お客様各位

カタログ等資料中の旧社名の扱いについて

拝啓　時下ますますご清栄のこととお喜び申し上げます。平素は格別のご高配を賜り厚く御礼申し上げます。

さて、２０１７年５月１日を以ってルネサス セミコンダクタ パッケージ&テスト ソリューションズ株式会社の半導体製造装置をはじめとする各種産業用制御ボードの受託開発・製造および画像認識システム開発・製造・販売事業を日立マクセル株式会社へ譲渡したことにより、当該事業は日立マクセル株式会社の子会社として新設されるマクセルシステムテック株式会社に承継されております。

従いまして、ドキュメント等資料中には、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。

敬具

２０１７年５月１日

マクセルシステムテック株式会社

【発行】マクセルシステムテック（http://www.systemtech.maxell.co.jp/）

【お問い合わせ先】denki-support@maxell.co.jp

maxell

マクセルシステムテック株式会社

Smalight® OS V4
リファレンスマニュアル
RZ/A版

ルネサス セミコンダクタ パッケージ&テスト ソリューションズ株式会社

ご注意

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切な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が所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、弊社は責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、弊社は、予告なしに、本資料に記載した製品または仕様を変更することがあります。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、弊社はその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。弊社は、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、弊社へご照会ください。
7. 本資料の転載、複製については、文書による弊社の事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら弊社までご照会ください。

はじめに

このマニュアルは、リアルタイム・マルチタスク OS： Smalight OS (スマライトオーエス、SMArt & LIGHT Operating System) の使用方法について説明します。
Smalight OS をご使用になる前に本マニュアルをよく読んで理解してください。また下記関連マニュアルもお読みの上、理解してください。

本製品は、ルネサス エレクトロニクス(株)製 RZ ファミリ RZ/A シリーズ(ARM 社 Cortex-A9 MPCore(1Core 構成))のデバイスに対応するものです。

【関連マニュアル】
- 使用するデバイスのハードウェアマニュアル
- 使用するコンパイラのマニュアル

目次

1. リアルタイムOS概説 1

1.1 リアルタイムOS 1
1.2 タスク 1
1.3 マルチタスク 1
1.4 コンテキスト 1
1.5 リアルタイムOSのスタック 2

2. Smalight OS概説 3

2.1 概要 3
2.2 特徴 3
2.3 サービスコール 4
2.4 前提条件 5
2.4.1 開発環境 5
2.4.2 構築環境 5

3. 機能 6

3.1 システム状態 6
3.2 タスクの状態 7
3.3 タスクのスケジューリング 8
3.3.1 タスク優先度の設定 9
3.3.2 タスクスケジューリングの例 10
3.3.3 タスクディスパッチ禁止状態 12
3.4 タスク間同期・通信と排他制御 13
3.4.1 イベントフラグ 13
3.4.2 セマフォ 15
3.4.3 データキュー 16
3.5 時間管理 18
3.5.1 周期タイマハンドラ 18
3.5.2 システム時刻 18
3.5.3 周期ハンドラ 19
3.5.4 時間管理に関する注意事項 20
3.6 割込みハンドラ 21
3.6.1 vdisp無割込みハンドラ 21
3.6.2 vdisp有割込みハンドラ 22
3.6.3 多重割込み 23

4. サービスコール、コールバック 25

4.1 システム状態とサービスコール 25
4.2 コールバック 27
4.3 各サービスコールの動作 28
4.3.1 サービスコール動作別の分類 28
4.3.2 タスクスイッチ型サービスコールの基本動作 28
4.3.3 i付きサービスコールの基本動作 29
4.3.4 関数型サービスコールの基本動作 30
4.4 サービスコールの説明形式 31
4.5 サービスコールの基本的な振舞い 32
4.6 タスク管理 33
4.6.1 sta_tsk タスクの開始 33

4.6.2 ext_tsk 自タスクの終了 ... 34
4.6.3 slp_tsk タスクの起床待ち ... 35
4.6.4 tslp_tsk タスクの起床待ち(タイムアウト付き) ... 36
4.6.5 wup_tsk, iwup_tsk タスクの起床 ... 37
4.6.6 can_wup タスク起床要求のキャンセル ... 38
4.6.7 vrot_rdq, ivrot_rdq タスクのローテーション ... 39
4.6.8 sus_tsk, isus_tsk 他タスクのサスペンド ... 40
4.6.9 rsm_tsk, irsm_tsk サスペンドの解除 ... 41
4.6.10 dis_dsp ディスパッチの禁止 ... 42
4.6.11 ena_dsp ディスパッチの許可 ... 43
4.7 イベントフラグ ... 44
4.7.1 wai_flg イベントフラグ待ち ... 44
4.7.2 twai_flg イベントフラグ待ち(タイムアウト付き) ... 45
4.7.3 set_flg , iset_flg イベントフラグの設定 ... 46
4.7.4 clr_flg イベントフラグのクリア ... 47
4.7.5 vevtflg_init イベントフラグの初期化 ... 48
4.7.6 VEVTFLG_ATTR イベントフラグ属性の設定(マクロ) ... 49
4.8 セマフォ ... 50
4.8.1 wai_sem セマフォの獲得 ... 50
4.8.2 twai_sem セマフォの獲得(タイムアウト付き) ... 51
4.8.3 sig_sem , isig_sem セマフォの返却 ... 52
4.8.4 vsem_init セマフォの初期化 ... 53
4.8.5 VSEM_ATTR セマフォ属性の設定(マクロ) ... 54
4.9 データキュー ... 55
4.9.1 rcv_dtq データキューからの受信 ... 55
4.9.2 trcv_dtq データキューからの受信(タイムアウト付き) ... 56
4.9.3 snd_dtq , isnd_dtq データキューへの送信 ... 57
4.9.4 tsnd_dtq データキューへの送信(タイムアウト付き) ... 58
4.9.5 fsnd_dtq , ifsnd_dtq データキューへの強制送信 ... 59
4.9.6 vdtq_init データキューの初期化 ... 60
4.9.7 VDTQ_ATTR データキューの属性設定 ... 61
4.10 時間管理 ... 62
4.10.1 set_tim システム時刻の設定 ... 62
4.10.2 get_tim システム時刻の取得 ... 63
4.10.3 vsystim_init 時間管理の初期化 ... 64
4.10.4 ivsig_tim タイムティックの供給（周期タイマ処理） ... 65
4.11 周期ハンドラ ... 66
4.11.1 sta_cyc 周期ハンドラの開始 ... 66
4.11.2 stp_cyc 周期ハンドラの停止 ... 67
4.11.3 vcyc_init 周期ハンドラの初期化 ... 68
4.11.4 VCYC_ATTR 周期ハンドラの属性設定 ... 69
4.11.5 VCYC_CHG 周期ハンドラの起動周期変更 ... 70
4.11.6 callback_cychdr 周期ハンドラ本体(コールバック) ... 71
4.12 割込み ... 72
4.12.1 vdisp vdisp有割込みハンドラをディスパッチして終了 ... 72
4.12.2 callback_int 割込みハンドラ本体(コールバック) ... 73
4.13 その他の初期化 ... 74
4.13.1 vslos_init OSの起動 ... 74
4.13.2 kinit OS初期化処理(コールバック) ... 75
4.13.3 uinit ユーザ初期化処理(コールバック) ... 76
4.13.4 stack_init タスクスタックの初期化(コールバック) ... 77
4.14 その他 ... 78
4.14.1 idle アイドル処理(コールバック) ... 78

5. アプリケーションプログラムの作成 ... 79

5.1 作成の手順 ... 79
5.2 システムの起動処理 ... 80

5.2.1 システム起動時の処理フロー ... *80*
5.2.2 初期化処理 ... *81*
5.3 タスク ... 84
5.4 割込みハンドラ ... 85
5.4.1 ベクタハンドラ ... *85*
5.4.2 割込みベスタIDテーブルと割込みスタック ... *86*
5.4.3 vdisp無割込みハンドラ ... *88*
5.4.4 vdisp有割込みハンドラ ... *89*
5.4.5 割込みスタックについて ... *90*
5.5 時間管理 ... 91
5.5.1 時間管理の対象サービスコール ... *91*
5.5.2 時間管理の初期化 ... *91*
5.6 イベントフラグ ... 93
5.6.1 イベントフラグの対象サービスコール ... *93*
5.6.2 イベントフラグの初期化 ... *93*
5.7 セマフォ ... 94
5.7.1 セマフォの対象サービスコール ... *94*
5.7.2 セマフォの初期化 ... *94*
5.8 データキュー ... 95
5.8.1 データキューの対象サービスコール ... *95*
5.8.2 データキューの初期化 ... *95*
5.9 周期ハンドラ ... 96
5.9.1 周期ハンドラの対象サービスコール ... *96*
5.9.2 周期ハンドラの作成 ... *96*
5.9.3 周期ハンドラの初期化 ... *97*
5.10 周期タイマハンドラの初期化 ... 98
5.11 周期タイマハンドラの作成 ... 98

6. 構築 ... 99

6.1 ファイル･ディレクトリ構成と操作 ... 99
6.2 構築手順 ... 100
6.3 OS定義ファイル ... 102
6.4 コンフィギュレーションファイル ... 103
6.4.1 カーネルマスクレベルの設定 ... *103*
6.4.2 タスクの設定 ... *104*
6.4.3 周期タイマハンドラ周期時間の設定 ... *106*
6.4.4 イベントフラグ数の設定 ... *106*
6.4.5 セマフォ数の設定 ... *107*
6.4.6 周期ハンドラ数の設定 ... *107*
6.4.7 データキュー数の設定 ... *108*
6.5 DS-5 ... 109
6.5.1 Cコンパイラ ... *109*
6.5.2 アセンブラ ... *109*
6.5.3 リンカ ... *110*
6.5.4 ソースファイルの追加 ... *110*
6.5.5 ロードモジュールの生成 ... *110*
6.6 スタック使用量の算出 ... 111
6.6.1 タスクスタック ... *111*
6.6.2 割込みスタック ... *112*
6.6.3 OSスタック ... *115*

