お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

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

User's Manual

RENESAS

RI600/4 V.1.00

ユーザーズマニュアル

RX600 シリーズ用リアルタイムOS

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2009.08

本資料ご利用に際しての留意事項

1．本資料は、お客様に用途に応じた適切な弊社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弊社または第三者の知的財産権その他の権利の実施、使用を許諾または保証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など全ての情報の使用に起因する損害、第三者の知的財産権その他の権利に対する侵害に関し、弊社は責任を負いません。
3．本資料に記載の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、あるいはその他軍事用途の目的で使用しないでください。また、輸出に際しては、「外国為替および外国貿易法」その他輸出関連法令を遵守し、それらの定めるところにより必要な手続を行ってください。
4．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などの全ての情報は本資料発行時点のものであり、弊社は本資料に記載した製品または仕様等を予告なしに変更することがあります。弊社の半導体製品のご購入およびご使用に当たりましては、事前に弊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、弊社ホームページ(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。
5．本資料に記載した情報は、正確を期すため慎重に制作したものですが、万一本資料の記述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、弊社はその責任を負いません。
6．本資料に記載の製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を流用する場合は、流用する情報を単独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。弊社は、適用可否に対する責任は負いません。
7．本資料に記載された製品は、各種安全装置や運輸・交通用、医療用、燃焼制御用、航空宇宙用、原子力、海底中継用の機器・システムなど、その故障や誤動作が直接人命を脅かしあるいは人体に危害を及ぼすおそれのあるような機器・システムや特に高度な品質・信頼性が要求される機器・システムでの使用を意図して設計、製造されたものではありません（弊社が自動車用と指定する製品を自動車に使用する場合を除きます）。これらの用途に利用されることをご検討の際には、必ず事前に弊社営業窓口へご照会ください。なお、上記用途に使用されたことにより発生した損害等について弊社はその責任を負いかねますのでご了承願います。
8．第７項にかかわらず、本資料に記載された製品は、下記の用途には使用しないでください。これらの用途に使用されたことにより発生した損害等につきましては、弊社は一切の責任を負いません。
   1）生命維持装置。
   2）人体に埋め込み使用するもの。
   3）治療行為（患部切り出し、薬剤投与等）を行うもの。
   4）その他、直接人命に影響を与えるもの。
9．本資料に記載された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、実装条件およびその他諸条件につきましては、弊社保証範囲内でご使用ください。弊社保証値を越えて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、弊社はその責任を負いません。
１０．弊社は製品の品質および信頼性の向上に努めておりますが、特に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。弊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、誤動作防止設計などの安全設計（含むハードウェアおよびソフトウェア）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。
１１．本資料に記載の製品は、これを搭載した製品から剥がれた場合、幼児が口に入れて誤飲する等の事故の危険性があります。お客様の製品への実装後に容易に本製品が剥がれることがなきよう、お客様の責任において十分な安全設計をお願いします。お客様の製品から剥がれた場合の事故につきましては、弊社はその責任を負いません。
１２．本資料の全部または一部を弊社の文書による事前の承諾なしに転載または複製することを固くお断りいたします。
１３．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弊社営業窓口までご照会ください。

はじめに

本マニュアルは、RX600 シリーズマイコン用リアルタイム OS 製品「RI600/4」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、(社)トロン協会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、(社)トロン協会ホームページ(http://www.assoc.tron.org/)から入手が可能です。μITRON 仕様の著作権は(社)トロン協会に属しています。
- 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

1. 本マニュアルの構成 1
2. 概要 2
2.1 特長 2
2.2 提供ソフトウェア構成 3
2.3 動作環境 3
3. カーネル入門 4
3.1 カーネルの動作原理 4
3.2 サービスコール 6
3.3 オブジェクト 6
3.4 タスク 8
3.4.1 タスクの状態 8
3.4.2 タスクのスケジューリング(優先度とレディキュー) 10
3.4.3 タスクの待ち行列 11
3.5 システムの状態 12
3.5.1 タスクコンテキストと非タスクコンテキスト 12
3.5.2 ディスパッチ禁止/許可状態 12
3.5.3 CPUロック/ロック解除状態 13
3.5.4 ディスパッチ保留状態 13
3.6 処理の単位と優先順位 14
3.7 割込み 15
3.7.1 割込みの種類 15
3.7.2 PSWレジスタのIビットおよびIPLビットの扱い 16
3.7.3 割込みの禁止 17
3.7.4 使用できるサービスコール 17
3.7.5 ノンマスカブル割込み、例外について 18
3.7.6 RXマイコンの高速割込み機能 18
3.8 スタック 18
4. カーネルの機能 19
4.1 モジュール構成 19
4.2 モジュール概要 20
4.3 タスク管理機能 22
4.4 タスク付属同期機能 25
4.5 セマフォ 27
4.5.1 優先度逆転問題 28
4.6 イベントフラグ 29
4.7 データキュー 31
4.8 メールボックス 33

4.9 ミューテックス 35
4.9.1 ベース優先度と現在優先度 37
4.10 メッセージバッファ 38
4.11 固定長メモリプール 40
4.12 可変長メモリプール 42
4.12.1 空き領域の断片化について 43
4.13 時間管理機能 44
4.13.1 タスクのタイムアウト 44
4.13.2 タスクの遅延 45
4.13.3 周期ハンドラ 45
4.13.4 アラームハンドラ 47
4.13.5 時間の精度 48
4.13.6 注意事項 49
4.14 システム状態管理機能 50
4.15 割込み管理機能 52
4.16 システム構成管理機能 52
4.17 オブジェクトリセット機能 53
4.18 カーネルのアイドリング 53

5. サービスコールリファレンス 54

5.1 ヘッダファイル 54
5.2 基本データ型 54
5.3 サービスコールのリターン値とエラーコード 55
5.3.1 概要 55
5.3.2 メインエラーコードとサブエラーコード 55
5.4 システム状態とサービスコール 55
5.4.1 タスクコンテキストと非タスクコンテキスト 55
5.4.2 CPUロック状態 56
5.4.3 ディスパッチ禁止状態 56
5.4.4 カーネル管理外の割込みハンドラなど 56
5.5 μITRON 仕様外の仕様 56
5.6 タスク管理機能 57
5.6.1 タスクの起動(act_tsk, iact_tsk) 58
5.6.2 タスクの起動要求のキャンセル(can_act, ican_act) 59
5.6.3 タスクの起動(起動コード指定)(sta_tsk, ista_tsk) 60
5.6.4 自タスクの終了(ext_tsk) 61
5.6.5 タスクの強制終了(ter_tsk) 62
5.6.6 タスク優先度の変更(chg_pri, ichg_pri) 63
5.6.7 タスク優先度の参照(get_pri, iget_pri) 64
5.6.8 タスクの状態参照(ref_tsk, iref_tsk) 65
5.6.9 タスクの状態参照(簡易版)(ref_tst, iref_tst) 67
5.7 タスク付属同期機能 69
5.7.1 起床待ち(slp_tsk, tslp_tsk) 70
5.7.2 タスクの起床(wup_tsk, iwup_tsk) 71

5.7.3 タスク起床要求のキャンセル(can_wup, ican_wup) 72
5.7.4 待ち状態の強制解除(rel_wai, irel_wai) 73
5.7.5 強制待ち状態への移行(sus_tsk, isus_tsk) 74
5.7.6 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) 75
5.7.7 タスク遅延(dly_tsk) 76
5.8 同期・通信(セマフォ)機能 77
5.8.1 セマフォ資源の返却(sig_sem, isig_sem) 78
5.8.2 セマフォ資源の獲得(wai_sem, pol_sem, ipol_sem, twai_sem) 79
5.8.3 セマフォの状態参照(ref_sem, iref_sem) 80
5.9 同期・通信(イベントフラグ)機能 81
5.9.1 イベントフラグのセット(set_flg, iset_flg) 82
5.9.2 イベントフラグのクリア(clr_flg, iclr_flg) 83
5.9.3 イベントフラグ待ち(wai_flg, pol_flg, ipol_flg, twai_flg) 84
5.9.4 イベントフラグの状態参照(ref_flg, iref_flg) 86
5.10 同期・通信(データキュー)機能 87
5.10.1 データキューへの送信(snd_dtq,psnd_dtq,ipsnd_dtq,tsnd_dtq, fsnd_dtq, ifsnd_dtq) 88
5.10.2 データキューからの受信(rcv_dtq, prcv_dtq, iprcv_dtq, trcv_dtq) 90
5.10.3 データキューの状態参照(ref_dtq, iref_dtq) 91
5.11 同期・通信(メールボックス)機能 92
5.11.1 メールボックスへの送信(snd_mbx, isnd_mbx) 93
5.11.2 メールボックスからの受信(rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx) 95
5.11.3 メールボックスの状態参照(ref_mbx, iref_mbx) 97
5.12 拡張同期・通信(ミューテックス)機能 98
5.12.1 ミューテックスのロック(loc_mtx, ploc_mtx, tloc_mtx) 99
5.12.2 ミューテックスのロック解除(unl_mtx) 100
5.12.3 ミューテックスの状態参照(ref_mtx) 101
5.13 同期・通信(メッセージバッファ)機能 102
5.13.1 メッセージバッファへの送信(snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf) 103
5.13.2 メッセージバッファからの受信(rcv_mbf, prcv_mbf, trcv_mbf) 105
5.13.3 メッセージバッファの状態参照(ref_mbf, iref_mbf) 107
5.14 メモリプール管理(固定長メモリプール)機能 108
5.14.1 固定長メモリブロックの獲得(get_mpf, pget_mpf, ipget_mpf, tget_mpf) 109
5.14.2 固定長メモリブロックの返却(rel_mpf, irel_mpf) 111
5.14.3 固定長メモリプールの状態参照(ref_mpf, iref_mpf) 112
5.15 メモリプール管理(可変長メモリプール)機能 113
5.15.1 可変長メモリブロックの獲得(get_mpl, tget_mpl, pget_mpl, ipget_mpl) 114
5.15.2 可変長メモリブロックの返却(rel_mpl) 116
5.15.3 可変長メモリプールの状態参照(ref_mpl, iref_mpl) 117
5.16 時間管理(システム時刻管理)機能 118
5.16.1 システム時刻の設定(set_tim, iset_tim) 119
5.16.2 システム時刻の参照(get_tim, iget_tim) 120
5.16.3 タイムティックの供給(isig_tim) 121
5.17 時間管理(周期ハンドラ)機能 122
5.17.1 周期ハンドラの動作開始(sta_cyc, ista_cyc) 123

5.17.2 周期ハンドラの動作停止(stp_cyc, istp_cyc) 124
5.17.3 周期ハンドラの状態参照(ref_cyc, iref_cyc) 125
5.18 時間管理(アラームハンドラ)機能 126
5.18.1 アラームハンドラの動作開始(sta_alm, ista_alm) 127
5.18.2 アラームハンドラの動作停止(stp_alm, istp_alm) 128
5.18.3 アラームハンドラの状態参照(ref_alm, iref_alm) 129
5.19 システム状態管理機能 130
5.19.1 タスクの優先順位の回転(rot_rdq, irot_rdq) 131
5.19.2 実行状態のタスクIDの参照(get_tid, iget_tid) 132
5.19.3 CPUロック状態への移行(loc_cpu, iloc_cpu) 133
5.19.4 CPUロック状態の解除(unl_cpu, iunl_cpu) 135
5.19.5 ディスパッチの禁止(dis_dsp) 136
5.19.6 ディスパッチの許可(ena_dsp) 137
5.19.7 コンテキストの参照(sns_ctx) 138
5.19.8 CPUロック状態の参照(sns_loc) 139
5.19.9 ディスパッチ禁止状態の参照(sns_dsp) 140
5.19.10 ディスパッチ保留状態の参照(sns_dpn) 141
5.19.11 カーネルの起動(vsta_knl, ivsta_knl) 142
5.19.12 システムダウン(vsys_dwn, ivsys_dwn) 143
5.20 割込み管理機能 144
5.20.1 割込みマスクの変更(chg_ims, ichg_ims) 145
5.20.2 割込みマスクの参照(get_ims, iget_ims) 146
5.20.3 カーネル管理割込みハンドラからの復帰(ret_int) 147
5.21 システム構成管理機能 148
5.21.1 バージョン情報の参照(ref_ver, iref_ver) 149
5.22 オブジェクトリセット機能 151
5.22.1 データキューのリセット(vrst_dtq) 152
5.22.2 メールボックスのリセット(vrst_mbx) 153
5.22.3 メッセージバッファのリセット(vrst_mbf) 154
5.22.4 固定長メモリプールのリセット(vrst_mpf) 155
5.22.5 可変長メモリプールのリセット(vrst_mpl) 156
5.23 定数とマクロ 157
5.23.1 ヘッダファイル 157
5.23.2 定義内容 158
6. アプリケーション記述方法 161
6.1 ヘッダファイル 161
6.2 変数の扱い 161
6.3 タスク 162
6.3.1 カーネルへの登録 162
6.3.2 コーディング 162
6.3.3 起動時のCPU状態 163
6.4 割込みハンドラ 164
6.4.1 カーネルへの登録 164
6.4.2 コーディング 164

6.4.3 起動時のCPU状態 165
6.5 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) 166
6.5.1 カーネルへの登録 166
6.5.2 コーディング 166
6.5.3 起動時のCPU状態 167
6.6 システムダウンルーチン 168
6.6.1 概要 168
6.6.2 コーディング 168
6.6.3 起動時のCPU状態 169
6.7 浮動小数点演算命令を使用する場合の注意 170
6.8 DSP 機能をサポートしたマイコンを使用する場合の注意 171
7. ロードモジュール生成手順 173
7.1 概要 173
7.2 スタートアップファイル(resetprg.c)の作成 175
7.3 カーネルライブラリ 180
7.4 セクション一覧 180
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 181
7.6 CPU の動作モード 181
8. コンフィギュレータ(cfg600) 182
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 182
8.2 cfg ファイル内の表現形式 182
8.3 デフォルト cfg ファイル 184
8.4 cfg ファイルの定義項目 185
8.4.1 システム定義(system) 186
8.4.2 system.contextの注意事項 188
8.4.3 システムクロック定義(clock) 191
8.4.4 タスク定義(task[]) 193
8.4.5 セマフォ定義(semaphore[]) 195
8.4.6 イベントフラグ定義(flag[]) 197
8.4.7 データキュー定義(dataqueue[]) 199
8.4.8 メールボックス定義(mailbox[]) 200
8.4.9 ミューテックス定義(mutex[]) 202
8.4.10 メッセージバッファ定義(message_buffer[]) 203
8.4.11 固定長メモリプール定義(memorypool[]) 205
8.4.12 可変長メモリプール定義(variable_memorypool[]) 207
8.4.13 周期ハンドラ定義(cyclic_hand[]) 209
8.4.14 アラームハンドラ定義(alarm_hand[]) 211
8.4.15 可変ベクタ割込み定義(interrupt_vector[]) 212
8.4.16 固定ベクタ割込み定義(interrupt_fvector[]) 214
8.5 コンフィギュレータの実行 216
8.5.1 コンフィギュレータ概要 216
8.5.2 環境設定 218

8.5.3 コンフィギュレータ起動方法 218
8.5.4 コマンドオプション 218
8.6 エラーメッセージ 219
8.6.1 エラー形式とエラーレベル 219
8.6.2 メッセージ一覧 219
9. テーブル生成ユーティリティ(mkritbl) 222
9.1 概要 222
9.2 環境設定 223
9.3 テーブル生成ユーティリティ起動方法 223
9.4 注意事項 223
10. GUI コンフィギュレータ 224
11. サンプルプログラム 225
11.1 概要 225
11.2 ソースリスト 226
11.3 サンプル cfg ファイル 227
12. スタックサイズの算出方法 228
12.1 スタックの種類 228
12.2 “Call Walker” 228
12.3 各タスクのユーザスタックサイズの算出方法 229
12.4 システムスタックサイズの算出方法 230

1. 本マニュアルの構成

本マニュアルは、以下の章から構成されています。

■2. 概要

本製品の概要について解説しています。

■3. カーネル入門

本製品を使用する上で必要となる考え方や、本製品の基幹であるカーネルに関する基本的な項目を解説しています。

■4. カーネルの機能

カーネルの機能について解説しています。

■5. サービスコールリファレンス

カーネルのサービスコール仕様を記載しています。

■6. アプリケーション記述方法

タスクやハンドラの記述方法を説明しています。

■7. ロードモジュール生成手順

ロードモジュールの作成手順の概要を説明しています。

■8. コンフィギュレータ(cfg600)

コマンドラインコンフィギュレータ cfg600 へ入力する cfg ファイルの作成方法、および cfg600 の使用方法を説明しています。

■9. テーブル生成ユーティリティ(mkritbl)

テーブル生成ユーティリティ mkritbl の使用方法を説明しています。

■10. GUI コンフィギュレータ

GUI コンフィギュレータについて紹介しています。GUI コンフィギュレータの使用方法は、オンラインヘルプを参照してください。

■11. サンプルプログラム

サンプルアプリケーションについて説明しています。

■12. スタックサイズの算出方法

タスクやハンドラなどに必要なスタックサイズの算出方法を説明しています。

2. 概要

2.1 特長

(1) μITRON4.0 仕様に準拠

RI600/4 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ました。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 RI600/4 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

RI600/4 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、RI600/4 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。また、High-performance Embedded Workshop のリアルタイム OS デバッグ機能は、RI600/4 にも対応しています。

(5) コンフィギュレータによる容易なカーネル構築

RI600/4 では、コマンドラインコンフィギュレータ cfg600 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。

また、GUI 化されたコンフィギュレータも用意しています。GUI コンフィギュレータを使用すると、cfg ファイルの記述形式を習得しなくとも、容易にカーネルを構築することができます。

2.2 提供ソフトウェア構成

(1) カーネル

RX600 シリーズマイコン用のリアルタイム OS 本体です。

(2) cfg600(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) mkritbl(テーブル生成ユーティリティ)

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。

(4) GUI 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。GUI 画面上でカーネル構築情報を入力し、cfg ファイルを出力します。

2.3 動作環境

表 2.1に、動作環境を示します。

表2.1 動作環境

項目	動作環境
対象 CPU	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 ®2000, Windows®XP , WIndows Vista®
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」

3. カーネル入門

3.1 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 3.1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。

