

お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日

ルネサスエレクトロニクス株式会社

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

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

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したのですが、誤りが無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、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とその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

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

HI7000シリーズ テクニカルQ&A

アプリケーションノート

ルネサスF-ZTAT™マイクロコンピュータ

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

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

本版で改訂させた箇所

修正項目	頁	修正内容（詳細はマニュアル参照）
全体	-	社名変更に伴い、マニュアル本文中の「日立製作所」、「株式会社日立製作所」、「日立半導体」、「日立 XX」という表記をすべて「株式会社ルネサステクノロジ」に変更。カテゴリー呼称の「シリーズ」を「グループ」に変更。

はじめに

ルネサス SuperH RISC engine ファミリは、高性能の演算処理を実現すると共に各種周辺機器を内蔵し、低消費電力で動作する機器組み込み用新世代シングルチップ RISC マイコンです。

本アプリケーションノートでは、ルネサス SuperH RISC engine ファミリのこれらの機能・性能をさらに活かすために、 μ ITRON*仕様に準拠した機器組み込み用リアルタイム・マルチタスク OS (Operating System)、HI7000 シリーズ (Renesas Industrial Realtime Operating System SH7000 Series) を使用する上でユーザから多く寄せられた質問についての回答を記載しています。

なお、本アプリケーションノートに掲載されているコーディング例を実際にご使用になる場合は、必ず動作確認の上ご使用くださいますようお願いいたします。

関連マニュアル

関連マニュアルは以下のとおりです。

- ・ HI7000 ユーザーズマニュアル
- ・ HI7000 構築マニュアル
- ・ HI7700 ユーザーズマニュアル
- ・ HI7700 構築マニュアル
- ・ SH シリーズ C コンパイラユーザーズマニュアル
- ・ SuperH RISC engine ファミリ C コンパイラ編アプリケーションノート
- ・ 使用する SH マイコンのハードウェアマニュアル
- ・ 使用する SH マイコンのプログラミングマニュアル
- ・ HI7000 リリースノート
- ・ HI7400 リリースノート
- ・ HI7700 リリースノート

【注】* μ ITRON は、“ Micro Industrial TRON TRON ” の略称です。

TRON は、“ The Real Time Operating system Nucleus Nucleus ” の略称です。

- 目次 -

1. 構築関連	1-1
1.1 タスクの登録	1-1
1.2 同一優先度タスクの登録	1-2
1.3 初期登録タスク以外でのタスク登録	1-3
1.4 タスクの起動（初期登録）	1-4
1.5 HI7000 シリーズのエンディアン	1-5
2. 初期化関連	2-1
2.1 セクションの初期化	2-1
2.2 スタック領域の初期化	2-3
2.3 周辺デバイスの初期化	2-5
2.4 メッセージ領域の初期化	2-10
2.5 メモリブロック領域の初期化	2-11
3. タスク関連	3-1
3.1 タスクの生成	3-1
3.2 休止（DORMANT）状態のタスクの情報	3-3
3.3 タスクの待ちと起床要求のタイミング	3-4
3.4 共有スタックの定義方法	3-6
4. スケジューリング関連	4-1
4.1 ラウンドロビンスケジューリングの実現方法	4-1
4.2 タスクの沈み込み	4-5
4.3 タスクのデッドロック	4-6
5. 事象・同期・通信関連	5-1
5.1 タスクの待ち解除とタイムアウトのタイミング	5-1
5.2 待ちとポーリングの違い	5-3
5.3 イベントフラグのクリア	5-4
5.4 タイムアウト監視中のタスクとシステムクロック	5-6
5.5 待ち状態に入るシステムコールでのタイムアウト指定	5-7
5.6 タスク間の簡単な同期処理	5-8
5.7 タスクの起床要求のキューイングと解除	5-9
5.8 タスク付属同期機能の使用例	5-10
5.9 イベントフラグの使用例	5-11
5.10 セマフォの使用例	5-12
5.11 メールボックスの使用例	5-13
6. オブジェクト関連	6-1
6.1 オブジェクトの ID 番号	6-1
6.2 イベントフラグの初期値	6-2
6.3 セマフォの初期値	6-3

6.4	メールボックスの初期化.....	6-4
6.5	メモリブロック・ページブロックの初期化.....	6-5
6.6	オブジェクト数とメモリ (RAM) サイズの関係.....	6-6
7.	標準関数関連	7-1
7.1	関数呼び出しと拡張 SVC の違い.....	7-1
7.2	拡張 SVC の発行.....	7-3
7.3	リエントラントな共通関数.....	7-5
7.4	共通関数の排他制御.....	7-6
8.	メモリ関連	8-1
8.1	カーネルの RAM 領域を小さくしたい.....	8-1
8.2	カーネルの ROM 領域を小さくしたい.....	8-3
8.3	タスクのスタック領域を小さくしたい.....	8-4
8.4	共通関数の排他制御.....	8-5
9.	割込み関連	9-1
9.1	割込みハンドラの動き.....	9-1
9.2	割込みハンドラが起動しない.....	9-2
9.3	割込みハンドラのスタック切り替え方法.....	9-3
9.4	割込みハンドラの終了方法.....	9-5
9.5	割込み同時発生時の動作順序.....	9-6
9.6	未定義割込みの発生.....	9-7
9.7	割込み許可の方法.....	9-8
9.8	カーネル実行中の割込み発生.....	9-9
10.	タイマ、タイマハンドラ関連	10-1
10.1	システムクロックの値.....	10-1
10.2	システムクロックが遅れる (誤差が生じる).....	10-2
10.3	タイマデバイスを変えたい.....	10-3
11.	ハードウェア関連	11-1
11.1	必要となるハードウェア.....	11-1
11.2	レジスタの使用制限.....	11-2
11.3	レジスタ値の変更.....	11-7
12.	デバッグ関連	12-1
12.1	タスクの実行順序を見る.....	12-1
12.2	トレースバッファの指定と使い方.....	12-2
13.	その他	13-1
13.1	プログラムを特定アドレスからスタートしたい.....	13-1
13.2	システムコールの発行方法.....	13-2

Q&A 集

ユーザから多く寄せられた質問についての回答を記載します。記述形式は図に示すようになっています。

X.X タイトル	
質 問	質問内容です。
回 答	質問に対する回答です。

図 記述形式

1. 構築関連

1.1 タスクの登録

質 問

カーネルビルドファイル (hibuild.usr) またはセットアップファイル (hisetup.usr) にタスクを登録する場合、定義する情報を教えてください。

回 答

タスクを登録する場合、以下の情報を定義します。

【タスク情報定義パート】

項番	マクロ名	意 味	指定可能範囲
1	hi_maxtskid	最大タスク ID	1 ~ 1023
2	hi_ststskid	スタティックスタックを使用する最大タスク ID	0 ~ hi_maxtskid
3	hi_tskstksz	ダイナミックスタック領域のトータルサイズ	160 以上の 4 の倍数
4	hi_maxtskpri	最大タスク優先度	1 ~ 255
5	hi_maxdspno*	最大 DSP 数	0 ~ 65535

【注】 * HI7400 を使用する場合のみ定義が必要です。アプリケーションで同時に使用する DSP の数を定義します。つまり、DSP 属性を持つタスクやハンドラが同時に実行する最大の数を定義します。0 を定義すると、DSP サポート機能を使用しないという意味になります。

【スタティックスタック情報定義パート】

項番	定義項目	意 味
1	スタックサイズ	スタック領域のサイズ (108 以上の 4 の倍数)
2	スタック配列	スタック領域の確保
3	_0Hinitsp テーブル	スタティックスタック使用タスクのスタック情報

1.2 同一優先度タスクの登録

質 問

セットアップファイルに同一優先度のタスクを複数登録可能ですか。

回 答

同一優先度のタスクを複数登録することは可能です。タスクは、休止 (DORMANT) 状態となります。以下に、同一優先度のタスク登録例を示します。

【セットアップファイル (hisetup.usr) のオブジェクト初期登録パート】

```
CRE_TSK (6, 0, TA_HLNG, usr_task6, 1, 0, 256, 0, TASK6)
CRE_TSK (7, 0, TA_HLNG, usr_task7, 1, 0, 256, 0, TASK7)
CRE_TSK (8, 0, TA_HLNG, usr_task8, 1, 0, 256, 0, TASK8)
```

同一優先度

また、カーネルビルドファイルにおいても、同一優先度のタスクを登録することができます。

【カーネルビルドファイル (hibuild.usr) のオブジェクト初期登録パート】

```
CRE_TSK (6, 0, TA_HLNG, usr_task6, 1, 0, 256, 0, TASK6)
CRE_TSK (7, 0, TA_HLNG, usr_task7, 1, 0, 256, 0, TASK7)
CRE_TSK (8, 0, TA_HLNG, usr_task8, 1, 0, 256, 0, TASK8)
```

同一優先度

1.3 初期登録タスク以外でのタスク登録

質 問

カーネルにタスクを登録する場合、セットアップファイル (hisetup.usr) での初期登録タスク以外での登録が可能ですか。

回 答

カーネルへのタスクの登録方法は以下の2つがあります。

カーネルビルドファイル (hibuild.usr)、およびセットアップファイル (hisetup.usr) でタスクを登録する。(初期登録)

cre_tsk、vcre_tsk、vscr_tsk システムコールでタスクを登録する。

共に、カーネルにタスクが登録され、休止 (DORMANT) 状態となります

また、それぞれの特長を以下に示します。必要に応じて使い分けてください。

1. カーネルビルドファイル、およびセットアップファイルでタスクを登録した場合 (初期登録)

システム起動処理中にタスクはカーネルに登録され、休止 (DORMANT) 状態となります。

たとえば、あるタスクが

(1) システム起動処理後、直ちに動作しなければならない。

(2) 割込みから起動もしくは起床されるため、常に登録されていなければならない。

などの場合は、この方法でタスクを登録してください。

登録したタスクを起動する方法は、STA_TSK マクロによる起動と、システム初期化ハンドラから sta_tsk システムコールを発行し起動する場合があります。

2. cre_tsk、vcre_tsk、vscr_tsk システムコールでタスクを登録した場合

初期登録タスク以外のタスクの登録は、システム初期化ハンドラ、タスクまたは割込みハンドラから cre_tsk、vcre_tsk、vscr_tsk システムコールを発行することにより行ないます。

上記「1. カーネルビルドファイル、およびセットアップファイルでタスクを登録した場合」に該当しない場合は、この方法でタスクを登録することができます。実行可能 (READY) 状態とするには sta_tsk システムコールでタスクを起動します。

1.4 タスクの起動（初期登録）

質 問

カーネルビルドファイル（hibuild.usr）またはセットアップファイル（hisetup.usr）に登録した初期登録タスクはいつ起動しますか。

回 答

カーネルは、カーネルビルドファイル（hibuild.usr）およびセットアップファイル（hisetup.usr）に登録された初期登録タスクは自動で起動することはありません。

カーネルビルドファイル、およびセットアップファイルの「オブジェクト初期登録パート」で STA_TSK マクロを使用することでタスクを起動することができます。

【STA_TSK マクロによるタスクの起動】

オブジェクト初期登録パートへ記述します。

```
CRE_TSK (1, 0, TA_HLNG, usr_task1, 1, 0, 256, 0, TASK1)
```

```
VSCR_TSK (2, 0, TA_HLNG, usr_task2, 2, 0, 0, TASK2)
```

```
VCRE_TSK (3, usr_task3, 3)
```

```
STA_TSK (1, stacd)
```

↑
タスクの起動

また、システム初期化ハンドラ、タスクおよび割込みハンドラから必要に応じ、sta_tsk システムコールで起動します。

1.5 HI7000 シリーズのエンディアン

質 問

HI7000 シリーズを動作させる場合のエンディアンについて教えてください。

回 答

HI7000 シリーズを動作させる場合のエンディアンを以下に示します。

【HI7000 または HI7400】

カーネルは、ビッグエンディアン空間で動作するよう設計されています。また、カーネルがアクセスする領域（カーネルデータ領域、スタック領域、パケット領域、リターンパラメータ格納領域等）もビッグエンディアン空間にする必要があります。

なお、詳細については構築マニュアル「5.2.1 必要なファイルとセクション構成」を参照してください。

【HI7700】

HI7700 は、ビッグエンディアン空間、またはリトルエンディアン空間のどちらでも動作するように設計されています。提供メディアからインストーラを起動し、使用するエンディアンを選択します。

なお、詳細については「HI7700 ソフトウェア引渡し添付資料」を参照してください。

2. 初期化関連

2.1 セクションの初期化

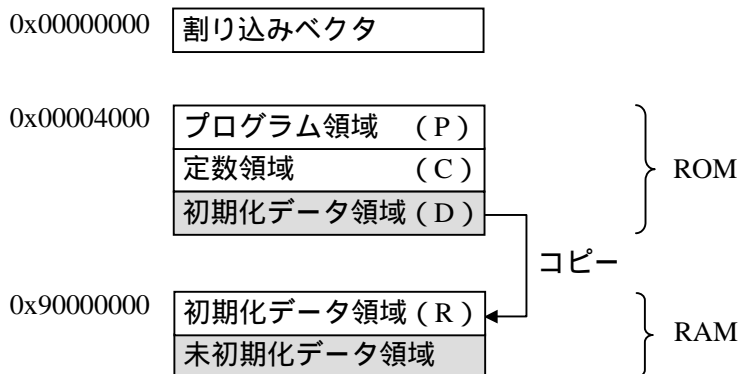
質 問

C 言語で記述したアプリケーションで初期化が必要なセクションがあります。初期化する方法を教えてください。

回 答

リンケージエディタの ROM 化支援機能を有効にし、C 言語で記述したアプリケーションから生成される初期化データ領域 (D) は、実行時に ROM から RAM へコピーしなければなりません。

以下のメモリ割り付け例でコピーの方法を説明します。



コピーは、カーネルが起動する前の「CPU 初期化ルーチン」で行ってください。

ただし、外付け RAM などの初期設定 (バスコントローラなど) や、ROM/RAM チェックが必要な場合は、それらの初期化が終わったあとにコピーを行ってください。

なお、詳細については、SH シリーズ C コンパイラユーザーズマニュアル「3.2 メモリ領域の割り付け」および H シリーズリンケージエディタ、ライブラリアン、オブジェクトコンバータユーザーズマニュアル「2.7 ROM 化支援機能」を参照してください。

以下、HI7000 を例にコピーを行うプログラムのサンプルを示します。

```
extern void _0Hrs_knl(void);          /* kernel reset routine      */
/* extern void _INITSCT(void); */    /* section-initialize routine */

#pragma section _hi_cpuini
#pragma noregsave(hi_cpuini)

void hi_cpuini(void)
{
    register VW *p;                    /* pointer to memory      */
    register unsigned char *p_rom,*p_ram; /* pointer to memory      */

    /*** Initialize Hardware Environment ***/

    /*** Initialize Software Environment ***/

    /*** Section Initialize ***/

    for(p =(VW *)RAMEND; p >=(VW *)RAMSTA; p--) /* RAM clear */
        *p = 0x00000000;

    /* 初期化データを ROM から RAM へコピー */
    for(p_ram = (unsigned char *)_D_BGN, p_rom = (unsigned char *)_D_ROM ;
        p_ram < (unsigned char *)_D_END ; p_ram++,p_rom++)
        *p_ram = *p_rom;

    _0Hrs_knl();                       /* go to kernel reset routine */
}
```

ポインタ p の型は、RAM のアクセスサイズにより変化します。

なお、CPU 初期化ルーチンについては、カーネルユーザズマニュアル「第 4 章 C 言語によるプログラムの作成方法」を参照してください。

2.2 スタック領域の初期化

質 問

タスク、および割り込みハンドラのスタック領域は初期化されますか。

回 答

カーネルは、スタック領域の初期化を行っていません。初期化が必要な場合は、ユーザプログラムで実施してください。

タスク用スタック領域の定義は、セットアップファイル (hisetup.usr) の「タスク情報定義パート」および「スタティックスタック情報定義パート」で行います。なお、詳細は「構築マニュアル」を参照してください。

【タスク情報定義パート】

ダイナミックスタック領域のトータルサイズを定義します。

```
/*#####*/
/* [2. Task information] */
/* (中略) */
/*#####*/
#define hi_maxtskid 10 /* (1)Maximum task ID */
#define hi_stskid 5 /* (2)Maximum task ID using static stack */
#define hi_tskstksz 0x10000 /* (3)Task stack area size */
#define hi_maxtskpri 5 /* (4)Maximum task priority */
```

↑ 各タスクのスタティックスタック領域のサイズ

【スタティックスタック情報定義パート】

各タスクのスタティックスタック領域のサイズを定義します。

```
/*#####*/
/* [3. Static task stack information] */
/* (中略) */
/*#####*/
/***** (1)Each stack area size *****/
#define stksz1 0x100 /* (1-1)Stack-1 size */
#define stksz2 0x200 /* (1-2)Stack-2 size */
#define stksz3 0x100 /* (1-3)Stack-3 size */
#define stksz4 0x200 /* (1-4)Stack-4 size */
```

↑ ダイナミックスタック領域のトータルサイズ

割り込みハンドラ用スタック領域の定義方法を以下に示します。

【HI7000 または HI7400】

割り込みレベル単位でスタック領域を確保します。

以下に、カーネル割り込みマスクレベル以下の割り込みハンドラでのスタック領域確保を示します。

```
#define low_irq_stksz 192
VW    low_irq_stk[low_irq_stksz / sizeof(VW)];
static const VP p_low_irq_stk = (VP)&low_irq_stk[low_irq_stksz / sizeof(VW)];
#pragma interrupt (low_irq_hdr(sp = p_low_irq_stk, tn = 25))
void   low_irq_hdr(void)
{
  ID    tskid;          /* task id          */
  ER    ercd;          /* error code       */
      . . . . .
      . . . . .
      ercd = wup_tsk(tskid); /* wakp up task */
      . . . . .
      . . . . .
}

```

割り込みハンドラで使用するスタックの確保

【HI7700】

セットアップファイルの「割り込みハンドラスタック情報定義パート」で行います。なお、詳細は各カーネルの「構築マニュアル」を参照してください。

```
/*#####*/
/* [7. Interrupt Handler stack Information] */
/* (中略) */
/*#####*/
#define hi_irqstksz 1024 /* Interrupt Handler stack size */

