動します。

```
C:¥> mkritblpx  <ディレクトリ名またはファイル名>
```

通常は、アプリケーションのコンパイル時に生成される"mrc"ファイルが格納されたディレクトリを引数に指定します。複数のディレクトリ、ファイルを指定することができます。
なお、カレントディレクトリにある"mrc"ファイルは無条件に入力となります。

また、カレントディレクトリに、cfg600px が出力した vector.tpl が存在する必要があります。

9.4 注意事項

アプリケーションのコンパイルによって生成された mrc ファイルを漏れなく指定してください。漏れがある場合、サービスコールモジュールがリンクされない場合があります。リンクされなかったサービスコールを呼び出すとシステムダウンとなります。

10. サンプルプログラム

High-performance Embedded Workshop では、RI600/PX のプロジェクトを生成することができます。本章では、RI600/PX プロジェクトについて解説します。

10.1 サンプルプログラムの概要

ドメインは、「マスタドメイン」、「アプリケーションドメイン A」、「アプリケーションドメイン B」の 3 つです。

マスタドメイン(domid #1)は、「信頼されたドメイン」です。マスタドメインは、アプリケーションドメイン A・B の実行に必要な各種カーネルオブジェクトを動的に生成します。マスタドメインに属するタスク(MasterDom_Task)は、cfg ファイルによって静的に生成・起動されます。

アプリケーションドメイン A(domid #2)とアプリケーションドメイン B(domid #3)は、「信頼されたドメイン」ではありません。

アプリケーションドメイン A に属するタスクは AppDomA_Task、アプリケーションドメイン B に属するタスクは AppDomB_Task です。

AppDomA_Task と AppDomB_Task は、セマフォ(ID_SEM1)を使って排他制御しながら共有変数 g_ulSharedData をアクセスします。

また、AppDomA_Task は、データキュー(ID_DTQ1)にデータを送信し、AppDomB_Task はそれを受信します。

表10.1　カーネルオブジェクト

種別	IDなど	解説
ドメイン	1	• マスタドメイン • 信頼されたドメイン • 所属タスク：MasterDom_Task
	2	• アプリケーションドメイン A • 信頼されていないドメイン • 所属タスク：AppDomA_Task
	3	• アプリケーションドメイン B • 信頼されていないドメイン • 所属タスク：AppDomB_Task
タスク	ID_MASTERDOMTASK	cfgファイルで生成・起動される。
	ID_DOM_A_TASK	MasterDom_Taskによって生成・起動される。
	ID_DOM_B_TASK	MasterDom_Taskによって生成・起動される。
セマフォ	ID_SEM1	• MasterDom_Task によって、生成される。 • AppDomA_Task と AppDomB_Task からの変数"g_ulSharedData"へのアクセスの排他制御に使用。
データキュー	ID_DTQ1	• MasterDom_Task によって、生成される。 • AppDomA_Task と AppDomB_Task の間での通信に使用。 • データキュー領域は BS セクションとして生成される。BS セクションは非メモリオブジェクト。
可変長メモリプール	ID_MPL1	• MasterDom_Task によって、生成される。 • 単なるダミー • プール領域は BU_SH セクションであり、memory_object[4]内。
周期ハンドラ	ID_CYC1	• cfg ファイルで生成される。 • AppDomA_Task と AppDomB_Task のレディキューを回転する。
アラームハンドラ	ID_ALM1	• cfg ファイルで生成される。 • 単なるダミー
割り込みハンドラ	可変ベクタ#64	• cfg ファイルで定義される。 • 単なるダミー

10.2 RI600/PX プロジェクトの生成

1. [ようこそ!]ダイアログボックスから[新規プロジェクトワークスペースの作成]オプションを選択し、[OK]ボタンをクリックするか、[ファイル -> 新規ワークスペース]を選択してください。[新規プロジェクトワークスペース]ダイアログボックスが開きます。
2. [ワークスペース名]に新規ワークスペース名を入力してください。
3. [CPU種別]として"RX"、[ツールチェイン]として"Renesas RX Standard"を選択してください。
4. [プロジェクトタイプ]として"Application"を選択し、[OK]ボタンをクリックしてください。選択したプロジェクトの作成をガイドするウィザードが開きます。
5. [新規プロジェクト - 2/11- Select RTOS]で、[RTOS]として"RI600/PX"を選択してください。