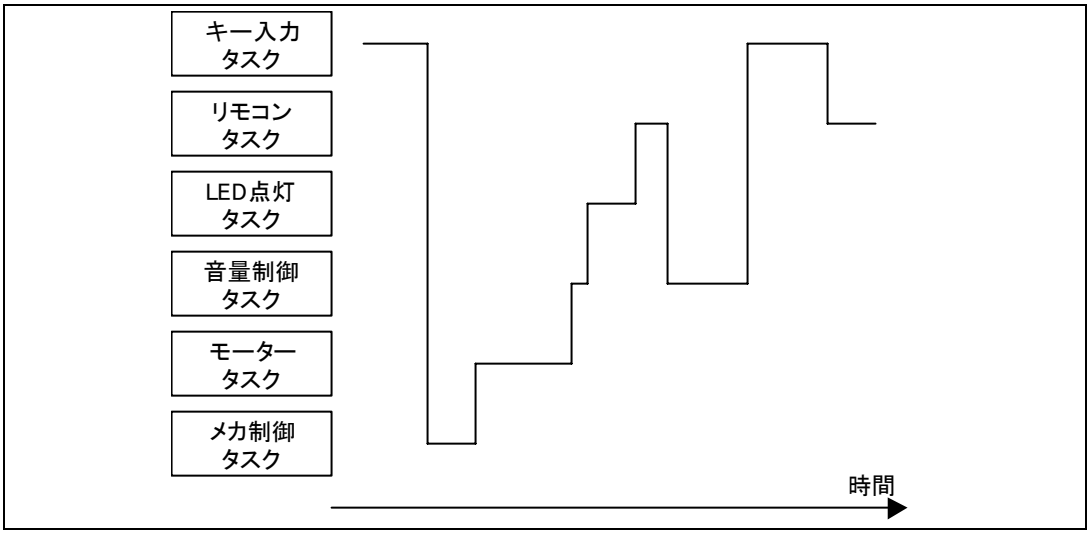

図3.1 マルチタスク動作

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 3.2参照)。

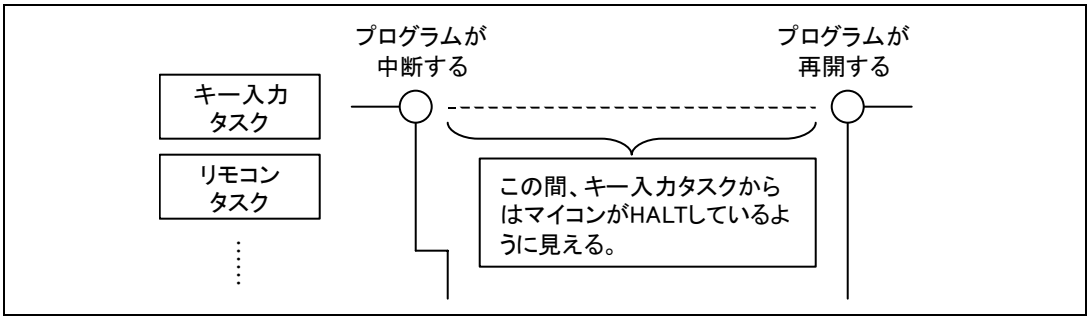

図3.2 タスクの中断と再開

図 3.2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 3.3参照)。

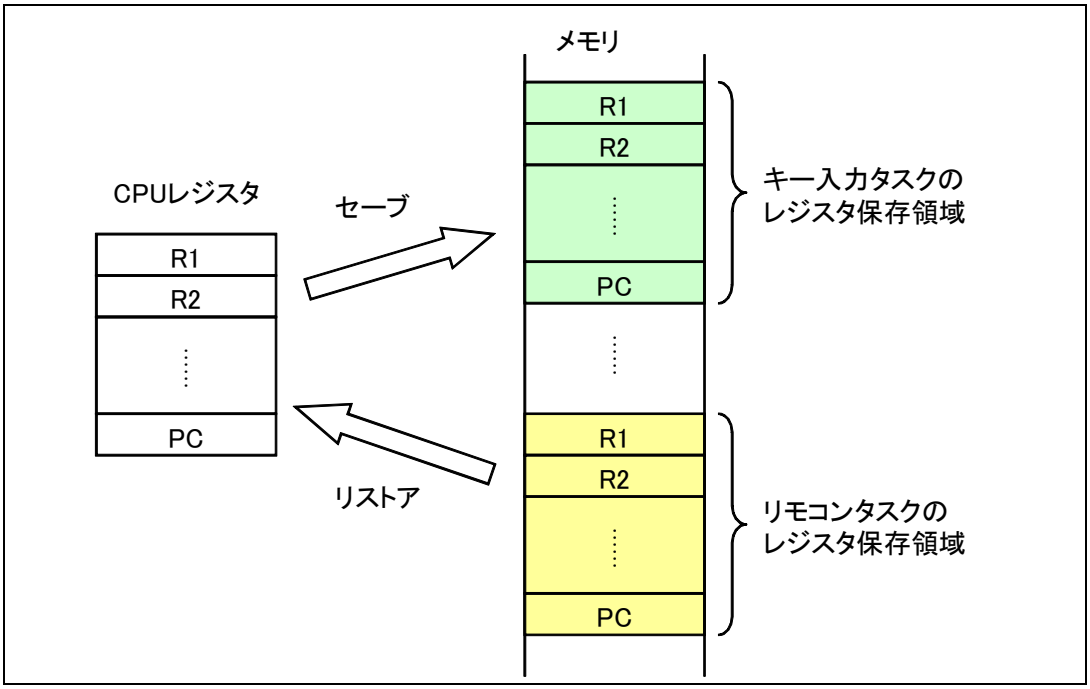

図3.3 タスクの切り替え

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。

3.2 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 3.4参照)。

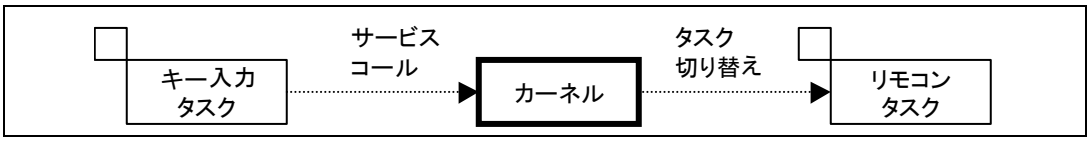

図3.4 サービスコール

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);
```

3.3 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 RI600/4 では、タスクの識別をそのタスクの名前 (ID 名称) で指定し、その ID 名称からタスクの ID 番号への変換を RI600/4 に付属しているプログラム"コンフィギュレータ cfg600"が自動的に行います。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 3.5は、タスクを識別する様子を示したものです。

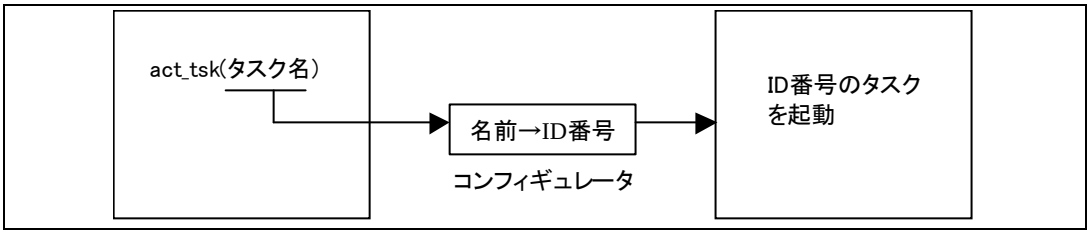

図3.5 タスクの識別

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。

3.4 タスク

3.4.1 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 3.6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。

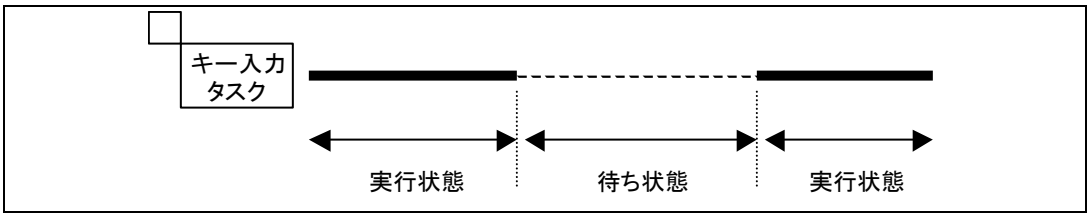

図3.6 タスクの状態

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 3.7に、タスクの状態遷移図を示します。

(1) 休止状態(DORMANT)

カーネルに登録された後、まだ起動されていない状態、または終了後の状態です。

(2) 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。

(3) 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。

(4) 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。

(5) 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。

(6) 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。

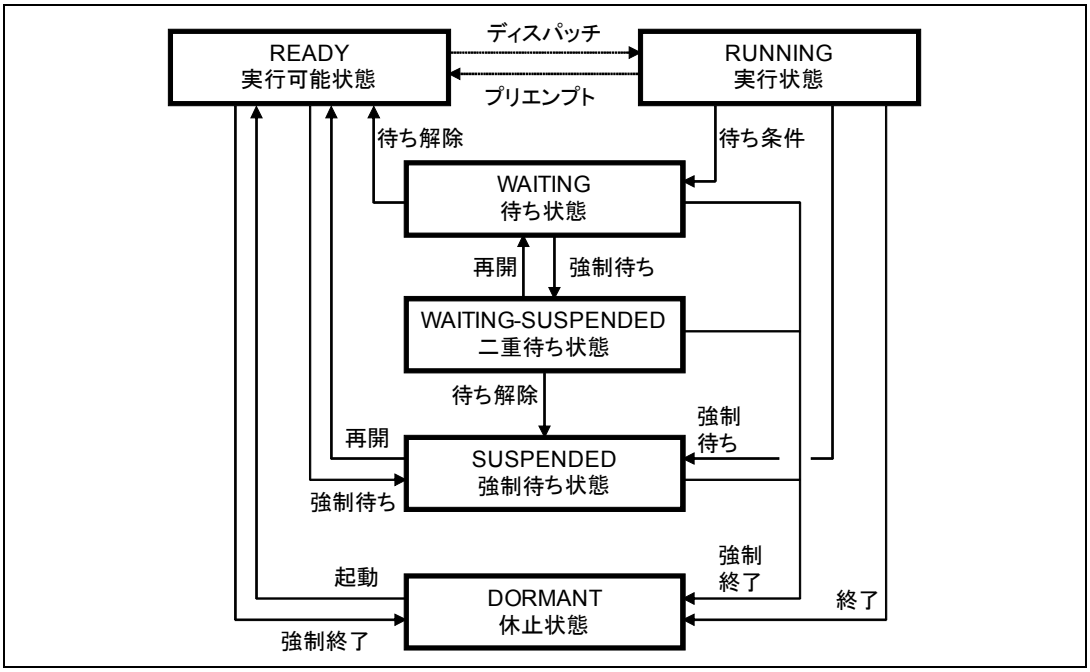

図3.7 タスクの状態遷移図

3.4.2 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 3.8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。

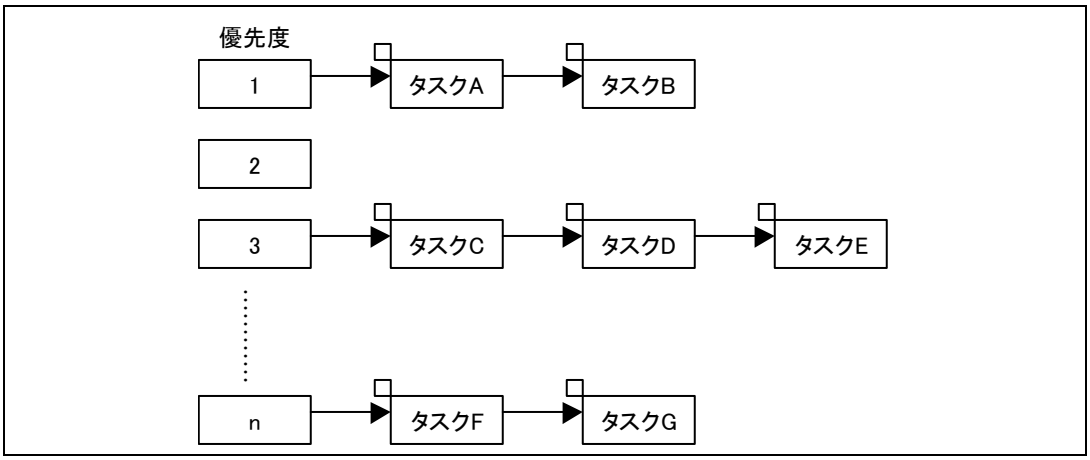

図3.8 レディキュー(実行待ち状態)

3.4.3 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 3.9および図 3.10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。

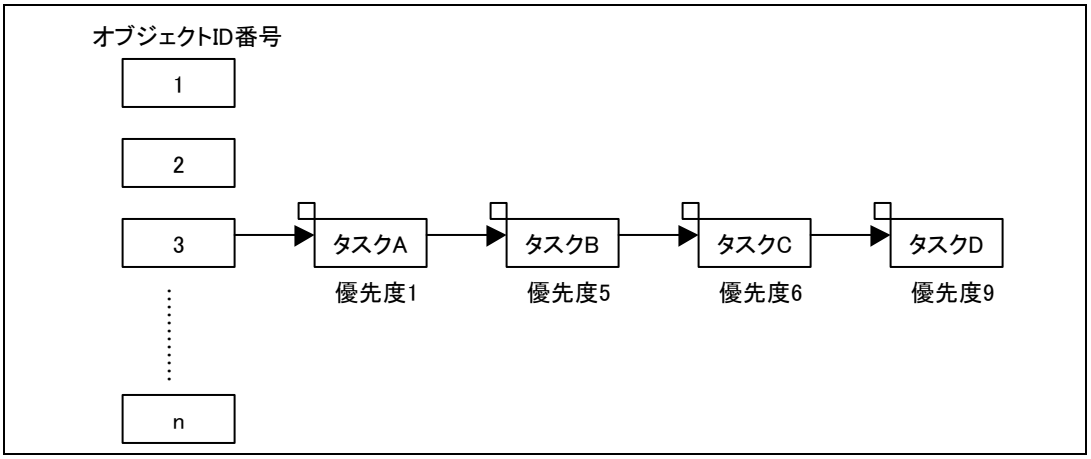

図3.9 TA_TPRI 属性の待ち行列

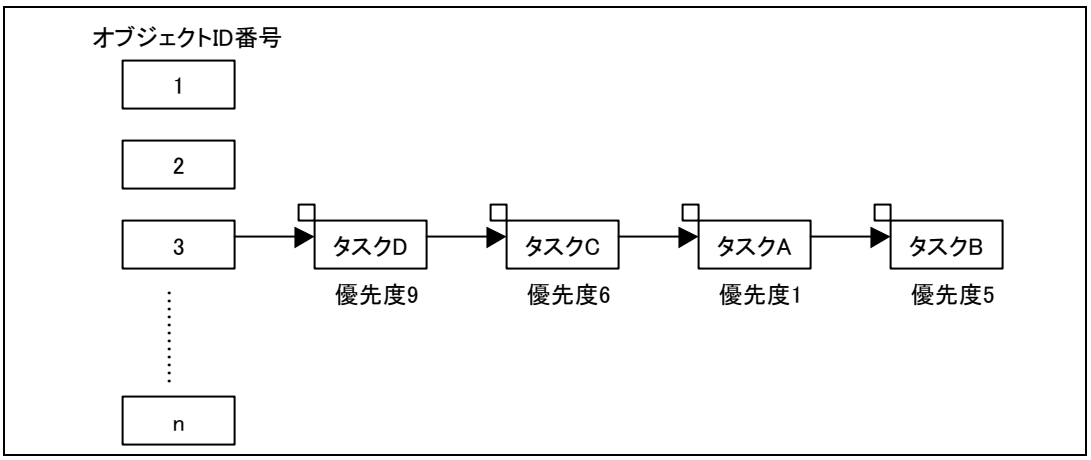

図3.10 TA_TFIFO 属性の待ち行列

3.5 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。

3.5.1 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 3.1に示します。

表3.1 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	3.5.2, 3.5.3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）

3.5.2 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
- chg_ims で割込みマスクを 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。

3.5.3 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 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 3.2のように制限されます。

表3.2 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 *	

【注】 CPU ロック状態は解除されます。

CPU ロック状態は、カーネル管理割込み処理間、あるいはカーネル管理割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。

3.5.4 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。

3.6 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) 割込みハンドラ
割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。

また、各処理単位は以下の優先順位で処理されます。
(1) 割込みハンドラ、タイムイベントハンドラ
(2) ディスパッチャ (カーネルの処理の一部)
(3) 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。

割込みハンドラは、割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。

3.7 割込み

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

3.7.1 割込みの種類

割込みは、カーネル管理割込みとカーネル管理外割込みに分類されます。

- **カーネル管理割込み**

  カーネル割込マスクレベル (system.system_IPL)より割込み優先レベルが低い割込みをカーネル管理割込みといいます。
  カーネル管理割込みハンドラでは、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込みが発生した場合、カーネル管理割込みを受け付け可能となるまで割込み受理が遅延されます。

- **カーネル管理外割込み**

  カーネル割込マスクレベル (system.system_IPL)より割込み優先レベルが高い割込みをカーネル管理外割込みといいます。また、固定ベクタ割込みも、カーネル管理外割込みの扱いとなります。
  カーネル管理外割込みハンドラでは、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込みが発生した場合、直ちに割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込み応答が可能です。

図 3.11に、カーネル割込みマスクレベルを 10 に設定した場合の、カーネル管理割込みとカーネル管理外割込みの関係を示します。

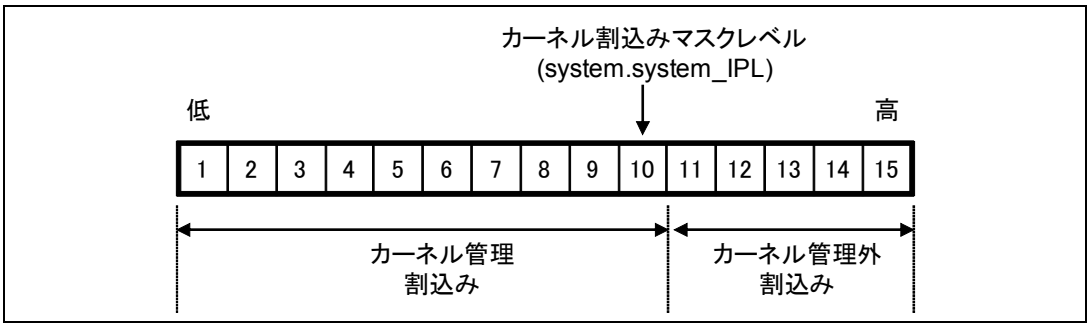

図3.11 割込み優先レベルと割込みの種類

3.7.2 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

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、全割込みが許可された状態です。

タスクでは、I ビットを 0 にしてはなりません。なお、RI600/4 では I ビットを 0 にする方法は提供されていません。

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。

(2) 割込みハンドラ

I ビットの初期値は、cfg ファイルで割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。

割込みハンドラ起動時の IPL ビットは、当該割込み優先レベルとなります。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 ビットは割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。

3.7.3 割込みの禁止

割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込みは、カーネル管理割込みのみです。カーネル管理外割込みを禁止したい場合は、(b), (c)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。「3.5.3 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

(2) 特定の割込み要因を禁止する

特定の割込み要因を禁止するには、割込みコントローラ(ICU)の割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。

3.7.4 使用できるサービスコール

(1) 割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。

(2) カーネル管理外割込みハンドラなど、PSW.IPL>カーネル割込みマスクレベル(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。

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

3.7.5 ノンマスカブル割込み、例外について

ノンマスカブル割込みは、カーネル管理外割込みの扱いとなります。

また、命令実行起因で発生する以下の例外も、カーネル管理外割込みの扱いとなります。これらの例外に対する割込みハンドラからサービスコールを呼び出した場合、E_CTX エラーは検出されないことに注意してください。

- 未定義命令例外
- 特権命令例外
- アクセス例外
- 浮動小数点例外
- INT 命令例外
- BRK 命令例外

3.7.6 RX マイコンの高速割込み機能

RX マイコンは「高速割込み」機能をサポートしています。ひとつの割込み要因だけを高速割込みとすることができます。高速割込みは、割込み優先レベル 15 として扱われます。高速割込みを使用する場合は、割込み優先レベル 15 の割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込みを使用する場合は、その割込みはカーネル管理外割込みとして扱う必要があります。つまり、カーネル割込みマスクレベル(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。

3.8 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定義する際に、スタックサイズとセクション名を指定することで、指定されたセクション名で各タスクのスタック領域が生成されます。

- **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。

4. 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。

4.1 モジュール構成

カーネルは、図 4.1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・割込みハンドラ・アラームハンドラおよび周期ハンドラから構成されます。

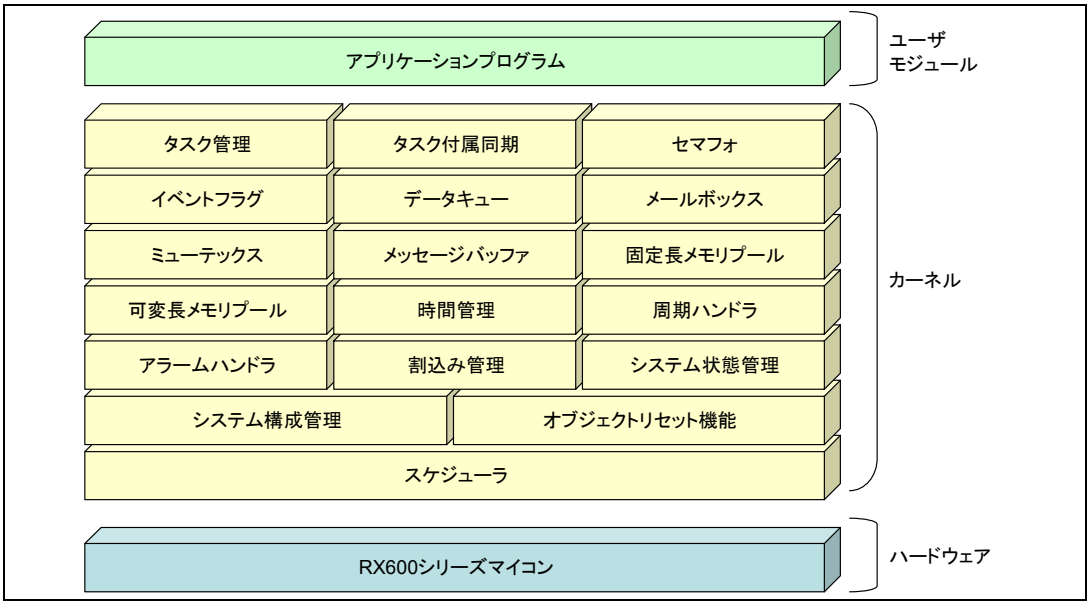

図4.1 カーネルの構成

4.2 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) 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。

(5) 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。

(6) 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。

(7) 割込み管理機能

割込みハンドラからの復帰処理を行います。

(8) 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。

(9) 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。

(10) 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。

(11) 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。

4.3 タスク管理機能

タスク管理機能とは、タスクの起動、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、cfg ファイルに task[]を記述します。task[]では、以下の情報を指定します。

- ID 名称
- タスクの開始アドレス
- ユーザスタックサイズ
- スタックを配置するセクション
- 初期優先度
- 初期起動の指定
- 拡張情報

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。

(2) 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。

(3) 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。

(4) 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。

(5) 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。

(6) 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 4.2参照)。
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 4.3参照)。

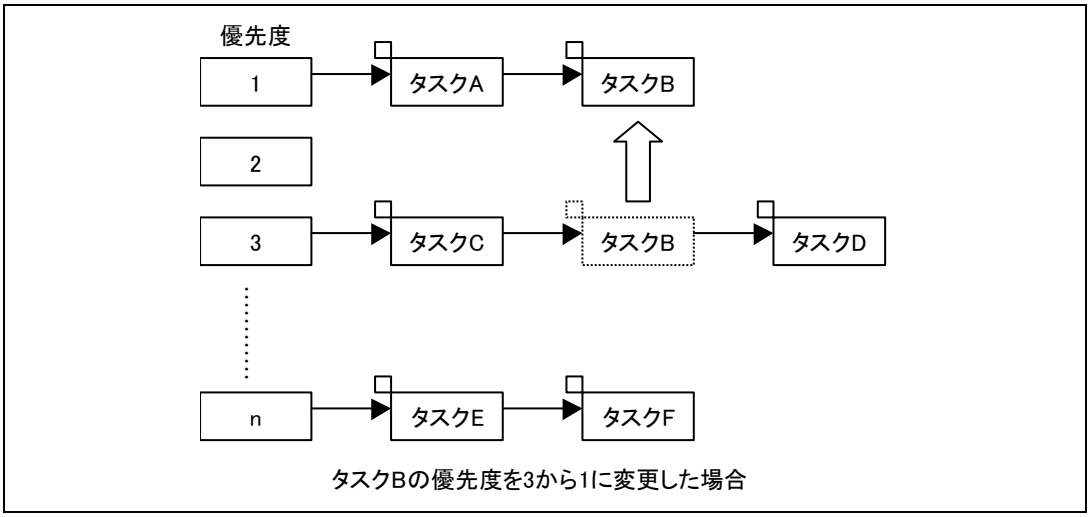

図4.2　実行可能状態または実行状態のタスクの優先度を変更した時のレディキュー

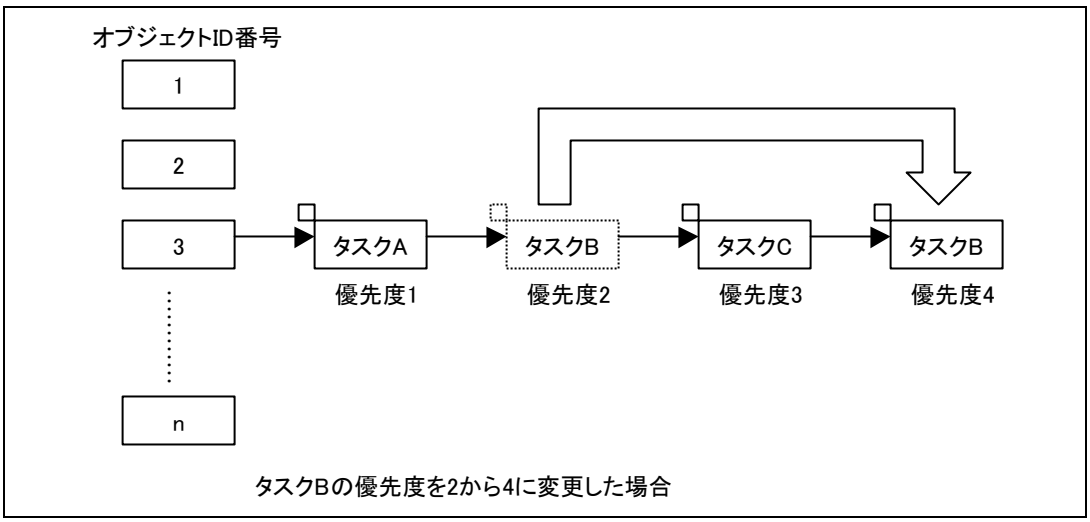

図4.3　TA_TPRI 属性のオブジェクト待ち状態のタスクの優先度を変更した時の待ち行列

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、「4.9 ミューテックス」を参照してください。

(7) 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。