```

割り込みハンドラのスタック領域のサイズ

2.3 周辺デバイスの初期化

質問

周辺デバイスの初期化はいつ行いますか。

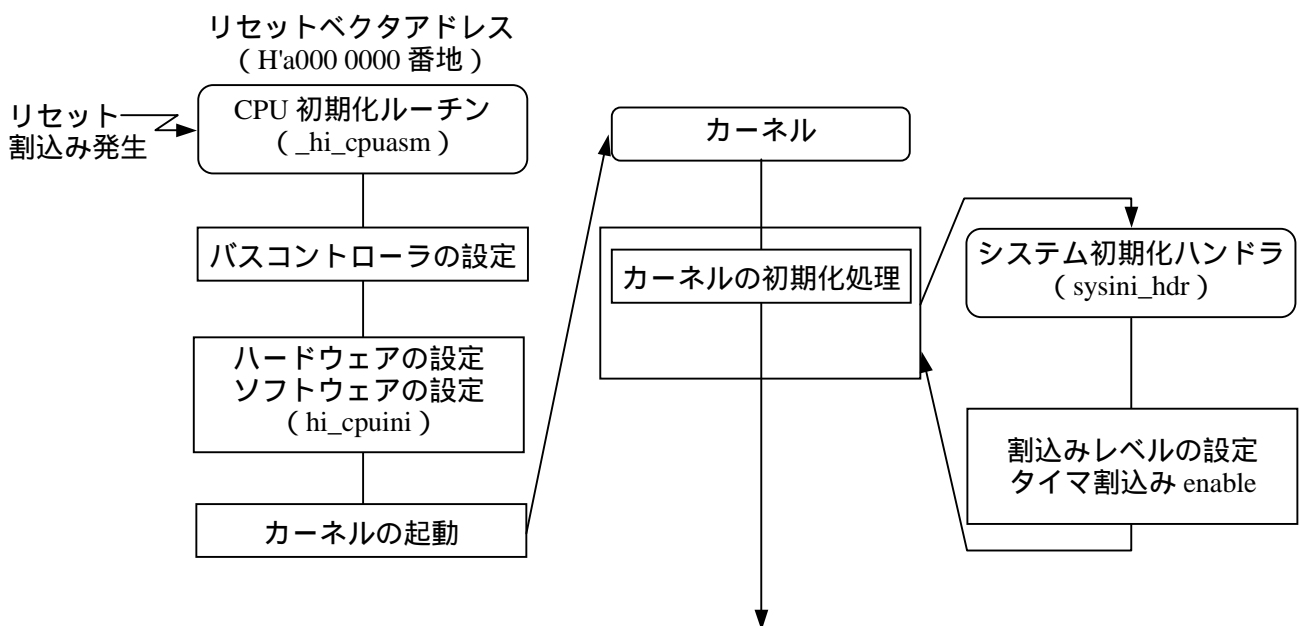
回答

ハードウェアの初期化や、プログラムの実行環境の初期設定は、CPU 初期化ルーチンで行います。

たとえば、バスコントローラ、ポートの初期設定、初期化データ領域のコピーなどが該当します。ただし、C 言語記述の CPU 初期化ルーチンを実行する前に、メモリの使用環境整備（カーネル動作前に、メモリのリード/ライトを可能にする）のために、アセンブリ言語で記述しバスコントローラの初期化を実施する必要があります。

また、タイマや、SCI などカーネルもしくはタスクが動作後でも可能なものはタスクで初期化を行い、タイミング的にカーネルやタスクが動作後に初期化を行う場合は、システム初期化ハンドラで初期化を行います。

以下に、HI7700 の CPU 初期化ルーチンの処理概要を示します。



【HI7700 バスコントローラの初期化例】

```

*****
;
;*          HI7700 CPU initialize routine for SH7708/7702( Ver. 1.0 )      ;*:
;*          Copyright (c) Renesas Technology                               ;*:
;*          Licensed Material of Renesas Technology                       ;*:
;*          HI7700(HS0770ITxE1Sxx) V1.0                                  ;*:
*****
;
;          .program          hicpuasm
;          .heading         "hicpuasm : CPU initialize routine"
;          .export          _hi_cpuasm
;          .import          _hi_cpuini
;          .import          __0Hpon_sp
;          .import          __0Hman_sp
;          .section         P_hicpuasm,code,align=4
;
;
*****
;* EXPEVT address, data
*****
;
CCN_BASE      .equ  h'ffffffd0          ; INTC(exception) base address
EXPEVT       .equ  h'ffffffd4-CCN_BASE ; EXPEVT      address offset
;
;
PON_CODE     .equ  h'000                ; power-on reset exception code
;
;
*****
;* BSC address
*****
;
BSC_BASE     .equ  h'ffffff60          ; BSC      base address
BCR1        .equ  h'ffffff60-BSC_BASE ; BCR1    address offset
BCR2        .equ  h'ffffff62-BSC_BASE ; BCR2    address offset
WCR1        .equ  h'ffffff64-BSC_BASE ; WCR1    address offset
WCR2        .equ  h'ffffff66-BSC_BASE ; WCR2    address offset
MCR         .equ  h'ffffff68-BSC_BASE ; MCR     address offset
DCR         .equ  h'ffffff6a-BSC_BASE ; DCR     address offset
PCR         .equ  h'ffffff6c-BSC_BASE ; PCR     address offset
RTCSR       .equ  h'ffffff6e-BSC_BASE ; RTCSR   address offset
RTCNT       .equ  h'ffffff70-BSC_BASE ; RTCNT   address offset
RTCOR       .equ  h'ffffff72-BSC_BASE ; RTCOR   address offset
RFCR        .equ  h'ffffff74-BSC_BASE ; RFCR    address offset
;
SDMR_CS2    .equ  h'ffffd000          ; SDMR(CS2) base address
SDMR_CS3    .equ  h'ffffe000          ; SDMR(CS3) base address
;
CMF_BIT     .equ  h'0080              ; CMF bit in RTCSR
;

```

```

;*****
;* BSC initial data
;* After reset, you must initialize BSC for memory(stack) access at first.
;* Please modify these definition in order to your hardware.
;*****
BCR1_DATA .equ h'0000 ; BCR1 initial data
BCR2_DATA .equ h'3ffc ; BCR2 initial data
WCR1_DATA .equ h'3fff ; WCR1 initial data
WCR2_DATA .equ h'ffff ; WCR2 initial data
MCR_DATA .equ h'0000 ; MCR initial data
DCR_DATA .equ h'0000 ; DCR initial data
PCR_DATA .equ h'0000 ; PCR initial data
RTCSR_DATA .equ h'a500 + h'00 ; RTCSR initial data
RTCNT_DATA .equ h'a500 + h'00 ; RTCNT initial data
RTCOR_DATA .equ h'a500 + h'00 ; RTCOR initial data
RFCR_DATA .equ h'a400 + h'000 ; RFCR initial data
;
STP_REFRESH .equ h'a500 ; RTCSR initial data(stop count-up)
;
SDMR2_DATA .equ h'0230 ; SDMR_CS2 initial data
SDMR3_DATA .equ h'0230 ; SDMR_CS3 initial data
;
IDLE_TIME .equ h'566 ; loop counter for idle-time
REFRESH_CNT .equ h'8 ; counter for dummy refresh
;
;*****
;*NAME = _hi_cpuasm
;*DATE = 96/10/07 ;
;*FUNCTION = CPU initialize routine ;
;*****
_hi_cpuasm:
;**** Initialize BSC
; mov.l #BSC_BASE,r0 ; set BCR base address to gbr
; ldc r0,gbr
;
; mov.w #BCR1_DATA,r0 ; Initialize BCR1
; mov.w r0,@(BCR1,gbr)
;
; mov.w #BCR2_DATA,r0 ; Initialize BCR2
; mov.w r0,@(BCR2,gbr)
;
; mov.w #WCR1_DATA,r0 ; Initialize WCR1
; mov.w r0,@(WCR1,gbr)
;
; mov.w #WCR2_DATA,r0 ; Initialize WCR2
; mov.w r0,@(WCR2,gbr)
;
; mov.w #MCR_DATA,r0 ; Initialize MCR
; mov.w r0,@(MCR,gbr)
;

```

```

;      mov.w  #DCR_DATA,r0          ; Initialize DCR
;      mov.w  r0,@(DCR,gbr)
;
;
;      mov.w  #PCR_DATA,r0          ; Initialize PCR
;      mov.w  r0,@(PCR,gbr)
;
;
;      mov.w  #STP_REFRESH,r0       ; stop refresh
;      mov.w  r0,@(RTCSR,gbr)
;
;
;      mov.w  #RTCNT_DATA,r0        ; Initialize RTCNT
;      mov.w  r0,@(RTCNT,gbr)
;
;
;      mov.w  #RTCOR_DATA,r0        ; Initialize RTCOR
;      mov.w  r0,@(RTCOR,gbr)
;
;
;      mov.w  #RFCR_DATA,r0         ; Initialize RFCR
;      mov.w  r0,@(RFCR,gbr)
;*** Initialize SDRAM
;
;      mov.l  #IDLE_TIME,r0         ; loop for idle-time
;hicpuasm010:
;      add    #-1,r0
;      cmp/eq #0,r0
;      bf    hicpuasm010
;
;
;      mov.l  #SDMR_CS2,r0          ; Initialize SDMR(CS2)
;      mov.l  #SDMR2_DATA*4,r2
;      mov.b  r1,@(r0,r2)          ; write dummy data(r1)
;
;
;      mov.l  #SDMR_CS3,r0          ; Initialize SDMR(CS3)
;      mov.l  #SDMR3_DATA*4,r2
;      mov.b  r1,@(r0,r2)          ; write dummy data(r1)
;
;
;      mov.w  #RTCSR_DATA,r0        ; Initialize RTCSR
;      mov.w  r0,@(RTCSR,gbr)
;
;
;      mov    #0,r1                 ; loop for dummy refresh
;      mov.w  #REFRESH_CNT,r2
;hi_cpuasm020:
;      mov.w  @(RTCSR,gbr),r0
;      tst    #CMF_BIT,r0          ; check CMF bit
;      bt    hi_cpuasm020
;
;
;      add    #1,r1                 ; loop counter up
;      cmp/eq r1,r2                 ; if end dummy refresh
;      bt    hi_cpuasm030           ; then goto hi_cpuasm030
;      mov.w  #RTCSR_DATA,r0       ; clear CMF bit
;      bra    hi_cpuasm020
;      mov.w  r0,@(RTCSR,gbr)
;
;

```

```

;hi_cpuasm030:
;
;***** Initialize sp and jump to hi_cpuini() written by C-language
mov.l #CCN_BASE,r2 ; get CCN base address
mov.l #PON_CODE,r3 ; get exception code to power-on
mov.l @(EXPEVT,r2),r0 ; get exception code
cmp/eq r3,r0 ; if exception != power-on
bf hi_cpuasm050 ; then hi_cpuasm050
;
mov.l #_0Hpon_sp,r2 ; get stack address
;
hi_cpuasm040:
mov r2,r15 ; set SP
;
mov.l #_hi_cpuini,r0 ; get hi_cpuini address
jmp @r0 ; jump to hi_cpuini()
nop ; never return to this point
;
hi_cpuasm050:
mov.l #_0Hman_sp,r2 ; get stack address
bra hi_cpuasm040
nop
;
.pool
;
.end

```

C 言語で記述した
CPU 初期化ルーチンへジャンプ

上記例を参考に、HI7000、HI7400 カーネルのバスコントローラ初期化ルーチンを作成してください。
また、C 言語による CPU 初期化ルーチンについては、カーネルユーザズマニュアル「C 言語記述による CPU 初期化ルーチンの記述例」を参照してください。

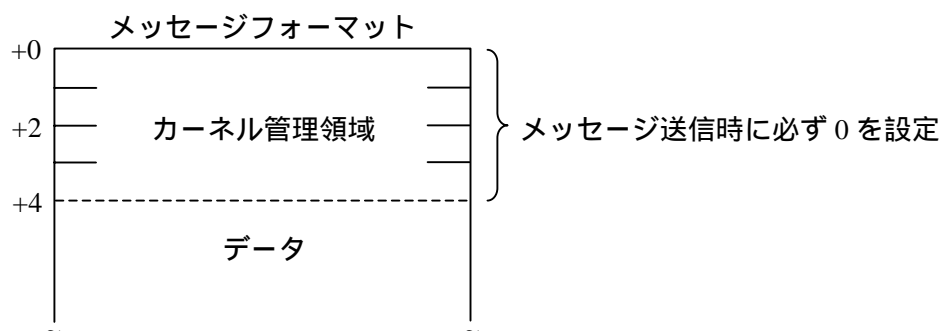
2.4 メッセージ領域の初期化

質 問

メッセージ領域の初期化は必要ですか。

回 答

メッセージは送信する前に先頭の4バイトを0に初期化しなければなりません。



データの領域は、ユーザが自由に使用できる領域です。ユーザ任意のデータを設定してください。
また、メッセージを送信後、受信される前にメッセージのカーネル管理領域を破壊すると、メッセージの送受信が正常に行われません。注意してください。

2.5 メモリブロック領域の初期化

質 問

獲得したメモリブロックの内容は、ゼロクリアされていますか。

回 答

可変長メモリプール、または固定長メモリプールに限らず、メモリプール領域から獲得したメモリブロックの内容は不定です。必要があれば、獲得したブロックのサイズ分初期化してください。サイズを超えて初期化を行うとカーネル管理データが破壊されますので注意してください。

【HI7700】

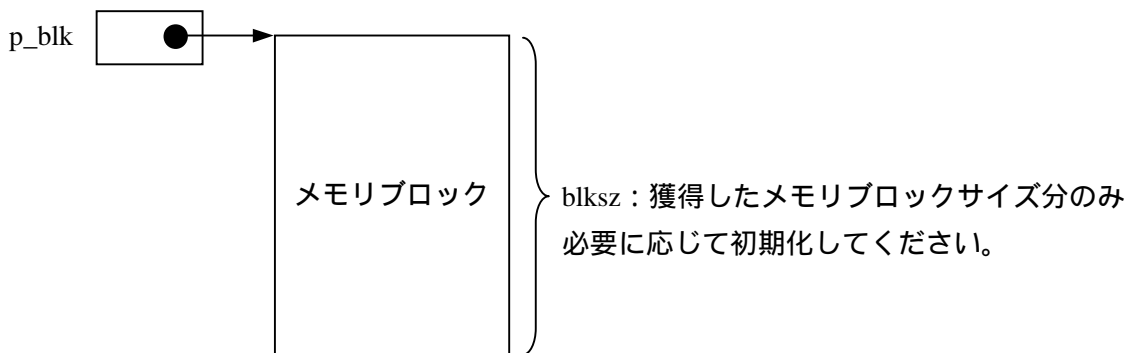
ページブロックの内容も、メモリブロックと同様に不定です。必要であれば、獲得したページブロックのサイズ分初期化してください。

【コーディング例】

```
ER ercd = get_blk(VP *p_blk, ID mplid, INT blkosz);
```

<パラメータ>

VP *p_blk メモリブロックの先頭アドレスを返す領域の先頭アドレス
ID mplid 可変長メモリプール ID
INT blkosz メモリブロックサイズ (バイト数)



3. タスク関連

3.1 タスクの生成

質 問

タスクの生成はどのように行いますか。

回 答

タスクの生成方法には 2 種類あり、生成するタスクが使用するスタックの種類によって分かります。スタックについての詳細は、カーネルユーザズマニュアル「2.5.5 タスクのスタック」を参照してください。

(1) ダイナミックスタックを使用するタスクの生成方法

- ・ cre_tsk システムコール
- ・ vasn_tsk システムコール
- ・ カーネルビルドファイルまたはセットアップファイルでの初期登録
(CRE_TSK マクロ)

【コーディング例】

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

ID	tskid	タスク ID	
T_CTSK	*pk_ctsk	タスク生成情報の先頭アドレス	
typedef	struct	t_ctsk{	
	VP	exinf;	拡張情報
	ATR	tskatr;	タスク属性
	FP	task;	タスク起動アドレス
	PRI	itskpri;	タスク起動時優先度
	UB	cpumode;	タスク CPU モード
	INT	stksz;	タスクスタックサイズ
	TMO	quantum;	タイムスライス時間
	B	*name;	タスク名称を格納した領域の先頭アドレス
		}	T_CTSK;

(2) スタティックスタックを使用するタスクの生成方法

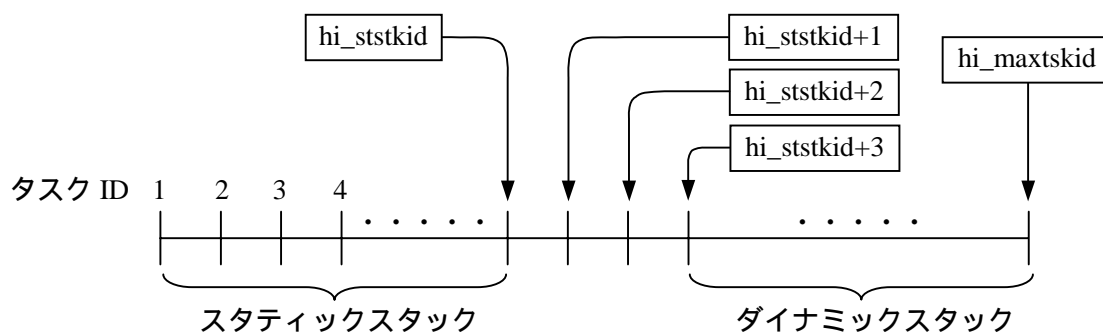
- vcre_tsk システムコール
- vscr_tsk システムコール
- カーネルビルドファイルまたはセットアップファイルでの初期登録
(VCRE_TSK, VSCR_TSK マクロ)

【コーディング例】

```
ER ercd = vscr_tsk((ID) tskid, (T_CTSK *)pk_ctsk);
```

```
typedef struct t_ctsk{  
    VP          exinf;          拡張情報  
    ATR         tskatr;        タスク属性  
    FP          task;          タスク起動アドレス  
    PRI         itskpri;       タスク起動時優先度  
    UB          cpumode;       タスク CPU モード  
    INT         stksz;         (予備)  
    TMO         quantum;      タイムスライス時間  
    B           *name;         タスク名称を格納した領域の先頭アドレス  
} T_CTSK;
```

タスクスタックとタスク ID の関係を以下に示します。



ダイナミックスタック使用タスク ID に空きが生じている状態で vasn_tsk システムコールを発行した場合は、ID の小さい方からカーネルは割り付けを行います。

3.2 休止 (DORMANT) 状態のタスクの情報

質 問

休止 (DORMANT) 状態のタスクは、休止状態に遷移する前のタスクの情報を保持していますか。

回 答

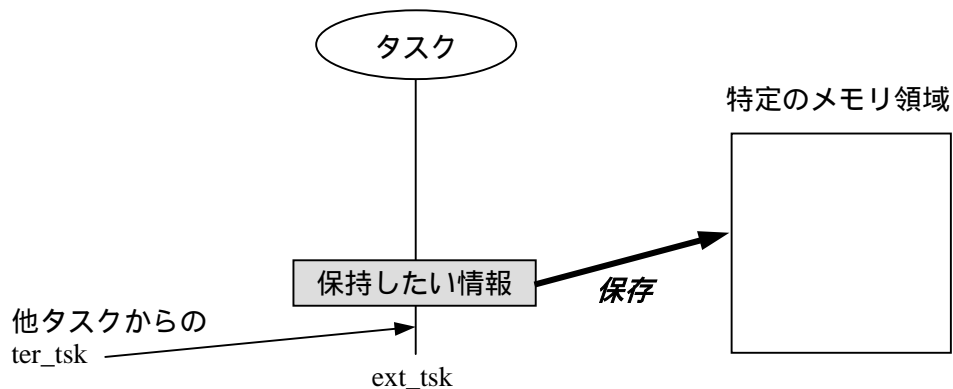
タスクが休止 (DORMANT) 状態に遷移した時点でタスクの情報は失われます。

なお、タスクの情報とは以下のものを指します。

**レジスタ情報、タスクステータス、優先度、起床要求、待ちビットパターン、
待ちイベントフラグ ID、待ちメールボックス ID、待ちセマフォ ID、待ちメモリプール ID**

これらの情報が必要な場合は、ext_tsk システムコールまたは他タスクからの ter_tsk システムコール発行前に、特定のメモリ領域に情報を保存しておく必要があります。