10.3 生成ファイル

ここでは、主要な生成ファイルについて説明します。

(1) resetprg.c

「7.2 スタートアップファイル(resetprg.c)の作成」を参照してください。

(2) <プロジェクト名>.cfg

cfg ファイルです。
clock.template の設定は、使用する MCU に応じて変更してください。

(3) <プロジェクト名>.c

表 10.1に示したタスク・ハンドラが実装されています。

10.4 メモリマップ

[ ]で示したセクションは、16 バイト境界に配置するためにリンカの aligned_section オプションを指定しています。

10.4.1 RAM 領域

表10.2 RAM 領域

アドレス	セクション並び(リンカ設定)	解説	メモリオブジェクト
0～0x0001FFFF	SI	システムスタック	非メモリオブジェクト
	BRI_RAM,RRI_RAM	カーネルデータ	
	BS,BS_1,BS_2,RS,RS_1,RS_2	ハンドラ専用データ	
	[SURI_STACK]	ユーザスタック	
	[BU_MASTERDOM],BU_MASTERDOM_1, BU_MASTERDOM_2, RU_MASTERDOM,RU_MASTERDOM_1, RU_MASTERDOM_2	マスタドメイン専用データ	memory_object[1]
	[BU_DOM_A],BU_DOM_A_1,BU_DOM_A_2, RU_DOM_A,RU_DOM_A_1,RU_DOM_A_2	アプリケーションドメインA専用データ	memory_object[2]
	[BU_DOM_B],BU_DOM_B_1,BU_DOM_B_2, RU_DOM_B,RU_DOM_B_1,RU_DOM_B_2	アプリケーションドメインB専用データ	memory_object[3]
	[BURI_HEAP],BU_SH,BU_SH_1,BU_SH_2, RU_SH,RU_SH_1,RU_SH_2	共有データ	memory_object[4]

10.4.2 ROM 領域

表10.3 ROM 領域

アドレス	セクション並び(リンカ設定)	解説	メモリオブジェクト
0xFFFF0000～ 0xFFFFFF7F	[PU_MASTERDOM],CU_MASTERDOM, CU_MASTERDOM_1,CU_MASTERDOM_2, DU_MASTERDOM, DU_MASTERDOM_1,DU_MASTERDOM_2	マスタドメイン専用コード・定数	memory_object[5]
	[PU_DOM_A],CU_DOM_A,CU_DOM_A_1, CU_DOM_A_2,DU_DOM_A,DU_DOM_A_1, DU_DOM_A_2	アプリケーションドメインA専用コード・定数	memory_object[6]
	[PU_DOM_B],CU_DOM_B,CU_DOM_B_1, CU_DOM_B_2,DU_DOM_B,DU_DOM_B_1, DU_DOM_B_2	アプリケーションドメインB専用コード・定数	memory_object[7]
	[PU_SH],WU_SH,WU_SH_1,WU_SH_2,LU_SH, CU_SH,CU_SH_1,CU_SH_2, DU_SH,DU_SH_1,DU_SH_2	共有コード・定数	memory_object[8]
	INTERRUPT_VECTOR	可変割り込みベクタテーブル	非メモリオブジェクト
	PRI_KERNEL	カーネルコード	
	CRI_ROM,DRI_ROM	カーネル定数	
	C$*,PS,CS,CS_1,CS_2,DS,DS_1,DS_2	ハンドラ専用コード・定数	
0xFFFFFF80～ 0xFFFFFFFF	FIX_INTERRUPT_VECTOR	固定割り込みベクタテーブル	

10.4.3 メモリオブジェクト

メモリオブジェクトは全部で 8 個あり、cfg ファイルで登録します。

メモリオブジェクトは 16 バイト境界のアドレスから始まらなければなりません。このため、リンク時にメモリオブジェクトの先頭セクション(start_address)に対して aligned_section オプションを指定する必要があります。

```
// Master domain data
memory_object[1]{
    start_address = BU_MASTERDOM;
    end_address   = RU_MASTERDOM_2;
    acptn1       = 0x0001;              マスタドメインのみドリードアクセス可能
    acptn2       = 0x0001;              マスタドメインのみライトアクセス許可
    acptn3       = 0;                   全ドメインの実行禁止
};
```