7. サンプルプログラム ... 116

付録A 変更履歴 ... 117

付録B 制限事項 ... 118

付録C　NEON/VFPに関する注意事項 119
C.1 NEON/VFPレジスタの有効化 119
C.2 fpscrレジスタの初期化 119
付録D　スタック仕様 120
付録E　データ型、リターンコード 121

図・表・リスト目次

図3-1 システム状態 6
図3-2 タスク状態遷移図 7
図3-3 タスクのスケジューリング 8
図3-4 タスクスケジュールの例 10
図3-5 割込み禁止状態でのタスク切替え 11
図3-6 イベントフラグの使用例 14
図3-7 イベントフラグを利用したデータ送信 14
図3-8 セマフォの使用例 15
図3-9 データキューの概念 16
図3-10 データキューの使用例 17
図3-11 時間管理の注意事項 18
図3-12 周期ハンドラの起動タイミング 19
図3-13 周期ハンドラの呼び出し 19
図3-14 vdisp無割込みハンドラの基本動作 21
図3-15 vdisp有割込みハンドラの基本動作 22
図3-16 多重割込み動作例 23
図3-17 多重割込み実装上の注意事項① 24
図3-18 多重割込み実装上の注意事項② 24
図4-1 タスクスイッチ型サービスコールの基本動作 28
図4-2 i付きサービスコールの基本動作 29
図4-3 データキューへの強制送信時、空きがない場合の動作(データ数=3) 59
図5-1 アプリケーション作成フロー 79
図5-2 システム起動時のフロー 80
図5-3 vdisp無割込みハンドラのフロー 88
図5-4 vdisp有割込みハンドラのフロー 89
図6-1 構築手順① 100
図6-2 構築手順② 101
図6-3 関数のスタック計算方法 111
図6-4 タスクのスタック計算方法 111
図6-5 vdisp無割込みハンドラのスタック切り替えタイミング 112
図6-6 vdisp有割込みハンドラのスタック切り替えタイミング 113
図6-7 周期タイマハンドラのスタック切り替えタイミング 114
図D-1 スタック仕様 120

表2-1 サービスコール一覧 4
表2-2 開発環境 5
表2-3 DS-5 プロジェクト情報 5
表3-1 タスク優先度の設定例① 9
表3-2 タスク優先度の設定例② 9
表3-3 タスク優先度の設定例③ 9
表3-4 割込みハンドラの定義 21
表4-1 システム状態とサービスコール① 25
表4-2 システム状態とサービスコール② 26
表4-3 コールバックルーチン 27
表4-4 タスクスイッチ型サービスコール 28
表4-5 i付きサービスコール 29
表4-6 関数型サービスコール 30
表5-1 時間管理関連サービスコール一覧 91

表5-2 イベントフラグ関連サービスコール一覧 93
表5-3 セマフォ関連サービスコール一覧 94
表5-4 データキュー関連サービスコール一覧 95
表5-5 周期ハンドラ関連サービスコール一覧 96
表6-1 ファイル・ディレクトリ構成 99
表6-2 カーネルマスクレベル設定① 103
表6-3 Cコンパイラの設定 109
表6-4 アセンブラの設定 109
表6-5 リンカの設定 110
表6-6 属性と配置名 110
表7-1 サンプルプログラム一覧 116
表E-1 データ型一覧 121
表E-2 リターンコード一覧 121

リスト5-1 リセットプログラム(vhandlerRES.s) 81
リスト5-2 リセットプログラム(main.c) 81
リスト5-3 kinitコールバック(kinit.c) 82
リスト5-4 uinitコールバック(user.c) 83
リスト5-5 タスク記述方法 84
リスト5-6 タスク起動状態の設定方法(stackini.c) 84
リスト5-7 ベクタテーブル定義(vector.s) 85
リスト5-8 ベクタIDテーブル(vectbl.c) 86
リスト5-9 割込みスタック定義(istack.c) 87
リスト5-10 vdisp無割込みハンドラ本体 88
リスト5-11 vdisp有割込みハンドラ本体 89
リスト5-12 時間管理の有効設定 (slosdef.h) 91
リスト5-13 時間管理の初期化(kinit.c) 91
リスト5-14 イベントフラグの有効設定 (slosdef.h) 93
リスト5-15 イベントフラグの初期化(kinit.c) 93
リスト5-16 セマフォの有効設定 (slosdef.h) 94
リスト5-17 セマフォの初期化(kinit.c) 94
リスト5-18 データキューの有効設定 (slosdef.h) 95
リスト5-19 データキューの初期化(kinit.c) 95
リスト5-20 周期ハンドラ記述方法 96
リスト5-21 周期ハンドラの有効設定 (slosdef.h) 97
リスト5-22 周期ハンドラの初期化(kinit.c) 97
リスト5-23 周期タイマハンドラの初期化(user.c) 98
リスト5-24 周期タイマハンドラの例(user.c) 98
リスト6-1 OS定義ファイルの設定(slosdef.h) 102
リスト6-2 カーネルマスクレベルの設定(config.c) 103
リスト6-3 タスクの設定①(config.c) 104
リスト6-4 タスクの設定②(config.c) 105
リスト6-5 周期タイマハンドラ周期時間の設定例(config.c) 106
リスト6-6 イベントフラグ数の設定(config.c) 106
リスト6-7 セマフォ数の設定(config.c) 107
リスト6-8 周期ハンドラ数の設定(config.c) 107
リスト6-9 データキュー数の設定(config.c) 108

1. リアルタイム OS 概説

1.1 リアルタイムＯＳ

OS の中で特に時間の制約が厳しい、リアルタイム性が求められるといった要求に特化した OS をリアルタイム OS といいます。

例えば、車に搭載されるカーナビゲーションシステムでは、目的地を通り過ぎてから目的地のガイドされたり、現在の位置を示すマップ表示が遅かったりしたのでは、カーナビゲーションシステムとしての機能が満たされません。

この様にリアルタイム性が求められるシステムで使用される OS がリアルタイム OS です。主に組み込みシステムでリアルタイム OS が採用されています。リアルタイム OS の代表格には、μTRON 仕様準拠 OS があります。

1.2 タスク

OS から見た処理の単位をタスクといいます。逆にユーザから見た処理の単位はジョブといいます。OS 上でジョブを実現するための手段として、タスクという処理単位で実装され、1つのジョブはひとつ、または複数のタスクからなりたっています。

1.3 マルチタスク

OS が複数のタスクを切り替えながら実行することをマルチタスクと言います（逆に同時に1つのタスクしか実行しない方式をシングルタスクと言います）。

タスク切り替えのことをディスパッチといいます。また、タスクスケジュールを行う OS の機能をディスパッチャといいます。ディスパッチャで実行タスクを決定する処理をタスクスケジューリングといいます。タスクを切り替えることにより、入出力待ちなどでタスクが待ち状態となっても、他のタスクが動作できるため、システム全体のスループットが向上します。

短い時間でタスクの切り替えを行うことで、複数のアプリケーション(タスク)が平行して動作している様に見えます。

1.4 コンテキスト

コンテキストとは CPU の状態（プログラムカウンタ、スタックポインタ、レジスタ　など）を指します。

各々のタスクは個別にコンテキストをもっており、実行中の状態から、他のタスクへ切り替わるタイミングで、コンテキストを保存します。再度、そのタスクが実行されるとき、保存されたコンテキストから、中断前の状態に戻すことで矛盾なく、実行を再開することができます。

コンテキストの保存・復帰が行われるタイミングとしては、OS のサービスコール発行時、割込み発生時などがあります。

一般的にコンテキストはスタック領域、OSの管理領域に保存されます。

1.5　リアルタイム OS のスタック

リアルタイム OS のスタックは、タスクスタック、割込みスタック、OSスタックがあります。

(1)タスクスタック

タスクスタックは、タスクを実行するための必要な情報で、関数内のローカル変数、コンテキストなどが格納されます。タスクの切り替えが発生しても継続して実行するためには、タスク毎にスタックを準備する必要があります。

(2)割込みスタック

割込みスタックは、割込み関数内のローカル変数、割込み発生元への戻り情報などが格納されます。割込み発生元のスタックをそのまま使用する実装方法もありますが、多重割り込みを考慮した場合、割り込み発生元のスタック使用量の見積が困難となる、また、スタックサイズも大きくなることから、リアルタイム OS では割込みレベル毎に割込みスタックを準備し、スタックを切り替えて使用します。

(3)OS スタック

OS スタックは OS 実行中に使用されるスタックです。

リアルタイム OS が主に採用される組み込みシステムの特性上、使用できるメモリ容量に制限があります。限りあるメモリを有効に使用するため、スタック使用量を計算して設定する必要があります。

スタックサイズを必要以上に指定した場合、使われない無駄なメモリが発生します。また、スタックサイズが少ない場合、指定されたスタックを超えてメモリの内容が不正に書き換えられ動作結果が不定となりますので、適切なスタックサイズを指定することが重要となります。

2. Smalight OS 概説

2.1 概要

本リアルタイム OS は小容量コンパクトなリアルタイム・マルチタスク OS です。

2.2 特徴

(1)ITRON 仕様ライクな API

ITRON 仕様に完全に準拠したものでありませんが、ITRON 仕様ライクな API 提供でサービスコールのイメージがつかみやすく、また、将来 ITRON 仕様準拠 OS への置き換え時、移植が容易です。

(2)タスクスケジュール方式:優先度に基づいたスケジューリング

優先度ベースのスケジューリングに加えて、最下位優先度タスクについては FCFS(First Come First Service)によるスケジューリングをサポートしています。