(8) 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。

(9) 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。

4.4 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 4.4参照)。

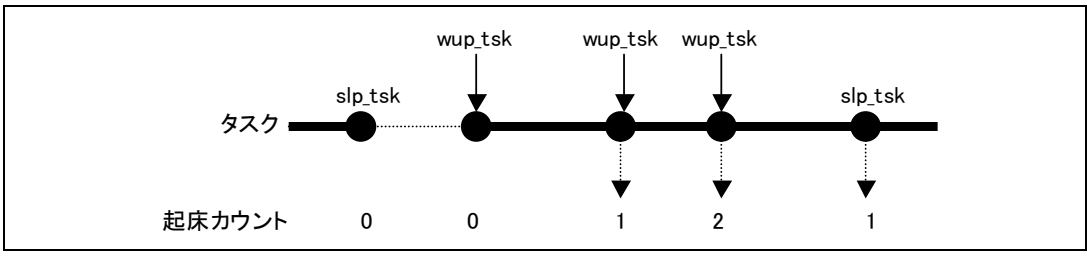

図4.4 起床要求の蓄積

(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 4.5参照)。

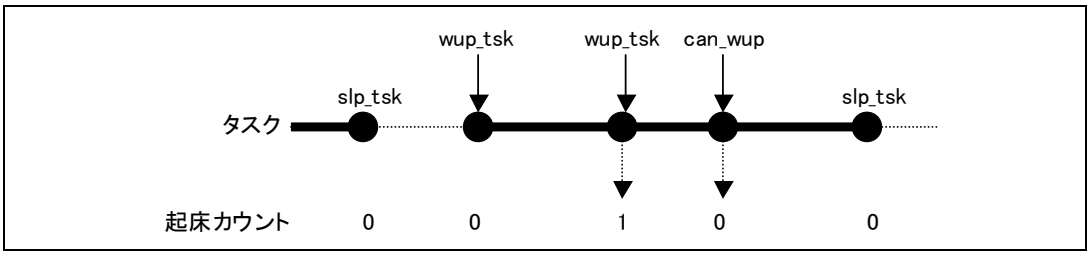

図4.5 起床要求のキャンセル

(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 4.6参照)。

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。

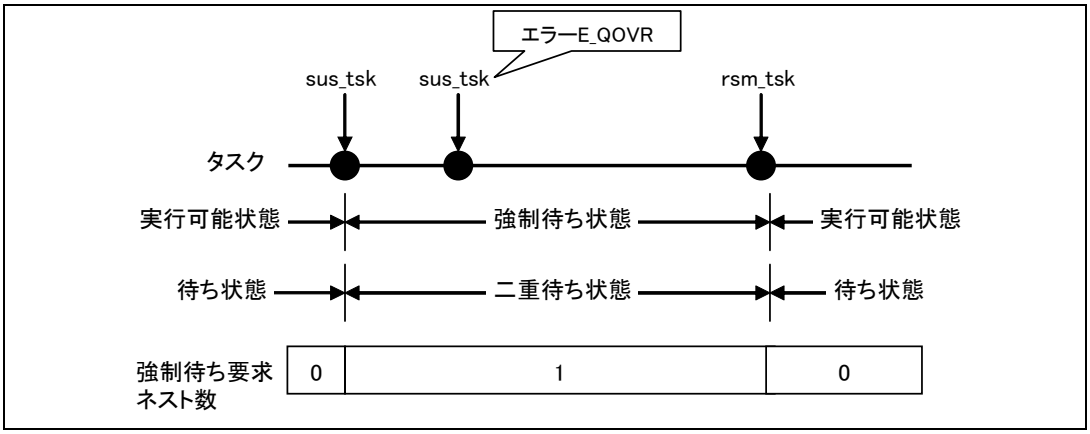

図4.6 タスクの強制待ちと再開

(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。

4.5 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。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、cfg ファイルに semaphore[]を記述します。semaphore[]では、以下の情報を指定します。

- ID 名称
- 待ち行列の並び方(TA_TFIFO(FIFO 順)または TA_TPRI(タスク優先度順))
- セマフォカウンタ初期値
- セマフォカウンタ最大値

図 4.7にセマフォの動作例を示します。

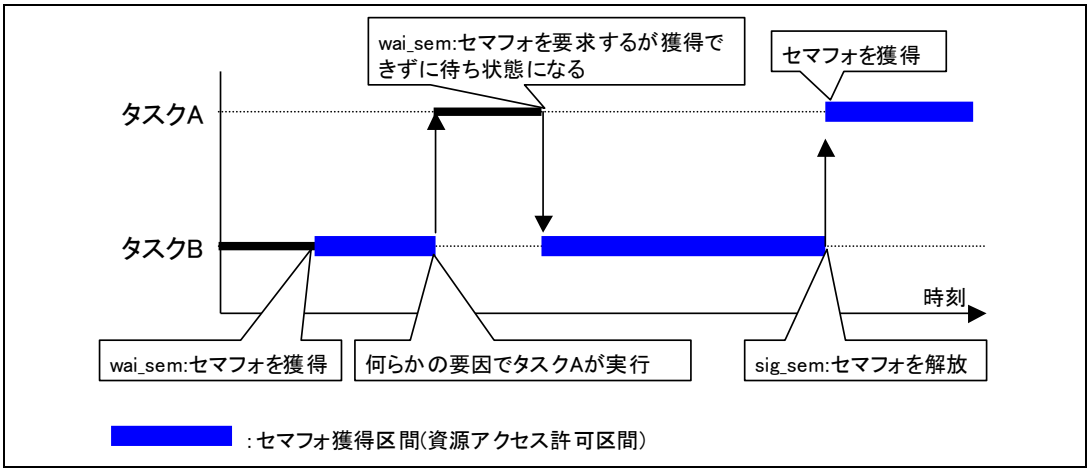

図4.7 セマフォの動作例

セマフォを操作するサービスコールには、次のものがあります。

(1) セマフォを獲得する(wai_sem, twai_sem)

セマフォを獲得します。セマフォカウンタが正なら、セマフォカウンタを 1 減算します。セマフォカウンタが 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。

(2) 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、セマフォカウンタが 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。

(3) 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、なければセマフォカウンタを 1 加算します。

(4) セマフォの状態を参照する(ref_sem, iref_sem)

セマフォのカウンタ値や待ちタスク ID を参照します。

4.5.1 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 4.8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。

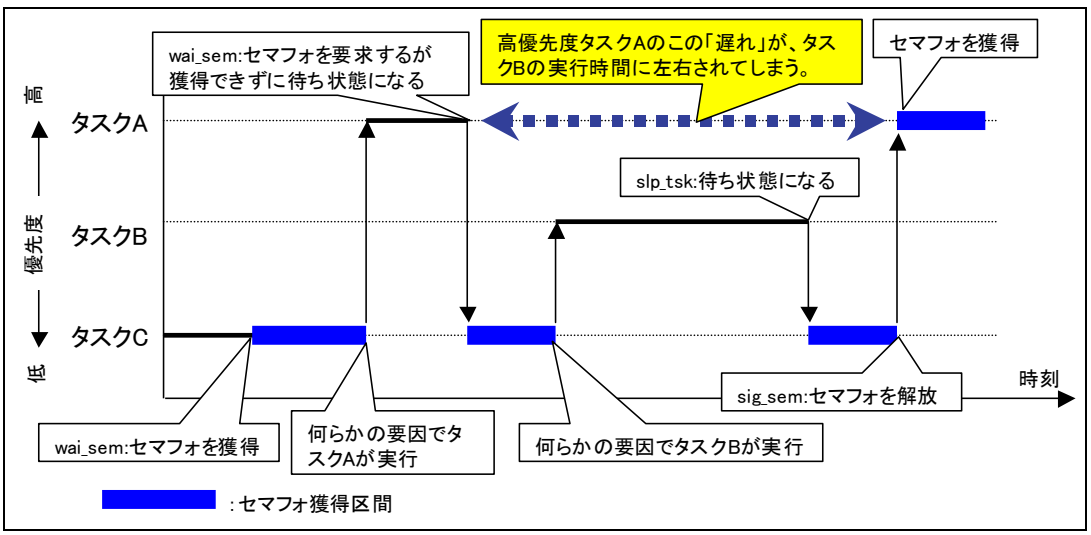

図4.8 優先度逆転現象

4.6 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群のすべて(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 4.9に示します。

図 4.9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。

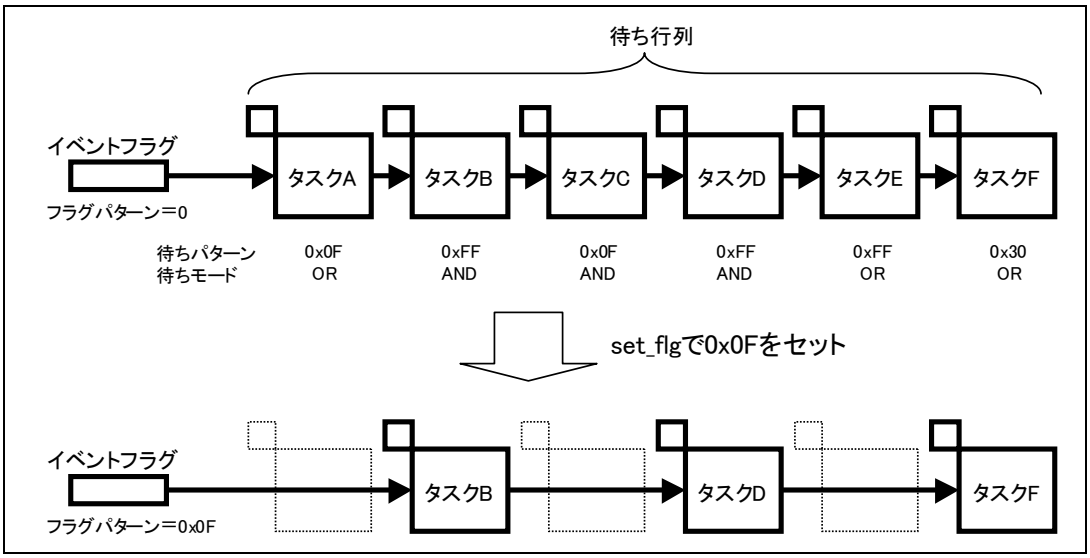

図4.9 イベントフラグの動作例

イベントフラグを生成するには、cfg ファイルに flag[]を記述します。flag[]では、以下の情報を指定します。

- ID 名称
- 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)

イベントフラグを操作する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。

(2) 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。

(3) 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。

(4) 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。

(5) 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。

4.7 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 4.10に、データキューの概要を示します。

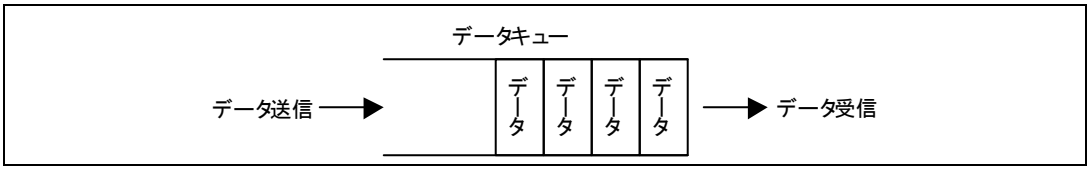

図4.10 データキュー

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、cfg ファイルに dataqueue[]を記述します。dataqueue[]では、以下の情報を指定します。

- ID 名称
- 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

データキューを操作するサービスコールには、次のものがあります。

(1) 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。

(2) 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。

(3) 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。

(4) 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。

(5) 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。

(6) 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。

4.8 メールボックス

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4.11に、メールボックスの概要を示します。

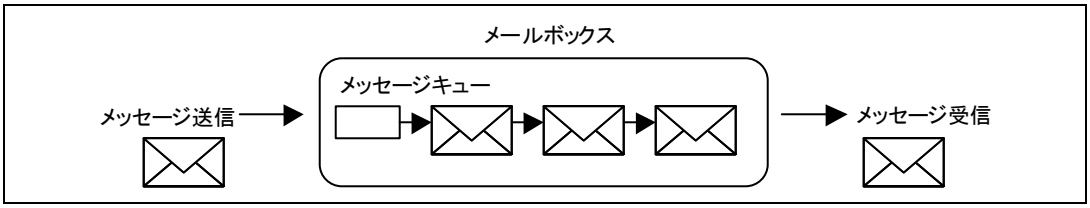

図4.11 メールボックス

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、cfg ファイルに mailbox[]を記述します。mailbox[]では、以下の情報を指定します。

- ID 名称
- 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

メールボックスを操作するサービスコールには、次のものがあります。

(1) 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。

(2) 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。

(3) 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。

(4) 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。

4.9 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。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 4.12に、ミューテックスの動作例を示します。

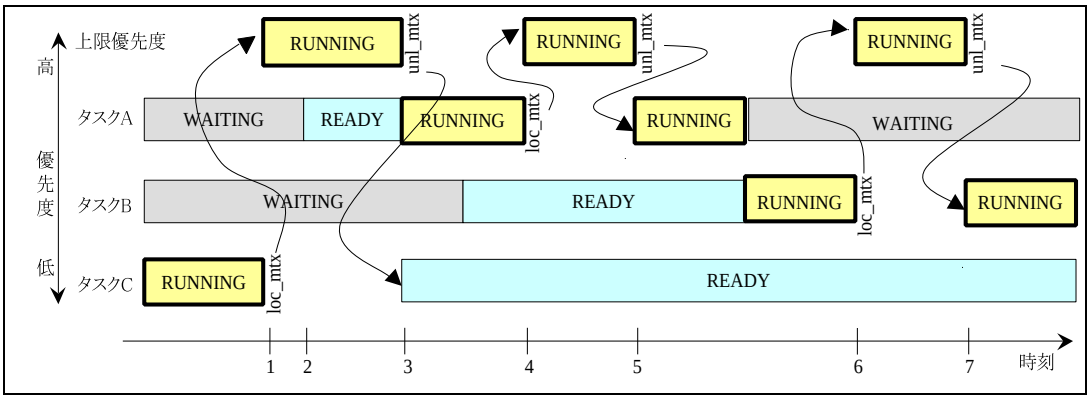

図4.12 ミューテックスの動作例

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、cfg ファイルに mutex[]を記述します。mutex[]では、以下の情報を指定します。

- ID 名称
- 上限優先度

ミューテックスを操作する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。

(2) 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。

(3) 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。

(4) 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。

4.9.1 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、E_ILUSE エラーになります。

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。

4.10 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 4.13に、メッセージバッファの概要を示します。

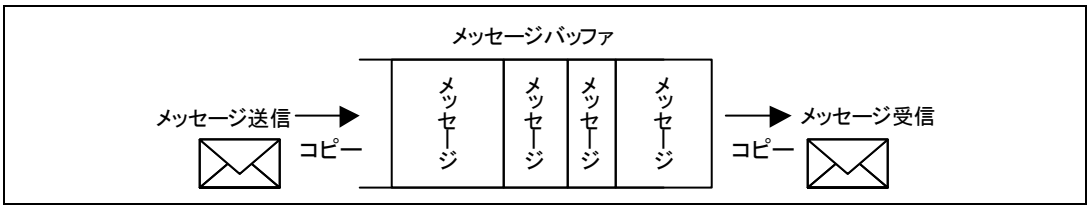

図4.13 メッセージバッファ

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、cfg ファイルに message_buffer[]を記述します。message_buffer[]では、以下の情報を指定します。

- ID 名称
- バッファサイズ
- バッファを配置するセクション
- 最大メッセージサイズ

メッセージバッファを操作するサービスコールには、次のものがあります。

(1) 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に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。

(2) 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。

(3) 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。

(4) 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。

(5) 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。

4.11 固定長メモリプール

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 4.14に固定長メモリプールの概要を示します。

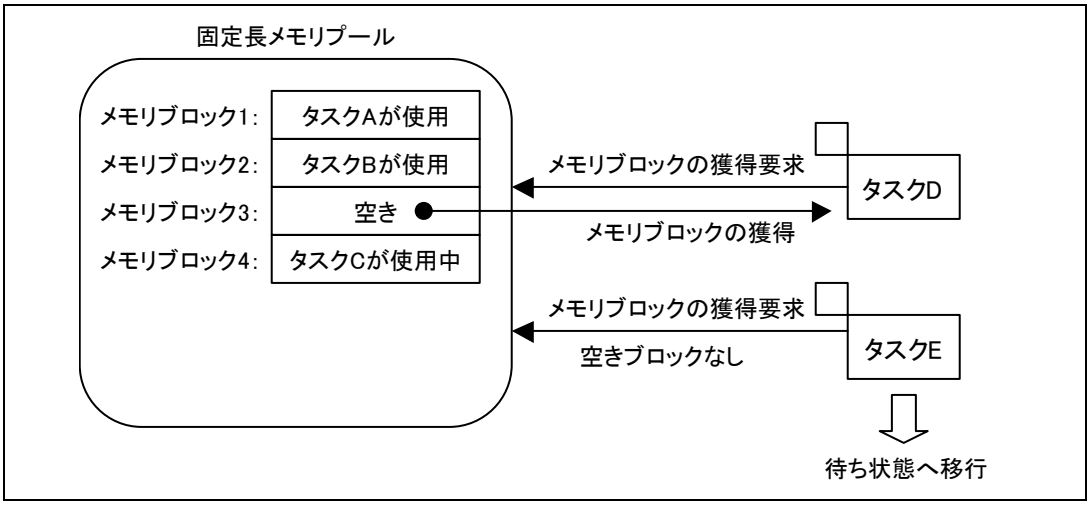

図4.14 固定長メモリプール

固定長メモリプールを生成するには、cfg ファイルに memorypool[]を記述します。memorypool[]では、以下の情報を指定します。

- ID 名称
- 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
- メモリプールを配置するセクション

固定長メモリプールを操作するサービスコールには、次のものがあります。

(1) 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。

(2) 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。

(3) 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。

(4) 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。

4.12 可変長メモリプール

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、cfg ファイルに variable_memorypool[]を記述します。variable_memorypool[]では、以下の情報を指定します。なお、メモリ獲得待ち行列は、常に FIFO 順で管理されます。

- ID 名称
- 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
- メモリプールを配置するセクション

可変長メモリプールを操作するサービスコールには、次のものがあります。

(1) 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。

(2) 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。

(3) 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。

(4) 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。

4.12.1 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 4.15)。

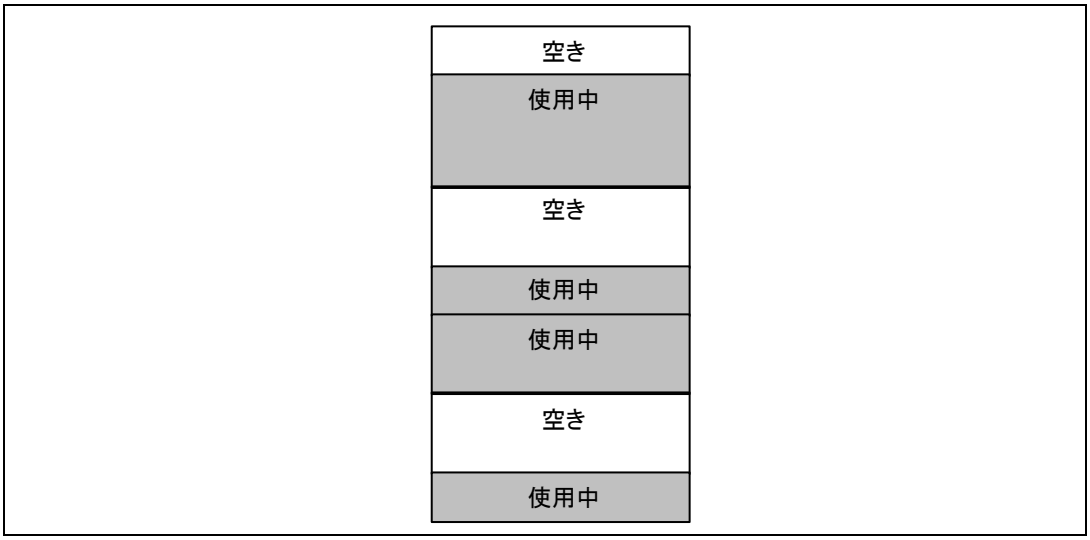

図4.15 空き領域の断片化

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、「8.4.12 **可変長メモリプール定義**(variable_memorypool[])」を参照してください。

4.13 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 1ms です。

4.13.1 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。

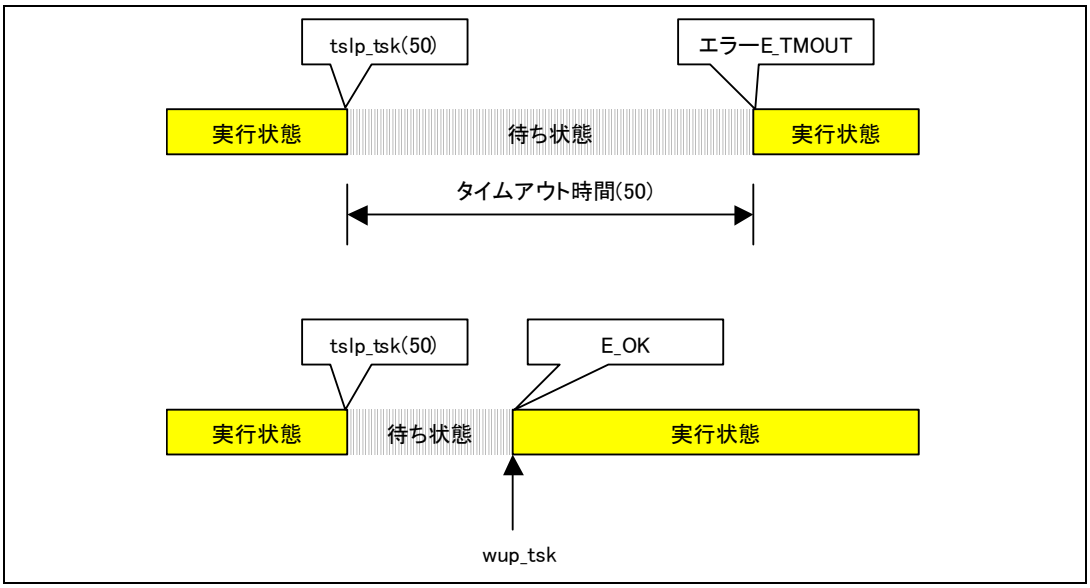

図4.16 タイムアウト

4.13.2 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。

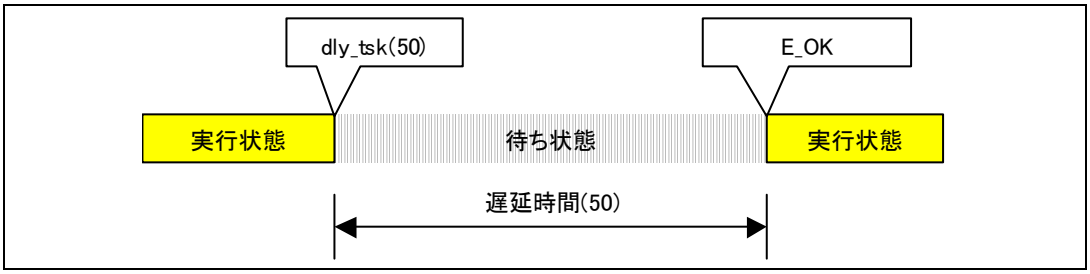

図4.17 タスクの遅延

4.13.3 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、cfg ファイルに cyclic_hand[]を記述します。cyclic_hand[]では、以下の情報を指定します。

- ID 名称
- 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 4.18および図 4.19に、周期ハンドラの動作例を示します。

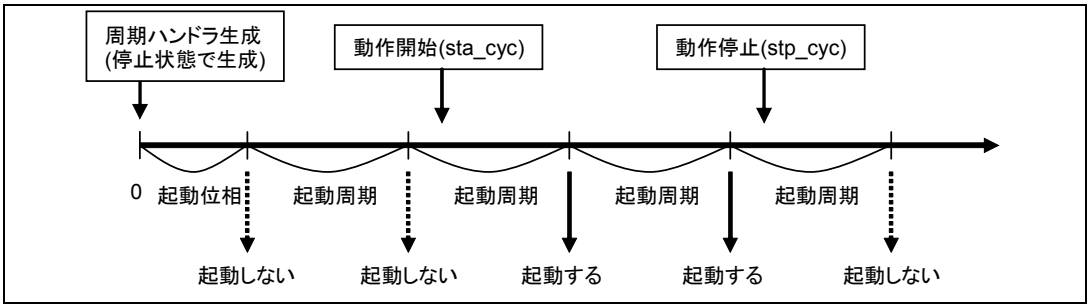

図4.18 起動位相を保存する場合の動作

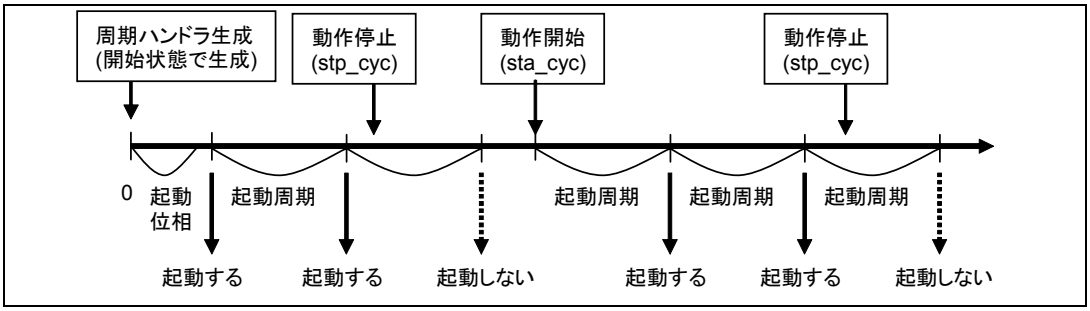

図4.19 起動位相を保存しない場合の動作

周期ハンドラを操作する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。

(2) 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。

(3) 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。

4.13.4 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 4.20にアラームハンドラの動作例を示します。

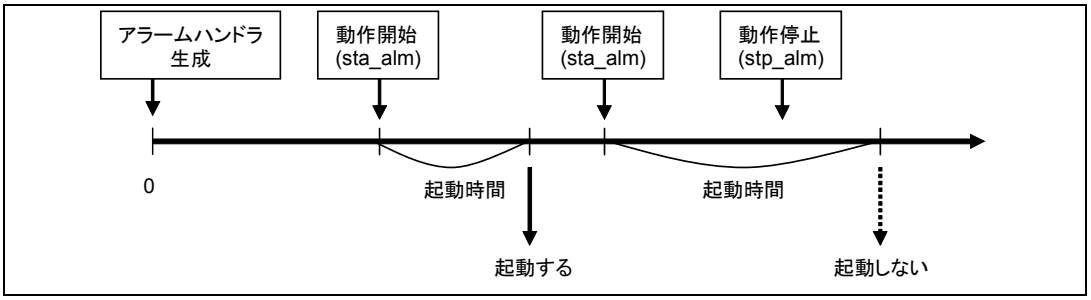

図4.20 アラームハンドラの動作例