図10.1 memory_object[1]：**マスタドメイン専用データ**

```
// App-domain A data
memory_object[2]{
    start_address = BU_DOM_A;
    end_address   = RU_DOM_A_2;
    acptn1       = 0x0002;          アプリケーションドメイン A のみリード可能
    acptn2       = 0x0002;          アプリケーションドメイン A のみライト可能
    acptn3       = 0;               全ドメインの実行禁止
};
```

図10.2 memory_object[2]：**アプリケーションドメイン** A **専用データ**

```
// App-domain B data
memory_object[3]{
    start_address = BU_DOM_B;
    end_address   = RU_DOM_B_2;
    acptn1       = 0x0004;          アプリケーションドメイン B みリード可能
    acptn2       = 0x0004;          アプリケーションドメイン B みライト可能
    acptn3       = 0;               全ドメインからの実禁止
};
```

図10.3 memory_object[3]：**アプリケーションドメイン** B **専用データ**

```
// Shared data
memory_object[4]{
    start_address = BURI_HEAP;
    end_address   = RU_SH_2;
    acptn1       = TACP_SHARED;         全ドメインからリード可能
    acptn2       = TACP_SHARED;         全ドメインからライト可能
    acptn3       = 0;                   全ドメインの実行禁止
};
```

図10.4 memory_object[4]：**共有データ**

```
// Master domain code and const
memory_object[5]{
    start_address = PU_MASTERDOM;
    end_address   = DU_MASTERDOM_2;
    acptn1       = 0x0001;              マスタドメインのみリード可能
    acptn2       = 0;                   全ドメインからライト禁止
    acptn3       = 0x0001;              マスタドメインのみ実行可能
};
```

図10.5 memory_object[5]：**マスタドメイン専用コード・定数**

```
// App-domain A code and const
memory_object[6]{
    start_address = PU_DOM_A;
    end_address   = DU_DOM_A_2;
    acptn1       = 0x0002;          アプリケーションドメイン A のみリード可能
    acptn2       = 0;               全ドメインからライト禁止
    acptn3       = 0x0002;          アプリケーションドメイン A ののみ実行可能
};
```

図10.6 memory_object[6]：**アプリケーションドメイン** A **専用コード・定数**

```
// App-domain B code and const
memory_object[7]{
    start_address = PU_DOM_B;
    end_address   = DU_DOM_B_2;
    acptn1       = 0x0004;          アプリケーションドメイン B のみリード可能
    acptn2       = 0;               全ドメインからライト禁止
    acptn3       = 0x0004;          アプリケーションドメイン B ののみ実行可能
};
```

図10.7 memory_object[7]：**アプリケーションドメイン** B **専用コード・定数**

```
// Shared code and const
memory_object[8]{
    start_address = PU_SH;
    end_address   = DU_SH_2;
    acptn1       = TACP_SHARED;     全ドメインからのリード可能
    acptn2       = 0;               全ドメインからライト禁止
    acptn3       = TACP_SHARED;     全ドメインの実行可能
};
```

図10.8 memory_object[8]：**共有コード・定数**

10.4.4 ユーザスタック

ユーザスタックはメモリオブジェクト外とする必要があります。本サンプルでは、全タスクのユーザスタックをデフォルトと同じ SURI_STACK セクションとしています。

(1) MasterDom_Task のユーザスタック

MasterDom_Task は cfg ファイルで静的に生成されます。

```
task[]{
    name          = ID_MASTERDOMTASK;
    entry_address   = MasterDom_Task();
    initial_start   = ON;
    stack_size     = 256;
    priority       = 1;
//  stack_section   = SURI_STACK;
    exinf         = 1;
    domain_num     = 1;
};
```

"stack_section"を省略すると、ユーザスタックは SURI_STACK セクションに生成されます。

図10.9 MasterDom_Task の生成

(2) AppDomA_Task と AppDomB_Task のユーザスタック

AppDomA_Task と AppDomB_Task は、MasterDom_Task が呼び出す acre_tsk によって動的に生成されます。acre_tsk には、それぞれのタスクのユーザスタックの先頭アドレスとサイズを渡します。