なお、各情報は ref_tsk, can_wup, flg_sts, mbx_sts, sem_sts, mpl_sts システムコールで得ることができます。



3.3 タスクの待ちと起床要求のタイミング

質 問

タスクが slp_tsk システムコールを発行し待ち (WAIT) 状態に遷移中、そのタスクに wup_tsk システムコールが発行された場合、結果はどうなりますか。

回 答

結果は、wup_tsk システムコールが受け付けられるタイミングによって異なります。

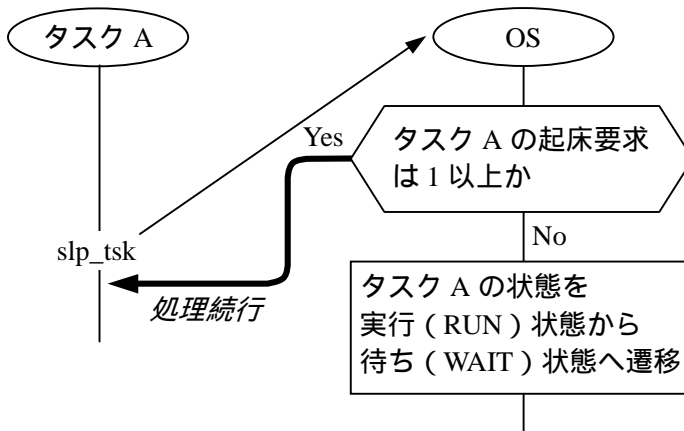
slp_tsk システムコールの処理は、

- (1) 自タスクの起床要求 (wupcnt) を調べ、1 以上なら待ち状態にならずに処理を続行
- (2) 自タスクの起床要求 (wupcnt) が、0 ならタスクを待ち状態に遷移を行います。

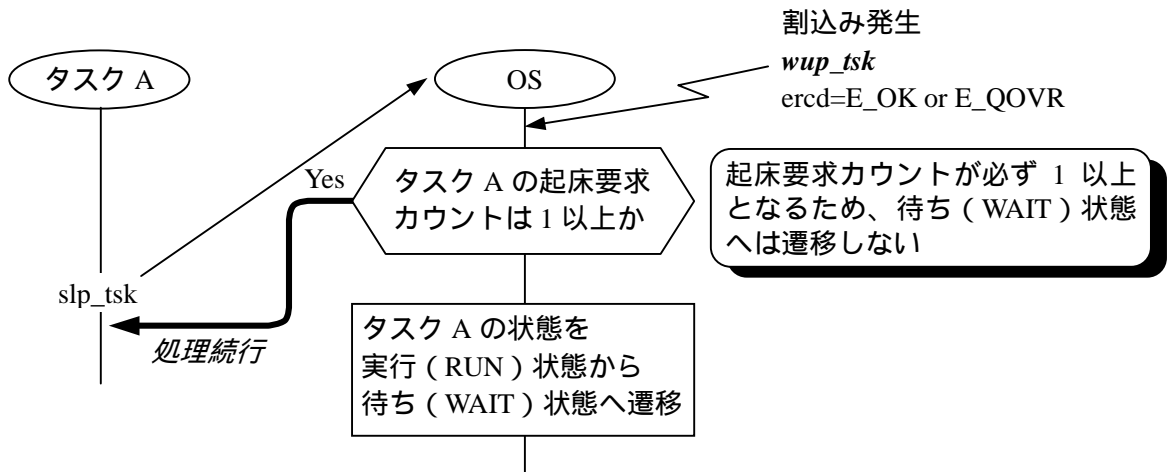
この slp_tsk システムコール処理の前に wup_tsk システムコールが受け付けられた場合、wup_tsk システムコール処理は「タスクは待ち (WAIT) 状態でない」と判断するため、このタスクの起床要求を 1 加算しエラーコードは E_OK となります。このとき、要求数が最大値を超えればエラーコードとして E_QOVR が返ります。このあと、slp_tsk システムコールの処理が行われるため、タスクは待ち (WAIT) 状態にならずに、処理を続行します。

これに対して、slp_tsk システムコール処理の後に wup_tsk システムコールが受け付けられた場合、wup_tsk システムコール処理は「タスクは待ち (WAIT) 状態」と判断するため、このタスクの待ち (WAIT) 状態が解除され常に E_OK となります。

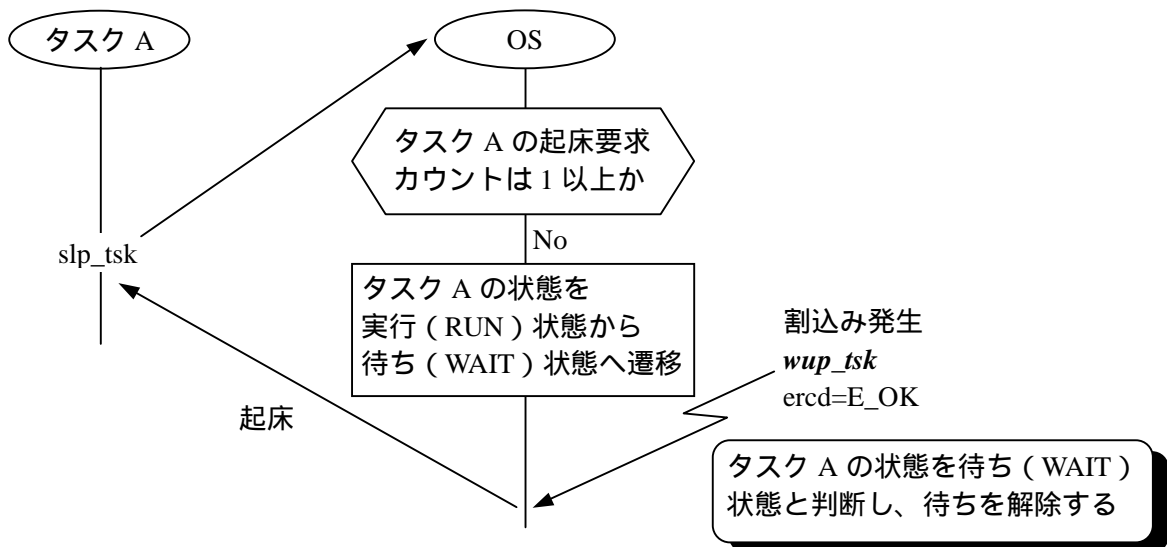
【slp_tsk システムコールの処理】



【wup_tsk 受け付けが slp_tsk 処理の前の場合】



【wup_tsk 受け付けが slp_tsk 処理の後の場合】



3.4 共有スタックの定義方法

質 問

複数のタスクで共有スタックを使用します。定義方法を教えてください。

回 答

スタック共有の定義は、セットアップファイルで行ないます。セットアップファイルでタスクスタックポインタに同じ値を定義したタスク ID 同士は、それらのタスク間で1つのスタックを共有することになります。スタックを共有するタスク群では、同時に1タスクのみがスタックを占有して実行する権利が与えられます。

以下に、スタック共有の具体的定義方法を示します。なお、共有スタック機能の詳細については、カーネルユーザズマニュアル「2.5.6 共有スタック機能」を参照してください。

【セットアップファイルのスタティックスタック情報定義パート】

```
/*#####*/
/* [3. Static task stack information] */
/* (中略) */
/*#####*/
/***** (1)Each stack area size *****/
#define stksz1 0x100 /* (1-1)Stack-1 size */

/* (中略) */

/***** (3)_0Hinitsp *****/
(VP)&stk1[0], /* (3-1) ID1's stack top address */
(VP)&stk1[stksz1/sizeof(VW) - 2], /* ID1's stack bottom address */
(VP)&stk2[0], /* (3-2) ID2's stack top address */
(VP)&stk2[stksz2/sizeof(VW) - 2], /* ID2's stack bottom address */
(VP)&stk3[0], /* (3-3) ID3's stack top address */
(VP)&stk3[stksz3/sizeof(VW) - 2], /* ID3's stack bottom address */
(VP)&stk4[0], /* (3-4) ID4's stack top address */
(VP)&stk4[stksz1/sizeof(VW) - 2], /* ID4's stack bottom address */
(VP)&stk4[0], /* (3-5) ID5's stack top address */
(VP)&stk4[stksz4/sizeof(VW) - 2], /* ID5's stack bottom address */
```

↑ スタック配列 stk4 を ID4 と 5 のタスクで共有

4. スケジューリング関連

4.1 ラウンドロビンスケジューリングの実現方法

質 問

ラウンドロビンスケジューリングを実現する具体的な手段を教えてください。

回 答

ラウンドロビンスケジューリングには、レディキューを操作する `rot_rdq` システムコールを使用する場合と、タイムスライス機能を使用する 2 通りの方法があります。

以下に、ラウンドロビンスケジューリングの実現例を示します。

【`rot_rdq` システムコールによるラウンドロビンスケジューリング】

タイマ機能と `rot_rdq` システムコールを組み合わせ、ラウンドロビンスケジューリングを行うことが可能です。すなわち、一定周期で起動されるタイマ割り込みハンドラから `rot_rdq` システムコールを発行することによってレディキューを回転させます。

以下に、その実現例を示します。

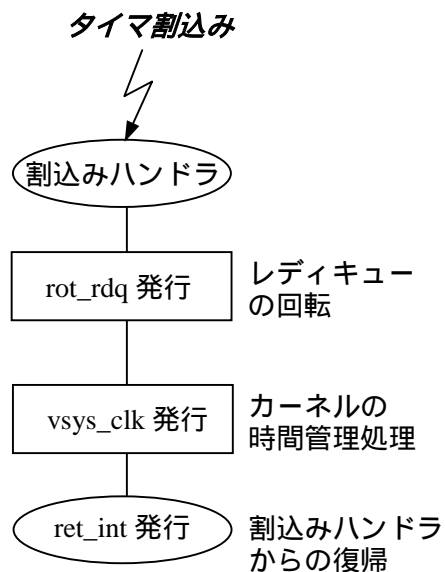
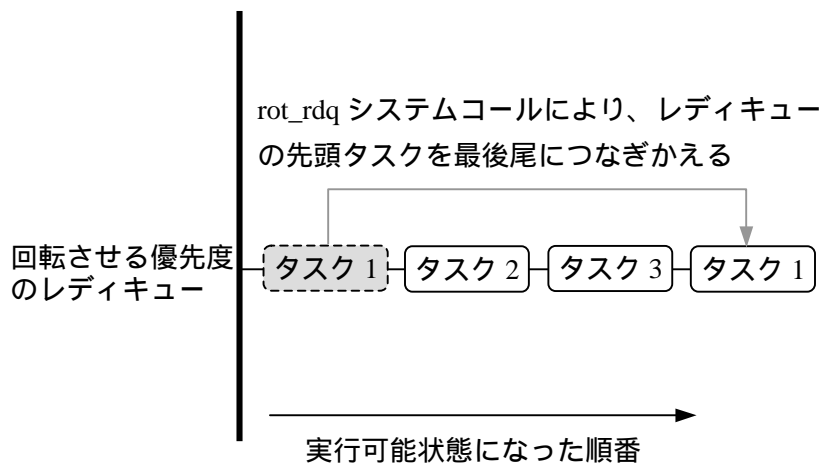
ラウンドロビンスケジューリングを行う基本的な手順は次の通りです。

- (1) ラウンドロビンスケジューリングの対象となる複数のタスクの優先度を同一にします。なる複数のタスクの優先度を同一にします
- (2) タイマを使用し、一定周期で割り込みを発生させます。
- (3) タイマ割り込みハンドラから、対象とするタスクの優先度を指定して `rot_rdq` システムコールを発行後、`vsys_clk` システムコールによりカーネルの時間管理処理を要求し、`ret_int` システムコールにより割り込みハンドラから復帰します。

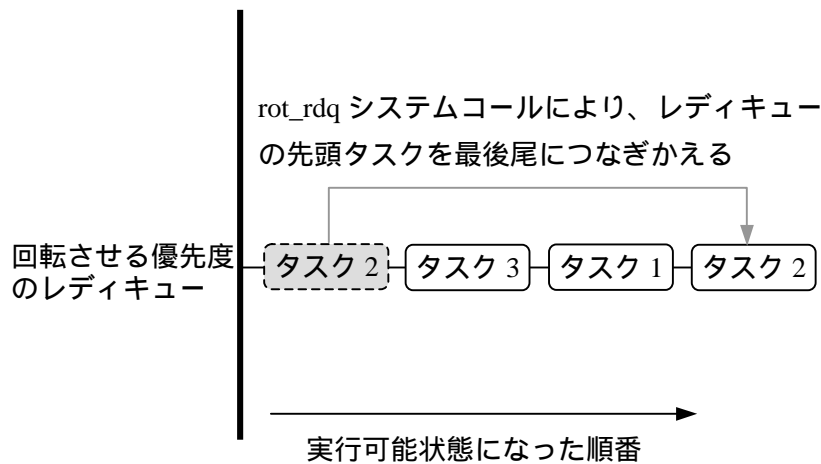
以上により、タイマ割り込み周期でレディキューを回転することができる、ラウンドロビンスケジューリングを実現できます。

< rot_rdq システムコールによるラウンドロビンスケジューリングの動作 >

タイマ割り込み 1 回目



タイマ割り込み 2 回目



タスク 1 からタスク 3 の優先度のレディキューを、一定周期で回転させラウンドロビンスケジューリングを実現します。

コーディング

HI7000 (タイマ割込みハンドラ)

```
/* ***** */
/* NAME      = hi_tmriq ;                               */
/* DATE      = 95/01/11                                 */
/* FUNCTION   = Timer interrupt handler                 */
/* ***** */
#pragma interrupt (hi_tmriq(sp = p_intstk13, tn = 25))
void hi_tmriq(void)
{
    register VP gbrsave;                                /* save variable of GBR */
    gbrsave = (VP)get_gbr();                            /* save GBR to gbr save */
    set_gbr((VP)IOBASE);                               /* set I/O base add ress to GBR */
    gbr_read_byte(TCSR);                               /* dummy read TCSR */
    gbr_and_byte(TCSR, TCSR_BASE);                    /* clear TCSR flag */
    rot_rdq(tskpri);                                  /* rotate ready queue */
    vsys_clk();                                       /* request kernel t imer processing */
    set_gbr(gbrsave);                                /* load GBR form gb rsave */
}

```

対象とするレディキューを回転させます

HI7700 (タイマ割込みハンドラ)

```
/* ***** */
/* SPECIFICATIONS ;                                     */
/* NAME      = hi_tmriq ;                               */
/* AUTHOR    = Renesas Technology Corp. ;              */
/* FUNCTION   = HI7700 hadrware timer interrupt handler ; */
/* ATTRIBUTE = PUBLIC ;                               */
/* HISTROY   = V1.0 ;                                 */
/* END OF SPECIFICATIONS ;                             */
/* ***** */
void hi_tmriq(void)
{
    VP      gbrsave;                                /* save variable of GBR */
    gbrsave = (VP)get_gbr();                            /* save GBR to gbr save */
    set_gbr((VP)IOBASE);                               /* set I/O base add ress to GBR */
    gbr_write_word(TCR, TCRDAT);                      /* clear UNF */
    rot_rdq(tskpri);                                  /* rotate ready que ue */
    vsys_clk();                                       /* request kernel t imer processing */
    set_gbr(gbrsave);                                /* load GBR form gb rsave */
}

```

対象とするレディキューを回転させます

【タイムスライス機能によるタイムスライススケジューリング】

「タスクが実行 (RUN) 状態を維持する最大連続時間：タイムスライス時間 (quantum)」は、タスクごとにタスク生成時に指定します。また、vchg_qua システムコールを用いて動的に変更することが可能です。

<タスク生成のコーディング>

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

パラメータ

ID	tskid	タスク ID	
T_CTSK	*pk_ctsk	タスク生成情報の先頭アドレス	
typedef	struct	t_ctsk{	
	VP	exinf;	拡張情報
	ATR	tskatr;	タスク属性
	FP	task;	タスク起動アドレス
	PRI	itskpri	タスク起動時優先度
	UB	cpumode;	タスク CPU モード
	INT	stksz;	タスクスタックサイズ
	TMO	quantum;	タイムスライス時間
	B	*name	タスク名称を格納した領域の先頭アドレス
	}	T_CTSK;	

↑ タイムスライス時間を設定します

<タイムスライス時間変更のコーディング>

```
ER ercd = vchg_qua((ID)tskid, (TMO)quantum);
```

パラメータ

ID	tskid	タスク ID
TMO	quantum	タイムスライス時間

↑ タイムスライス時間を設定します

4.2 タスクの沈み込み

質 問

タスクの沈み込みはどんなときに発生しますか。また、その回避策を教えてください。

回 答

タスクの沈み込みは、常にそのタスクよりも高い優先度のタスクあるいは割り込みハンドラが実行されてしまう場合や、割り込みをマスクしたタスクが CPU を占有してしまう場合など、そのタスクがいつまでも実行できない（プロセッサが割り付けられない）ときに発生します。

沈み込みが発生する主な原因の 1 つに、次のようなことが考えられます。

優先度の高いタスクの処理が、待ち（WAIT）状態に遷移することでタスク切り替えが発生することを想定します。ところが、待ち状態に遷移するシステムコールを発行する前に待ち要因が成立しているため、待ち状態に遷移せずタスク切り替えが発生せず、優先度の高いタスクが実行し続けます。

このような場合の対策としては、次のような方法が考えられます。

- (1) 必ず待ち状態に遷移するように待ち状態に遷移するシステムコールを発行する前で対象オブジェクトの状態を参照し、その状態によって処理フローを変化させるなど、方式の変更を検討する。
- (2) タイマ割り込みを使用し、一定時間で `rot_rdq` システムコールを発行する、または、タイムスライス機能を使用する（ラウンドロビンスケジューリング）。

沈み込みを解決する一般的な対策として、次のような方法が考えられます。

- (1) タスクの分割、および優先度を再検討する。
- (2) 沈み込みが発生するタスクの優先度を高くする。
- (3) 割り込みハンドラの処理を単純化するなどにより、処理時間の短縮を図る。
- (4) 割り込み発生頻度を少なくする。