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、cfg ファイルに alarm_hand[]を記述します。alarm_hand[]では、以下の情報を指定します。

- ID 名称
- アラームハンドラの開始アドレス
- 拡張情報

アラームハンドラを操作する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。

(2) 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。

(3) 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。

4.13.5 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 4.21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。

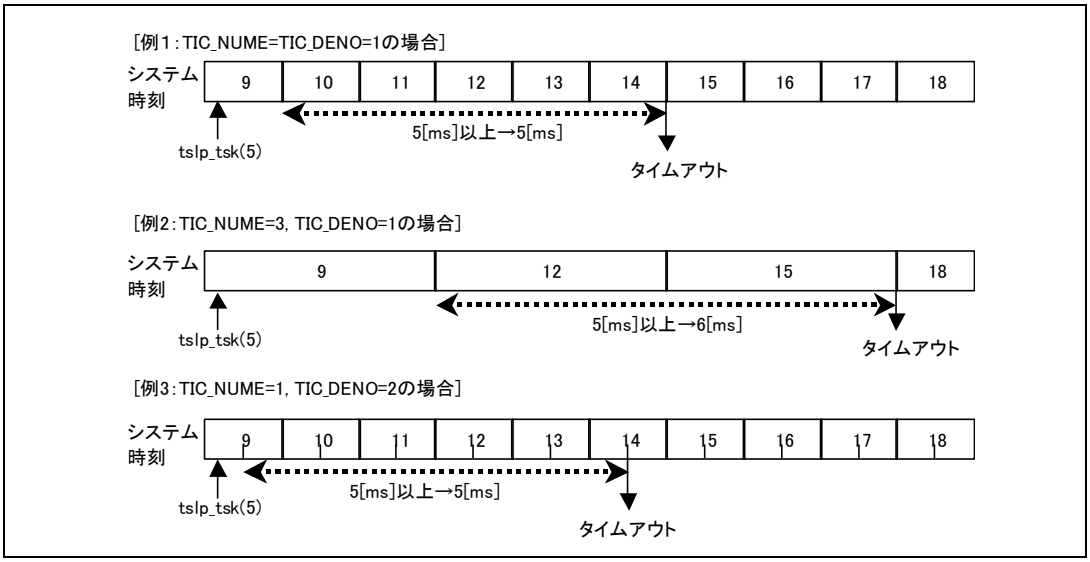

図4.21 時間の精度(tslp_tsk)

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。

4.13.6 注意事項

タイマ割込み発生時には、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、タイマ割込みレベル(clock.IPL)以下の割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。

4.14 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。

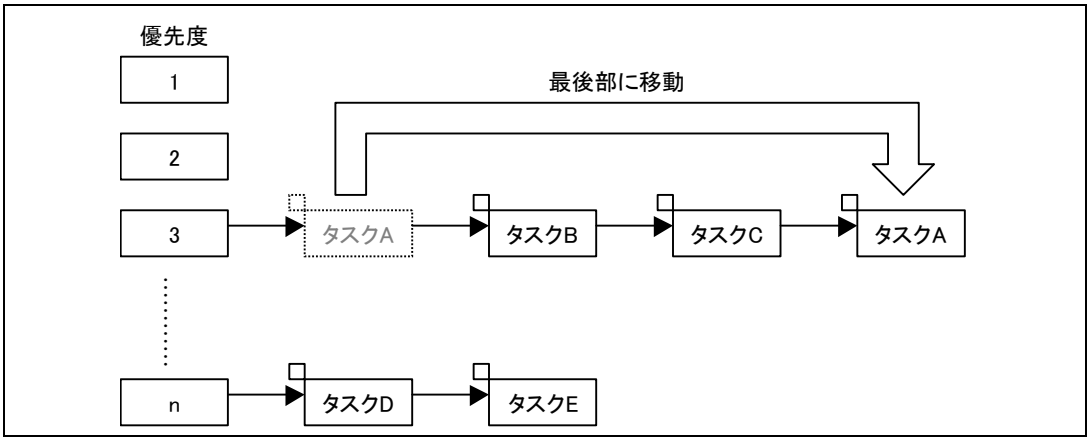

図4.22 rot_rdq によるレディキューの操作

(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、カーネルを初期化します。

(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。

4.15 割込み管理機能

割込みが発生すると、割込みハンドラが起動されます。割込みハンドラは、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、「3.7 割込み」も参照してください。

(1) 割込みマスクを変更する(chg_ims, ichg_ims)

割込みマスク(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) 割込みマスクを参照する(get_ims, iget_ims)

現在の割込みマスク(PSW レジスタの IPL ビット)を参照します。

(3) カーネル管理割込みハンドラから復帰する(ret_int)

カーネル管理割込みハンドラから復帰します。

ret_int サービスコールは、割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。

4.16 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(「5.23.2 定義内容」参照)から取得することもできます。

4.17 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。

4.18 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、割込みが発生するのを待ちます。

CPU の省電力モードを利用するには、通常は最低優先度のタスクで省電力モードに移行するようにします。

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、RI600/4 が提供する kernel.h と、cfg600 が出力する kernel_id.h をインクルードしてください。

5.2 基本データ型

基本データ型の詳細を、表 5.1に示します。

表5.1 基本データ型

No.	データ型	定義内容	No.	データ型	定義内容
1	B	符号付き 8 ビット整数	18	ER	符号付き 32 ビット整数
2	H	符号付き 16 ビット整数	19	ID	符号付き 16 ビット整数
3	W	符号付き 32 ビット整数	20	ATR	符号無し 16 ビット整数
4	D	符号付き 64 ビット整数	21	STAT	符号無し 16 ビット整数
5	UB	符号無し 8 ビット整数	22	MODE	符号無し 16 ビット整数
6	UH	符号無し 16 ビット整数	23	PRI	符号付き 16 ビット整数
7	UW	符号無し 32 ビット整数	24	SIZE	符号無し 32 ビット整数
8	UD	符号無し 64 ビット整数	25	TMO	符号付き 32 ビット整数
9	VB	符号付き 8 ビット整数 *	26	RELTIM	符号無し 32 ビット整数
10	VH	符号付き 16 ビット整数 *	27	SYSTIM	以下のメンバから構成される構造体
11	VW	符号付き 32 ビット整数 *			上位：符号無し 16 ビット整数
12	VD	符号付き 64 ビット整数 *			下位：符号無し 32 ビット整数
13	VP	void 型変数へのポインタ	28	VP_INT	符号付き 32 ビット整数 *
14	FP	void 型関数へのポインタ	29	ER_UINT	符号付き 32 ビット整数
15	INT	符号付き 32 ビット整数	30	FLGPTN	符号無し 32 ビット整数
16	UINT	符号無し 32 ビット整数	31	IMASK	符号無し 16 ビット整数
17	BOOL	符号付き 32 ビット整数			

【注】* これらのデータタイプの変数の値を参照する場合や代入する場合には、明示的に型変換(キャスト)を行う必要があります。

5.3 サービスコールのリターン値とエラーコード

5.3.1 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。

5.3.2 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。

5.4 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。

5.4.1 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。

5.4.2 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)
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

5.4.3 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。

5.4.4 カーネル管理外の割込みハンドラなど

カーネル管理外割込みハンドラなど、PSW.IPL>カーネル割込みマスクレベル(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一時的に PSW.IPL がカーネル割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。

5.5 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

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

5.6 タスク管理機能

表 5.2に、タスク管理機能の仕様を示します。
タスクは、cfg ファイルの task[]定義によって生成します。

表5.2 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 *3 TA_ASM：アセンブリ言語記述 *3 TA_ACT：起動属性
6	タスクスタック	個別のセクションに割り当て可能

【注】 *1 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義したタスク数を意味します。
*2 cfg600 が kernel_id.h に出力するマクロで、system.priority に指定した値です。
*3 現在の実装では、これらによる動作の差異はありません。

表5.3 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	act_tsk	[S]	タスクの起動	○		○	○	○	
2	iact_tsk	[S]			○	○	○	○	
3	can_act	[S]	タスク起動要求のキャンセル	○		○	○	○	
4	ican_act				○	○	○	○	
5	sta_tsk	[B]	タスクの起動(起動コード指定)	○		○	○	○	
6	ista_tsk				○	○	○	○	
7	ext_tsk	[S][B]	自タスクの終了	○		○	○	○	○
8	ter_tsk	[S][B]	タスクの強制終了	○		○	○	○	
9	chg_pri	[S][B]	タスク優先度の変更	○		○	○	○	
10	ichg_pri				○	○	○	○	
11	get_pri	[S]	タスク優先度の参照	○		○	○	○	
12	iget_pri				○	○	○	○	
13	ref_tsk		タスクの状態参照	○		○	○	○	
14	iref_tsk				○	○	○	○	
15	ref_tst		タスクの状態参照(簡易版)	○		○	○	○	
16	iref_tst				○	○	○	○	

【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 タスクの起動(act_tsk, iact_tsk)

■C 言語 API

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

■引数

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=TSK_SELF(0)
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)
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

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。

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。

act_tsk では、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。

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

■C 言語 API

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

■引数

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=TSK_SELF(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

■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。

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

■C 言語 API

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

■引数

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 (1)tskid<=0, VTMAX_TSK < tskid
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_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

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。この時、タスク起動時に行うべき処理(表 5.4参照)を行います。

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。

5.6.4 自タスクの終了(ext_tsk)

■C 言語 API

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

■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のとおりです。

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。

5.6.5 タスクの強制終了(ter_tsk)

■C 言語 API

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

■引数

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
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_ILUSE	サービスコール不正使用(tskid のタスクが自タスク)
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

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。

5.6.6 タスク優先度の変更(chg_pri, ichg_pri)

■C 言語 API

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

■引数

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
	(1)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=TSK_SELF(0)
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_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

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(task[].priority)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。

5.6.7 タスク優先度の参照(get_pri, iget_pri)

■C 言語 API

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

■引数

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_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=TSK_SELF(0)
E_OBJ	オブジェクト状態エラー(tskid のタスクが休止状態)
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

■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。

5.6.8 タスクの状態参照(ref_tsk, iref_tsk)

■C 言語 API

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

■引数

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

■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 {
           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;
```