(3)各種オブジェクトの最大数(タスク,イベントフラグ,セマフォ,データキュー,周期ハンドラ):127 個

(4)サンプルのベクタテーブル、ユーザタスク、割込みハンドラを提供しています。

2.3　サービスコール

以下にサービスコール一覧を示します。

表2-1 サービスコール一覧

区分	サービスコール名称	説明
タスク管理関連	sta_tsk	タスクの開始
	ext_tsk	自タスクの終了
	slp_tsk	タスクの起床待ち
	tslp_tsk	タスクの起床待ち(タイムアウト付き)
	wup_tsk, iwup_tsk	タスクの起床
	can_wup, ican_wup	タスク起床要求のキャンセル
	vrot_rdq, ivrot_rdq	タスクのローテーション
	sus_tsk, isus_tsk	他タスクのサスペンド
	rsm_tsk, irsm_tsk	サスペンドの解除
	dis_dsp	ディスパッチの禁止
	ena_dsp	ディスパッチの許可
イベントフラグ	wai_flg	イベントフラグ待ち
	twai_flg	イベントフラグ待ち(タイムアウト付き)
	set_flg, iset_flg	イベントフラグの設定
	clr_flg, iclr_flg	イベントフラグのクリア
	vevtflg_init	イベントフラグの初期化
	VEVTFLG_ATTR(*1)	イベントフラグの属性設定
セマフォ	wai_sem	セマフォの獲得
	twai_sem	セマフォの獲得(タイムアウト付き)
	sig_sem, isig_sem	セマフォの返却
	vsem_init	セマフォの初期化
	VSEM_ATTR(*1)	セマフォの属性設定
データキュー	rcv_dtq	データキューからの受信
	trcv_dtq	データキューからの受信(タイムアウト付き)
	snd_dtq, isnd_dtq	データキューへの送信
	tsnd_dtq	データキューへの送信(タイムアウト付き)
	fsnd_dtq, ifsnd_dtq	データキューへの強制送信
	vdtq_init	データキューの初期化
	VDTQ_ATTR(*1)	データキューの属性設定
時間管理	set_tim, iset_tim	システム時刻の設定
	get_tim, iget_tim	システム時刻の取得
	vsystim_init	時間管理の初期化
	ivsig_tim	タイムティックの供給（周期タイマ処理）
周期ハンドラ	sta_cyc, ista_cyc	周期ハンドラの開始
	stp_cyc, istp_cyc	周期ハンドラの停止
	vcyc_init	周期ハンドラの初期化
	VCYC_ATTR(*1)	周期ハンドラの属性設定
	VCYC_CHG(*1)	周期ハンドラの起動周期変更
割込み関連	vdisp	vdisp有割込みハンドラをディスパッチして終了
その他の初期化	vslos_init	OSの起動

(*1) マクロです。

2.4 前提条件

2.4.1 開発環境

本製品の開発環境を以下に示します。他の開発環境への移行は、ユーザの責任において行ってください。

表2-2 開発環境

ツール名
ARM社 統合開発環境「ARM Development Studio 5(DS-5)」 Version 5.21 Compiler toolchain(armcc、armasm、armlink) Version 5.05

2.4.2 構築環境

本製品では、以下の DS-5 プロジェクトを提供しています。

表2-3 DS-5 プロジェクト情報

ワークスペース	プロジェクト	説明
¥RZA_v***vvvl***¥armcc	¥smalight-os(*1)	Smalight OS 本体用
	¥u-ap	ユーザアプリケーション用

vvv : バージョン／リビジョン
l : 提供形態(s:ソース付属あり、r:ソース付属なし)
(*1) 提供形態によっては存在しない場合があります

3. 機能

3.1 システム状態

システム状態は、システム初期化部、OS 部、タスク部、割込み部で管理します。

システム初期化部はシステムの初期化を行う処理で、kinit, uinit、stack_init コールバックルーチンがシステム初期化部に属します。OS 部は OS 実行中の状態です。タスク部はタスク実行中の状態です。割込み部は割込み実行中の状態です(周期ハンドラの実行中は割込み部です)。

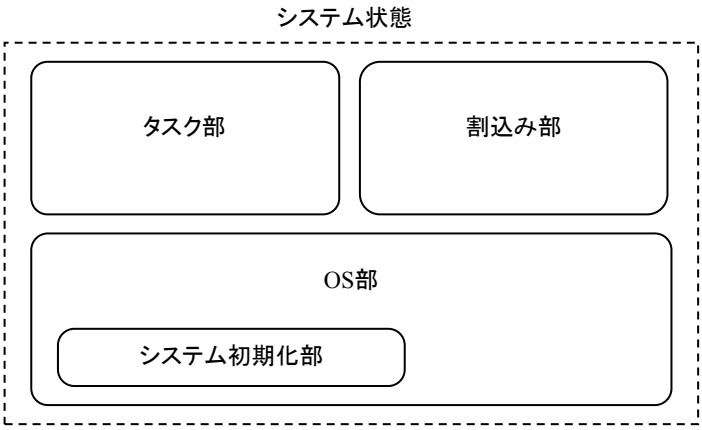

図3-1 システム状態

3.2　タスクの状態

タスク状態は Running, Ready, Waiting, Suspended, Dormant の 5 種類で管理します。以下に状態遷移図を示します。

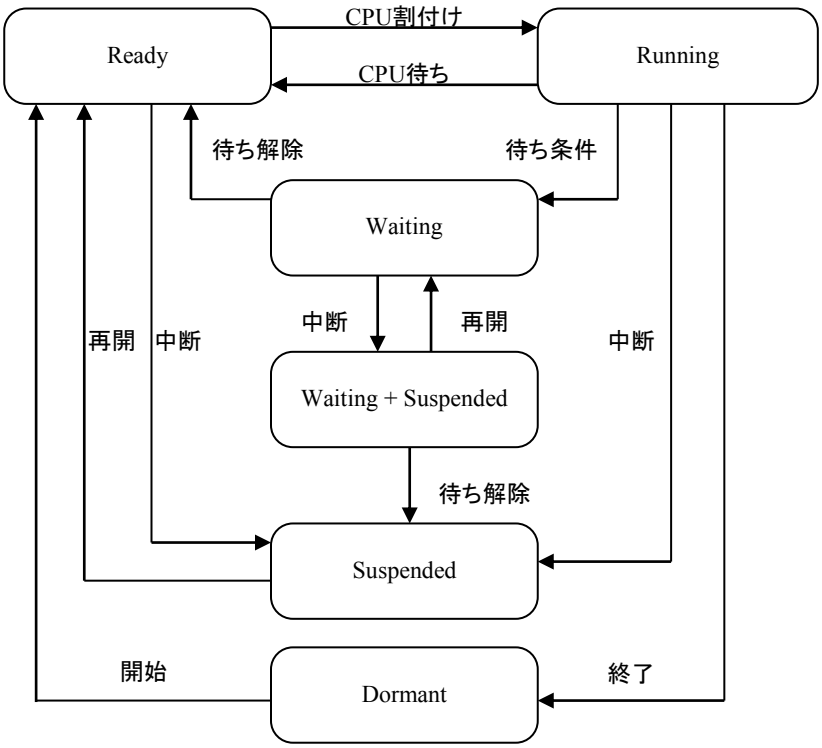

図3-2 タスク状態遷移図

3.3　タスクのスケジューリング

本 OS は優先度ベースのスケジューリングと FCFS(First Come First Service)によるスケジューリングの2つの方式をサポートしています。この2つの方式は同時に使用することができます。

優先度ベースのスケジューリングの対象となるタスクをプライオリティタスク(Priority Task)といいます。FCFS(First Come First Service)によるスケジューリングに対象となるタスクをローテーションタスク(Rotation Task)といいます。ローテーションタスクは、最低優先度のタスクとして管理されます。

プライオリティタスクの優先度は、タスク ID 番号と同じ値で、値が小さい程、優先度が高くなります（つまり、プライオリティタスクは同一優先度レベルのタスクが存在しません）。プライオリティタスクが Ready 状態になるとタスク優先度に従って Running 状態になります。一番優先度の高いプライオリティタスクが一度 Running 状態になると Dormant 状態、Waiting 状態、もしくは Suspended 状態になるまで、タスクの切替えは発生しません。

ローテーションタスクは、優先度が一番低いタスクで、同一優先度に複数のタスクを定義することができます。実行順序は FCFS (First Come First Service)にてで管理されます。

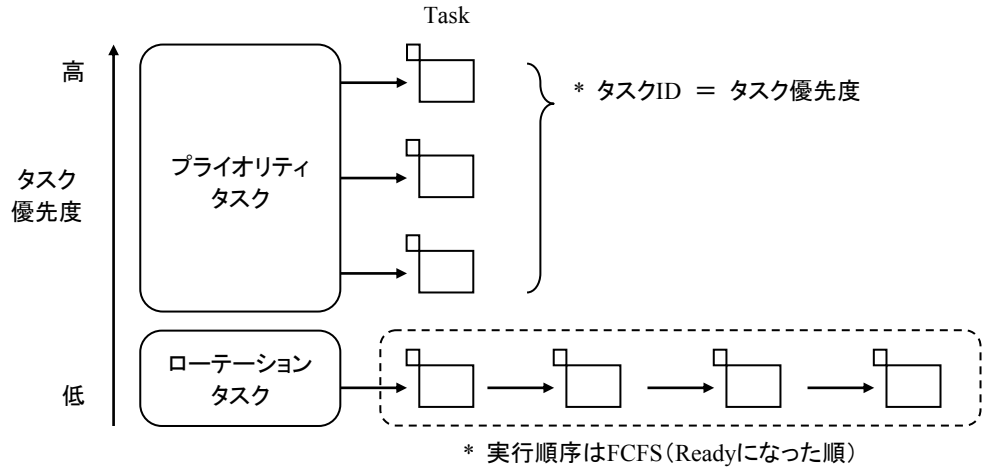

図3-3 タスクのスケジューリング

3.3.1　タスク優先度の設定

プライオリティタスク、ローテーションタスクは構築により静的に決定します。構築により、どの様にタスク優先度が設定されるかを具体的に説明します。構築情報では、タスク数とプライオリティタスク数を指定します。これにより、各タスクの優先度が決定します。

（1）タスク数=4, プライオリティタスク数=4

表3-1 タスク優先度の設定例①

タスクID	種類	優先度
1	プライオリティタスク	1番優先度が高い
2	プライオリティタスク	2番目に優先度が高い
3	プライオリティタスク	3番目に優先度が高い
4	プライオリティタスク	4番目に優先度が高い