4.3 タスクのデッドロック

質 問

タスクのデッドロックとは、どんなときに発生しますか。また、その回避策を教えてください。

回 答

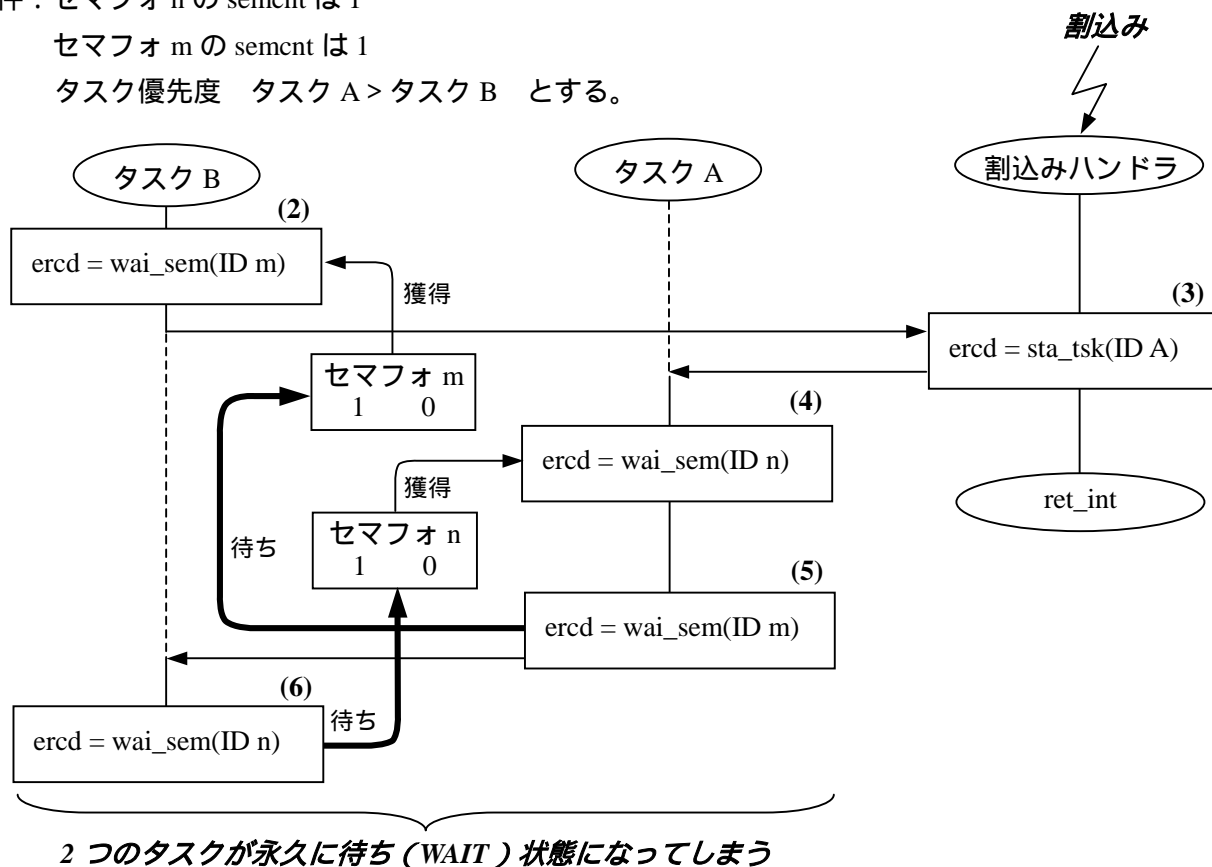
タスクのデッドロックとは、2 つ以上のタスクが、ある状態から永久に抜け出せなくなった状態をいいます。この状態は、セマフォの機能を使用した場合などに発生しやすいと考えられています。デッドロックが発生する簡単な例を示します。

【タスクのデッドロックの例】

条件：セマフォ n の semcnt は 1

セマフォ m の semcnt は 1

タスク優先度 タスク A > タスク B とする。



- (1) はじめ、セマフォ n、m のセマフォカウント (semcnt) は 1 です。
- (2) タスク B がセマフォ m に対して wai_sem システムコールを発行し獲得しました。セマフォカウントは 0 となります。
- (3) 割込みハンドラから、タスク A が起動されました。タスク A はタスク B より優先度が高いため、タスク切り替えが発生します。
- (4) タスク A がセマフォ n に対して wai_sem システムコールを発行し獲得しました。セマフォカウントは 0 となります。
- (5) 続いてタスク A は、セマフォ m に対しても wai_sem システムコールを発行しました。しかし、タスク B がセマフォを獲得 (semcnt=0) しているので、待ち (WAIT) 状態に遷移します。
- (6) タスク B が再び実行 (RUN) 状態に遷移します。タスク B は、セマフォ n に対して wai_sem システムコールを発行しましたが、タスク A がセマフォを獲得 (semcnt=0) しているので、待ち状態に遷移します。

以上、タスク A とタスク B は、お互いが相手のセマフォの開放を待つ待ち (WAIT) 状態となり、この状態は永久に解除されません。

このデッドロックを回避するには、獲得 (待ち状態) と開放 (待ち状態解除) の関係を明確に定義し、矛盾がないかを確認します。セマフォによる同期や排他制御を行う場合は、1 つのセマフォ ID で管理するシステム設計を行ってください。

5. 事象・同期・通信関連

5.1 タスクの待ち解除とタイムアウトのタイミング

質 問

タスクの待ち状態解除処理中にタイムアウトが発生した場合、待ち状態だったタスクが発行したシステムコールのエラーコードはどうなりますか。

回 答

タスクのエラーコードは、E_OK が返る場合と E_TMOOUT が返る場合の 2 通りとなります。タスクの待ち状態解除処理が先に完了した場合は E_OK が返り、タイムアウト処理が先に完了した場合は E_TMOOUT が返ります。

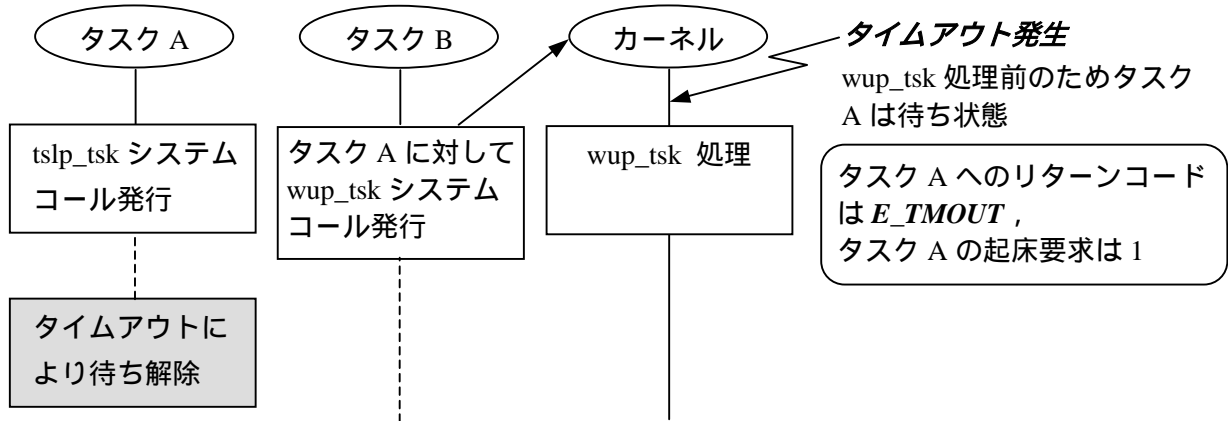
wup_tsk システムコール処理終了前にタイムアウトが発生した場合、タイムアウト処理は「タスクは待ち状態」と判断するため、このタスクへのエラーコードは E_TMOOUT が返ります。また、この処理終了後、wup_tsk システムコールの処理が行われるため、このタスクの起床要求は 1 となります。

これに対して、wup_tsk システムコール処理終了後にタイムアウトが発生した場合、タイムアウト処理は「タスクは待ち状態でない」と判断するため、このタスクへの処理は何も行わず、このタスクへのエラーコードは E_OK が返ります。この場合には、wup_tsk システムコールの処理は終了していますので、このタスクの起床要求は 0 となります。

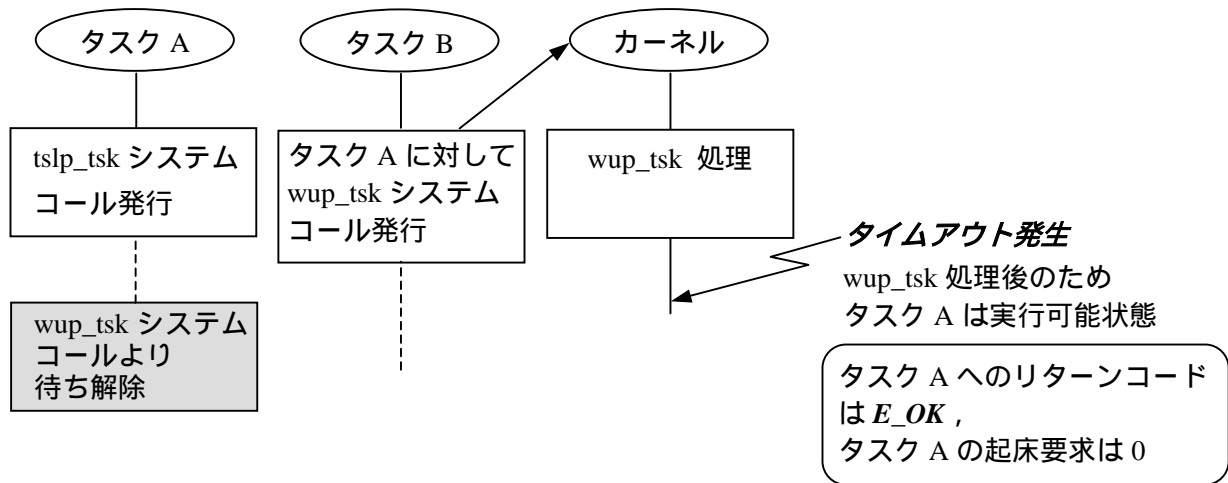
タスクの待ちとタイムアウトのタイミング

以下の図において、タスク優先度は「A>B」とします。

【wup_tsk システムコール処理終了前にタイムアウト発生】



【wup_tsk システムコール処理終了後にタイムアウト発生】



5.2 待ちとポーリングの違い

質問

待ちとポーリングの違いは何ですか。それぞれのメリットを教えてください。

回答

待ちとポーリングの違いは、要求した資源がない場合に、獲得するまで待ち（WAIT）状態になるか、待ち（WAIT）状態にならずにエラーリターンし、処理を継続するかの違いです。

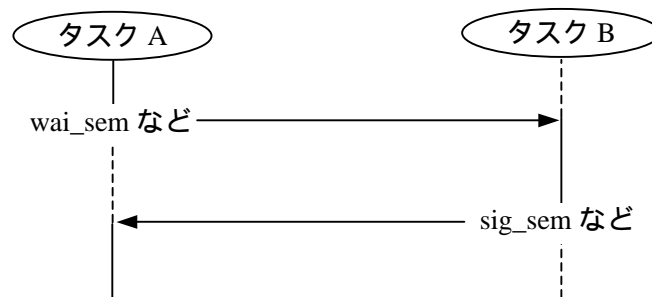
【待ちのメリット】

要求した資源を獲得するまで待ち（WAIT）状態になるため、資源を開放する側と同期がとれます。

get_blf, get_blk, rcv_mbf, rcv_msg, snd_mbf, vwai_tfl, wai_flg, wai_sem の各システムコール

さらに、HI7700 では vget_pbl システムコール

（タスク優先度：A > B）

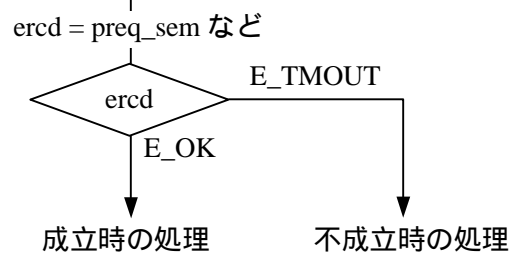


【ポーリングのメリット】

待ち（WAIT）状態にならないため、資源がない場合の処理が明示的に記述できます。

pget_blf, pget_blk, pol_flg, prcv_mbf, prcv_msg, preq_sem, psen_mbf, vpol_tfl の各システムコール。

さらに、HI7700 では vpget_pbl システムコール



5.3 イベントフラグのクリア

質問

イベントフラグのクリアは、いつどのようなタイミングで行いますか。
このときの注意事項はありますか。

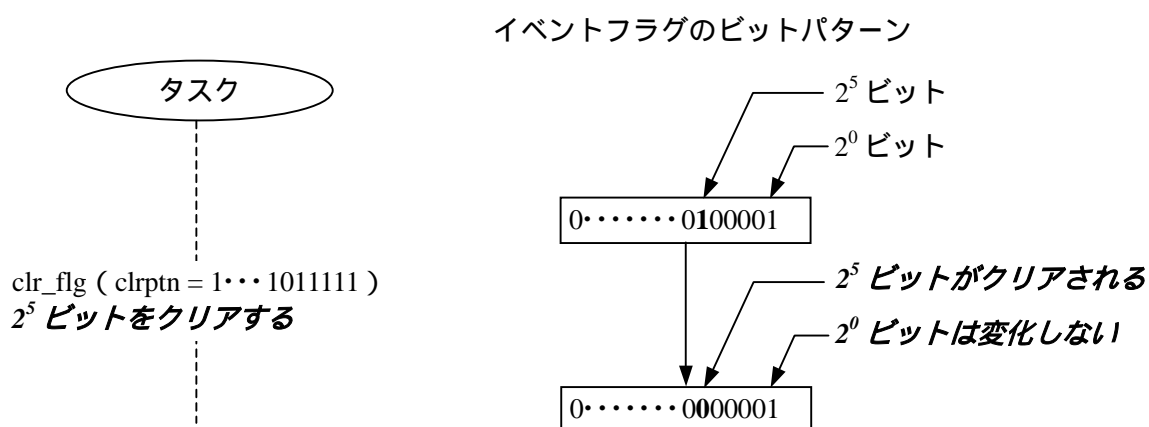
回答

clr_flg システムコールを発行するタイミング例は、タスクがイベントフラグに対して待ち状態に移行する場合に、wai_flg システムコール発行直前に clr_flg システムコールを発行し、値を初期化するケースが考えられます。このときの注意事項を以下に示します。

イベントフラグをクリアする方法には、2つの方法があります。1つは、clr_flg システムコールを発行する。もう1つは、wai_flg システムコール発行時にクリア指定 (TWF_CLR) を行うことです。

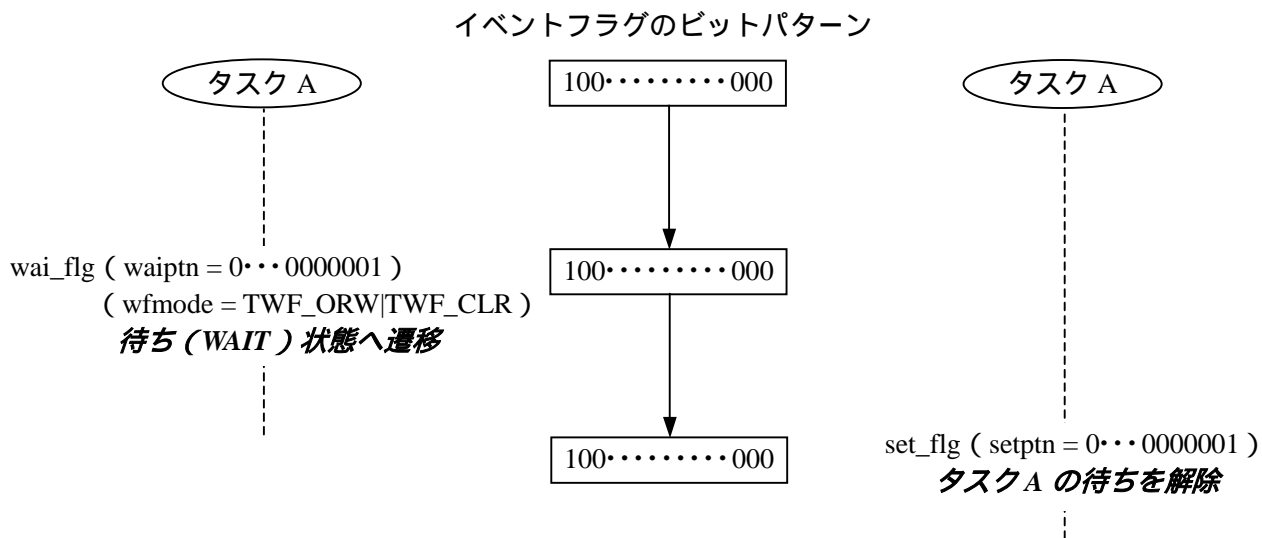
【clr_flg システムコール】

clrptn が 0 になっているビットをクリアすることは、発生したイベントをクリアすることなので、発生したイベントをクリアして良いかの判断を十分に行う必要があります。

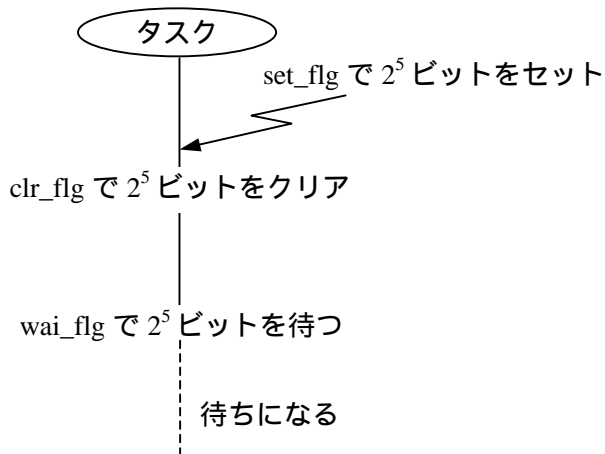


【wai_flg システムコール】

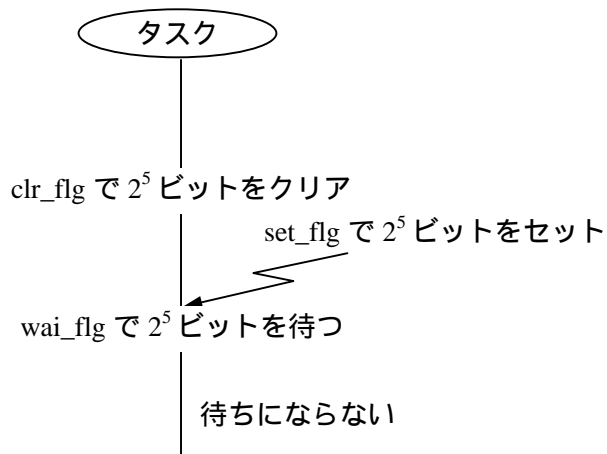
クリア指定 (TWF_CLR) によるイベントフラグのクリアでは、待ちタスクの条件が成立した場合に、対象イベントフラグの値を 0 にクリアします。すべてのビットをクリアして良いかの判断を十分に行う必要があります。



【クリア前にイベント発生】



【クリア後にイベント発生】



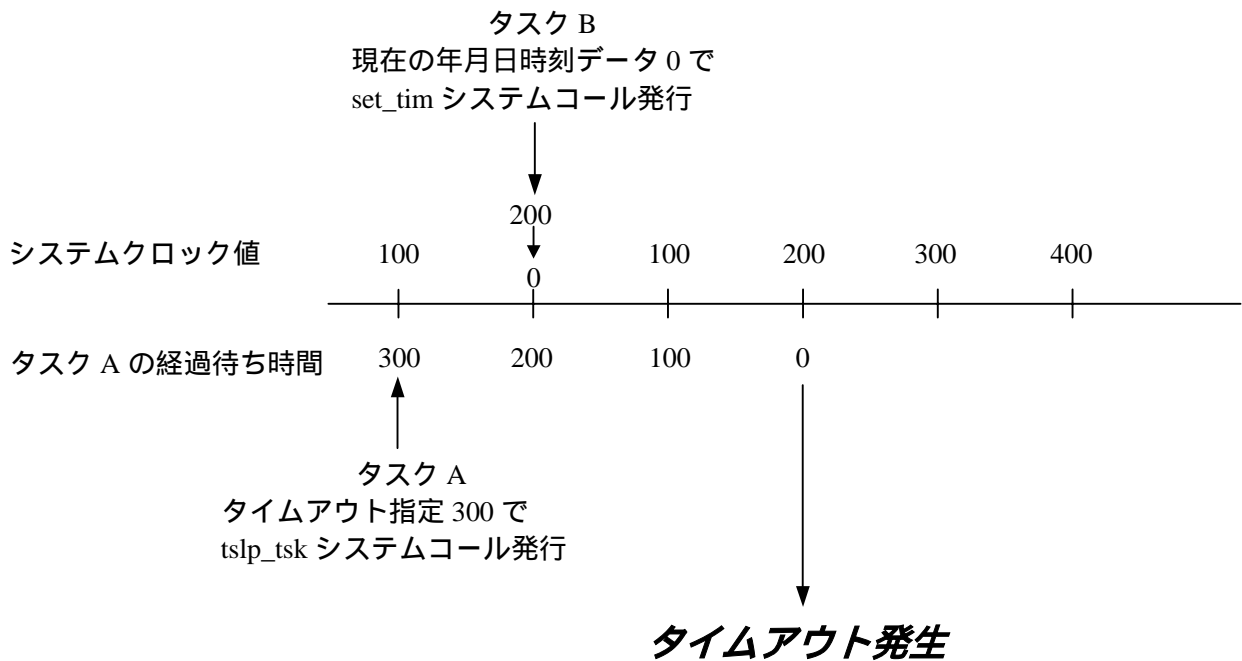
5.4 タイムアウト監視中のタスクとシステムクロック

質 問

システムクロックの値が 100 のときに、タスク A からタイムアウト指定 300 で `tslp_tsk` システムコールを発行し、システムクロックが 200 のときにタスク B から現在の年月日時刻データ 0 で `set_tim` システムコールを発行した場合は、タスク A のタイムアウトはどこで発生しますか。