■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=TSK_SELF(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

■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。

◆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

◆tskpri

タスクの現在優先度です。タスクが休止状態の場合は、タスクの初期優先度を返します。

◆tskbpri

タスクのベース優先度です。タスクが休止状態の場合は、タスクの初期優先度を返します。

◆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

◆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。

◆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。

◆actcnt

現在の起動要求キューイング数を返します。

◆wupcnt *

現在の起床要求キューイング数を返します。

◆suscnt *

現在の強制待ち要求ネスト数を返します。

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

■C 言語 API

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

■引数

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

■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 {
           STAT    tskstat;     タスク状態
           STAT    tskwait;     待ち要因
} T_RTST;
```

■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=TSK_SELF(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

■機能説明

tskid で示されたタスクについて、タスク状態と待ち要因を参照し、pk_rtst が指す領域に返します。
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。
pk_rtst 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。

◆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

◆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

5.7 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	○		○		○	
2	tslp_tsk	[S]	同上(タイムアウト有)	○		○		○	
3	wup_tsk	[S][B]	タスクの起床	○		○	○	○	
4	iwup_tsk	[S][B]			○	○	○	○	
5	can_wup	[S][B]	タスク起床要求のキャンセル	○		○	○	○	
6	ican_wup				○	○	○	○	
7	rel_wai	[S][B]	待ち状態の強制解除	○		○	○	○	
8	irel_wai	[S][B]			○	○	○	○	
9	sus_tsk	[S][B]	強制待ち状態への移行	○		○	○	○	
10	isus_tsk				○	○	○	○	
11	rsm_tsk	[S][B]	強制待ち状態からの再開	○		○	○	○	
12	irsm_tsk				○	○	○	○	
13	frsm_tsk	[S]	強制待ち状態からの強制再開	○		○	○	○	
14	ifrsm_tsk				○	○	○	○	
15	dly_tsk	[S][B]	自タスクの遅延	○		○		○	

【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 起床待ち(slp_tsk, tslp_tsk)

■C 言語 API

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

■引数

《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 (1)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_TMOUT	タイムアウト
E_RLWAI	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)
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

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。

5.7.2 タスクの起床(wup_tsk, iwup_tsk)

■C 言語 API

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

■引数

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=TSK_SELF(0)
E_OBJ	オブジェクト状態エラー(tskid のタスクが休止状態)
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)
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

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。

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。

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。

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

■C 言語 API

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

■引数

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=TSK_SELF(0)
E_OBJ	オブジェクト状態エラー(tskid のタスクが休止状態)
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

■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。

5.7.4 待ち状態の強制解除(rel_wai, irel_wai)

■C 言語 API

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

■引数

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
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_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

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。

5.7.5 強制待ち状態への移行(sus_tsk, isus_tsk)

■C 言語 API

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

■引数

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=TSK_SELF(0)
E_OBJ	オブジェクト状態エラー(tskid のタスクが休止状態)
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)
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

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。

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

■引数

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
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_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

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。

5.7.7 タスク遅延(dly_tsk)

■C 言語 API

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

■引数

dlytim	遅延時間

■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 (1)(0x7FFFFFFF - TIC_NUME)/TIC_DENO < dlytim
E_RLWAI	待ち状態強制解除(待ちの間に rel_wai サービスコールが呼び出された)
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

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。

5.8 同期・通信(セマフォ)機能

表 5.8に、セマフォ機能の仕様を示します。
セマフォは、cfg ファイルの semaphore[]定義によって生成します。

表5.8 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優先度順

【注】 *1 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義したセマフォ数を意味します。
*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5.9 セマフォ機能サービスコールー覧

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	sig_sem	[S][B]	セマフォ資源の返却	○		○	○	○	
2	isig_sem	[S][B]			○	○	○	○	
3	wai_sem	[S][B]	セマフォ資源の獲得	○		○		○	
4	pol_sem	[S][B]	同上(ポーリング)	○		○	○	○	
5	ipol_sem				○	○	○	○	
6	twai_sem		同上(タイムアウト有)	○		○		○	
7	ref_sem	[S]	セマフォの状態参照	○		○	○	○	
8	iref_sem				○	○	○	○	

【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 セマフォ資源の返却(sig_sem, isig_sem)

■C 言語 API

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

■引数

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 (1)semid<=0, VTMAX_SEM < semid
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)
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

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、そのセマフォのカウント値を 1 増やします。

なお、セマフォのカウント値の最大値は、semaphore[].max_count に定義します。

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

■C 言語 API

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

■引数

semid	セマフォ ID
《twai_sem》	
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)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 (1)semid<=0, VTMAX_SEM < semid
E_RLWAI	待ち状態強制解除(待ちの間に rel_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_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

■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。

5.8.3 セマフォの状態参照(ref_sem, iref_sem)

■C 言語 API

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

■引数

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

■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 {
           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;
```

■エラーコード

E_ID	不正 ID 番号 (1)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

■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 5.10に、イベントフラグ機能の仕様を示します。
イベントフラグは、cfg ファイルの flag[]定義によって生成します。

表5.10 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 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義したイベントフラグ数を意味します。
*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5.11 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	set_flg	[S][B]	イベントフラグのセット	○		○	○	○	
2	iset_flg	[S][B]			○	○	○	○	
3	clr_flg	[S][B]	イベントフラグのクリア	○		○	○	○	
4	iclr_flg				○	○	○	○	
5	wai_flg	[S][B]	イベントフラグ待ち	○		○		○	
6	pol_flg	[S][B]	同上(ポーリング)	○		○	○	○	
7	ipol_flg	[S]			○	○	○	○	
8	twai_flg	[S]	同上(タイムアウト有)	○		○		○	
9	ref_flg		イベントフラグの状態参照	○		○	○	○	
10	iref_flg				○	○	○	○	

【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 イベントフラグのセット(set_flg, iset_flg)

■C 言語 API

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

■引数

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 (1)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)

■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。

イベントフラグ値の更新の結果、そのイベントフラグで待っているタスクの待ち解除条件を満たせば、そのタスクの待ち状態を解除します。なお、待ち解除条件は待ち行列の順に評価されます。この時、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のすべてのビットをクリアし、サービスコールの処理を終了します。

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 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。

5.9.2 イベントフラグのクリア(clr_flg, iclr_flg)

■C 言語 API

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

■引数

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 (1)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

■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。

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

■C 言語 API

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

■引数

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
《twai_flg》	
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)waiptn=0 (2)wfmode が不正 (3)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 (1)flgid<=0, VTMAX_FLG < flgid
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_RLWAI	待ち状態強制解除(待ちの間に rel_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_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

■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。

5.9.4 イベントフラグの状態参照(ref_flg, iref_flg)

■C 言語 API

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

■引数

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 {
           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_ID	不正 ID 番号 (1)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

■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 5.12に、データキュー機能の仕様を示します。
データキューは、cfg ファイルの dataqueue[]定義によって生成します。

表5.12 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優先度順

【注】 *1 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義したデータキュー数を意味します。

表5.13 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	snd_dtq	[S]	データキューへの送信	○		○		○	
2	psnd_dtq	[S]	同上(ポーリング)	○		○	○	○	
3	ipsnd_dtq	[S]			○	○	○	○	
4	tsnd_dtq	[S]	同上(タイムアウト有)	○		○		○	
5	fsnd_dtq	[S]	データキューへの強制送信	○		○	○	○	
6	ifsnd_dtq	[S]			○	○	○	○	
7	rcv_dtq	[S]	データキューからの受信	○		○		○	
8	prcv_dtq	[S]	同上(ポーリング)	○		○	○	○	
9	iprcv_dtq				○	○	○	○	
10	trcv_dtq	[S]	同上(タイムアウト有)	○		○		○	
11	ref_dtq		データキューの状態参照	○		○	○	○	
12	iref_dtq				○	○	○	○	

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

■C 言語 API

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

■引数

dtqid	データキューID
data	データキューへ送信するデータ
《tsnd_dtq》	
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)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 (1)dtqid<=0, VTMAX_DTQ < dtqid
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_RLWAI	待ち状態強制解除(待ちの間に rel_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_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)
EV_RST	オブジェクトリセット(vrst_dtq)による待ち解除

■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 E_ILUSE エラーとなります。

(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

- 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。

- psnd_dtq, ipsnd_dtq の場合

  直ちにエラーE_TMOUTで終了します。

- 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、データキューの先頭のデータ(最も古いデータ)を削除した後、dataをデータキューの末尾に格納します。

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。

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

■C 言語 API

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

■引数

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
《trcv_dtq》	
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)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号
	(1)dtqid<=0, VTMAX_DTQ < dtqid
E_RLWAI	待ち状態強制解除(待ちの間に rel_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_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

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の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。

5.10.3 データキューの状態参照(ref_dtq, iref_dtq)

■C 言語 API

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

■引数

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

■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 {
           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;
```

■エラーコード

E_ID	不正 ID 番号 (1)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

■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 5.14に、メールボックス機能の仕様を示します。
メールボックスは、cfg ファイルの mailbox[]定義によって生成します。

表5.14 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 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義したメールボックス数を意味します。
*2 cfg600 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5.15 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	snd_mbx	[S][B]	メールボックスへの送信	○		○	○	○	
2	isnd_mbx				○	○	○	○	
3	rcv_mbx	[S][B]	メールボックスからの受信	○		○		○	
4	prcv_mbx	[S][B]	同上(ポーリング)	○		○	○	○	
5	iprcv_mbx				○	○	○	○	
6	trcv_mbx	[S]	同上(タイムアウト有)	○		○		○	
7	ref_mbx		メールボックスの状態参照	○		○	○	○	
8	iref_mbx				○	○	○	○	

【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 メールボックスへの送信(snd_mbx, isnd_mbx)

■C 言語 API

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

■引数

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

■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

《メールボックスのメッセージヘッダ》

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

■エラーコード

E_PAR	パラメータエラー (1)msgpri<=0, mailbox.max_pri < msgpri
E_ID	不正 ID 番号 (1)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)

■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 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。

```
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)

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

■引数

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

《trcv_mbx》

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

■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 {
           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   msgque;      メッセージヘッダ
           PRI     msgpri;      メッセージ優先度
} T_MSG_PRI;
```

■エラーコード

E_PAR	パラメータエラー (1)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 (1)mbxid<=0, VTMAX_MBX < mbxid
E_RLWAI	待ち状態強制解除(待ちの間に rel_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_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

■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。

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

■C 言語 API

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

■引数

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

■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

```
(1)T_RMBX
typedef    struct {
           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 {
           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   msgque;      メッセージヘッダ
           PRI     msgpri;      メッセージ優先度
} T_MSG_PRI;
```

■エラーコード

E_ID	不正 ID 番号 (1)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

■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 5.16に、ミューテックス機能の仕様を示します。
ミューテックスは、cfg ファイルの mutex[]定義によって生成します。

表5.16 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 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義したミューテックス数を意味します。
*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5.17 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	loc_mtx		ミューテックスのロック	○		○		○	
2	ploc_mtx		同上(ポーリング)	○		○	○	○	
3	tloc_mtx		同上(タイムアウト有)	○		○		○	
4	unl_mtx		ミューテックスのロック解除	○		○	○	○	
5	ref_mtx		ミューテックスの状態参照	○		○	○	○	

【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 ミューテックスのロック(loc_mtx, ploc_mtx, tloc_mtx)

■C 言語 API

```
ER ercd = loc_mtx(ID mtxid);
ER ercd = ploc_mtx(ID mtxid);
ER ercd = tloc_mtx(ID mtxid, TMO tmout);
```

■引数

mtxid　　ミューテックス ID

《tloc_mtx》

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)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号
	(1)mtxid<=0, VTMAX_MTX < mtxid
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_RLWAI	待ち状態強制解除(待ちの間に rel_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_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

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。

5.12.2 ミューテックスのロック解除(unl_mtx)

■C 言語 API

```
ER ercd = unl_mtx(ID mtxid);
```

■引数

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 (1)mtxid<=0, VTMAX_MTX < mtxid
E_ILUSE	サービスコール不正使用(呼び出しタスクは対象ミューテックスをロックしていない)
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

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の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。

5.12.3 ミューテックスの状態参照(ref_mtx)

■C 言語 API

```
ER ercd = ref_mtx(ID mtxid, T_RMTX *pk_rmtx);
```

■引数

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

■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 {
           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;
```

■エラーコード

E_ID	不正 ID 番号 (1)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) 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_mtx の呼び出し

■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 5.18に、メッセージバッファ機能の仕様を示します。
メッセージバッファは、cfg ファイルの message_buffer[]定義によって生成します。

表5.18 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 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義したメッセージバッファ数を意味します。

表5.19 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	snd_mbf	[R]	メッセージバッファへの送信	○		○		○	
2	psnd_mbf	[R]	同上(ポーリング)	○		○	○	○	
3	ipsnd_mbf				○	○	○	○	
4	tsnd_mbf	[R]	同上(タイムアウト有)	○		○		○	
5	rcv_mbf	[R]	メッセージバッファからの受信	○		○		○	
6	prcv_mbf	[R]	同上(ポーリング)	○		○	○	○	
7	trcv_mbf	[R]	同上(タイムアウト有)	○		○		○	
8	ref_mbf	[R]	メッセージバッファの状態参照	○		○	○	○	
9	iref_mbf				○	○	○	○	

【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への送信(snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf)

■C 言語 API

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

■引数

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)msgsz=0, msgsz＞最大メッセージサイズ *
	(2)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号
	(1)mbfid<=0, VTMAX_MBF < mbfid
E_RLWAI	待ち状態強制解除(待ちの間に rel_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_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)
EV_RST	オブジェクトリセット(vrst_mbf)による待ち解除

【注】最大メッセージサイズは、cfg ファイルの message_buffer[].max_msgsz に指定します。

■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。

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

■引数

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
《trcv_mbf》	
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)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号
	(1)mbfid<=0, VTMAX_MBF < mbfid
E_RLWAI	待ち状態強制解除(待ちの間に rel_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_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

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、送信待ち行列の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。

5.13.3 メッセージバッファの状態参照(ref_mbf, iref_mbf)