（2） タスク数=4, プライオリティタスク数=2

表3-2 タスク優先度の設定例②

タスクID	種類	優先度
1	プライオリティタスク	1番優先度が高い
2	プライオリティタスク	2番目に優先度が高い
3	ローテーションタスク	3番目に優先度が高い
4	ローテーションタスク	3番目に優先度が高い

（3） タスク数=4, プライオリティタスク数=0

表3-3 タスク優先度の設定例③

タスクID	種類	優先度
1	ローテーションタスク	1番優先度が高い
2	ローテーションタスク	1番優先度が高い
3	ローテーションタスク	1番優先度が高い
4	ローテーションタスク	1番優先度が高い

構築方法の詳細は「6.4.2 タスクの設定」を参照ください。

3.3.2　タスクスケジューリングの例

以下にサービスコールによるタスク制御の例を示します。例では、3 個のタスク(プライオリティタスク=1 個, ローテーションタスク=2 個)で各種サービスコールを発行した場合の状態遷移を示しています。

【解説】

① タスク 1 は、プライオリティタスクです。プライオリティタスクは優先度が高いタスクで、待ち要因が解除されるとすぐ Running 状態となります。

② タスク 2,3 は、ローテーションタスクです。お互いが実行可能な状態(Running, Ready 状態)のとき、vrot_rdq サービスコールを発行することでタスクの実行権が変更されます。

③ Waiting 状態のタスクに sus_tsk サービスコールを発行すると Waiting + Suspended 状態となります。両方の待ち要因が解除されることで Ready 状態となります。

④ 実行可能な状態(Running, Ready 状態)のタスクがない場合は、アイドル状態となります。

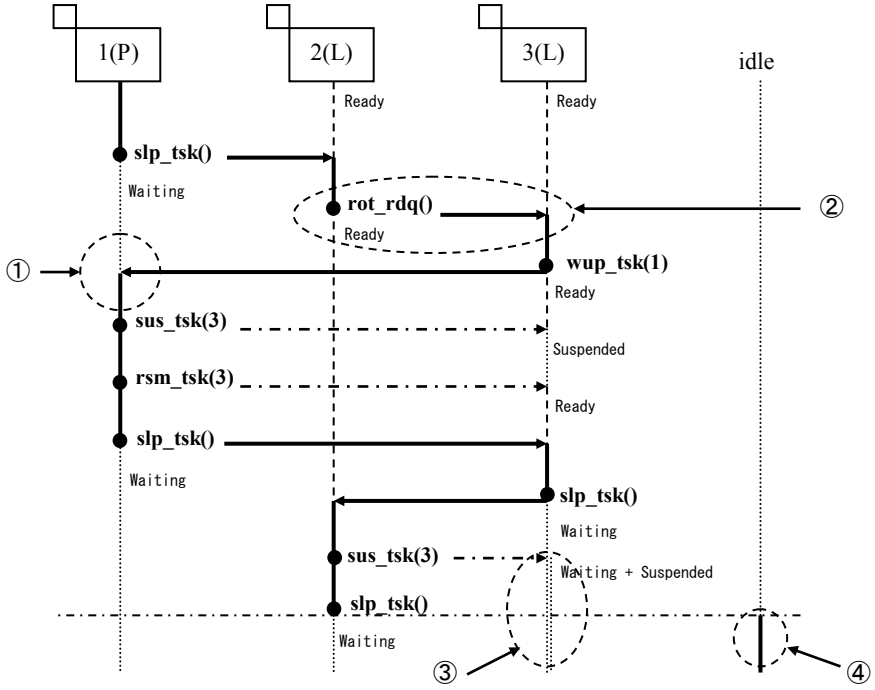

図3-4 タスクスケジュールの例

各サービスコールは基本的にエラーチェックを一切行いません。割込み禁止状態のタスクがタスクスイッチ型サービスコール(P28参照)を発行することでタスクの切替えが発生します。割込みを抑止(保留)したい場合にはタスクスイッチ型サービスコールを発行しないでください。

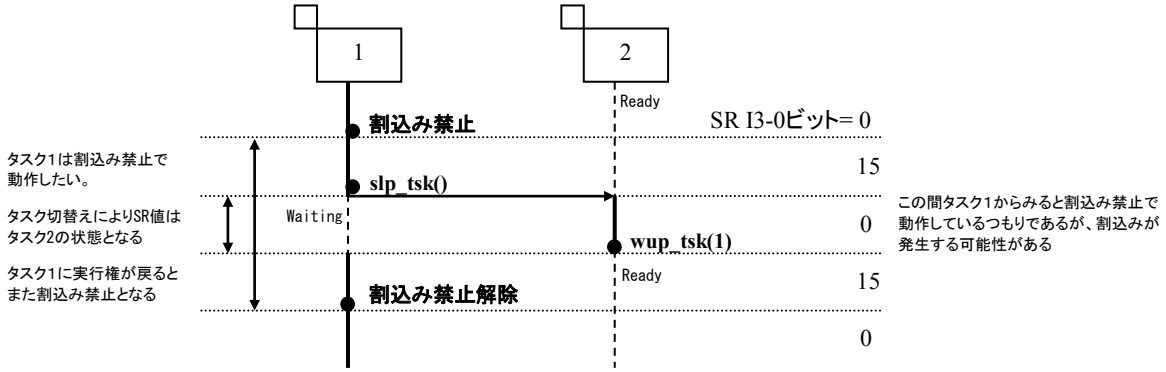

図3-5 割込み禁止状態でのタスク切替え

3.3.3　タスクディスパッチ禁止状態

本システムは、ディスパッチ禁止状態かディスパッチ許可状態のいずれかの状態をとります。タスクの実行開始直後は、ディスパッチ許可状態となっています。ディスパッチ禁止状態では、ディスパッチは起こりません。
ディスパッチ禁止状態では、タスク部から呼び出せるサービスコールに次の制限があります。

・ディスパッチ禁止で自タスクを広義の意味で待ち状態にする可能性のあるサービスコールは
呼び出してはならない。

ディスパッチ禁止状態での呼び出し禁止サービスコールは以下の通りです。
・slp_tsk、tslp_tsk(tim=TMO_POL 以外)
・wai_flg、twai_flg(tim=TMO_POL 以外)
・wai_sem、twai_sem(tim=TMO_POL 以外)
・rcv_dtq、trcv_dtq(tim=TMO_POL 以外)
・snd_dtq、tsnd_dtq(tim=TMO_POL 以外)

※　本システムでのサービスコールは Smalight のプログラムサイズを抑える工夫のひとつとして状態に対するのエラーチェックを行っておりません。許可されていない不正な状態でサービルコールを呼び出すことで、動作結果が不定となる場合があります。

3.4　タスク間同期・通信と排他制御

3.4.1　イベントフラグ

イベントフラグは、タスク間同期制御のひとつで少量のデータ通信機能としても利用できます。

イベントフラグを使用することにより、複数の事象発生の組み合せによるタスク間同期処理を行うことが出来ます。イベントフラグは、事象に対応したビットの集合体で、1つの事象発生を1ビットで表し、フラグのオン(1)/オフ(0)で行います。Smalight OS のイベントフラグは 32 ビットです。

(1) 基本動作

事象(イベント)発生を待つタスクは事象に対応したビットを指定してイベントフラグで待ちます。事象発生を通知するタスク(または割込みハンドラ)は事象に対応したビットをイベントフラグに設定することで、イベント待ちのタスクの待ちを解除します。

(2) イベントフラグの待ち条件

イベントフラグの待ち解除条件は待ち実施時（wai_flg,twai_flg コール時）に引数 wfmode として指定します。指定したビットのいずれかが発生するまで待つ TWF_ORW の指定 (OR 条件)、指定したビットの全てが発生するまで待つ TWF_ANDW の指定(AND 条件)があります。

イベントフラグ待ちが解除されたときは、待ち実施時（wai_flg,twai_flgコール時）に引数 p_ptn で指定した格納領域へ解除時のビットパターンが格納されます。OR 条件で解除された場合、待ち解除された要因(ビット)で確認できます。

(3) イベントフラグ待ちの待ち行列（待ちキュー）設定

イベントフラグで複数のタスクが同時に Waiting 状態(イベント待ち)となります。そのとき、タスクはイベントフラグの待ち行列に入ります。

待ち行列に入るとき、VEVFLG_TA_TFIFO(FIFO 順)属性、または、 VEVFLG_TA_TPRI(優先度順)属性に従ってキューイングされます。この属性で待ち解除されるタスクの順番が決定されます。

(4) 待ち解除時のイベントフラグクリア属性

VEVFLG_TA_CLR 属性(クリア属性)を設定したとき、待ち解除のタイミングで該当するイベントフラグの全てのビットを 0 クリアします。

クリア属性によりイベントフラグを使用した動作に違いがあります。同じイベントフラグで複数のタスクが待っているとき、クリア属性が設定されていなければ、待ち解除の条件を満たす全てのタスクが待ち解除されます。

クリア属性が設定されていれば、待ち行列をサーチして待ち解除の条件を満たすタスクを見つけた時点で、イベントフラグがクリアされます。他に条件が満たすタスクがあっても待ち解除されません。

以下にイベントフラグの使用例を示します。例は、クリア属性なし、タスク優先度はタスク 1 が高い場合を想定しています。

【解説】

① タスク 1 は clr_flg サービスコールを発行し、該当するフラグをクリア(0)します。
② タスク 1 は wai_flg（待ちパターン=1）サービスコールを発行し、指定したフラグがセットされてないので、フラグがセットされるのを待ちます。タスク 1 は waiting 状態になります。
③ タスク 2 で set_flg サービスコールを発行し、フラグを設定します。これにより待ち条件を満たしたタスク 1 は Ready 状態となり、Waiting 状態から解除されます。クリア属性なしなので、イベントフラグは 1 のままとなります。