回 答

タイムアウト監視中のタスクは `tslp_tsk` システムコール発行後、指定した相対的な時間（タイム割込み周期 × タイムアウト指定）が経過するまでタイムアウトにはなりません。したがって、タスク A のタイムアウト発生は、以下のようになります。



5.5 待ち状態に入るシステムコールでのタイムアウト指定

質 問

待ち状態に入るシステムコールでは、タイムアウトを指定することができますか。

回 答

以下のシステムコールを発行して待ちに入る場合、タイムアウト指定をすることができます。

機 能	システムコール名	HI7000	HI7400	HI7700
タスク付属同期	tslp_tsk			
タスク付属イベントフラグ	vtwai_tfl			
セマフォ	twai_sem			
イベントフラグ	twai_flg			
メールボックス	trcv_msg			
メッセージバッファ	tsnd_mbf , trcv_mbf			
メモリプール (可変長)	tget_blk			
メモリプール (固定長)	tget_blf			
ページプール	vtget_pbl	×	×	

5.6 タスク間の簡単な同期処理

質 問

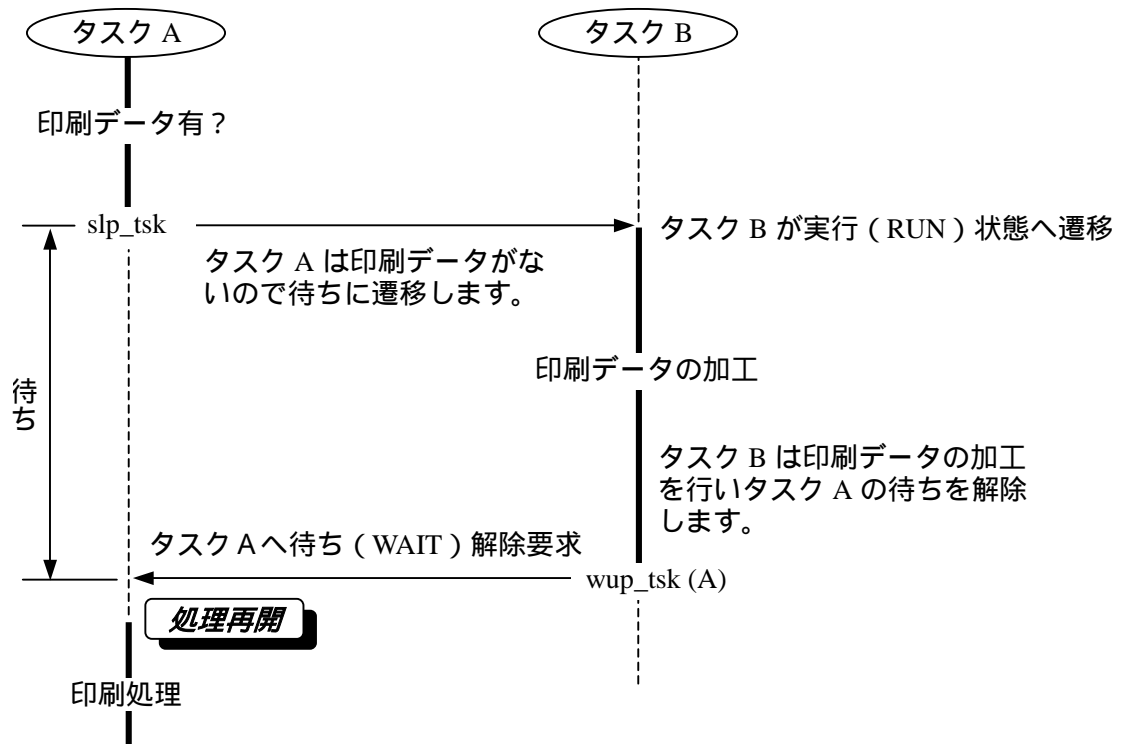
タスク間の簡単な同期処理について教えてください。

回 答

タスク間の同期処理は、タスクを決められた順番で実行しなければならない場合や、他のタスクからの回答を得てから動作を再開する場合などで有効な手段となります。

以下にタスク間の同期処理の例として、プリンタ出力を行う処理を示します。なお、下図において、タスク優先度は「A>B」とします。

【タスク間の同期処理例】



5.7 タスクの起床要求のキューイングと解除

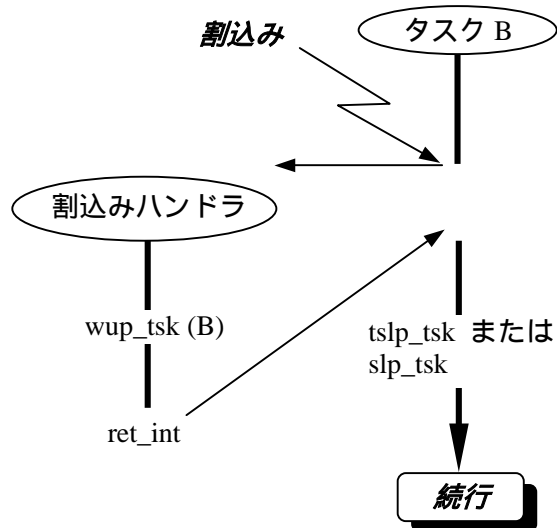
質問

タスクの起床要求がキューイングされる場合と、解除される場合を教えてください。

回答

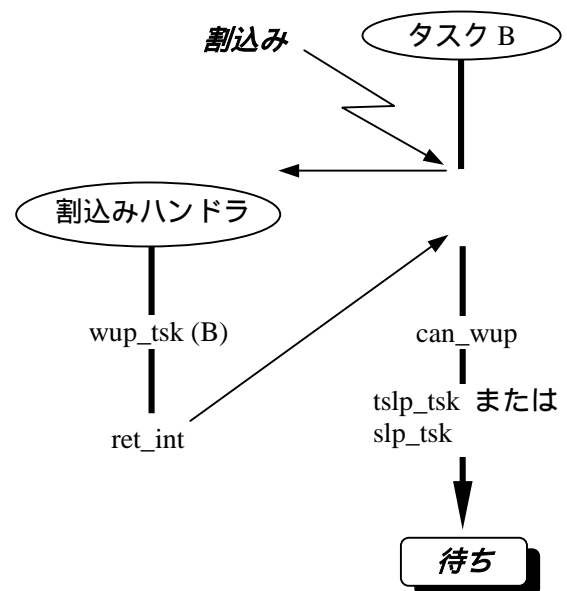
実行可能 (READY) 状態のタスクに対し待ち解除システムコールを発行すると、起床要求のキューイングとなります。また、起床要求のキューイングされているタスクに対して `can_wup` システムコールを発行すると、起床要求のキューイングが解除されます。

【起床要求キューイングの例】



起床要求がキューイングされているため、待ちに入るシステムコールを発行しても、待ち状態には遷移しません

【起床要求キューイング解除の例】



`can_wup` システムコールにより起床要求のキューイングが取り消され、待ちに入るシステムコールによりタスクは待ち状態に遷移します

5.8 タスク付属同期機能の使用例

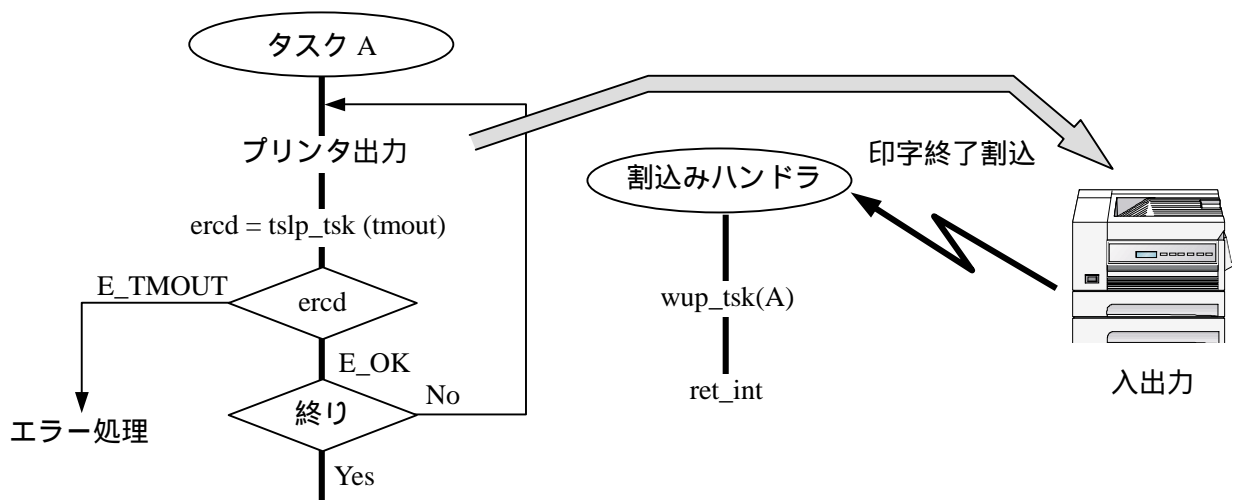
質 問

タスク付属同期機能の使用例を教えてください。

回 答

タスク付属同期機能はタスクに付属した同期機能で、タスクを待ち (WAIT) 状態にする機能とそれを起床する同期機能などがあります。この機能は、入出力装置と同期した処理を行う場合に有効な手段となります。

【タスク付属同期機能の使用例】



タスク A は、プリンタヘータを出力するタスクです。

タスク A がプリンタヘータを出力した後、tslp_tsk システムコールによりプリンタから紙が出力される間、待ち (WAIT) 状態となります。プリンタの印字終了割込みにより、割込みハンドラが起動され wup_tsk システムコールによりタスク A を起床します。

また、規定時間内に印字終了割込みが発生しない場合は、タイムアウトとなりエラー処理を行います。

5.9 イベントフラグの使用例

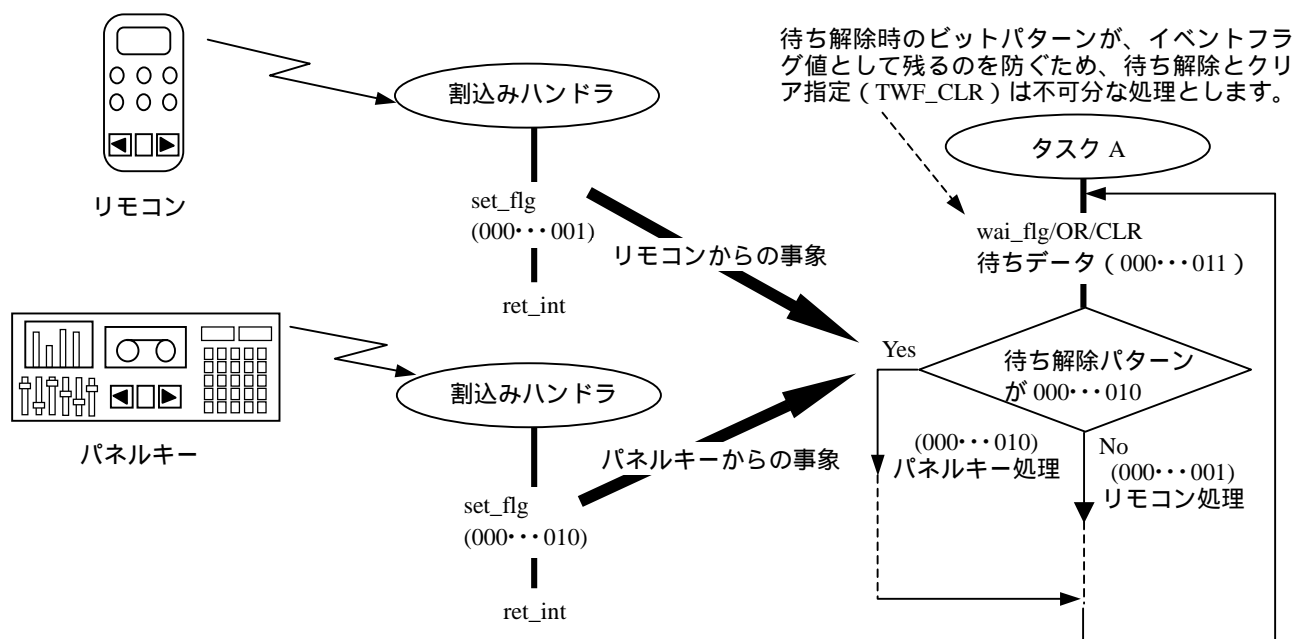
質問

イベントフラグを用いた同期処理の使用例を教えてください。

回答

イベントフラグは事象の有無をビット対応のフラグで表現する同期機能です。事象を知らせる側ではイベントフラグの特定ビットをセットしたりクリアしたりすることが可能で、事象を待つ側では、イベントフラグのビットパターンが特定の条件を満たすまでタスクを待ち (WAIT) 状態にしておくことができ、複数の条件が成立してから処理を実行する場合などに有効な機能です。

【イベントフラグを用いたオーディオ制御例】



タスク A は wai_flg システムコールによりリモコンからの事象発生または、パネルキーからの事象発生を OR 待ちで待ちます。リモコンおよび、パネルキーの各割込みハンドラでは、事象に応じたイベントフラグのビットパターンを設定し set_flg システムコールにより、事象発生のお知らせを行います。タスク A は事象の発生で待ち (WAIT) 状態を解除します。待ちを解除されたタスク A では要因解析を行い、リモコンまたは、パネルキーに応じた処理を行います。

5.10 セマフォの使用例

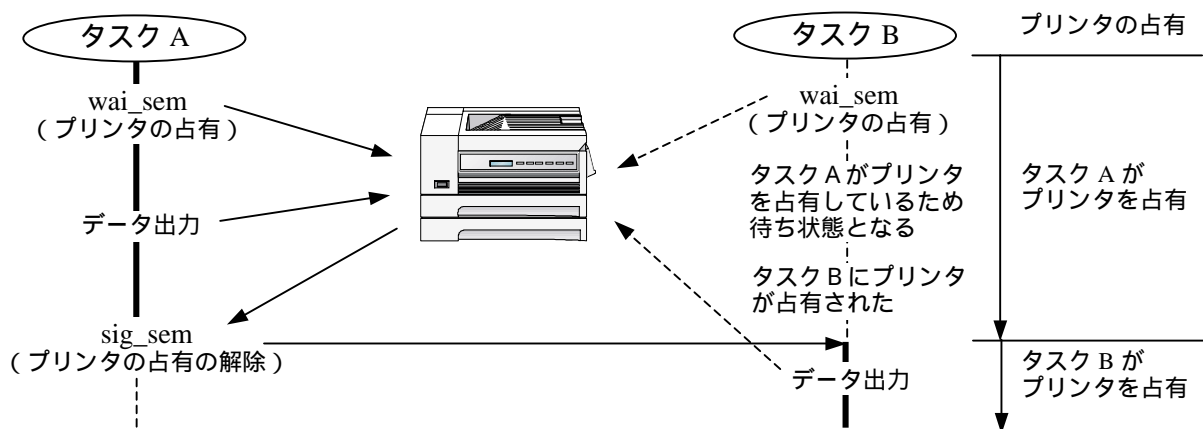
質 問

セマフォを用いた排他制御の使用例を教えてください。

回 答

セマフォは資源の有無や数を表現することにより、同期や排他制御を行うことができる機能です。セマフォを用いた排他制御を行うことにより、1つしかない資源を排他的に使用できるようになります。

【セマフォを用いたプリンタ制御例】



プリンタを使用するタスクでは、`wai_sem` システムコールを用いて、セマフォから資源を1つ獲得する要求をします。資源を獲得して、はじめてプリンタを使用する権利が与えられるというルールでプログラミングします。

はじめに、タスク A が `wai_sem` システムコールを用いてプリンタの使用権を獲得します。セマフォから資源を獲得したタスク A は、プリンタを使用することができます。

その後、タスク B が `wai_sem` システムコールを用いてプリンタの使用権を獲得しようとしていますが、セマフォのカウント値が0のため、セマフォから資源を獲得できるまで待ち状態になります。

タスク A はプリンタの使用が終了すれば、プリンタの使用権を返却し、他のタスクが使用権を獲得できるようにしなければなりません。

プリンタの使用が終了したタスク A は、`sig_sem` システムコールを用いてセマフォに資源を返却します。

これにより、タスク B はセマフォから資源が獲得でき待ち状態が解除され、プリンタを使用できます。

5.11 メールボックスの使用例

質 問

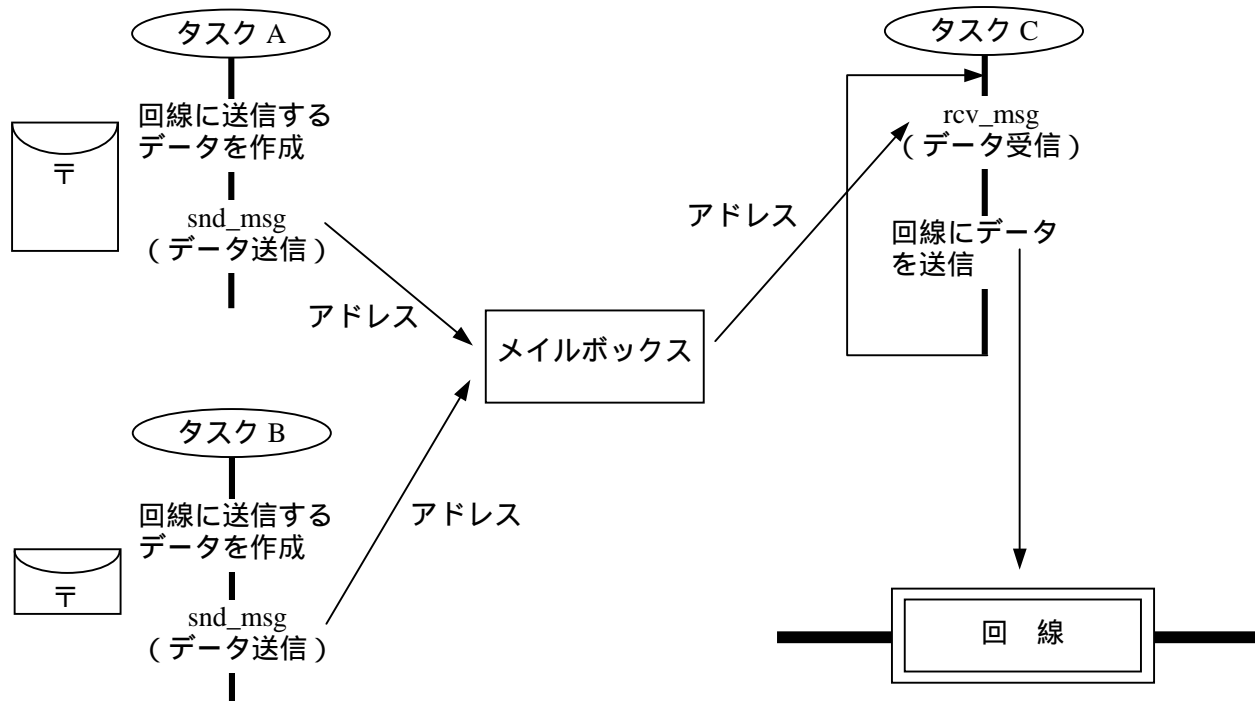
メールボックスを用いたメッセージ使用例を教えてください。

回 答

メールボックスは、メッセージを送受信することにより同期と通信を同時に行う機能です。セマフォやイベントフラグとは違い、メールボックスを用いてメッセージ通信（メッセージ先頭アドレスの送受信）を行うことにより、タスク間でデータの授受を行うことができます。