AppDomA_Task と AppDomB_Task のスタック領域は、共に#pragma section ディレクティブを使って SURI_STACK セクションに生成しています。

```
/************************************************************************
 Stack for AppDomA_Task and AppDomB_Task
************************************************************************/
#define DOM_A_STKSZ    0x100         // AppDomA_Task's stack size
#define DOM_B_STKSZ    0x100         // AppDomB_Task's stack size
#pragma section B SURI_STACK
static UW      s_ulDomA_Stk[DOM_A_STKSZ];     // Stack area for AppDomA_Task
static UW      s_ulDomB_Stk[DOM_B_STKSZ];     // Stack area for AppDomB_Task
```

図10.10 AppDomA_Task と AppDomB_Task のユーザスタック

10.5 セクションに関するビルドツールの設定

10.5.1 標準ライブラリ構築ツール

標準ライブラリのセクションは全ドメインからアクセス可能なメモリオブジェクトとしています。

表10.4 標準ライブラリのセクション

領域	セクション	メモリオブジェクト
コード	PU_SH	memory_object[8]
定数	CU_SH	
リテラル	LU_SH	
switch**文分岐テーブル**	WU_SH,WU_SH_1,WU_SH2_	
初期化データ	DU_SH,DU_SH_1,DU_SH_2	
未初期化データ	BU,BU_SH_1,BU_SH_2	memory_object[4]
初期化データ(RAM) (**リンカで指定**)	RU_SH,RU_SH_1,RU_SH_2	

10.5.2 C/C++コンパイラ

デフォルトのセクションは表 10.4と同じとし、これ以外のセクションとする場合には、個別にソースコードに#pragma section ディレクティブを記述することでセクションを切り替えることとしています。

10.5.3 リンカ

以下のセクションについて、aligned_section オプションを指定する必要があります。

1. memory_object[].start_addressに指定したセクション
2. task[].stack_sectionに指定したセクション(本サンプルでは指定していません)
3. SURI_STACKセクション

このため、メモリオブジェクトの先頭セクションと SURI_STACK に aligned_section を指定しています。

10.6 アクセス違反の対処例

本サンプルでは、以下の例を実装しています。詳細はサンプルコードを参照してください。

- タスク例外を発生させ、タスク例外処理ルーチンでタスクを再起動する。
- longjmp()を使って、正常な時点からやり直す。

11. スタック使用量の算出

11.1 スタックの種類

スタックには、システムスタックとユーザスタックの 2 種類があります。スタックサイズの計算方法は、ユーザスタックとシステムスタックで異なります。

- ユーザスタック
  タスクのスタックを、ユーザスタックと呼びます。ユーザスタックは、CPU の USP レジスタによってアクセスされます。
  ユーザスタックサイズは 16 の倍数、ユーザスタック領域の先頭は 16 バイト境界のアドレスでなければなりません。そうでない場合はエラーが検出されます。表 11.1に、ユーザスタックの指定方法を示します。

表11.1 ユーザスタックの指定方法

項目	cfg ファイルでの静的な生成(task[]定義)	サービスコールによる動的な生成 (cre_tsk, acre_tsk)
ユーザスタックサイズ	task[].stack_size に指定します。 16 の倍数でない場合は cfg600px がエラーを報告します。	T_CTSK.stksz に指定します。 16 の倍数でない場合は、cre_tsk, acre_tsk がエラーを返します。
ユーザスタック領域	cfg600px が task[].stack_section で指定したセクション内に生成します。 ユーザスタック領域の先頭が 16 バイト境界でない場合は、vsta_knl がエラーを検出し、システムダウンとなります。	T_CTSK.stk から stksz バイトをスタック領域として使用します。この領域はアプリケーション側で確保する必要があります。 スタック領域の先頭が 16 バイト境界でない場合は、cre_tsk, acre_tsk がエラーを返します。

- システムスタック
  タスク以外、すなわち各種ハンドラやカーネルが共通に使用するスタックで、システムにひとつだけ存在します。システムスタックサイズは、cfg ファイルの system.stack_size に指定します。セクション名は SI です。
  システムスタックは、CPU の ISP レジスタによってアクセスされます。

11.2 "Call Walker"