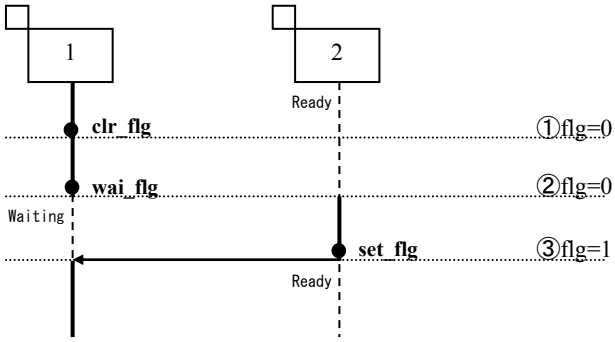

図3-6 イベントフラグの使用例

イベントフラグを利用して、32 ビットのデータ送信が可能です。以下に使用例を示します。例は、クリア属性あり、イベントフラグのチェック条件を OR 条件とします。タスク 1 は、待ちパターン 0xFFFFFFFF でイベントフラグ待ちに入ります。タスク 2 で、送信したいデータ(0x12345678)をイベントフラグ設定します。

イベントフラグのチェック条件が OR 条件なので、1 ビットでも ON になれば、待ち解除されることを利用したデータ送信となります。

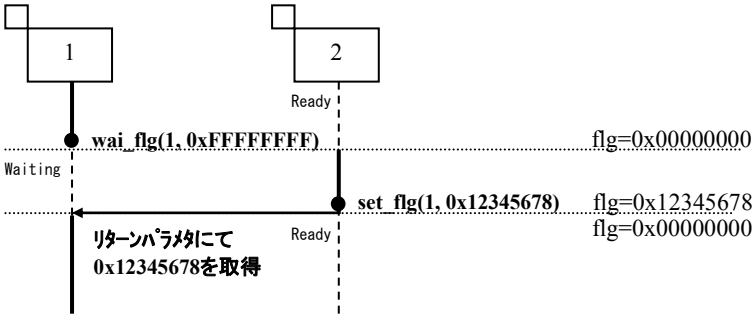

図3-7 イベントフラグを利用したデータ送信

3.4.2　セマフォ

セマフォは、資源(各種 I/O,共有メモリ、非リエントラント関数など)の排他制御に使用します。

複数のタスクから同時に使用することができない資源を複数のタスクで共有する場合に使用します。使用方法は、資源に対応するセマフォを用意し、資源を使用する前にセマフォを獲得し、資源の使用が終わったらセマフォを返却します。セマフォが獲得できない場合、他タスクで使用中と判断し、セマフォが返却されるのを待ちます。

(1) 基本動作

セマフォを獲得するとセマフォカウント(セマフォ資源数:資源の並列獲得可能な数)をデクリメント(-1)します。セマフォの獲得ではセマフォカウントが 1 以上のとき、Waiting 状態(セマフォ待ち)にならずにセマフォを獲得できます。セマフォカウントが 0 のとき、Waiting 状態(セマフォ待ち)に入ります。セマフォを返却すると、セマフォカウントをインクリメント(+1)します。

(2) セマフォ資源数の初期値

セマフォ資源数とは、セマフォで排他制御する資源の並行して獲得可能な数です。セマフォ資源数が 2 であれば、2 つのタスクが同時に資源を使用することができます。

(3) セマフォの待ち行列(待ちキュー)設定

セマフォの獲得で複数のタスクが同時に Waiting 状態(セマフォ待ち)となることがあります。そのとき、タスクはセマフォ待ちの待ち行列に入ります。

待ち行列に入るとき、VSEM_TA_TFIFO(FIFO 順)属性、または、 VSEM_TA_TPRI(優先度順)属性に従ってキューイングされます。この属性でセマフォを獲得するタスクの順番が決定されます。

以下にセマフォの使用例を示します。例は、セマフォ資源数の初期値は 1、タスク優先度はタスク 1 が高い場合を想定しています。

【解説】

① タスク 2 は wai_sem サービスコールを発行しセマフォを獲得します。資源数(semcnt)は 0 となります。
② タスク 1 は wai_sem サービスコールを発行しますが、資源数(semcnt)は 0 のため Waiting 状態になり、セマフォが返却されるのを待ちます。
③ タスク 2 で sig_sem サービスコールを発行しセマフォを返却します。これによりタスク 1 はセマフォを獲得し Ready 状態となります。タスク優先度によりタスク 1 が実行(Running 状態)されます。
④ タスク 1 は sig_sem サービスコールを発行しセマフォを返却します。資源数(semcnt)は 1 となります。

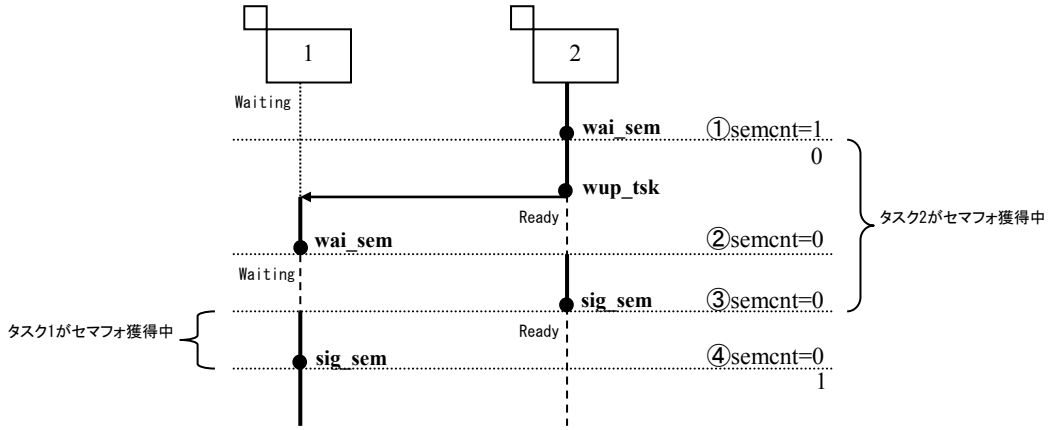

図3-8 セマフォの使用例

3.4.3　データキュー

データキューはタスク間データ通信に使用します。

データキューは 1 ワードのデータ転送を行います。データ送信するとデータキューにデータが送られます。データ受信はデータキューからデータを受信します。データキューはリングバッファで FIFO(First In First Out)で管理されます。Smalight OS のデータキューは 32 ビット / ワードで実装されます。

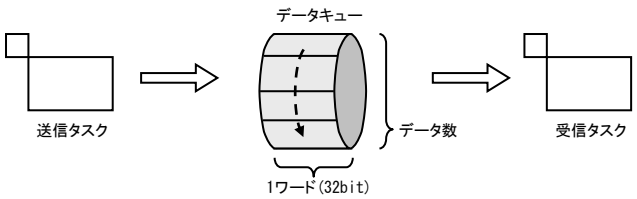

図3-9 データキューの概念

(1) 基本動作

データ送信時、データキューの空きがあればデータキューにデータを設定します。データキューの空きがない場合、タスクはデータキューに空きがでるまで Waiting 状態(データキュー待ち)となります。

データ受信時、データキューにデータが存在すればデータキューからデータを取り出します。受信すべきデータがデータキューに存在していない場合は Waiting 状態(データキュー待ち)となります。

(2) データキュー送信待ちの待ち行列（待ちキュー）設定

データキューへの送信で複数のタスクが同時に Waiting 状態(データキュー送信待ち)となることがあります。そのとき、タスクはデータキュー送信待ちの待ち行列(待ちキュー)に入ります。

待ち行列に入るとき、VDTQ_TA_TFIFO(FIFO 順)属性、または、 VDTQ_TA_TPRI(優先度順)属性に従ってキューイングされます。この属性で待ち解除されるタスクの順番が決定されます。

※ 尚、データキュー受信待ちの待ちキューは、常に FIFO 順でキューイングします。

(3) データ数

データキューを構成するリングバッファは、データ数 0～127 個の値を任意に設定できます。データ数が 0 個の場合、送受信すると送信側と受信側、双方が揃うまでタスクは Waiting 状態に入ります。データ数を多くすることで、データ送信時、Waiting 状態に入る可能性が低くなると言えます。

以下にデータキューの使用例を示します。例は、データ数は 1、タスク優先度は高い順にタスク 1、タスク 2、タスク 3 となります。

【解説】

① タスク 1 は snd_dtq サービスコールを発行しデータ H'11111111 を送信します。データキューの空きが 1 のため、データ送信が完了します。データキューの空きは 0 となります。

② タスク 1 は slp_tsk サービスコールを発行し Waiting 状態となります。Ready 状態のタスク 2 が実行(Running 状態)されます。

③ タスク 2 は snd_dtq サービスコールを発行しデータ H'22222222 を送信します。データキューの空きが 0 のため、データ送信されず Waiting 状態となり、データキューへの送信完了を待ちます。Ready 状態のタスク 3 が実行(Running 状態)されます。

④ タスク 3 は rcv_dtq サービスコールを発行し、データキューからデータ H'11111111 を受信します。データキューに空きが発生したので、データキューへの送信待ちで Waiting 状態のタスク 2 が送信完了し、実行(Running 状態)されます。タスク 3 は Ready 状態です。

⑤ タスク 2 は slp_tsk サービスコールを発行し Waiting 状態となります。Ready 状態のタスク 3 が実行(Running 状態)されます。