■C 言語 API

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

■引数

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

■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 {
           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;
```

■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) 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

■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 5.20に、固定長メモリプール機能の仕様を示します。
固定長メモリプールは、cfg ファイルの memorypool[]定義によって生成します。

表5.20 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優先度順

【注】 *1 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義した固定長メモリプール数を意味します。
*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5.21 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	get_mpf	[S][B]	固定長メモリブロックの獲得	○		○		○	
2	pget_mpf	[S][B]	同上(ポーリング)	○		○	○	○	
3	ipget_mpf				○	○	○	○	
4	tget_mpf	[S]	同上(タイムアウト有)	○		○		○	
5	rel_mpf	[S][B]	固定長メモリブロックの返却	○		○	○	○	
6	irel_mpf				○	○	○	○	
7	ref_mpf		固定長メモリプールの状態参照	○		○	○	○	
8	iref_mpf				○	○	○	○	

【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 固定長メモリブロックの獲得(get_mpf, pget_mpf, ipget_mpf, tget_mpf)

■C 言語 API

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

■引数

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
《tget_mpf》	
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)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 (1)mpfid<=0, VTMAX_MPF < mpfid
E_RLWAI	待ち状態強制解除(待ちの間に rel_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_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
EV_RST	オブジェクトリセット(vrst_mpf)による待ち解除

■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、以下を守ってください。

(1) cfgファイルのmemorypool[].siz_block(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) cfgファイルのmemorypool[].section(メモリプールのセクション)を、リンク時に目的の境界調整数のアドレスに配置する。

5.14.2 固定長メモリブロックの返却(rel_mpf, irel_mpf)

■C 言語 API

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

■引数

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 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 (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)

■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。

5.14.3 固定長メモリプールの状態参照(ref_mpf, iref_mpf)

■C 言語 API

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

■引数

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

■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 {
           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;
```

■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) 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

■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 5.22に、可変長メモリプール機能の仕様を示します。
可変長メモリプールは、cfg ファイルの variable_mempool[]定義によって生成します。

表5.22 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 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義した可変長メモリプール数を意味します。
*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5.23 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	get_mpl		固定長メモリブロックの獲得	○		○		○	
2	pget_mpl		同上(ポーリング)	○		○	○	○	
3	ipget_mpl				○	○	○	○	
4	tget_mpl		同上(タイムアウト有)	○		○		○	
5	rel_mpl		可変長メモリブロックの返却	○		○	○	○	
6	ref_mpl		可変長メモリプールの状態参照	○		○	○	○	
7	iref_mpl				○	○	○	○	

【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 可変長メモリブロックの獲得(get_mpl, tget_mpl, pget_mpl, ipget_mpl)

■C 言語 API

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

■引数

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
《tget_mpl》	
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)blksz が 0 (2)blksz が獲得可能な最大サイズを超えている (3)tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	不正 ID 番号 (1) mplid<=0, VTMAX_MPL < mplid
E_RLWAI	待ち状態強制解除(待ちの間に rel_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_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
EV_RST	オブジェクトリセット(vrst_mpl)による待ち解除

■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 4 です。

5.15.2 可変長メモリブロックの返却(rel_mpl)

■C 言語 API

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

■引数

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 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 (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)

■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。

5.15.3 可変長メモリプールの状態参照(ref_mpl, iref_mpl)

■C 言語 API

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

■引数

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

■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 {
           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;
```

■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) 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

■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 5.24に、システム時刻管理の仕様を示します。

表5.24 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、cfg600 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5.25 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	○		○	○	○	
2	iset_tim				○	○	○	○	
3	get_tim	[S]	システム時刻の参照	○		○	○	○	
4	iget_tim				○	○	○	○	
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 ercd = set_tim(SYSTIM *p_systim);
ER ercd = iset_tim(SYSTIM *p_systim);
```

■引数

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

■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 {
           UH      utime;       システム時刻（上位）
           UW      ltime;       システム時刻（下位）
} SYSTIM;
```

■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 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

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

■引数

p_systim　システム時刻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 {
           UH      utime;       システム時刻（上位）
           UW      ltime;       システム時刻（下位）
} SYSTIM;
```

■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 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

■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 5.26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5.26 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 *3 TA_ASM：アセンブリ言語記述 *3 TA_STA：周期ハンドラの動作開始 TA_PHS：起動位相の保存

【注】 *1 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義した周期ハンドラ数を意味します。
*2 TIC_NUME および TIC_DENO は、cfg600 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。
*3 現在の実装では、これらによる動作の差異はありません。

表5.27 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	sta_cyc	[S][B]	周期ハンドラの動作開始	○		○	○	○	
2	ista_cyc				○	○	○	○	
3	stp_cyc	[S][B]	周期ハンドラの動作停止	○		○	○	○	
4	istp_cyc				○	○	○	○	
5	ref_cyc		周期ハンドラの状態参照	○		○	○	○	
6	iref_cyc				○	○	○	○	

【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 周期ハンドラの動作開始(sta_cyc, ista_cyc)

■C 言語 API

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

■引数

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 (1)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

■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 が指定されておらず、既に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。

5.17.2 周期ハンドラの動作停止(stp_cyc, istp_cyc)

■C 言語 API

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

■引数

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 (1)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

■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。

5.17.3 周期ハンドラの状態参照(ref_cyc, iref_cyc)

■C 言語 API

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

■引数

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

■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 {
           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;
```

■エラーコード

E_ID	不正 ID 番号 (1)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

■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 5.28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5.28 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 *3 TA_ASM：アセンブリ言語記述 *3

【注】 *1 cfg600 が kernel_id.h に出力するマクロで、cfg ファイルに定義した周期ハンドラ数を意味します。
*2 TIC_NUME および TIC_DENO は、cfg600 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。
*3 現在の実装では、これらによる動作の差異はありません。

表5.29 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	sta_alm		アラームハンドラの動作開始	○		○	○	○	
2	ista_alm				○	○	○	○	
3	stp_alm		アラームハンドラの動作停止	○		○	○	○	
4	istp_alm				○	○	○	○	
5	ref_alm		アラームハンドラの状態参照	○		○	○	○	
6	iref_alm				○	○	○	○	

【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動作開始(sta_alm, ista_alm)

■C 言語 API

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

■引数

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 (1)(0x7FFFFFFF - TIC_NUME)/TIC_DENO < almtim
E_ID	不正 ID 番号 (1)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

■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。

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

■C 言語 API

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

■引数

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 (1)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

■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。

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

■C 言語 API

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

■引数

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

■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 {
           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;
```

■エラーコード

E_ID	不正 ID 番号 (1)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

■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5.30 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	○		○	○	○	
2	irot_rdq	[S][B]			○	○	○	○	
3	get_tid	[S][B]	実行状態のタスク ID の参照	○		○	○	○	
4	iget_tid	[S]			○	○	○	○	
5	loc_cpu	[S][B]	CPU ロック状態への移行	○		○	○	○	○
6	iloc_cpu	[S]			○	○	○	○	○
7	unl_cpu	[S][B]	CPU ロック状態の解除	○		○	○	○	○
8	iunl_cpu	[S]			○	○	○	○	○
9	dis_dsp	[S][B]	ディスパッチの禁止	○		○	○	○	
10	ena_dsp	[S][B]	ディスパッチの許可	○		○	○	○	
11	sns_ctx	[S]	コンテキストの参照	○	○	○	○	○	○
12	sns_loc	[S]	CPU ロック状態の参照	○	○	○	○	○	○
13	sns_dsp	[S]	ディスパッチ禁止状態の参照	○	○	○	○	○	○
14	sns_dpn	[S]	ディスパッチ保留状態の参照	○	○	○	○	○	○
15	vsta_knl	[V]	カーネルの起動	○	○	○	○	○	○
16	ivsta_knl	[V]		○	○	○	○	○	○
17	vsys_dwn	[V]	システムダウン	○	○	○	○	○	○
18	ivsys_dwn	[V]		○	○	○	○	○	○

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

■引数

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、tskid=TSK_SELF(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)

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

■引数

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_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

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

■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(カーネル割込みマスクレベル)以下のレベルの割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
- 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)

CPU ロック状態と CPU ロック解除状態の間の遷移は、loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, ext_tsk サービスコールによってのみ発生します。カーネル管理割込みハンドラ、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 ercd = unl_cpu(void);
ER ercd = iunl_cpu(void);
```

■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

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、dis_dsp サービスコールによるディスパッチ禁止状態は解除されません。

5.19.5 ディスパッチの禁止(dis_dsp)

■C 言語 API

```
ER ercd = dis_dsp(void);
```

■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) 割込みは受け付けられます。
(3) 待ち状態になるサービスコールを呼び出せません。

ディスパッチ禁止状態は、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) chg_imsサービスコールで割込みマスク(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とディスパッチ許可状態の間の遷移は、dis_dsp, ena_dsp, ext_tsk サービスコールによってのみ発生します。

本サービスコールによって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 ercd = ena_dsp(void);
```

■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

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 state = sns_ctx(void);
```

■リターン値

TRUE 非タスクコンテキスト

FALSE タスクコンテキスト

ただし、コンテキストエラー発生時はエラーE_CTX を返します。

■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を調べます。
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 state = sns_loc(void);
```

■リターン値

TRUE　　CPU ロック状態

FALSE　　CPU ロック解除状態

ただし、コンテキストエラー発生時はエラーE_CTX を返します。

■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 ロック状態かどうかを調べます。
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 state = sns_dsp(void);
```

■リターン値

TRUE　　ディスパッチ禁止状態

FALSE　　ディスパッチ許可状態

ただし、コンテキストエラー発生時はエラーE_CTX を返します。

■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かどうかを調べます。
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 state = sns_dpn(void);
```

■リターン値

TRUE　　ディスパッチ保留状態である

FALSE　　ディスパッチ保留状態でない

ただし、コンテキストエラー発生時はエラーE_CTX を返します。

■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かどうかを調べます。

以下のいずれかの条件を満たすときは、ディスパッチ保留状態です。

(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);
```

■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) INTBレジスタを、cfg600によって生成された可変ベクタテーブル先頭アドレスに初期化
(2) カーネル内部テーブルを初期化
(3) cfgファイルで指定された各種オブジェクトの生成
(4) マルチタスク環境へ移行

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、割込みマスクがカーネル割込みマスクレベル(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);
```

■引数

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、「6.6 **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、割込みマスクがカーネル割込みマスクレベル(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 割込み管理機能

表5.31 割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		割込みマスクの変更	○		○	○	○	
2	ichg_ims				○	○	○	○	
3	get_ims		割込みマスクの参照	○		○	○	○	
4	iget_ims				○	○	○	○	
5	ret_int	[S][B]	カーネル管理割込みハンドラからの復帰		○	○	○	○	

【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 割込みマスクの変更(chg_ims, ichg_ims)

■C 言語 API

```
ER ercd = chg_ims(IMASK imask);
ER ercd = ichg_ims(IMASK imask);
```

■引数

imask	割込みマスク値

■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 (1) imask に 0～15 以外の値を指定
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 の割込みマスク(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、割込みマスクがカーネル割込みマスクレベル(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、起動時よりも割込みマスク値を下げてはなりません。

2. 割込みマスク(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. 割込みマスクがカーネル割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 割込みマスクの参照(get_ims, iget_ims)

■C 言語 API

```
ER ercd = get_ims(IMASK *p_imask);
ER ercd = iget_ims(IMASK *p_imask);
```

■引数

p_imask	割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）

■機能説明

現在の割込みマスクレベル(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、割込みマスクがカーネル割込みマスクレベル(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)

■機能説明

カーネル管理割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込みマスク(PSW.IPL)は、カーネル割込みマスクレベル(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、カーネル管理外割込みをカーネル管理割込みハンドラとして定義した場合

(2) カーネル管理割込みハンドラ内で、PSW.IPLをカーネル割込みマスクレベル(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

割込みには、カーネル管理割込みとカーネル管理外割込みの 2 種類があります。これについては、「3.7.1 割込みの種類」を参照してください。

カーネル管理割込みハンドラは、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、カーネル管理外割込みハンドラの場合は、可変ベクタ割込みは 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5.32 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	[S]	バージョン情報の参照	○		○	○	○	
2	iref_ver				○	○	○	○	

【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 ercd = ref_ver(T_RVER *pk_rver);
ER ercd = iref_ver(T_RVER *pk_rver);
```

■引数

pk_rver	バージョン情報を返すパケットへのポインタ

■リターン値

正常終了（E_OK）

■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 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

■パケットの構造

```
typedef    struct {
           UH      maker;       メーカー
           UH      prid;        形式番号
           UH      spver;       仕様書バージョン
           UH      prver;       製品バージョン
           UH      prno[4];     製品管理情報
} T_RVER;
```

■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) 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、ルネサステクノロジを意味する 0x0115 です。

(2) 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、0x0015 が返ります。

(3) 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。

- ビット 15～12：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。
- ビット 11～0：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) 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。製品添付のリリースノートを参照してください。例えば、V.1.00 Release 00 の prver は、0x0100 となります。

(5) 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5.33 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	○		○	○	○	
2	vrst_mbx	[V]	メールボックスのリセット	○		○	○	○	
3	vrst_mbf	[V]	メッセージバッファのリセット	○		○	○	○	
4	vrst_mpf	[V]	固定長メモリプールのリセット	○		○	○	○	
5	vrst_mpl	[V]	可変長メモリプールのリセット	○		○	○	○	

【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 ercd = vrst_dtq( ID dtqid );
```

■引数

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 (1)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)

■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 ercd = vrst_mbx( ID mbxid );
```

■引数

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 (1)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)

■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 ercd = vrst_mbf( ID mbfid );
```

■引数

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)

■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 ercd = vrst_mpf( ID mpfid );
```

■引数

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)

■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 ercd = vrst_mpl( ID mplid );
```

■引数

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)

■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 定数とマクロ

5.23.1 ヘッダファイル

定数とマクロは、以下のヘッダファイルで定義されています。図 5.3に、これらのファイルのインクルード関係を示します。

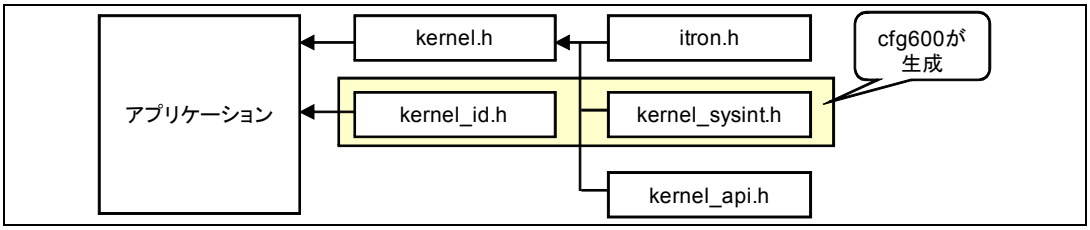

図5.3 ヘッダファイルのインクルード関係

(1) kernel.h

kernel.h は、”インストールディレクトリ¥inc600”ディレクトリにあります。アプリケーションからは、kernel.h をインクルードしてください。

kernel.h には、カーネル仕様が定義されます。

(2) kernel_id.h

kernel_id.h は cfg600 によって生成されます。アプリケーションからは、kernel_id.h をインクルードしてください。

kernel_id.h には、cfg ファイルで指定された ID 名称や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。

(3) kernel_sysint.h

kernel_sysint.h は cfg600 によって生成されます。

kernel_sysint.h には、サービスコールを INT 命令で呼び出すための定義が含まれます。

(4) itron.h

itron.h は、”インストールディレクトリ¥inc600”ディレクトリにあります。

itron.h には、ITRON 仕様共通定義が定義されます。

(5) kernel_api.h

kernel_api.h は、”インストールディレクトリ¥inc600”ディレクトリにあります。

kernel_api.h には、サービスコール関数宣言が記述されています。

5.23.2 定義内容

表5.34 マクロと定数

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
オブジェクト属性	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_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	kernel.h	ポーリング
	TMO_FEVR	-1	kernel.h	永久待ち
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
オブジェクトの状態	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
	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

分類	マクロ	定義内容	定義場所	説明
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	0x0115	kernel.h	カーネルのメーカーコード
	TKERNEL_PRID	0x0015	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
	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 TSK	task[]定義数	kernel_id.h	最大タスク ID
	VTMAX_SEM	semaphore[]定義数	kernel_id.h	最大セマフォ ID
	VTMAX_FLG	flag[]定義数	kernel_id.h	最大イベントフラグ ID
	VTMAX_DTQ	dataqueue[]定義数	kernel_id.h	最大データキューID
	VTMAX_MBX	mailbox[]定義数	kernel_id.h	最大メールボックス ID
	VTMAX_MTX	mutex[]定義数	kernel_id.h	最大ミューテックス ID
	VTMAX_MBF	message_buffer[]定義数	kernel_id.h	最大メッセージバッファ ID
	VTMAX_MPF	memorypool[]定義数	kernel_id.h	最大固定長メモリプール ID
	VTMAX_MPL	variable_memorypool[]定義数	kernel_id.h	最大可変長メモリプール ID
	VTMAX_CYH	cyclic_hand[]定義数	kernel_id.h	最大周期ハンドラ ID
	VTMAX_ALH	alarm_hand[]定義数	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
	VTMAX_AREASIZE	0x20000000	kernel.h	各種領域サイズの最大値
	VTKNL_LVL	system.system_IPL	kernel_id.h	カーネル割込みマスクレベル
	VTIM_LVL	clock.IPL	kernel_id.h	タイマ割込み優先レベル
エラーコード	E_NOSPT	-11	itron.h	未サポート機能
	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_ILUSE	-28	itron.h	サービスコール不正使用
	E_OBJ	-41	itron.h	オブジェクト状態エラー
	E_QOVR	-43	itron.h	キューイングオーバーフロー
	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
	EV_RST	-127	itron.h	オブジェクトリセットによる待ち解除

分類	マクロ	定義内容	定義場所	説明
エラーコード処理マクロ	ER ercd = ERCD(ER mercd, ER sercd)		itron.h	メインエラーコード(mercd)とサブエラーコード(sercd)からエラーコードを生成
	ER mercd = MERCD(ER ercd)		itron.h	エラーコードからメインエラーコードを取り出す
	ER sercd = SERCD(ER ercd)		itron.h	エラーコードからサブエラーコードを取り出す

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、cfg600 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 カーネルへの登録

カーネルへタスクを登録することを、「タスクの生成」と呼びます。

タスクの生成は、cfg ファイルの task[]に、タスク開始関数名などの情報を定義することで行います。詳細は、「8.4.4 タスク定義(task[])」を参照してください。