なお、メッセージ送信後は、受信側タスクがメッセージを受信する前にメッセージの内容を変更しないようにしてください。

【複数のタスクから送られてきたメッセージを回線に送信する例】



6. オブジェクト関連

6.1 オブジェクトの ID 番号

質 問

オブジェクトの ID 番号について教えてください。

回 答

ID 番号とは、タスクやセマフォなどのオブジェクト生成時に割り当てられた番号のことで、各オブジェクトは、この番号によって識別します。

ID 番号はオブジェクト種別ごとに独立し、タスク ID = 1 とセマフォ ID = 1 は全く異なる意味を持ちます。システム設計を行う際は、あらかじめ「1 番のタスクは ~ をするタスク」、「1 番のセマフォは ~ をするセマフォ」と整理して決めておくと、アプリケーションプログラムを開発する上での混乱を防止することができます。

以下に、オブジェクトを生成し、ID 番号を割り付ける方法を示します。

(1) システムコールを用いる方法

```
ercd = cre_tsk(1, &t_ctsk) ; /* ID = 1 のタスクを生成 */
```

これ以降、生成したオブジェクトは、ここで指定した ID 番号を用いて操作できます。

(2) カーネルビルドファイルおよびセットアップファイルを用いる方法

カーネルビルドファイルおよびセットアップファイルの「オブジェクト初期登録パート」で初期登録用マクロにより、オブジェクトを生成します。

```
CRE_TSK(2, 0, TA_HLNG, usr_task2, 1, 0, 256, 0, TASK2) /* ID = 2 のタスクを生成 */
```

(3) ID 番号の自動割り当て

HI7000 シリーズでは、オブジェクトの生成時に自動的に ID 番号を割り当てるシステムコールをサポートしています。

```
ercd = vasn_tsk(&id, &t_ctsk) ; /* t_ctsk パケットで指定した内容でタスクを生成し、 */  
/* 生成したタスク ID を id に返す */
```

6.2 イベントフラグの初期値

質 問

イベントフラグの初期値を教えてください。

回 答

イベントフラグの初期値は、イベントフラグ生成時に指定します。

以下の方法でイベントフラグを生成します。

- cre_flg システムコール
- vasn_flg システムコール
- カーネルビルドファイルまたはセットアップファイルでの初期登録 (CRE_FLG マクロ)

【コーディング例】

```
ER ercd = cre_flg((ID)flgid, (T_CFLG *)pk_cflg);
ID      イベントフラグ ID
T_CFLG イベントフラグ生成情報の先頭アドレス
typedef struct t_cflg{
    VP      exinf;      拡張情報
    ATR     flgatr;     イベントフラグ属性
    UINT    iflgptn;    イベントフラグの初期値
    B *name;           イベントフラグ名称を格納した領域の先頭アドレス
} T_CFLG;
```

6.3 セマフォの初期値

質 問

セマフォの初期値を教えてください。

回 答

セマフォの初期値は、セマフォ生成時に指定します。

以下の方法でセマフォを生成します。

- cre_sem システムコール
- vasn_sem システムコール
- カーネルビルドファイルまたはセットアップファイルでの初期登録 (CRE_SEM マクロ)

【コーディング例】

```
ER ercd = cre_sem((ID)semid, (T_CSEM *)pk_csem);
ID      セマフォ ID
T_CSEM  セマフォ生成情報の先頭アドレス
typedef struct t_csem{
    VP      exinf;      拡張情報
    ATR     sematr;     セマフォ属性
    UINT    isemptn;    セマフォの初期値
    B       *name;     セマフォ名称を格納した領域の先頭アドレス
} T_CSEM;
```

6.4 メールボックスの初期化

質 問

メールボックスの初期化は必要ですか。

回 答

メールボックスの初期化は必要ありません。

ただし、メッセージは送信する前に先頭の4バイトを0に初期化する必要があります。

詳細は、「2.4 メッセージ領域の初期化」を参照してください。

6.5 メモリブロック・ページブロックの初期化

質 問

獲得したメモリブロック、またはページブロックの内容はゼロクリアされていますか。

回 答

可変長メモリプール、固定長メモリプール、またはページプールに限らず、獲得したブロックの内容は不定です。必要であれば、獲得したブロックサイズ分初期化してください。サイズを超えて初期化を行った場合、カーネル管理領域が破壊され動作保証ができませんので注意してください。

コーディング例については、「2.5 メモリブロック領域の初期化」を参照してください。

6.6 オブジェクト数とメモリ（RAM）サイズの関係

質 問

RAM サイズをできるだけ少なくするためのオブジェクト生成数について教えてください。

回 答

オブジェクトで使用する RAM サイズは、セットアップファイルで指定するオブジェクトの定義数によって決まります。RAM サイズを小さくするために、システムで必要とするオブジェクト数のみ確保してください。

以下に、RAM 使用サイズ例を示します。RAM 使用サイズの詳細は、各プログラムのコンパイルリストを参照してください。

【条件】

	ケース 1	ケース 2	ケース 3
優先度	5	10	40
タスク数	5	10	40
タスクスタック数	4	10	40
DSP 使用数 (HI7400 のみ)	0	10	40
セマフォ	5	10	20
イベントフラグ	5	10	20
メールボックス	5	10	20
タイマ機能	未使用	使用	使用
タイムスライス	未使用	未使用	使用
周期起動ハンドラ	未使用	未使用	10
トレース機能	未使用	未使用	使用
名称機能	未使用	未使用	使用
使用割込みレベル数	2	7	14

【RAM 使用サイズ】

単位：バイト

	HI7700			HI7000			HI7400			
	ケース 1	ケース 2	ケース 3	ケース 1	ケース 2	ケース 3	ケース 1	ケース 2	ケース 3	
RAM 使用サイズ	1522	3760	13552	1696	5408	26240	1696	5488	26560	
内訳	カーネルワーク	744	1364	6600	796	1500	6884	796	1580	7204
	アプリスタック	808	2396	6952	900	3908	19356	900	3908	19356

7. 標準関数関連

7.1 関数呼び出しと拡張 SVC の違い

質 問

関数呼び出しと拡張 SVC の違いを教えてください。

回 答

関数（以下、サブルーチンとよびます）呼び出しと拡張 SVC の違いを以下に示します。

- (1) サブルーチンは、その呼び出し元の一部であり、タスクがサブルーチンを呼び出した場合、サブルーチンはあくまでもタスクとして動作します。

これに対して拡張 SVC 発行時は、拡張 SVC ハンドラ実行状態となり、呼び出し元とは異なる状態すなわち、拡張 SVC ハンドラ実行状態として扱われます。

拡張 SVC ハンドラ実行状態とタスク実行状態は、次の点が異なります。

条件割り込みハンドラからのシステムコール	条 件	
	拡張 SVC ハンドラ実行中	サブルーチン（タスク）実行中
ter_tsk システムコール	休止（DORMANT）状態へは SVC ハンドラ終了まで遅延	すぐに休止（DORMANT）状態
sus_tsk システムコール	強制待ち（SUSPEND）状態へは SVC ハンドラ終了まで遅延	すぐに強制待ち（SUSPEND）状態

したがって、拡張 SVC ハンドラは最後まで動ききることになります。

- (2) 拡張 SVC は拡張機能コードとよぶ、通常のシステムコールの機能コードと同様なコードを用いて拡張 SVC ハンドラを呼び出します。したがって、タスクと拡張 SVC ハンドラを一緒にリンクする必要がありません。

これに対してサブルーチンは、タスクと一緒にリンクする必要があります。

- (3) サブルーチン呼び出しの事象は、トレース情報には取得されません。

これに対して拡張 SVC 呼び出しの事象はトレース情報として取得されます。

使用方法：

- (1) `def_svc` システムコール、またはシステム構築において拡張 SVC ハンドラをカーネルに登録します。
- (2) 呼び出す場合は、登録時に `s_fncd` で設定した拡張機能コードを用いて、通常のシステムコールと同様に呼び出します。ただし、TRAPA 命令の番号 (#30) が異なるので注意が必要です。
- (3) 拡張 SVC ハンドラから復帰する場合には、`ret_svc` システムコールを発行します。

注意事項：

- (1) タスクスタックを計算する場合には、呼び出す拡張 SVC ハンドラで使用するスタックサイズを加算する必要があります。拡張 SVC ハンドラから拡張 SVC ハンドラを呼び出すなどのネストがある場合にはそれらもタスクスタックとして加算する必要があります。
- (2) 拡張 SVC ハンドラで使用するレジスタは保証する必要がありません。拡張 SVC ハンドラでは `#pragmanoregsave` 宣言することで、スタック使用量を削減することができます。詳細は、SH シリーズ C コンパイラ ユーザーズマニュアルの「レジスタ退避・回復の制御」を参照してください。

7.2 拡張 SVC の発行

質 問

拡張 SVC の発行方法を教えてください。

回 答

アプリケーション (拡張 SVC ハンドラ) をカーネルに組み込み、このプログラムを拡張されたシステムコールとして通常のシステムコールと同様の手続きで発行することができます。

【登録】

拡張 SVC ハンドラは、以下の方法で定義します。

- ・ def_svc システムコール
- ・ vasn_svc システムコール
- ・ カーネルビルドファイルまたはセットアップファイルでの初期登録 (DEF_SVC マクロ)

<コーディング>

```
ER ercd = def_svc((FN)s_fncd, (T_DSVC *)pk_dsvc);
FN      s_fncd      拡張機能コード
T_DSVC  *pk_dsvc    拡張 SVC ハンドラ定義情報の先頭アドレス
typedef struct t_dsvc{
    ATR    svcatr;    拡張 SVC ハンドラ属性
    FP     svchdr;    拡張 SVC ハンドラアドレス
}T_DSVC;
```

【発行】

vsys_cal システムコールにより、拡張 SVC を発行します。

<コーディング>

```
ER ercd = vsys_cal((FN)fncd, (VW)para1, (VW)para2, (VW)para3, (VW)para4);
FN      fncd        拡張 SVC の機能コード
VW      para1       パラメータ 1
VW      para2       パラメータ 2
VW      para3       パラメータ 3
VW      para4       パラメータ 4
```

機能コードを同じにします

また、拡張 SVC の汎用 C インタフェースとして vsys_cal システムコールを用意しています。これを用いても拡張 SVC を発行できますが、vsys_cal システムコールでは必ずパラメータを 4 つ指定しなければならないという制限があります。そこで、ユーザの拡張 SVC を trapa_svc() に define で定義すれば、無駄がなくなるだけでなく、プログラムがわかりやすくなります。

なお、拡張 SVC の詳細については、カーネルユーザズマニュアルを参照してください。

【HI7000 または HI7400】

「2.13 拡張 SVC」

「4.3.7 拡張 SVC ハンドラ」

【HI7700】

「2.12 拡張 SVC」

「4.3.8 拡張 SVC ハンドラ」

7.3 リエントラントな共通関数

質 問

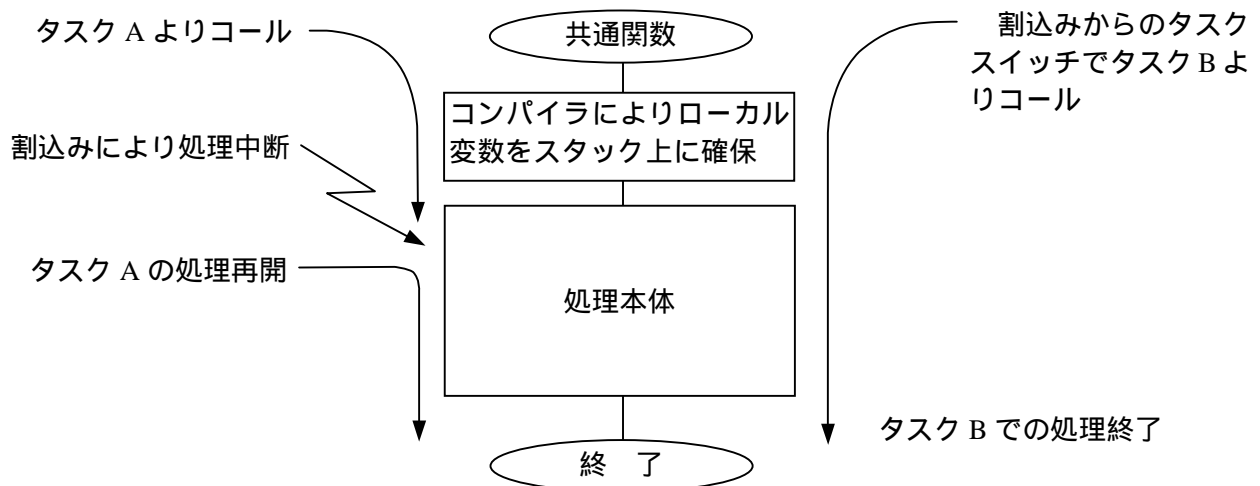
リエントラントな関数は、どのように作成しますか。

回 答

リエントラントな関数内では、共通変数などへの設定・更新・参照を行うことができません。したがって、関数内部で使用するローカル変数はすべてスタック上に確保する必要があります。

たとえば、ある関数の実行中に割り込みなどで中断され、再び同じ関数が呼び出された場合など、中断された処理で使用中のデータを後から呼び出された処理が変更したり、後から呼び出された処理で設定したデータを、中断再開後の処理が変更すると、データの矛盾が発生します。このように、データの操作中に重複して呼び出される関数は、リエントラントな関数にします。

なお、リエントラントな関数にできない処理については、関数の排他制御を行います。



- : タスク A より共通関数をコール
- : 処理中に割り込みが発生し、共通関数の処理を中断
- : 割り込みからのタスクスイッチでタスク B を起動し、タスク B も共通関数をコール
- : タスク B が先に共通関数の処理を完了
- : タスク A による共通関数処理再開 (タスク B により共通変数の操作を行うと、後から実行するタスク A との間にデータの矛盾が発生する)

7.4 共通関数の排他制御

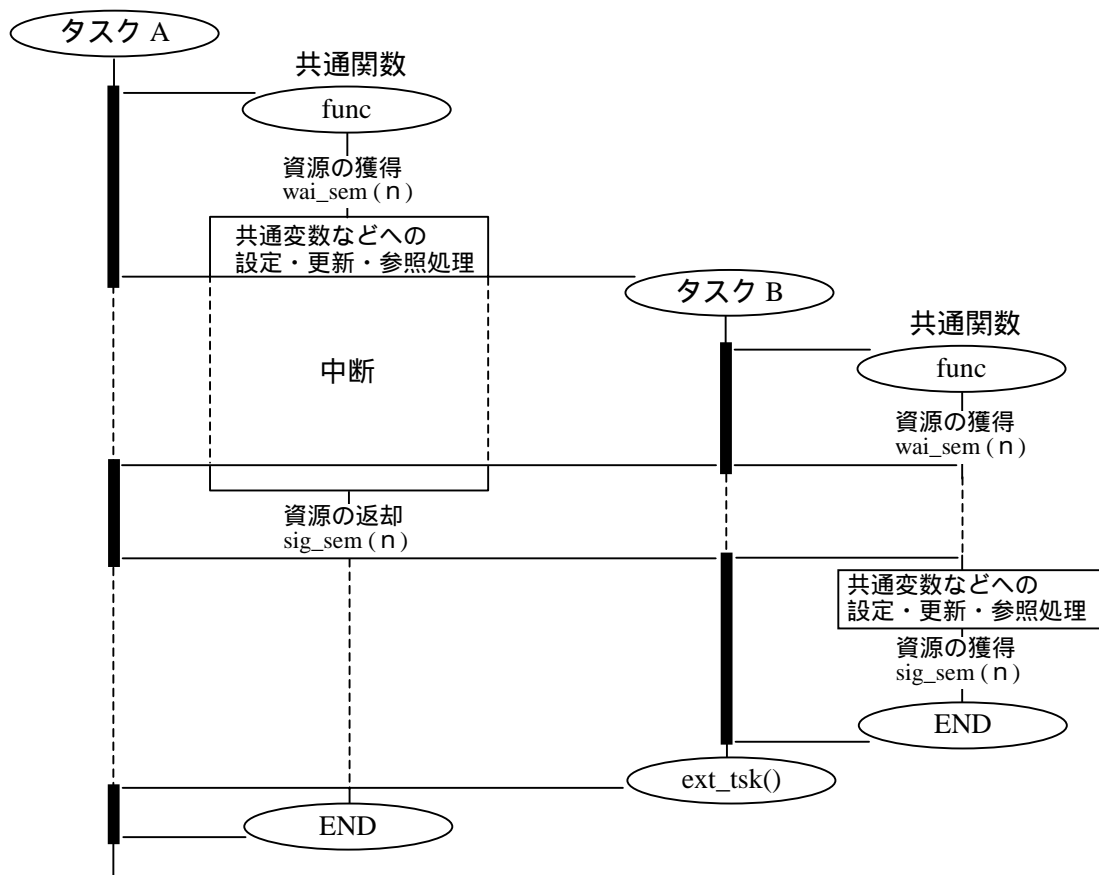
質問

関数の排他制御はどのように行いますか。

回答

関数の排他制御は主にセマフォを用いて行います。以下に、セマフォを用いた共通関数の排他制御例を示します。

タスク優先度：A < B



共通変数などへの設定・更新・参照などは、必ず資源の獲得を行ってから実施し、処理終了後は獲得した資源を返却します。また、各資源ごとに専用のセマフォ ID を用いる必要があります。

8. メモリ関連

8.1 カーネルの RAM 領域を小さくしたい

質 問

カーネルで使用する RAM 領域を可能な限り小さくしたいので、着眼点を教えてください。

回 答

カーネルの RAM 領域を最小とするには、各オブジェクトの数を必要数だけ登録します。また、使用する機能のみでカーネルを構築します。

RAM 領域を小さくするためには、使用しないオブジェクトがカーネルで生成されないようにすることが大切です。

以下に、RAM 領域の内訳を示します。

(1) カーネル作業領域

- ・システム初期化ハンドラで使用するスタックサイズを小さくしてください。
- ・未使用となるオブジェクトがないように定義してください。

(2) 割込みハンドラ用スタック領域 (タイマなど)