⑥ タスク 3 は rcv_dtq サービスコールを発行し、データキューからデータ H'22222222 を受信します。データキューの空きは 0 となります。

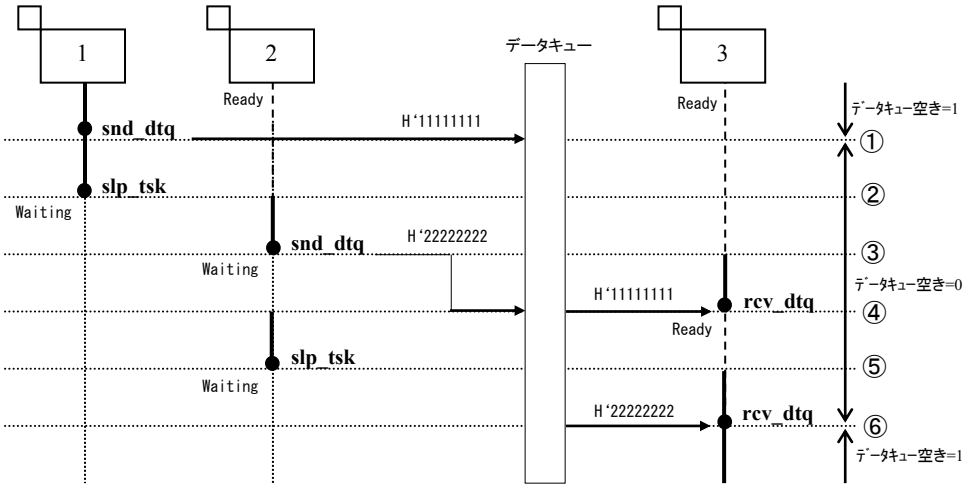

図3-10 データキューの使用例

3.5　時間管理

時間管理として、以下の機能を提供します。

・タイムアウト付きサービスコール(tslp_tsk, twai_flg, twai_sem, trcv_dtq, tsnd_dtq)
・システム時刻の設定・参照(set_tim, get_tim)
・周期ハンドラ (vcyc_init, sta_cyc, stp_cyc)

3.5.1　周期タイマハンドラ

時間管理の 3 つの機能を実現するために、周期的なタイマ割込みハンドラを用意する必要があります。本割込みハンドラを周期タイマハンドラといいます。周期タイマハンドラから、ivsig_tim サービスコールを発行することで時間管理を実現します。

3.5.2　システム時刻

システム時刻とは、符号無し 48bit のカウンタで、周期タイマハンドラによって更新されます。

サービスコール(tslp_tsk 等)のタイムアウト時間パラメータや周期ハンドラの周期時間の単位は 1msec ですが、周期タイマハンドラの周期時間は任意に 1msec 以上の値を設定できます。詳細は「6.4.3　周期タイマハンドラ周期時間の設定」を参照ください。

図 3-11に周期時間を 250msec に設定した場合の tslp_tsk(1)と tslp_tsk(500)で、タイムアウトにより Waiting 状態から Ready 状態に遷移する様子を示します。周期タイマハンドラの周期時間は小さい方が誤差は小さくなりますが、システムトータルで考えると周期タイマハンドラの動作回数が増えることでシステム全体のパフォーマンスが劣化する場合がありますので注意が必要です。

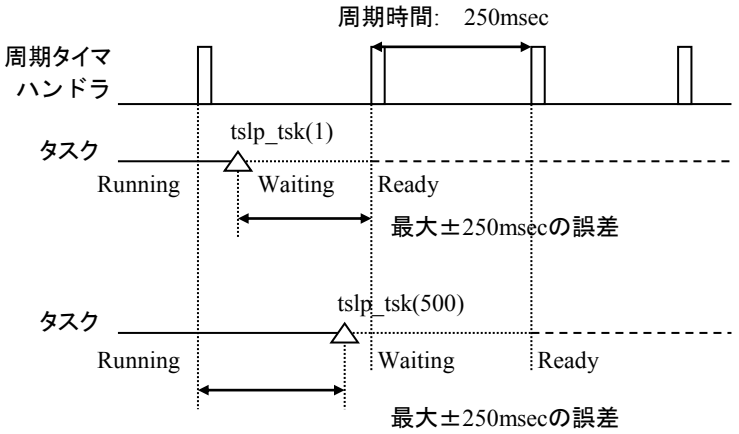

図3-11 時間管理の注意事項

3.5.3　　周期ハンドラ

周期的に実行される処理を呼び出す機能を周期ハンドラといいます。 周期ハンドラの開始から停止までの間、指定した起動周期で、周期ハンドラが呼び出されます。

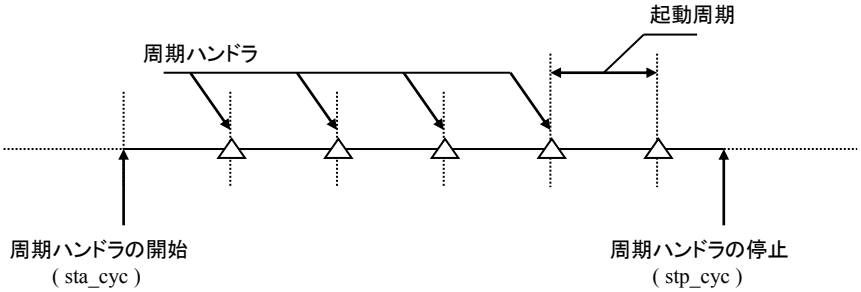

図3-12 周期ハンドラの起動タイミング

周期ハンドラの初期化、および、属性の設定（起動周期、周期ハンドラ関数の指定）は、システム初期化処理で行います。属性の設定で VCYC_TA_STA 属性（初期状態で周期ハンドラ動作が開始）を指定すると、周期ハンドラの動作が開始した状態となります。

周期ハンドラは、周期タイマハンドラ（ユーザ任意の周期割込）で呼び出す ivsig_tim サービスコールの延長で呼び出されます。それにより割込みマスクレベルは、カーネルマスクレベルで動作します。周期ハンドラを呼び出す時には、周期タイマハンドラの割込みレベルに設定してコールされます(本文中に記載されるカーネルマスクレベルの詳細は「6.4.1　カーネルマスクレベルの設定」を参照ください)。

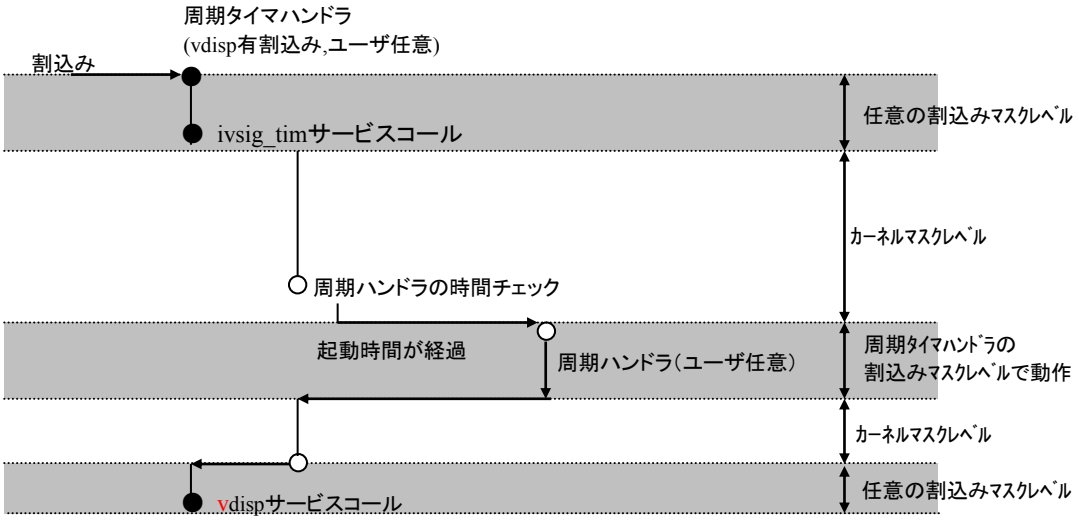

図3-13 周期ハンドラの呼び出し

3.5.4 時間管理に関する注意事項

(1) 時間管理機能のオーバーヘッドについて

周期タイマハンドラ(ユーザ任意)にてコールする ivsig_tim サービスコールにて、以下の処理が実行されます。

- システム時刻の更新
- 登録された周期ハンドラの管理、及び、起動周期を経過した全ての周期ハンドラの起動と実行
- t 付きサービスコールにて待ち状態タスクのタイムアウト処理

※ 構築や、使用するサービスコールによって、上記処理が発生しない場合もあります

これらの処理はカーネルマスクレベル、または、使用する周期タイマハンドラの割込みレベルにて実行されるため、ivsig_tim サービスコールの実行時間が長くなるとシステム全体のパフォーマンスが劣化、システム時刻の遅延等の問題が起こる可能性があります。例えば、周期タイマハンドラ(ユーザ任意)の周期時間が 1msec とき、周期起動が 1msec の周期ハンドラが起動され、ハンドラ処理時間が 1msec 以上の実行時間を要した場合、永久にハンドラ処理が繰り返し実行され、他のタスク処理が実行されない事態となる可能性もあります。

これらの問題を回避するため、以下の点について注意が必要です。

- 周期ハンドラは可能な限り短く記述してください。
- 登録する周期ハンドラを極力少なくしてください。
- 周期ハンドラの周期時間、t 付きサービスコールにて指定するタイムアウト時間はなるべく大きな値を指定してください。
- 周期タイマハンドラ(ユーザ任意)の周期は、なるべく大きな値としてください。

3.6　割込みハンドラ

本 OS では割込みハンドラを、以下の 2 種類に分類しています。この分類により作成方法が異なります。表中に記載されるカーネルマスクレベルの詳細は「6.4.1　カーネルマスクレベルの設定」を参照ください。

表3-4 割込みハンドラの定義

割込み区分	説明
vdisp無割込みハンドラ	OSサービスコールを発行しない、割込み発生元へ戻る割り込み。 なお、以下の割り込みは、必ずvdisp無割込みハンドラとする必要があります。 1．NMI割り込み（FIQモードとなる割り込み） 2．カーネルマスクレベルを超えるレベルの割込み
vdisp有割込みハンドラ	OSサービスコールを発行してタスク制御やディスパッチを行うための割込み。カーネルマスクレベル以下のレベルの割込み。 割込みの最後でvdispサービスコールを発行します。vdispサービスコールを発行することにより割込み発生元には戻らず、ディスパッチャへ制御を遷します。 vdisp有割込みハンドラの割り込みとして使用可能なのは、カーネルマスクレベル以下のCPUコアがIRQモードとなる割り込みのみです。※

※　CPU コアが IRQ モードとなる割り込みは、NMI 以外の割り込みコントローラによる割り込みです。

3.6.1　vdisp 無割込みハンドラ

vdisp 無割込みハンドラでは、OS サービスコールを発行してはいけません。vdisp 無割込みハンドラは割込み発生元に戻るため、必要最小限のレジスタ退避のみで処理することができます。以下に vdisp 無割込みハンドラの基本動作を示します。vdisp 無割込みハンドラでサービスコールを発行した場合の動作は保証されません。