6.3.2 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"
void task(VP_INT exinf)   /* (1) */
{
  /* 処理 */              /* (2) */
  ext_tsk();              /* (3) */
}
```

図6.1 タスク開始関数のコーディング例

(1) タスクの開始関数のAPIは、ここに記載の通りです。なお、タスク開始関数のプロトタイプ宣言は、cfg600がkernel_id.hに出力します。
exinf(R1レジスタ)には、sta_tsk, ista_tskによって起動された場合はそれらのサービスコールで指定した起動コード(stacd)が、act_tsk, iact_tskまたはcfgファイルのtask[].initial_start=ON指定によって起動された場合は、task[].exinfに指定した値が渡ります。

(2) 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。

(3) 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"
void task(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

6.3.3 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。例えば、コンパイラが提供する組み込み関数のうち、ヘッダ smachine.h で提供される機能は特権命令を使用するので、注意してください。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、全割込みが許可された状態です。

表6.2 タスク起動時の PSW

No.	ビット	初期値	解説
1	IPL	0	全割込み許可
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	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 割込みハンドラ

6.4.1 カーネルへの登録

カーネルへ割込みハンドラを登録することを、「割込みハンドラの定義」と呼びます。

割込みハンドラの定義は、cfg ファイルの interrupt_vector[](可変ベクタ用)または interrupt_fvector[](固定ベクタ用)に、ハンドラ開始関数名などの情報を定義することで行います。詳細は、「8.4.15 可変ベクタ割込み定義(interrupt_vector[])」および「8.4.16 固定ベクタ割込み定義(interrupt_fvector[])」を参照してください。

6.4.2 コーディング

図 6.3に、割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6.3 割込みハンドラ開始関数のコーディング例

(1) 割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、割込みハンドラ開始関数のプロトタイプ宣言は、cfg600がkernel_id.hに出力します。

(2) カーネル管理割込みハンドラでは、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。

6.4.3 起動時の CPU 状態

割込みハンドラは、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、当該割込み優先レベルで割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込み(interrupt_fvector[])は、すべての割込みを禁止した状態となります。

表6.3 割込みハンドラ起動時の PSW

No.	ビット	初期値	解説
1	IPL	当該割込み優先レベル	
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	

6.5 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)

6.5.1 カーネルへの登録

カーネルへ周期ハンドラ、アラームハンドラを登録することを、それぞれ「周期ハンドラの生成」「アラームハンドラの生成」と呼びます。

周期ハンドラの生成は、cfg ファイルの cyclic_hand[]に、ハンドラ開始関数名などの情報を定義することで行います。詳細は、「8.4.13 周期ハンドラ定義(cyclic_hand[])」を参照してください。

アラームハンドラの生成は、cfg ファイルの alarm_hand[]に、ハンドラ開始関数名などの情報を定義することで行います。詳細は、「8.4.14 アラームハンドラ定義(alarm_hand[])」を参照してください。

6.5.2 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 6.4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"
void time_handler(VP_INT exinf)    /* (1) */
{
  /* 処理 */                 /* (2) */
}                            /* (3) */
```

図6.4 タイムイベントハンドラ開始関数のコーディング例

(1) 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(R1レジスタ)には、周期ハンドラの場合はcyclic_hand[].exinf、アラームハンドラの場合はalarm_hand[].exinfに指定した値が渡ります。なお、タイムイベントハンドラ開始関数のプロトタイプ宣言は、cfg600がkernel_id.hに出力します。

(2) 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。

(3) 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込みハンドラから関数コールされます。

6.5.3 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、割込み許可状態については、タイマ割込みレベル(clock.IPL)で割込みをマスクした状態になります。

表6.4 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	タイマ割込みレベル(clock.IPL)	
2	I	1	
3	PM	0	スーパバイザモード
4	U	0	ISP(割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 システムダウンルーチン

6.6.1 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[3]
- ext_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

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。

6.6.2 コーディング

図 6.5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)    /* (1) */
{
  /* 処理 */                /* (2) */
  while(1);                /* (3) */
}
```

図6.5 システムダウン開始関数のコーディング例

(1) 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。
渡される引数の仕様を、表6.5に示します。
(2) 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。
(3) 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。

[3] 「9.4 注意事項」参照

表6.5 システムダウンルーチンに渡される引数

No.	引数	レジスタ	解説
1	type	R1	エラー種別 (1) ret_int でのエラー：-1 (2) ext_tsk でのエラー：-2 (3) 組み込まれていないサービスコールの呼び出し：-3 (4) 未定義の可変ベクタ割込み：-16 (5) 未定義の固定ベクタ割込み：-17 (5) vsys_dwn, ivsys_dwn の呼び出し：正の値を使用してください。
2	inf1	R2	システム異常情報 1 (1) ret_int でのエラー：E_CTX (2) ext_tsk でのエラー：E_CTX (3) 組み込まれていないサービスコールの呼び出し：E_NOSPT (4) 未定義の可変ベクタ・固定ベクタ割込み： (a) cfg600 で-U オプションを指定しない場合：不定 (b) cfg600 で-U オプションを指定した場合：ベクタ番号 (5) vsys_dwn, ivsys_dwn の呼び出し：ユーザ任意
3	inf2	R3	システム異常情報 2 (1) ret_int でのエラーの場合 2；タスクコンテキストからの ret_int の呼び出し 3：カーネル管理外割込みなど、PSW.IPL > system.system_IPL の状態からの ret_int の呼び出し 5：CPU ロック状態からの ret_int の呼び出し (2) ext_tsk でのエラーの場合 1：非タスクコンテキストからの ext_tsk の呼び出し 4：カーネル管理外割込みなど、PSW.IPL > system.system_IPL の状態からの ext_tsk の呼び出し (3) 組み込まれていないサービスコールの呼び出し：不定 (4) 未定義の可変ベクタ・固定ベクタ割込み：CPU の割込み処理によってスタックに退避された PC (5) vsys_dwn, ivsys_dwn の呼び出し：ユーザ任意
4	inf3	R4	システム異常情報 3 (1) ret_int でのエラー：不定 (2) ext_tsk でのエラー：不定 (3) 組み込まれていないサービスコールの呼び出し：不定 (4) 未定義の可変ベクタ・固定ベクタ割込み：CPU の割込み処理によってスタックに退避された PSW (5) vsys_dwn, ivsys_dwn の呼び出し：ユーザ任意

6.6.3 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。
また、割込み許可状態は、システムダウン発生時と同じになります。

表6.6 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	
3	PM	0	スーパバイザモード
4	U	0	ISP(割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 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 6.6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6.6 浮動小数点演算命令を使用するハンドラの例

6.8 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) ハンドラ

アプリケーション内に前述の DSP 機能命令を実行するタスクやハンドラがひとつでもある場合は、すべてのハンドラが明示的に ACC レジスタを保証する必要があります。図 6.7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)
{
  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6.7　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

RI600/4 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 RI600/4 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。GUI コンフィギュレータを用いて cfg ファイルを作成することもできます。

(4) コマンドラインコンフィギュレータ(cfg600)の実行

cfg600 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。

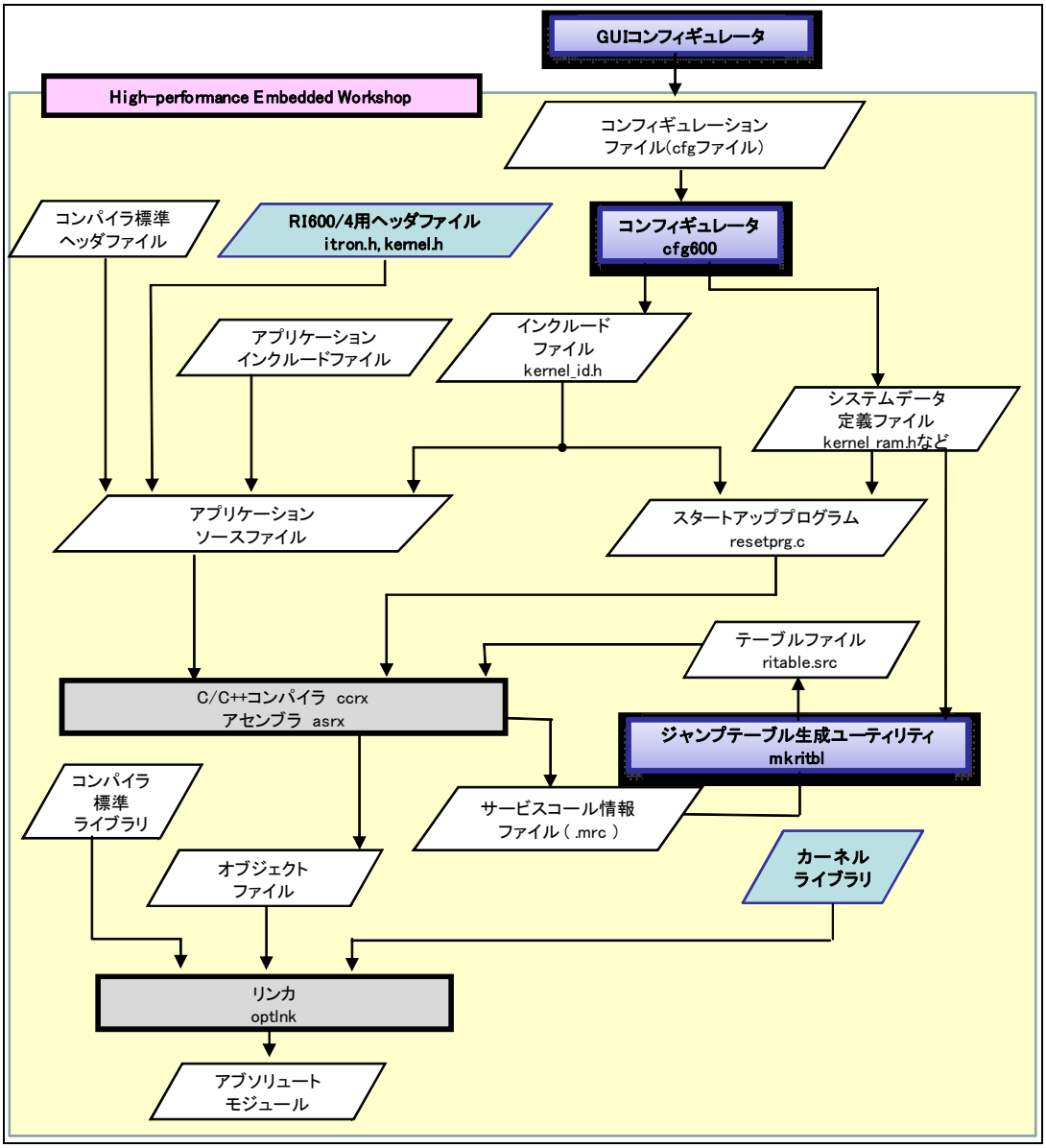

図7.1 ロードモジュール生成フロー

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) システムダウンルーチン(_RI_sys_dwn__())
(3) 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
  高速割込みを使用する場合は、FINTV レジスタを高速割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)
- システムタイマの初期化
  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、cfg600 が出力する ri_cmt.h をインクルードし、"void _RI_init_cmt()"を呼び出してください。
- カーネルを起動(ivsta_knl の呼び出し)

リセットからカーネル起動までは、割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"宣言が必要です。この宣言と、cfg600 が kernel_ram.h に出力する#pragma stacksize 宣言により、コンパイラによって関数先頭でスタックポインタをシステムスタックに初期化するコードが生成されます。

(2) システムダウンルーチン(_RI_sys_dwn__())

「6.6**システムダウンルーチン**」を参照してください。

(3) kernel_rom.h, kernel_ram.h の取り込み

kernel_rom.h および kernel_ram.h は cfg600 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 RI600-4 */
#include "kernel_id.h"     /* generated by cfg600 */
#include "kernel_ram.h"    /* generated by cfg600 */
#include "kernel_rom.h"    /* generated by cfg600 */
```

(4) 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。

(5) 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"
6.    #include    "kernel.h"                  // Provided by RI600/4
7.    #include    "kernel_id.h"               // Generated by cfg600
8.
9.    #if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
10.   #include    "ri_cmt.h"                  // Generated by cfg600
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_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 );
19.   #ifdef __cplusplus
20.   }
21.   #endif
22.
23.   #ifdef __cplusplus              // Use SIM I/O
24.   extern "C" {
25.   #endif
26.   extern void _INIT_IOLIB(void);
27.   extern void _CLOSEALL(void);
28.   #ifdef __cplusplus
29.   }
30.   #endif
31.
32.   #define FPSW_init 0x00000100
33.
34.   //extern void srand(_UINT);     // Remove the comment when you use rand()
35.   //extern _SBYTE *_s1ptr;                // Remove the comment when you use strtok()
36.
37.   #ifdef __cplusplus              // Use Hardware Setup
38.   extern "C" {
39.   #endif
40.   extern void HardwareSetup(void);
41.   #ifdef __cplusplus
42.   }
43.   #endif
44.
```

```
45.    //#ifdef __cplusplus            // Remove the comment when you use global class object
46.    //extern "C" {                  // Sections C$INIT and C$END will be generated
47.    //#endif
48.    //extern void _CALL_INIT(void);
49.    //extern void _CALL_END(void);
50.    //#ifdef __cplusplus
51.    //}
52.    //#endif
53.
54.    #pragma section ResetPRG
55.
56.    #pragma entry PowerON_Reset_PC
57.
58.    ////////////////////////////////////////////////////////////////////////
59.    // Power-on Reset Program
60.    ////////////////////////////////////////////////////////////////////////
61.    void PowerON_Reset_PC(void)
62.    {
63.        set_fpsw(FPSW_init);
64.
65.        _INITSCT();
66.
67.        _INIT_IOLIB();                  // Use SIM I/O
68.
69.    //  errno=0;                        // Remove the comment when you use errno
70.    //  srand((_UINT)1);                    // Remove the comment when you use rand()
71.    //  _s1ptr=NULL;                    // Remove the comment when you use strtok()
72.
73.        HardwareSetup();                // Use Hardware Setup
74.
75.    //  set_fintv(<handler address>);   // Initialize FINTV register
76.
77.
78.    #if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
79.        _RI_init_cmt();                 // Initialize CMT for RI600/4
80.                                        // Do comment-out when clock.timer is either NOTIMER or OTHER.
81.    #endif
82.
83.        nop();
84.
85.    //  _CALL_INIT();                   // Remove the comment when you use global class object
86.
87.        vsta_knl();                     // Start RI600/4
88.                                        // Never return from vsta_knl
89.
90.        _CLOSEALL();                    // Use SIM I/O
91.
```

```
92.     //  _CALL_END();                    // Remove the comment when you use global class object
93.
94.         brk();
95.
96.     }
97.
98.     ////////////////////////////////////////////////////////////////////////
99.     // System-down routine for RI600/4
100.    ////////////////////////////////////////////////////////////////////////
101.    #pragma section P PRI_KERNEL
102.    #pragma section B BRI_RAM
103.    struct SYSDWN_INF{
104.        W   type;
105.        VW  inf1;
106.        VW  inf2;
107.        VW  inf3;
108.    };
109.
110.    volatile struct SYSDWN_INF _RI_sysdwn_inf;
111.
112.    void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 )
113.    {
114.        // Now PSW.I=0 (all interrupts are masked.)
115.
116.        _RI_sysdwn_inf.type = type;
117.        _RI_sysdwn_inf.inf1 = inf1;
118.        _RI_sysdwn_inf.inf2 = inf2;
119.        _RI_sysdwn_inf.inf3 = inf3;
120.
121.        while(1)
122.        ;
123.    }
124.
125.    ////////////////////////////////////////////////////////////////////////
126.    // RI600/4 system data
127.    ////////////////////////////////////////////////////////////////////////
128.    #include "kernel_ram.h"     // generated by cfg600
129.    #include "kernel_rom.h"     // generated by cfg600
```

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(MDE 端子)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 セクション一覧

RI600/4 では、以下のセクションを使用しています。リンク時には、これらのセクションを適切なアドレスに配置してください。

CPU のエンディアン(MDE 端子)とは異なるエンディアンのメモリを使用する場合、そのメモリには以下のセクションを配置してはなりません。

- PRI_KERNEL セクション
  カーネルのプログラムセクションです。
  セクション属性は”CODE”、アライメント数は 1 です。
- CRI_ROM セクション
  カーネルの定数セクションです。
  セクション属性は”ROMDATA”、アライメント数は 4 です。
- FIX_INTERRUPT_VECTOR セクション
  固定割込みベクタテーブルのセクションです。このセクションは、0xFFFFFF80 番地に配置する必要があります。
  セクション属性は”ROMDATA”、アライメント数は 4 です。
- INTERRUPT_VECTOR セクション
  可変割込みベクタテーブルのセクションです。
  セクション属性は”ROMDATA”、アライメント数は 4 です。
- SI セクション
  システムスタックのセクションです。
  セクション属性は”DATA”、アライメント数は 4 です。
- SURI_STACK セクション(タスクスタック)
  タスクのスタックに付与するセクション名は、cfg ファイルの task[].stack_section 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。
  これらのセクションの属性は”DATA”、アライメント数は 4 です。
- BRI_RAM セクション
  カーネルの変数セクションです。
  セクション属性は”DATA”、アライメント数は 4 です。
- BRI_HEAP セクション(メッセージバッファ、可変長メモリプール、固定長メモリプール領域)
  これらの領域に付与するセクション名は、cfg ファイルの各オブジェクトの定義文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 BRI_HEAP となります。
  これらのセクションの属性は”DATA”、アライメント数は 4 です。

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、mkritbl への入力となります。

アプリケーションをライブラリ化する場合は、ライブラリ化時に生成された mrc ファイルも mkritbl に入力してください。

kernek.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。

7.6 CPU の動作モード

タスクはユーザモード(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、特権命令は以下の場合に使用されます。

- コンパイラの smachine.h によって提供される組み込み関数を使用
- アセンブリ言語プログラムで、特権命令を記述

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。

8. コンフィギュレータ(cfg600)

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、RI600/4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(cfg600)です。

cfg600 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。[4]
- 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。

[4] 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)
- 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[])
- 可変割込みベクタ定義(interrupt_vector[])
- 固定割込みベクタ定義(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込みは、「カーネル管理外割込み」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	○	○	×	×
FPSW	○	○	○	×
ACC	○	○	×	○
FPSW,ACC	○	○	○	○
MIN	○	×	×	×
MIN,FPSW	○	×	○	×
MIN,ACC	○	×	×	○
MIN,FPSW,ACC	○	×	○	○

【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-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

タスクでの浮動小数点演算命令	タスクでのDSP機能命令	system.context
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

タスクでの浮動小数点演算命令	タスクでのDSP機能命令	system.context
あり	あり	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

タスクでの浮動小数点演算命令	タスクでのDSP機能命令	system.context
あり	あり	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

タスクでの浮動小数点演算命令	タスクでのDSP機能命令	system.context
あり	あり	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 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 と base 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、cfg600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。
CMT0, CMT1, CMT2, CMT3のいずれかを選択した場合、カーネル起動前に初期化関数”void _RI_init_cmt(void)”をコールする必要があります。_RI_init_cmt()をコールするには、cfg600が出力するri_cmt.hをインクルードする必要があります。また、この初期化関数は全割込みを禁止した状態で呼び出してください。通常は、resetprg.cのPowerON_Reset_PC()からコールするようにしてください。
テンプレートファイルは、RI600/4によって提供されます。最初のバージョンでは、rx610.tplのみが提供されます。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。
テンプレートファイル名とマイコン品種名は、必ずしも対応していないことに注意してください。例えば、rx610.tplは、RX610グループ以外のマイコンでも使用できる場合があります。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、cfg600はエラーを報

告しませんが、cfg600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rx610.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) タイマ割込み優先レベル(IPL)

【説明】システムタイマの割込み優先レベルを定義します。
システムタイマの割込みハンドラ処理中は、ここで定義した割込みレベルより低いレベルの割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。
cfg600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 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;
};
```

■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】空いているID番号を小さいほうから順に自動的に割り当てます。

【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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。RI600/4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、cfg600によって生成されます。
【定義形式】数値
【定義範囲】44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。
【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

8.4.5 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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

8.4.6 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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

8.4.7 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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

8.4.8 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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。

8.4.9 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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】RI600/4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

8.4.10 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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。
【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_HEAP)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

8.4.11 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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。
【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_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】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

8.4.12 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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。
【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_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、heap_sizeおよびmax_memsizeを元にcfg600が下記表から選択したバリエーション(1～12種類)のいずれかとなります。ただし、この振る舞いは今後のバージョンで変更される可能性があります。

項番	メモリブロックサイズ (16 進数)
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)

8.4.13 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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～(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