- ・同一レベルの割込みハンドラのスタックは、共有してください。
- ・割込み要因をできるだけ少なくしてください。
- ・発生しない割込みに対するスタックは確保しないでください。

(3) タスク用スタック領域

- ・シーケンシャルにしか実行されないタスク間では、共有スタック機能の使用を検討してください。

(4) メモリプール領域

- ・メモリブロック数を余分に定義しないでください。

(5) メッセージバッファ領域

- ・メッセージバッファサイズを余分に定義しないでください。

(6) トレース機能用スタック領域、トレースバッファ領域

- ・トレース機能を使用しない場合は、領域を確保しないでください。
- ・トレースバッファ領域の確保は、エミュレータの貸し出しメモリなどを使用してください。

(7) その他 (ユーザ用)

- ・タスクが独自に使用するスタックサイズを小さくしてください。
- ・グローバル変数領域を小さくしてください。
- ・拡張 SVC ハンドラで使用するスタックサイズを小さくしてください。

HI7700 では、ページプール機能を用い RAM 領域を小さくすることも検討してください。

各カーネルが使用する RAM 領域サイズの詳細については、構築マニュアル「付録 A . 作業領域サイズの算出」を参照してください。

8.2 カーネルの ROM 領域を小さくしたい

質 問

カーネルの ROM 領域を可能な限り小さくしたいので、着眼点を教えてください。

回 答

カーネルの ROM 領域を小さくするためには、システムで使用しない機能を明確にするなど、ユーザーシステムに最適なカーネルを構築することが必要です。カーネルユーザズマニュアルの「第 2 章 カーネル」を参照し、システム構築時に必要としない機能を組み込まないようにしてください。

以下に、ROM 領域を小さくするための着眼点を示します。

(1) システムコールの選択

- ・使用しないシステムコールを組み込まないようにします。

(2) パラメータチェック機能

- ・リンケージで結合するカーネルライブラリは「パラメータチェック機能なし」のライブラリを選択します。

(3) ユーザタスク

- ・ユーザプログラムサイズを小さくします。
- ・共通処理はサブルーチンにするか、拡張 SVC としてください。
- ・同一機能のタスクであれば、ID の変更などで実体を 1 つにします。

8.3 タスクのスタック領域を小さくしたい

質 問

タスクのスタック領域を小さくしたいのですが、タスク単体およびタスク全体での着眼点を教えてください。

回 答

タスクスタックサイズの最小値は、HI7000 シリーズが常に必要とする固定サイズのみとなります。

【タスク単体での着眼点】

タスクを`#pragma noregsave` 宣言する。詳細は、SH シリーズ C コンパイラ ユーザーズマニュアル「2.3.10 レジスタ待避・回復の制御」を参照してください。

タスクを非タスク（割込みマスクレベル 0）で実行しない

関数のネスト数を少なくする

タスクをアセンブリ言語で記述する（C 言語でタスクを記述した場合、タスクスタックサイズが増大する）

【タスク全体での着眼点】

シーケンシャルに実行されるタスク間では、共有スタックを使用する

タスクスタックの内訳については、構築マニュアル「A.2 スタックサイズの算出」を参照してください。

8.4 共通関数の排他制御

質 問

割込みハンドラのスタック領域を小さくしたいので、着眼点を教えてください。

回 答

割込みハンドラのスタック領域を小さくする着眼点を以下に示します。

【着眼点】

割込みハンドラ内の処理で使用するレジスタを少なくし、スタックへのレジスタ退避を最小限にしてください。

同一レベルの割込みハンドラのスタックは、共有してください。

割込みレベルをできるだけ少なくし、割込みネスト数を減らしてください。

割込みハンドラから関数コールをしないでください。(割込みハンドラをC言語で記述した場合で、割込みハンドラから関数コールしなければ、コンパイラは割込みハンドラ内で使用するレジスタのみをセーブします。)

割込みハンドラから、システムコールを一切発行しなければ、構築マニュアル「表 A - 4 割込みハンドラ用スタック領域のメモリ容量算出表」項番 2 で示す、カーネル使用サイズ 144 バイトは不要となります。

9. 割り込み関連

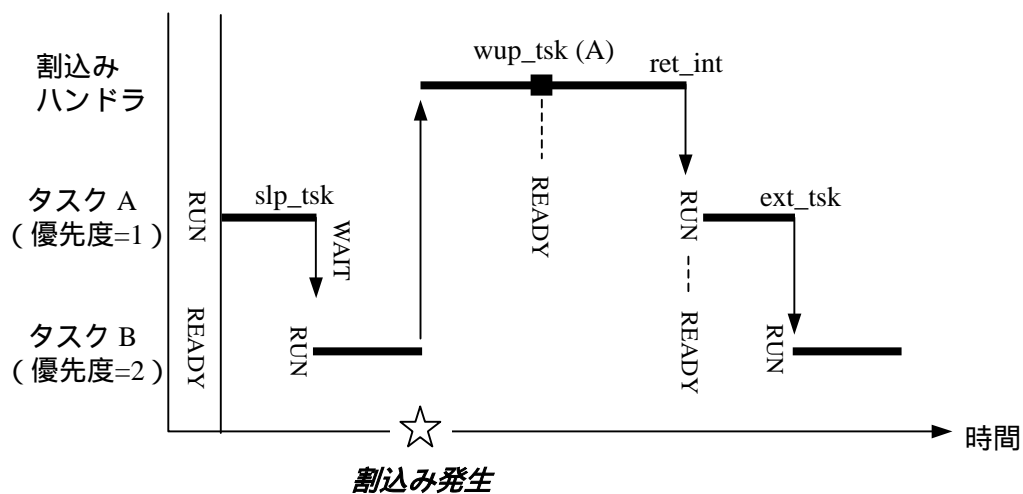
9.1 割り込みハンドラの動き

質 問

割り込みが発生した場合の割り込みハンドラおよび、タスクはどのように動作しますか。

回 答

タスク実行中に割り込みが発生しタスクの処理が中断され、割り込みハンドラ終了後に中断されたタスクが処理を再開するまでの様子を、以下に示します。



- : タスク A が `slp_tsk` システムコールを発行したため、優先度の低いタスク B が実行 (RUN) 状態となります。
- : 割り込みが発生し、タスク B の処理が中断され割り込みハンドラが起動されます。
- : 割り込みハンドラから `wup_tsk` システムコールを発行し、タスク A を起床します。割り込みハンドラは `ret_int` システムコール発行により終了します。
- : 実行可能 (READY) 状態となっているタスクの中で、最も優先度の高いタスク A が実行 (RUN) 状態となります。 `ext_tsk` システムコールによりタスク A は休止 (DORMANT) 状態となります。
- : 割り込みにより処理中断されていたタスク B が再開します。

9.2 割込みハンドラが起動しない

質 問

割込みハンドラが起動しません。何が原因として考えられますか。

回 答

割込みハンドラが起動しない場合は、以下の内容を確認してください。

(1) **デバイス初期化の確認**

デバイスの初期設定内容を確認してください(対象デバイスのハードウェアマニュアル参照)。

(2) **割込みベクタテーブルの確認**

割込みハンドラの実アドレスが、ベクタテーブルに登録されていることを確認してください。

(3) **システムダウンの確認**

未定義の割込み、例外の発生等でシステムダウンルーチンを実行している可能性があります。システムダウンルーチン(hi_sysdwn)に渡される情報と要因を確認をしてください。詳細は、カーネルユーザズマニュアル「2.16 システムダウン」を参照してください。

(4) **割込み頻度の確認**

対象デバイスより優先度の高いデバイスの割込み発生頻度が高く、対象デバイスの割込みをCPUが受け付けられないことが考えられます。優先度の高いデバイスの割込み頻度を確認してください。

(5) **ハードウェア障害の確認**

対象となるデバイスが割込みを発生していることを確認してください。

9.3 割込みハンドラのスタック切り替え方法

質 問

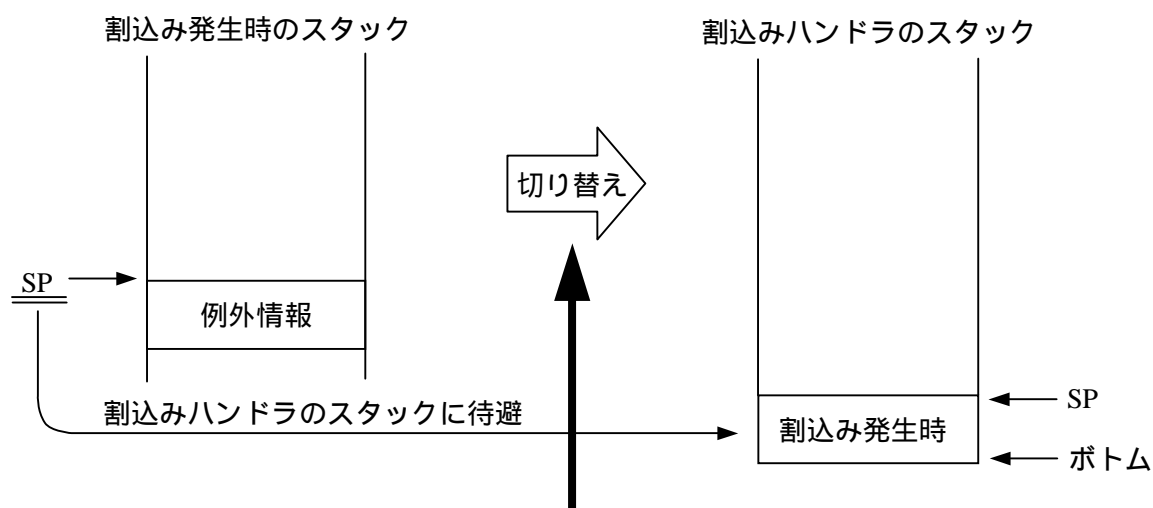
割込みハンドラのスタック切り替え方法を具体的に教えてください。

回 答

C 言語記述の割込みハンドラでは、C コンパイラの割り込み関数定義のオプションにより実現することが可能です。詳細は、SuperH RISC engine ファミリ C コンパイラ編 アプリケーションノート「割り込み関数の定義」を参照してください。

ただし、HI7700 では割込みハンドラのスタックをカーネルが管理するため、割込みハンドラではスタックを切り替えないでください。

アセンブリ言語で割込みハンドラを記述した場合は、以下のようになります。



SP (レジスタ) に割込みハンドラのスタックのアドレスを設定します。これで、割込みハンドラのスタックに切り替わったことになります。

このとき、割込み発生時の SP を保証する必要があるので、割込みハンドラのスタックに待避します。割込みハンドラのスタックのアドレスはユーザ任意で確保した領域のアドレスです。

【コーディング例】

割り込み関数 handler2 を宣言します。この関数は、配列 STK をスタックとして使用します。

< C 言語記述によるスタックポインタの切り替え >

```
extern int STK[100];
int *ptr = STK + 100;
#pragma interrupt(handler2(sp=ptr,tn=63)) /* 割り込み関数の宣言 */
void handler2(void)
{
..... /* 割り込み関数の記述処理 */
}
```

上記コーディングのアセンブリ言語展開コード

```
                .IMPORT      _STK
                .EXPORT      _ptr
                .EXPORT      _handler2
                .SECTION     P, CODE, ALIGN=4
_handler2:
                ; function: handler2
                MOV.L        R0, @-R15      ; 割り込まれた関数のスタックの R0 退避
                MOV.L        L211, R0
                MOV.L        @R0, R0       ; 割り込みハンドラ用スタックボトムアドレスを得る
                MOV.L        R15, @-R0     ; 割り込まれた関数のカレントスタックポインタ退避
                MOV          R0, R15
                .             ; 処理内で使用するレジスタの退避
                .             ; 割り込み関数処理
                .             ; 処理内で使用したレジスタの回復
                MOV.L        @R15+, R15
                MOV.L        @R15+, R0
                TRAPA        #63
L211:
                .DATA.L      _ptr
                .SECTION     D, DATA, ALIGN=4
_ptr:
                ; static: ptr
                .DATA.L      H'00000190+_STK
                .END
```

9.4 割込みハンドラの終了方法

質 問

割込みハンドラの終了方法を教えてください。

回 答

割込みハンドラの終了方法は、カーネルにより異なります。

【HI7000 または HI7400】

(1) カーネル割込みマスキングレベル以下の割込みハンドラの場合

- ・ #pragma interrupt により割込み関数を宣言し、ret_int システムコール (TRAPA #25) により終了するので、tn=25 を指定します。

(2) カーネル割込みマスキングレベルより高い割込みハンドラの場合

- ・ #pragma interrupt により割込み関数を宣言しますが、「トラップ命令リターン指定」を行いません。これにより、ret_int システムコールを発行せずに、RTE 命令で割込み発生前に復帰するコードが生成されます。

詳細については、HI7000 ユーザーズマニュアル「4.3.2 割込みハンドラ」を参照してください。

【HI7700】

(1) 通常の間数として記述し、#pragma interrupt 宣言は行わないでください。

詳細については、HI7700 ユーザーズマニュアルを参照「4.3.2 割込みハンドラ」を参照してください。

9.5 割込み同時発生時の動作順序

質 問

複数の割込みが同時に発生した場合、実行される順序を教えてください。

回 答

複数の割込みが同時に発生した場合は、カーネルの介在なしに割込みレベルの高い割込みが受け付けられます。

割込みレベルの詳細については、ハードウェアマニュアル「割込みコントローラ」を参照してください。

9.6 未定義割込みの発生

質 問

実際の割込み処理が存在するのに、割込みハンドラを定義しませんでした。
システムの動作はどうなりますか。

回 答

定義されていない割込みが発生すると、未定義割込みとなります。

未定義割込みが発生すると、システムダウンとなりシステムダウンルーチン (hi_sysdwn) が起動されます。システムダウンルーチンには入力パラメータとして、エラー種別、エラーコード、およびシステムダウン情報が渡されます。

システムダウンの詳細については、カーネルユーザズマニュアル「2.16 システムダウン」を参照してください。

9.7 割込み許可の方法

質 問

SR の I0 ~ 3 ビットを直接操作して割込み許可を行っても良いのですか。

回 答

割込み許可はカーネルがすべて管理しています。タスク側で直接 SR の I0 ~ 3 ビットを操作した場合、タスク制御が保証できなくなります。タスクの割込みマスクを変更したい場合は、chg_ims システムコールを使用してください。

9.8 カーネル実行中の割込み発生

質 問

カーネル実行中に割込みが発生した場合は、どのようになりますか。
また、割込みが多重に発生した場合は、どのようになりますか。

回 答

カーネル実行中は基本的にカーネル割込みマスクレベルで割込みを禁止しています。

(1) カーネル割込みマスクレベル以下の割込み

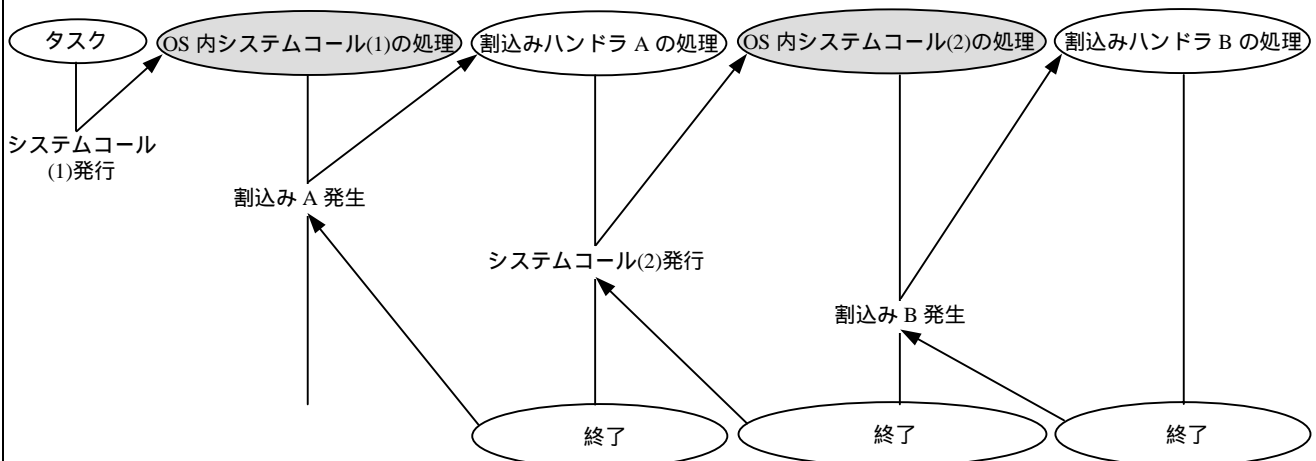
最大で割込みマスク時間だけ割込み処理が待たされる場合があります。受け付けられた割込みハンドラは必ず割込み発生元へリターンします。また、システムコール実行中に割込みがネストしても、必ず割込み発生元へリターンするのでシステムコール処理は矛盾なく完結します。

(2) カーネル割込みマスクレベルより高い割込み

割込みハンドラからシステムコールが発行された場合に動作の保証ができません。したがって、カーネル割込みマスクレベルより高いレベルの割込みハンドラからはシステムコールが発行できません。

【例：システムコールと割込みのネスト】

システムコール(1)処理中に割込みAが発生し、割込みAのハンドラ内でシステムコール(2)を発行したが、さらに割込みBが発生しました。このときの動作は、割込みBのハンドラ処理終了後、システムコール(2)の処理に戻り、これを完結します。その後、割込みAのハンドラ処理終了後、システムコール(1)の処理に戻り、システムコール処理を続けます。



10. タイマ、タイマハンドラ関連

10.1 システムクロックの値

質 問

システムクロックの値は、何を表していますか。

回 答

システムクロック値とは、システムが起動してから何回ハードウェアタイマによる割込みが発生したのかを示します。

ハードウェアタイマの周期が 10msec で、システムクロック値が 200 の場合は、

$$10\text{msec} \times 200 = 2,000\text{msec}$$

となり、システムが起動してから 2 秒経過していることを表します。

10.2 システムクロックが遅れる（誤差が生じる）

質 問

システムクロックに誤差が発生します。何が原因か教えてください。

回 答

カーネルのシステムクロックは、タイマ割込みハンドラから一定周期で `sys_clk` システムコールなどを発行することによって更新 (+1) されます。

したがって、論理的には以下ようになります。

$$\text{実時間} = \text{システムクロック値} \times \text{タイマ割込み周期}$$

タイマよりも高いレベルの割込みハンドラがタイマ周期時間以上連続して動作すると（正確にはタイマ割込みレベルがマスクされる時間が連続してタイマ周期以上となると）、タイマ割込みハンドラの動作回数が1回以上抜けることになり、システムクロック値が実時間よりも少なくなってしまう。

このようなことが起きないようにするため、理想的にはタイマ割込みレベルはシステムで最も高くしてください。それができない場合は、タイマよりも高い割込みハンドラの実行時間をタイマ周期よりも短くしてください。