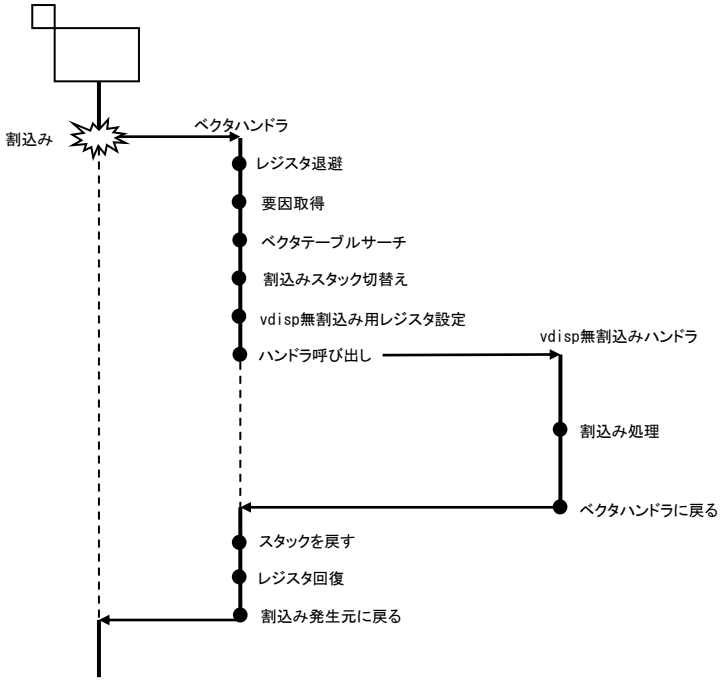

図3-14 vdisp無割込みハンドラの基本動作

3.6.2 vdisp 有割込みハンドラ

vdisp 有割込みハンドラは割込み発生元に戻らず、ディスパッチャへ制御を遷すことができます。以下に vdisp 有割込みハンドラの基本動作を示します。

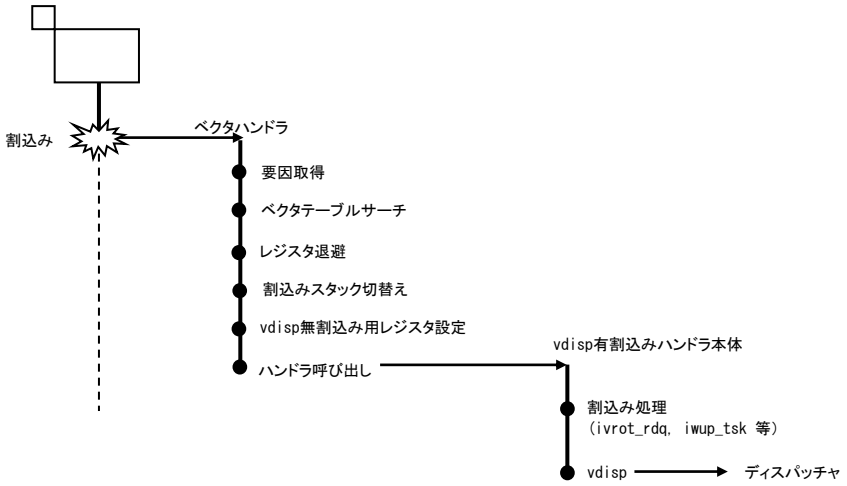

図3-15 vdisp有割込みハンドラの基本動作

3.6.3　　多重割込み

多重割込みの動作例を以下に示します。

【解説】

① タスクは基本的に割込み非マスク状態で動作させてください。割込みマスク状態で実行した場合、vdisp 有割込みが発生しても、ディスパッチされず割込み発生元に戻ります。タスク 1,2 はローテーションタスク (同一のタスク優先度)です。

② 割込みコントロール下位レベルの vdisp 有割込み実行中に、割込みコントロール上位レベルの vdisp 有割込みが発生したケースです。vdisp サービスコールを発行しても多重割込み中は、割込み発生元に戻ります。

③ NMI 割込みは IRQ モードとなる割込みではありません。IRQ モードとなる割込み以外で OS サービスコールは発行してはいけません。

④ 割込みコントロール下位レベルの vdisp 有割込みです。vdisp サービスコールを発行することで割込み発行元には戻らず、ディスパッチャへ制御を遷します。

⑤ 割込みコントロール上位レベルの vdisp 有割込みです。多重割込み中でないので、割込み発行元には戻らず、ディスパッチャへ制御を遷します。

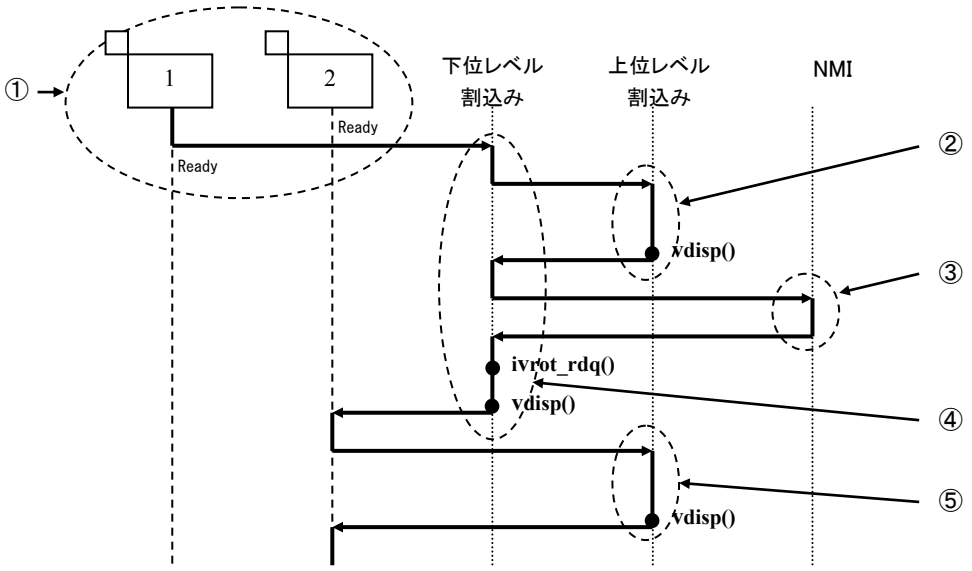

図3-16 多重割込み動作例

図 3-17に示す様に多重割込みで下位レベルの vdisp 無割込みの場合、上位レベル割込みにて発行したサービスコールによる状態変更が反映されず、OS 管理情報が不正となる可能性が生じます。この場合、以後の動作は保証されません。