8.4.14 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】空いているID番号を小さいほうから順に自動的に割り当てます。
【注意】ID番号は、1から漏れなく付与される必要があります。つまり、ID番号を指定する場合は、その値は定義するオブジェクト数以下でなければなりません。

(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

8.4.15 可変ベクタ割込み定義(interrupt_vector[])

interrupt_vector[]は、可変ベクタに対する割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込みが発生した場合は、システムダウンとなります。
なお、cfg600 はここで指定した割込みに関する割込み制御レジスタ(IPL 等)や、割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。

■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) カーネル管理割込み指定(os_int)

【説明】この割込みがカーネル管理割込みかどうかを定義します。
カーネル割込みマスクレベル(system.system_IPL)以下の割込みはカーネル管理割込み、それ以外はカーネル管理外割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込みはカーネル管理割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：カーネル管理割込み
- NO：カーネル管理外割込み

【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】cfg600は、entry_addressで指定された関数を、割込み関数としてkenrel_id.hに#pragma interrupt宣言を出力します。このpragma宣言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。

- E：多重割込みを許可する”enable”スイッチを渡します。
- F：高速割込みを指定する”fint”スイッチが渡されます。なお、高速割込みは必ずカーネル管理外割込み(os_int=NO)としなければなりません。
- S：割込みハンドラで使用するレジスタ数を制限する”save”スイッチが渡されます。

【省略時の扱い】何もスイッチを渡しません。

8.4.16 固定ベクタ割込み定義(interrupt_fvector[])

interrupt_fvector[]は、固定ベクタに対する割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込みの扱いとなります。

固定割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、RI600/4では各ベクタアドレスに対して表 8.8に示すようにベクタ番号を割り当てています。

本定義を行わないベクタ番号の割込みが発生した場合は、システムダウンとなります。ただし、リセットベクタ(ベクタ番号 31)を定義しない場合は、リセット関数は PowerON_Reset_PC()となります。

表8.8 固定割込みベクタ番号

ベクタアドレス	ベクタ番号	RX610 グループでの要因
FFFFFF80H	0	(予約領域)
FFFFFF84H	1	(予約領域)
FFFFFF88H	2	(予約領域)
FFFFFF8CH	3	(予約領域)
FFFFFF90H	4	(予約領域)
FFFFFF94H	5	(予約領域)
FFFFFF98H	6	(予約領域)
FFFFFF9CH	7	(予約領域)
FFFFFFA0H	8	(予約領域)
FFFFFFA4H	9	(予約領域)
FFFFFFA8H	10	(予約領域)
FFFFFFACH	11	(予約領域)
FFFFFFB0H	12	(予約領域)
FFFFFFB84	13	(予約領域)
FFFFFFB8H	14	(予約領域)
FFFFFFBCH	15	(予約領域)
FFFFFFC0H	16	(予約領域)
FFFFFFC4H	17	(予約領域)
FFFFFFC8H	18	(予約領域)
FFFFFFCCH	19	(予約領域)
FFFFFFD0H	20	特権命令例外
FFFFFFD4H	21	(予約領域)
FFFFFFD8H	22	(予約領域)
FFFFFFDCH	23	未定義命令例外
FFFFFFE0H	24	(予約領域)
FFFFFFE4H	25	浮動小数点例外
FFFFFFE8H	26	(予約領域)
FFFFFFECH	27	(予約領域)
FFFFFFF0H	28	(予約領域)
FFFFFFF4H	29	(予約領域)
FFFFFFF8H	30	ノンマスカブル割込み
FFFFFFFCH	31	リセット

なお、cfg600 はここで指定した割込みに関する割込み制御レジスタ(IPL 等)や、割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。

■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8.8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) PRAGMA 拡張機能に渡すスイッチ(pragma_switch))

【説明】cfg600は、entry_addressで指定された関数を、割込み関数としてkenrel_id.hに#pragma interrupt宣言を出力します。このpragma宣言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、ベクタ番号31(リセット)については、ここでの指定は無視され、#pragma interrupt宣言も出力しません。
【定義形式】シンボル
【定義範囲】以下を指定できます。

- S：割込みハンドラで使用するレジスタ数を制限する”save”スイッチが渡されます。

【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、mkritbl 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、mkritbl 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 8.1に示します。

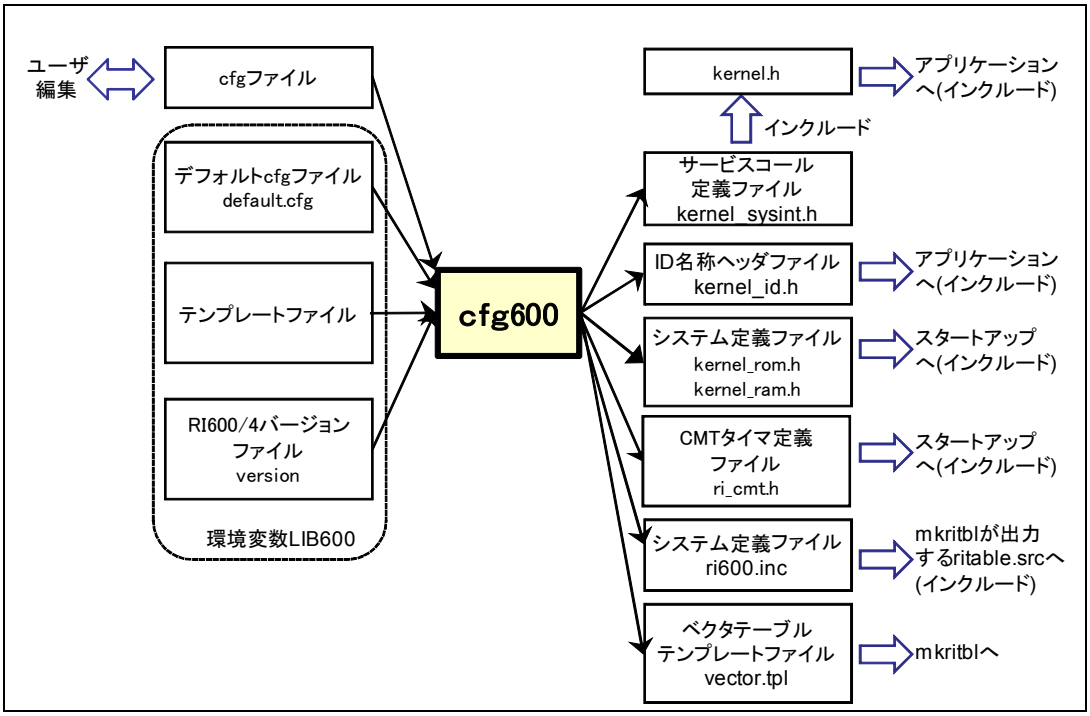

図8.1 コンフィギュレータ動作概要

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。

```
cfg600[△<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(「6.6 **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 8.9に示すように分類されます。

表8.9 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>
XXXの数値またはID番号が間違っています。

O1003 (E) Unknown token --> <XXX>
XXXは定義名として認識できません。

O1004 (E) XXX's ID number is too large.--> <ID 値>
XXX[]に指定したID番号が大きすぎます。ID番号は、当該オブジェクトの定義数(XXX[]の数)以下でなければなりません。

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でなけれなばりません。

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_meorypool.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。

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 not defined
(1)XXXが定義されていません。
(2)当該IDのオブジェクトの定義が抜けています。本エラーは、O1004と共に発生します。

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。

(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。

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ィリティ(mkritbl)

9.1 概要

mkritbl 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。

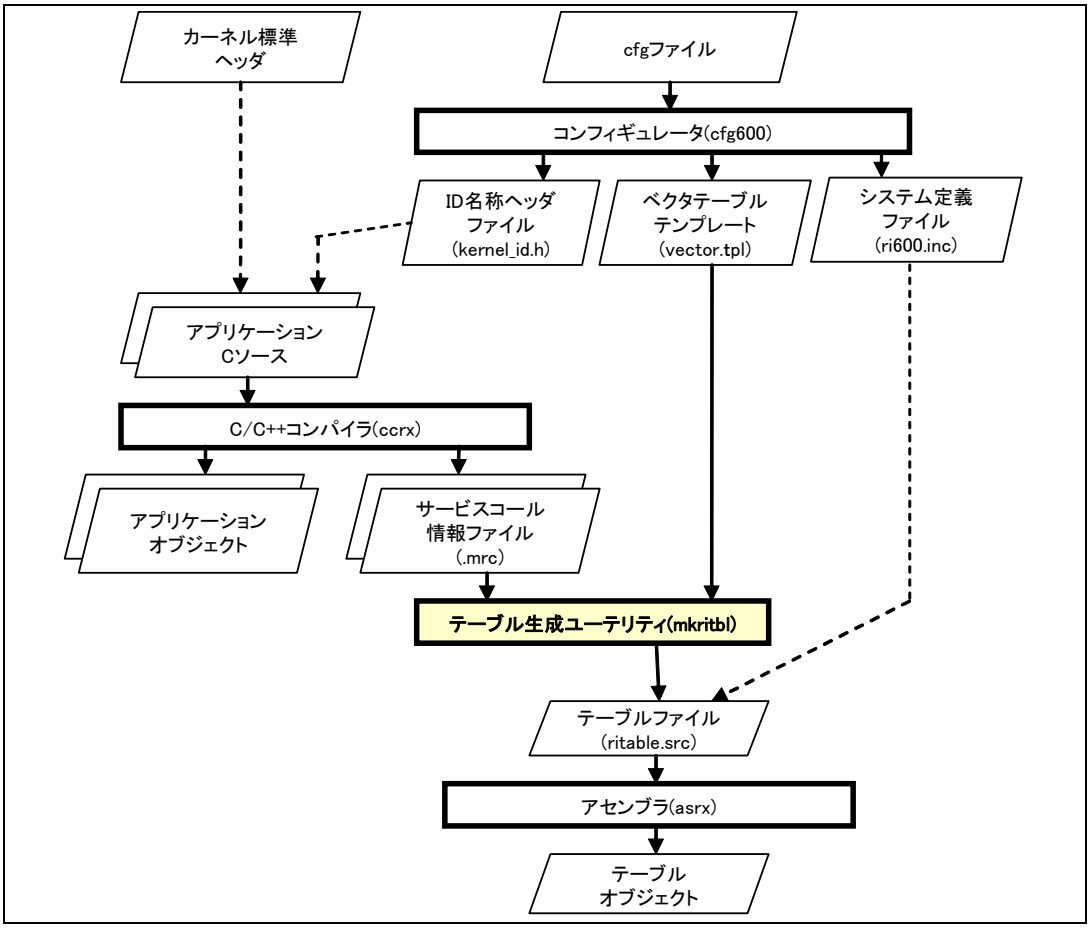

図9.1 mkritbl 概要

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。mkritbl 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、mkritbl は cfg600 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:¥> mkritbl  <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、cfg600 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. GUI コンフィギュレータ

GUI コンフィギュレータは、GUI 画面上で各種カーネルコンフィギュレーション情報を入力することで、cfg ファイルを生成するツールです。cfg ファイルは、cfg600 に入力します。

GUI コンフィギュレータを使用すれば、cfg ファイルの記法を習得しなくてもカーネルを構築することができます。

図 10.1に、GUI コンフィギュレータと cfg ファイルおよび cfg600 の関係を示します。

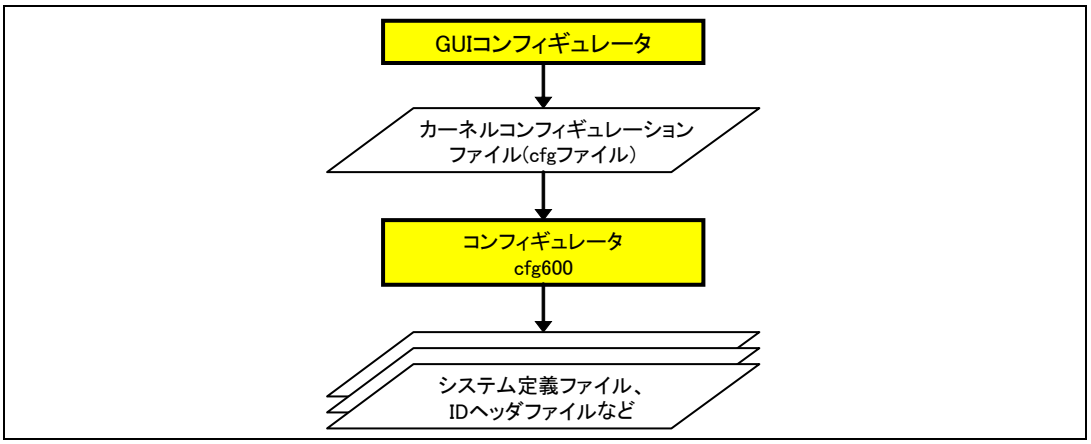

図10.1 GUI コンフィギュレータと cfg ファイルおよび cfg600 関係

GUI コンフィギュレータの使用方法については、オンラインヘルプを参照してください。

11. サンプルプログラム

11.1 概要

本章では、RI600/4 の応用例として、タスク間で交互に標準出力に文字列を出力するプログラムを示します。

表 11.1 サンプルプログラムの関数一覧

関数名	種類	ID 番号	優先度	機能
main()	タスク	1	1	task1,task2 を起動させます。
task1()	タスク	2	2	“task1 running”を出力します。
task2()	タスク	3	3	“task2 running”を出力します。
cyh1()	周期ハンドラ	1	－	task1 を起床します。

以下に、処理内容を説明します。

- main タスクは task1,task2,cyh1 を起動し、自タスクを終了させます。
- task1 は次の順で動作します。
  1. セマフォを獲得します(wai_sem)。
  2. 起床待ちに移行します(slp_tsk)。
  3. "task1 running"を出力します。
  4. セマフォを解放します(sig_sem)。
- task2 は次の順で動作します。
  1. セマフォを獲得します(wai_sem)。
  2. "task2 running"を出力します。
  3. セマフォを解放します(sig_sem)。
- cyh1 は 100ms 毎に起動され、task1 を起床します(iwup_tsk)。

11.2 ソースリスト

```
 1 /*****************************************************************
 2 *                       RI600/4  sample program
 3 *****************************************************************/
 4
 5 #include <stdio.h>
 6 #include “kernel.h”
 7 #include "kernel_id.h"
 8
 9
10 void task1( VP_INT exinf)
11 {
12     while(1){
13         wai_sem(ID_SEM1);
14         slp_tsk();
15         printf("task1 running¥n");
16         sig_sem(ID_SEM1);
17     }
18 }
19
20 void task2( VP_INT exinf)
21 {
22     while(1){
23         wai_sem(ID_SEM1);
24         printf("task2 running¥n");
25         sig_sem(ID_SEM1);
26     }
27 }
28
29 void cyh1( VP_INT exinf )
30 {
31     iwup_tsk(ID_TASK1);
32 }
```

11.3 サンプル cfg ファイル

```
 1 //*****************************************************************
 2 //      RI600/4 System Configuration File.
 3 //*****************************************************************
 4
 5 // System Definition
 6 system{
 7     stack_size  = 1024;
 8     priority    = 10;
 9     system_IPL  = 4;
10     message_pri = 10;
11     tic_nume    = 1;
12     tic_deno    = 1;
13     context     = NO;
14 };
15 //System Clock Definition
16 clock{
17     timer_clock = 25MHz;
18     timer       = CMT0;
19     IPL         = 4;
20 };
21 //Task Definition
22 task[]{
23     name        = ID_TASK1;
24     entry_address = task1();
25     initial_start = ON;
26     stack_size  = 512;
27     priority    = 1;
28 };
29 task[]{
30     name        = ID_TASK2;
31     entry_address   = task2();
32     initial_start = ON;
33     stack_size  = 512;
34     priority    = 2;
35 };
36 // Semaphore Definition
37 semaphore[]{
38     name        = ID_SEM1;
39     max_count   = 1;
40     initial_count   = 1;
41     wait_queue  = TA_TPRI;
42 };
43 // Cyclic Handler Definition
44 cyclic_hand[] {
45     name           = ID_CYC1;
46     entry_address  = cyh1();
47     interval_counter = 100;
48     start          = ON;
49     phsatr         = OFF;
50     phs_counter     = 0;
51     exinf          = 1;
52 };
```

12. スタックサイズの算出方法

12.1 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、タスクごとに確保する必要があります。各タスクのユーザスタックサイズは、cfg ファイルの task[].stack_size に指定します。
  また、各タスクのユーザスタックは、task[].stack_section で個別のセクションに割り当てることができます。
  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。
- 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。

12.2 “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。

12.3 各タスクのユーザスタックサイズの算出方法

各タスクのユーザスタックサイズは、以下の式で算出されます。

ユーザスタックサイズ = α + β

α

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)

β

タスクコンテキストサイズ
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 12.1を参照してください。

表 12.1 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

12.4 システムスタックサイズの算出方法

システムスタックを最も多く消費するのは、サービスコール処理中[5]に割込みが発生、さらに多重割込みが発生した場合です。すなわち、システムスタックの必要量 (最大サイズ)は以下の計算式で算出することができます。

$$\text{システムスタックの必要量} = \alpha + (\Sigma \beta i) + \gamma$$

α

使用するサービスコールの中で最大のシステムスタックサイズです。αの値は、カーネルのバージョンによって異なります。製品添付のリリースノートを参照してください。

βi

各割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。
iは、割込み優先レベルです。同じ優先レベルの割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βiとしてください。
なお、システムクロック割込みハンドラ(割込み優先レベル = clock.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。ε1, ε2およびε3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。

- ε1 + CYC
- ε2 + ALM
- ε3

♦CYC
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CYCとしてください。

♦ALM
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ALMとしてください。

γ

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、γを0として計算してください

[5] ユーザスタックからシステムスタックに切り替えた後

RX600シリーズ用リアルタイムOS
ユーザーズマニュアル
RI600/4 V.1.00
発行年月日　2009 年 8 月 28 日　Rev.1.00
発　　行　株式会社ルネサス テクノロジ 営業統括部
〒100-0004 東京都千代田区大手町 2-6-2
編　　集　株式会社ルネサスソリューションズ
グローバルストラテジックコミュニケーション本部
カスタマサポート部

株式会社 **ルネサス テクノロジ** 営業統括部　〒100-0004 東京都千代田区大手町2-6-2 日本ビル

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
西東京支社	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル)	(042) 524-8701
東北支社	〒980-0013	仙台市青葉区花京院1-1-20 (花京院スクエア)	(022) 221-1351
いわき支店	〒970-8026	いわき市平字田町120 (ラトブ)	(0246) 22-3222
茨城支店	〒312-0034	ひたちなか市堀口832-2 (日立システムプラザ勝田)	(029) 271-9411
新潟支店	〒950-0087	新潟市中央区東大通1-4-2 (新潟三井物産ビル)	(025) 241-4361
松本支社	〒390-0815	松本市深志1-2-11 (昭和ビル)	(0263) 33-6622
中部支社	〒460-0008	名古屋市中区栄4-2-29 (名古屋広小路プレイス)	(052) 249-3330
関西支社	〒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
北陸支社	〒920-0031	金沢市広岡3-1-1 (金沢パークビル)	(076) 233-5980
広島支店	〒730-0036	広島市中区袋町5-25 (広島袋町ビルディング)	(082) 244-2570
九州支社	〒812-0011	福岡市博多区博多駅前2-17-1 (博多プレステージ)	(092) 481-7695

※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。

■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：コンタクトセンタ　E-Mail: csc@renesas.com

Colophon 9.1

RI600/4 V.1.00
ユーザーズマニュアル

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

RJJ10J2562-0100