10.3 タイマデバイスを変えたい

質 問

タイマデバイスを変更したいのですが、手順を教えてください。

回 答

タイマデバイスを変更するには、以下の手続きが必要です。

- (1) 変更後のタイマデバイスにあわせた、タイマ初期化ルーチンの作成
- (2) 変更後のタイマデバイスにあわせた、タイマ割り込みハンドラの作成
- (3) 割り込みハンドラ定義

def_int システムコールを発行し、タイマ割り込みハンドラを定義します。

```
ER ercd = def_int((UINT)dintno, (T_DINT *)pk_dint);
UINT      dintno      割り込み定義番号 (ベクタ番号)
T_DINT    *pk_dint    割り込みハンドラ定義情報の先頭アドレス
typedef   struct      t_dint{
    ATR      intatr;      割り込みハンドラ属性
    FP      inthdr;      割り込みハンドラアドレス( (2) で作成したハンドラのアドレス)
    UINT     intsr;      将来拡張用
} T_DINT;
```

- (4) システム初期化ハンドラの作成

上記 (1) で作成したタイマ初期化ルーチンをシステム初期化ハンドラからコールします。

```
extern void hi_tmrini(void);
void sysini_hdr(void)
{
    ER ercd; /* error code */
    T_CTSK t_ctsk; /* cre_tsk packet */

    hi_tmrini(); /* call timer initialize routine */
    ercd = cre_tsk(1, &t_ctsk); /* create task */
    . . . . .
}

(1) で作成した関数名に変更
```

- (5) タイマ割り込みハンドラで使用するスタックサイズと領域の定義
- (6) 新規作成、および変更したファイルのコンパイルとリンク

11. ハードウェア関連

11.1 必要となるハードウェア

質 問

最低限必要なハードウェアを教えてください。

回 答

対象デバイス、ROM、RAM を搭載したユーザのシステムボードがあれば動作は可能です。
なお、必要に応じてコンソール、ユーザインタフェースボード、プリンタなどを用意します。
評価のためだけであれば、貸出しメモリのあるインサーキットエミュレータがあれば動作は可能です。
以下に、ROM サイズの参考値を示します。

【ROM サイズ】

カーネル ROM サイズは、全機能使用する場合での算出例

カーネル	HI7000	HI7400	HI7700
ROM サイズ	29.6KB	31.3KB	35.3KB

RAM サイズについては、本アプリケーションノートの「6.5 オブジェクト数とメモリ (RAM) サイズの関係」を参照してください。

11.2 レジスタの使用制限

質 問

レジスタの初期値は、システム初期化ハンドラ、割込みハンドラおよびタスクでどのようになっていますか。

回 答

各カーネルにおけるレジスタの初期値を以下に示します。

【HI7000】

(1) システム初期化ハンドラ

項番	項 目	初期値	終了時の制限	
1	SR	カーネル割込み マスクレベル	割込みマスクレベルは変更しないで ください	
2	使用できる レジスタ* ¹	R0 ~ R7	不定	
		MACH, MACL	不定	
		GBR	不定	
		R8 ~ R14, PR* ²	不定	起動時の状態に戻してください
3	スタック, スタックポインタ (R15)	カーネルスタック領域	起動時と同じ値に戻してください	

【注】 *1 ベクタベースレジスタ (VBR) はカーネルの初期化処理にて自動的に初期化されます。その後は絶対に変更しないでください。変更した場合、システムの正常な動作は保証されません。

*2 C 言語で記述する場合は意識する必要はありません。詳細は、カーネルユーザーズマニュアル「4.3.10 システム初期化ハンドラ」を参照してください。

(2) 割込みハンドラ

項番	項目	初期値	終了時の制限
1	SR	発生したレベルでマスク	割込み発生時の値に戻してください
2	使用できるレジスタ* ¹	R0 ~ R7* ²	発生時の値
		MACH, MACL* ²	
		GBR* ³	終了時に起動時の状態に戻してください
		R8 ~ R14, PR* ²	
3	スタック, スタックポインタ (R15)* ⁴	発生時の値	終了時に起動時と同じ値に戻してください

【注】 *1 ベクタベースレジスタ (VBR) はカーネルの初期化処理にて自動的に初期化されます。その後は絶対に変更しないでください。変更した場合、システムの正常な動作は保証されません。

*2 C 言語で記述する場合は意識する必要はありません。詳細は、カーネルユーザズマニュアル「4.3.2 割込みハンドラ」を参照してください。

*3 C 言語で記述する場合も、GBR は割込みハンドラ内で明示的に保証する必要があります。

*4 割込みハンドラ専用のスタックに切り替え、終了時に元に戻してください。

(3) タスク

項番	項目	初期値	終了時の制限
1	SR	0	起動時の状態 (0) に戻してください
2	使用できるレジスタ*	R0 ~ R7	起動時の値
		MACH, MACL	
		GBR	
		R8 ~ R14, PR	
3	スタック, スタックポインタ (R15)	起動時の値	

【注】 * ベクタベースレジスタ (VBR) はカーネルの初期化処理にて自動的に初期化されます。その後は絶対に変更しないでください。変更した場合、システムの正常な動作は保証されません。

【HI7400】

(1) システム初期化ハンドラ

項番	項目		初期値	終了時の制限
1	SR	IMASK	カーネル割込み マスクレベル	割込みマスクレベルは変更しないで ください
		RC	0	
		DMX, DMY	0	
		RF1 ~ 0	不定	
2	使用できる レジスタ*1	R0 ~ R7	不定	
		MACH, MACL	不定	
		GBR	不定	
		R8 ~ R14, PR*2	不定	起動時の状態に戻してください
3	スタック, スタックポインタ (R15)		カーネルスタック領域	起動時と同じ値に戻してください
4	RS, RE, MOD		不定	
5	A0G, A1G, A0, A1, M0, M1, X0, X1, Y0, Y1		不定	
6	DSR		不定	

【注】 *1 HI7000 と同様

*2 HI7000 と同様

(2) 割込みハンドラ

項番	項目		初期値	終了時の制限
1	SR	IMASK	発生したレベルでマスク	割込み発生時の値に戻してください
		RC	発生時の値	割込み発生時の値に戻してください
		DMX, DMY	発生時の値	割込み発生時の値に戻してください
		RF1 ~ 0	発生時の値	割込み発生時の値に戻してください
2	使用できる レジスタ*1	R0 ~ R7*2	発生時の値	起動時の状態に戻してください
		MACH, MACL*2		
		GBR*3		
		R8 ~ R14, PR*2		
3	スタック, スタックポインタ (R15) *4		カーネルスタック領域	起動時と同じ値に戻してください
4	RS, RE, MOD		発生時の値	割込み発生時の値に戻してください
5	A0G, A1G, A0, A1, M0, M1, X0, X1, Y0, Y1		発生時の値	割込み発生時の値に戻してください
6	DSR		発生時の値	割込み発生時の値に戻してください

【注】 *1, *2, *3, *4 HI7000 と同様

(3) タスク

項番	項目		初期値	終了時の制限
1	SR	IMASK	0	起動時の状態(0)に戻してください
		RC	0	
		DMX, DMY	0	
		RF1 ~ 0	0	
2	使用できるレジスタ*	R0 ~ R7	起動時の値	
		MACH, MACL		
		GBR		
		R8 ~ R14, PR		
3	スタック, スタックポインタ (R15)		カーネルスタック領域	起動時と同じ値に戻してください
4	RS, RE, MOD		不定	
5	A0G, A1G, A0, A1, M0, M1, X0, X1, Y0, Y1		不定	
6	DSR		不定	

【注】 * HI7000 と同様

【HI7000】

(1) システム初期化ハンドラ

項番	項目		初期値	終了時の制限
1	SR	MD	特権モード(1)に設定	
		RB	バンク0(0)が設定	起動時の状態に戻してください
		BL	例外ブロックは解除(0)	起動時の状態に戻してください
		I3 ~ I0	カーネル割込みマスクレベル	起動時の状態に戻してください
2	使用できるレジスタ* ¹	R0 ~ R7	不定	
		MACH, MACL	不定	
		GBR	不定	
		R8 ~ R14, PR* ²	不定	
3	スタック, スタックポインタ (R15)		カーネルスタック領域	起動時と同じ値に戻してください

【注】 *¹ ベクタベースレジスタ (VBR) はカーネルの初期化処理にて自動的に初期化されます。その後は絶対に変更しないでください。変更した場合、システムの正常な動作は保証されません。

*² C 言語で記述する場合は意識する必要はありません。詳細は、カーネルユーザーズマニュアル「4.3.10 システム初期化ハンドラ」を参照してください。

(2) 割込みハンドラ

項番	項 目		初期値	終了時の制限
1	SR	MD	特権モード(1)に設定	
		RB	ハンドラ定義時の値	終了時はバンク0(0)としてください
		BL	ハンドラ定義時の値	終了時は解除してください
		I3~I0	ハンドラ定義時の値	起動時の状態に戻してください
2	使用できるレジスタ*1	R0~R7	発生時の値	
		MACH, MACL*2	発生時の値	起動時の状態に戻してください
		GBR*3	発生時の値	
		R8~R14, PR*2	発生時の値	
3	スタック, スタックポインタ(R15)		カーネルスタック領域	起動時と同じ値に戻してください

【注】 *1 ベクタベースレジスタ(VBR)はカーネルの初期化処理にて自動的に初期化されます。その後は絶対に変更しないでください。変更した場合、システムの正常な動作は保証されません。

*2 C 言語で記述する場合は意識する必要はありません。詳細は、カーネルユーザズマニュアル「4.3.2 割込みハンドラ」を参照してください。

*3 C 言語で記述する場合も、GBR は割込みハンドラ内で明示的に保証する必要があります。

(3) タスク

項番	項 目		初期値	終了時の制限
1	SR	MD	特権モード(1)に設定	
		RB	バンク0(0)	終了時はバンク0(0)としてください
		BL	例外ブロック解除(0)	終了時は解除してください
		I3~I0	割込みマスク解除(0)	起動時の状態に戻してください
2	使用できるレジスタ*	R0~R7	R4: sta_tsk システムコールの起動コード(stacd) 他のレジスタは不定	
		MACH, MACL	不定	
		GBR	不定	
		R8~R14, PR	不定	
3	スタック, スタックポインタ(R15)		<スタティックスタック> スタック領域の最終アドレス-8 <ダイナミックスタック> スタック領域の最終アドレス	

【注】 * ベクタベースレジスタ(VBR)はカーネルの初期化処理にて自動的に初期化されます。その後は絶対に変更しないでください。変更した場合、システムの正常な動作は保証されません。

11.3 レジスタ値の変更

質 問

レジスタ値をタスクで変更した場合、終了時戻す必要がありますか。

回 答

タスク切り替えの際、レジスタはカーネルが切り替えるため、基本的にはタスクでレジスタ値を保証する必要はありません。詳細は、本アプリケーションノートの「11.2 レジスタの使用制限」を参照してください。

12. デバッグ関連

12.1 タスクの実行順序を見る

質 問

タスクがどのように実行されたのかなど、タスク切り替えをトレースできますか。

回 答

システムコールトレース機能を使用して、タスク切り替えを見ることが可能です。これは、タスクが発行したシステムコールの履歴を取得するという機能で、システムコールが発行された時刻（システムクロック）およびタスクがどのように切り替わったのかを知ることができます。

タスクの起動時刻、終了時刻については、トレースエントリのイベント属性（te_attr）が CONT 属性、または IDLE 属性時のシステムクロック値（te_ltime）で知ることができます。

なお、詳細は各カーネルのユーザーズマニュアル「2.18 トレース機能」を参照してください。

12.2 トレースバッファの指定と使い方

質 問

トレースバッファの指定はどのように行いますか。また、具体的な使用方法を教えてください。

回 答

トレースバッファは、トレースを取得する場合に必要で、以下のような場合に指定します。

- ・システムコールの発行順序を知りたい
- ・マルチタスク環境下での各タスクの動きを知りたい

など、デバッグを効率的に進める場合に有効です。

【トレースバッファの指定】

(1) カーネルビルドファイルおよびセットアップファイルによる指定

(a) トレース機能の選択

カーネルビルドファイル「レベル AV 機能選択パート」でトレース機能の組み込みを定義します。

```
/*#####*/
/* [1. Level-AV selection] */
/* (中略) */
/*#####*/
#define hi_vctcpy USE /* (1)Vector copy function */
#define hi_anavec USE /* (2)Analyze undefined Vector Number */
#define hi_name USE /* (3)Object name */
#define hi_slice USE /* (4)Time-slice function (vchg_qua SVC) */
#define hi_prique USE /* (5)Priority-order queue (except MPL) */
#define hi_wtcstk USE /* (6)Stack Watch */
#define hi_trace USE /* (7)System-call Trace */
```

↑ “USE” を指定します

(b) トレースバッファサイズの定義

セットアップファイル「トレース情報定義パート」でトレースバッファサイズを指定します。

```
/*#####*/
/* [7. Trace information] */
/* Usage :(1)Trace buffer size. 0 means no trace buffer. */
/*      #define hi_trcbufsz <size> */
/*      <size> : [56...] */
/*      (2)Optional data acquiring routine stack size. */
/*      #define hi_optstksz <size> */
/*      <size> : [0....] */
/*#####*/
#define hi_trcbufsz 0x10000 /* (1)Trace buffer area size */
#define hi_optstksz 0 /* (2)Optional routine stack size */
```

0または56以上の4の倍数で
トレースバッファサイズを指定します

詳細については、カーネルの構築マニュアルを参照してください。

(2) マルチタスクデバッガによる指定

ITRACE_BUFFER (トレースバッファの設定・表示) コマンドにより指定します。

<コマンドフォーマット>

```
#ITRACE_BUFFER[ <バッファ先頭アドレス> {<バッファ最終アドレス>|@<バッファバイト数>}](RET)
  バッファ先頭アドレス      : トレースバッファ先頭アドレス
  バッファ最終アドレス     : トレースバッファ最終アドレス
  バッファバイト数         : トレースバッファサイズ
```

詳細については、カーネル対応のマルチタスクデバッガ ユーザーズマニュアルを参照してください。

【トレースバッファの使い方】

(1) トレースバッファの内容を表示し、トレースエントリごとに発生イベントの解析を行います。

トレースバッファの古い情報から解析を行うことで、タスクの実行順序、システムコールの発行順序と結果などを得ることができます。

具体的な解析方法については、カーネルのユーザーズマニュアル「2.18.4 トレース取得データの解析例」を参照してください。

(2) マルチタスクデバッガの ITRACE コマンドを用いトレース情報の表示を行い、タスクの実行順序、システムコールの発行順序と結果を容易に得ることができます。また、ITRACE_SEARCH コマンドを用い、トレース情報の検索・表示を行うことができます。

詳細は、カーネル対応の E7000 マルチタスクデバッガ ユーザーズマニュアルを参照してください。

【マルチタスクデバッガ ITRACE コマンドの使用例】

最新のイベントから4イベント分を表示する。

```
#ITRACE -3(RET)
```

NO	ATR	ID	- NAME	EVENT	PC	R4	R5	R6	R7
-D'0003	S	NTSK		TER_TSK	00003DAC	00000002	00000000	00000000	00000000
-D'0002	R	NTSK		E_OK	00003DAC	00000002	00000000	00000000	00000000
-D'0001	C	0003	CTLTSK						
-D'0000	S	NTSK		DEBUGGER	00003DAC	00000000	00000000	00000000	00000000

13. その他

13.1 プログラムを特定アドレスからスタートしたい

質 問

プログラムを H'00200000 番地からスタートさせる方法を教えてください。

回 答

システムをリンカージェネレータで結合する際に、H'00200000 番地から配置するセクションを START オプション (サブコマンド) で指定します。

【HI7000 の himix.sub の変更例】

```
start  P_hireset,&          ; [MUST]Kernel initiation process      *
        P_hiknl,&          ; [MUST]Kernel                          *
        C_hibuild,&        ; [MUST]Kernel build data              *
        C_hisysmt,&        ; [MUST]Setup data                      *
        C_hissetup,&       ; [MUST]Initial registration, etc.      *
        P_hisysdwn,&       ; [MUST]System-down routine          *
        P_hicif,&          ; [MUST]SVC C-interface                *
        P_hicpuasm,&       ; CPU initialization routine          *
        P_hicpuini,&       ; CPU initialization sub-routine        *
        P_hitmrdv,&        ; Timer driver                          *
        C_hitmrdv,&        ; Timer driver constant                *
        P_hicnsdrv,&       ; Console driver                          *
        C_hicnsdrv,&       ; Console driver constant                *
        P_histdio,&        ; I/O handler for console            *
        C_histdio,&        ; I/O handler constant                *
        P_usrtask(00200000) ; user task                                *
```

↑
H'00200000 番地からスタート

13.2 システムコールの発行方法

質 問

システムコールの発行方法はサブルーチンコールですか、それともソフトウェア割込みですか。

回 答

システムコールの発行方法はソフトウェア割込みで、以下のトラップ番号を使用します。

TRAPA #16 ~ #31

ルネサスF-ZTAT™マイクロコンピュータ
アプリケーションノート
HI7000シリーズ テクニカルQ&A

発行年月日 2004年12月24日 Rev.2.00
発行 株式会社ルネサステクノロジ 営業企画統括部
〒100-0004 東京都千代田区大手町 2-6-2
編集 株式会社ルネサス小平セミコン 技術ドキュメント部

© 2004. Renesas Technology Corp., All rights reserved. Printed in Japan.

営業お問合せ窓口
株式会社ルネサス販売



<http://www.renesas.com>

本		社	〒100-0004	千代田区大手町2-6-2 (日本ビル)	(03) 5201-5350
京	支	社	〒212-0058	川崎市幸区鹿島田890-12 (新川崎三井ビル)	(044) 549-1662
西	支	社	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル2F)	(042) 524-8701
札	支	店	〒060-0002	札幌市中央区北二条西4-1 (札幌三井ビル5F)	(011) 210-8717
東	支	社	〒980-0013	仙台市青葉区花京院1-1-20 (花京院スクエア13F)	(022) 221-1351
い	支	店	〒970-8026	いわき市平小太郎町4-9 (損保ジャパンいわき第二ビル3F)	(0246) 22-3222
茨	支	店	〒312-0034	ひたちなか市堀口832-2 (日立システムプラザ勝田1F)	(029) 271-9411
新	支	店	〒950-0087	新潟市東大通1-4-2 (新潟三井物産ビル3F)	(025) 241-4361
松	支	社	〒390-0815	松本市深志1-2-11 (昭和ビル7F)	(0263) 33-6622
中	支	部	〒460-0008	名古屋市中区栄3-13-20 (栄センタービル4F)	(052) 261-3000
浜	支	店	〒430-7710	浜松市板屋町111-2 (浜松アクトタワー10F)	(053) 451-2131
西	支	部	〒541-0044	大阪市中央区伏見町4-1-1 (明治安田生命大阪御堂筋ビル)	(06) 6233-9500
北	支	社	〒920-0031	金沢市広岡3-1-1 (金沢パークビル8F)	(076) 233-5980
広	支	店	〒730-0036	広島市中区袋町5-25 (広島袋町ビルディング8F)	(082) 244-2570
鳥	支	店	〒680-0822	鳥取市今町2-251 (日本生命鳥取駅前ビル)	(0857) 21-1915
九	支	社	〒812-0011	福岡市博多区博多駅前2-17-1 (ヒロカネビル本館5F)	(092) 481-7695
鹿	支	店	〒890-0053	鹿児島市中央町12-2 (明治安田生命鹿児島中央町ビル)	(099) 284-1748

■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：カスタマサポートセンタ E-Mail: csc@renesas.com

HI7000 シリーズ テクニカル Q&A アプリケーションノート



ルネサスエレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJ05B0643-0200