コンパイラパッケージには、スタック算出ユーティリティである Call Walker が付属しています。Call Walker を使用すると、各関数ツリーで消費されるスタックサイズを確認することができます。

11.3 ユーザスタック使用量の算出

各タスクのユーザスタックの使用量は、以下の式で算出された値を 16 の倍数に切り上げた値です。

$$\text{ユーザスタックの使用量} = treesz\_task + ctxsz\_task + treesz\_tex + ctxsz\_tex$$

treesz_task：

タスク開始関数を起点とする関数ツリーで消費されるサイズ(Call Walker表示サイズ)です。

treesz_tex：

タスク例外処理ルーチン開始関数を起点とする関数ツリーで消費されるサイズ(Call Walker表示サイズ)です。タスク例外処理ルーチンを使用しない場合は、*treesz_tex*は0です。

ctxsz_task, ctxsz_tex：

タスクコンテキストサイズです。*ctxsz_task*はタスク用、*ctxsz_tex*はタスク例外処理ルーチン用です。
タスク例外処理ルーチンを使用しない場合は、*ctxsz_tex*は0です。。
タスクコンテキストサイズは、system.contextの設定によって異なります。表11.2を参照してください。

表11.2 タスクコンテキストサイズ

system.context	タスクコンテキストサイズ(バイト)
NO	68
FPSW	72
ACC	76
FPSW,ACC	80
MIN	44
MIN,FPSW	48
MIN,ACC	52
MIN,FPSW,ACC	56

11.4 システムスタック使用量の算出

システムスタックを最も多く消費するのは、サービスコール処理中[6]に割り込みが発生、さらに多重割り込みが発生した場合です。すなわち、システムスタックの必要量 (最大サイズ)は以下の計算式で算出することができます。

$$\text{システムスタックの使用量} = svcsz + (\sum_{k=1}^{15} treesz\_inthdr_k) + sysdwnsz$$

svcsz：

システムで使用するサービスコールの中で最大のシステムスタックサイズです。svcszの値は、カーネルのバージョンによって異なります。詳細はリリースノートを参照してください。

treesz_inthdr$_k$：

各割り込みハンドラ開始関数を起点とする関数ツリーで消費されるサイズ(Call Walker表示サイズ)です。
kは、割り込み優先レベルです。同じ優先レベルの割り込みが複数ある場合は、それらのハンドラの中で最大のサイズを*treesz_inthdr*$_k$としてください。
なお、システムクロック割り込みハンドラ(割り込み優先レベル = clock.IPL)の使用サイズは、以下に示す3つのサイズの最大値となります。*clocksz1*, *clocksz2*および*clocksz3*についてはリリースノートを参照してください。
なお、システムタイマを使用しないとき(clock.timer=NOTIMER)は、システムクロック割り込みハンドラが使用するサイズをシステムスタックサイズに加算する必要はありません。

- *clocksz1* + *cycsz*
- *clocksz* 2 + *almsz*
- *clocksz* 3

♦*cycsz*
周期ハンドラ開始関数を起点とする関数ツリーで消費されるサイズ(Call Walker表示サイズ)。
周期ハンドラが複数ある場合は、それらのハンドラの中で最大のサイズを*cycsz*としてください。

♦*almsz*
アラームハンドラ開始関数を起点とする関数ツリーで消費されるサイズ(Call Walker表示サイズ)。
アラームハンドラが複数ある場合は、それらのハンドラの中で最大のサイズを*almsz*としてください。

sysdwnsz：

システムダウンルーチン開始関数を起点とする関数ツリーで消費されるサイズ(Call Walker表示サイズ) + 40としてください。システムダウンルーチンに遷移するケースがない場合は、*sysdwnsz*を0としてください。

[6] ユーザスタックからシステムスタックに切り替えた後

RI600/PX V.1.00 **ユーザーズマニュアル**

発行年月日　2011 **年** 9 **月** 1 **日**　Rev.1.00

発行　**ルネサス エレクトロニクス株式会社**
〒211-8668　**神奈川県川崎市中原区下沼部** 1753

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）　(03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/inquiry

Colophon 1.0

RI600/PX V.1.00

ルネサス エレクトロニクス株式会社

R20UT0606JJ0100