多重割込みを前提に割込みからサービスコールを発行する場合、カーネルマスクレベル以下の割込みは vdisp 有割込みとして実装してください(図 3-18)。

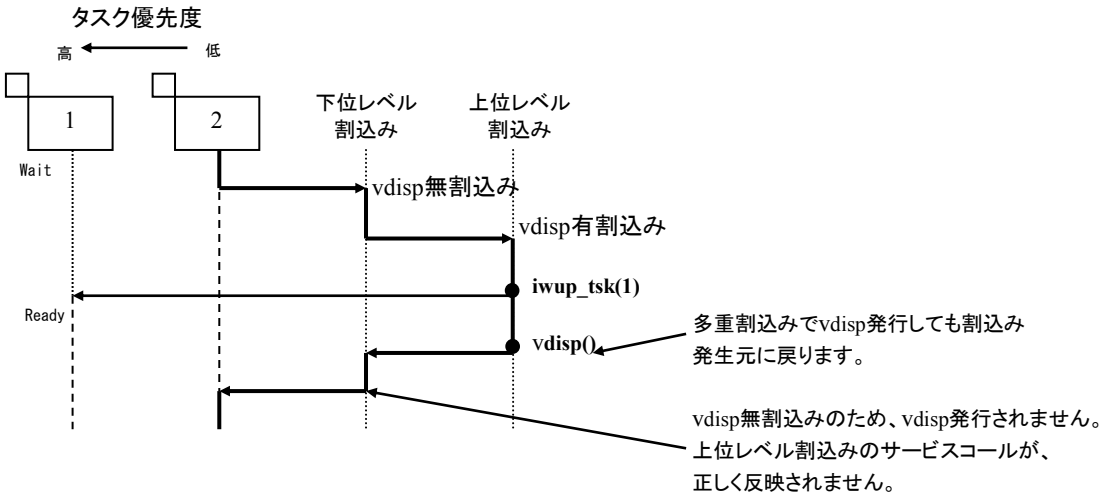

図3-17 多重割込み実装上の注意事項①

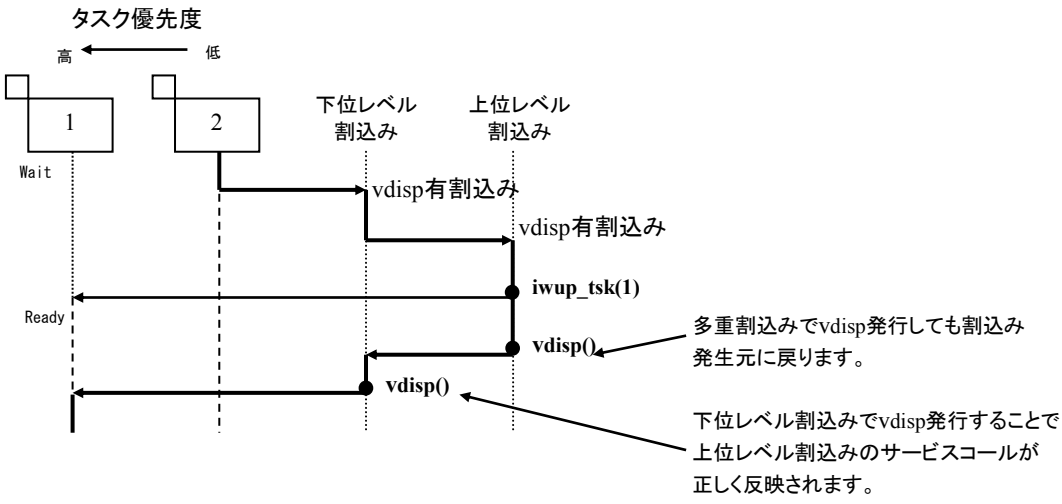

図3-18 多重割込み実装上の注意事項②

4. サービスコール、コールバック

4.1 システム状態とサービスコール

各サービスコールと発行可能なシステム状態の一覧を示します。
発行可能なシステム状態以外でサービスコールを発行した場合の動作は、保証されません。

表4-1 システム状態とサービスコール①

T:タスク部 E:ディスパッチ許可 D:ディスパッチ禁止 I:割込み部 O:システム初期化部
〇:発行可能、△:条件により発行可能、×:発行不可

区分	サービスコール	説明	発行可能なシステム状態				
			T	E	D	I	O
タスク管理	sta_tsk	タスクの開始	〇	〇	〇	×	×
	ext_tsk	自タスクの終了	〇	〇	×	×	×
	slp_tsk	タスクの起床待ち	〇	〇	×	×	×
	tslp_tsk	タスクの起床待ち(タイムアウト付き)	〇	〇	△	×	×
	wup_tsk	タスクの起床	〇	〇	〇	×	×
	iwup_tsk	タスクの起床	×	×	×	〇	×
	can_wup	タスク起床要求のキャンセル	〇	〇	〇	×	×
	ican_wup	タスク起床要求のキャンセル	×	×	×	〇	×
	vrot_rdq	タスクのローテーション	〇	〇	〇	×	×
	ivrot_rdq	タスクのローテーション	×	×	×	〇	×
	sus_tsk	他タスクのサスペンド	〇	〇	〇	×	×
	isus_tsk	他タスクのサスペンド	×	×	×	〇	×
	rsm_tsk	サスペンドの解除	〇	〇	〇	×	×
	irsm_tsk	サスペンドの解除	×	×	×	〇	×
	dis_dsp	ディスパッチの禁止	〇	〇	×	×	×
	ena_dsp	ディスパッチの許可	〇	×	〇	×	×
イベントフラグ	wai_flg	イベントフラグ待ち	〇	〇	×	×	×
	twai_flg	イベントフラグ待ち(タイムアウト付)	〇	〇	△	×	×
	set_flg	イベントフラグの設定	〇	〇	〇	×	×
	iset_flg	イベントフラグの設定	×	×	×	〇	×
	clr_flg	イベントフラグのクリア	〇	〇	〇	×	×
	iclr_flg	イベントフラグのクリア	×	×	×	〇	×
	vevtflg_init	イベントフラグの初期化	×	×	×	×	〇
	VEVTFLG_ATTR(*1)	イベントフラグの属性設定	×	×	×	×	〇
セマフォ	wai_sem	セマフォの獲得	〇	〇	〇	×	×
	twai_sem	セマフォの獲得(タイムアウト付)	〇	〇	△	×	×
	sig_sem	セマフォの返却	〇	〇	〇	×	×
	isig_sem	セマフォの返却	×	×	×	〇	×
	vsem_init	セマフォの初期化	×	×	×	×	〇
	VSEM_ATTR(*1)	セマフォの属性設定	×	×	×	×	〇

※ 小文字のサービスコールはサブルーチンコールです。
(*1) マクロです。

表4-2 システム状態とサービスコール②

T:タスク部 E:ディスパッチ許可 D:ディスパッチ禁止 I:割込み部 O:システム初期化部
○:発行可能、△:条件により発行可能、×:発行不可

区分	サービスコール	説明	発行可能なシステム状態				
			T	E	D	I	O
データキュー	rcv_dtq	データキューからの受信	○	○	×	×	×
	trcv_dtq	データキューからの受信(タイムアウト付)	○	○	△	×	×
	snd_dtq	データキューへの送信	○	○	×	×	×
	isnd_dtq	データキューへの送信	×	×	×	○	×
	tsnd_dtq	データキューへの送信(タイムアウト付)	○	○	△	×	×
	fsnd_dtq	データキューへの強制送信	○	○	○	×	×
	ifsnd_dtq	データキューへの強制送信	×	×	×	○	×
	vdtq_init	データキューの初期化	×	×	×	×	○
	VDTQ_ATTR(*1)	データキューの属性設定	×	×	×	×	○
時間管理	set_tim	システム時刻の設定	○	○	○	×	○
	iset_tim	システム時刻の設定	×	×	×	○	○
	get_tim	システム時刻の取得	○	○	○	×	○
	iget_tim	システム時刻の取得	×	×	×	○	○
	systim_init	時間管理の初期化	×	×	×	×	○
	ivsig_tim	タイムティックの供給(周期タイマ処理)	×	×	×	○	×
周期ハンドラ	sta_cyc	周期ハンドラの開始	○	○	○	×	×
	ista_cyc	周期ハンドラの開始	×	×	×	○	×
	stp_cyc	周期ハンドラの停止	○	○	○	×	×
	istp_cyc	周期ハンドラの停止	×	×	×	○	×
	vcyc_init	周期ハンドラの初期化	×	×	×	×	○
	VCYC_ATTR(*1)	周期ハンドラの属性設定	×	×	×	×	○
	VCYC_CHG(*1)	周期ハンドラの起動周期変更	○	○	○	○	○
割込み	vdisp	vdisp有割込みハンドラをディスパッチして終了	×	×	×	○	×
初期化	vslos_init	OSの起動	×	×	×	×	○

(*1) マクロです。

4.2 コールバック

OSが呼び出すコールバックルーチンの一覧を示します。
コールバックはユーザ作成ルーチンです。

表4-3 コールバックルーチン

区分	コールバック ルーチン名	説明	備考
周期ハンドラ	callback_cychdr	周期ハンドラ本体	名称はユーザ任意です
割込み	callback_int	割込みハンドラ本体	
その他初期化	kinit	OS初期化処理	名称は固定です
	uinit	ユーザ初期化処理	
	stack_init	タスクスタックの初期化処理	
その他	idle	アイドル処理	

4.3　各サービスコールの動作

4.3.1　サービスコール動作別の分類

各サービスコールを動作別に分類すると以下の様になります。以下、本分類に沿って各サービスコールの基本動作について説明します。

- タスクスイッチ型サービスコール
- i付きサービスコール
- 関数型サービスコール

4.3.2　タスクスイッチ型サービスコールの基本動作

タスクスイッチ型サービスコールは、タスク部から発行するとタスクスイッチする可能性があるサービスコールです。タスクスイッチ型サービスコールは以下の通りです。

表4-4 タスクスイッチ型サービスコール

タスクスイッチ型 サービスコール	sta_tsk, ext_tsk, ena_dsp, slp_tsk, tslp_tsk, wup_tsk, vrot_rdq, sus_tsk, rsm_tsk, wai_flg, twai_flg, set_flg, wai_sem, twai_sem, sig_sem, rcv_dtq, trcv_dtq, snd_dtq, tsnd_dtq, fsnd_dtq

タスクスイッチ型サービスコールの基本動作を説明します。

① タスクスタックにレジスタを退避します。
② カーネルマスクレベルに変更します。カーネルマスクレベルはコンフィギュレーションファイルの設定にて決定します。
③ スタックを OS スタックに切替えます。
④ 各サービスコールの処理を実行します。
⑤ 一時的にマスクレベルを解放します。これにより一時的に全ての保留中の割込みが実行されます。これは割込み禁止時間短縮のため、行われます。
⑥ ディスパッチャを実行します。ディスパッチャとは優先度やタスク状態に従い、実行するタスクを選択してタスクへ処理を遷移します。実行できるタスクがない場合は、アイドル状態となります。

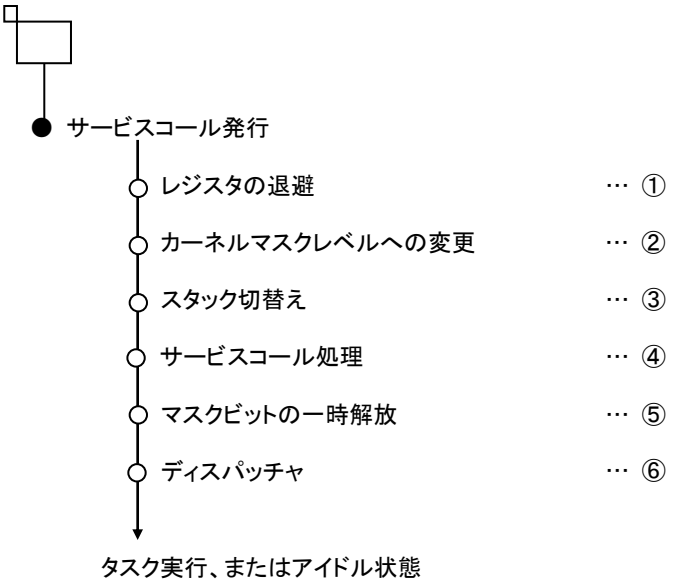

図4-1 タスクスイッチ型サービスコールの基本動作

4.3.3　i 付きサービスコールの基本動作

i付きサービスコールは、割込み部から発行するサービスコールです。i付きサービスコールは以下の通りです。

表4-5 i付きサービスコール

i付きサービスコール	iwup_tsk, ivrot_rdq, isus_tsk, irsm_tsk, iset_flg, isig_sem, isnd_dtq, ifsnd_dtq, ivsig_tim

i付きサービスコールの基本動作を説明します。

① カーネルマスクレベルに変更します。カーネルマスクレベルはコンフィギュレーションファイルの設定にて決定します。
② 各サービスコールの処理を実行します。
③ ①で変更したマスクレベルを元に戻します。
④ 発行元へ戻ります。
⑤ i 付きサービスコールではディスパッチャが実行されずに発行元へ戻ります。vdisp 有割込みハンドラの最後で必ず vdisp サービスコールを発行してください。

i付きサービスコールはタスクスイッチ型サービスコールと比較すると、ディスパッチャが実行されない等の違いがあります。

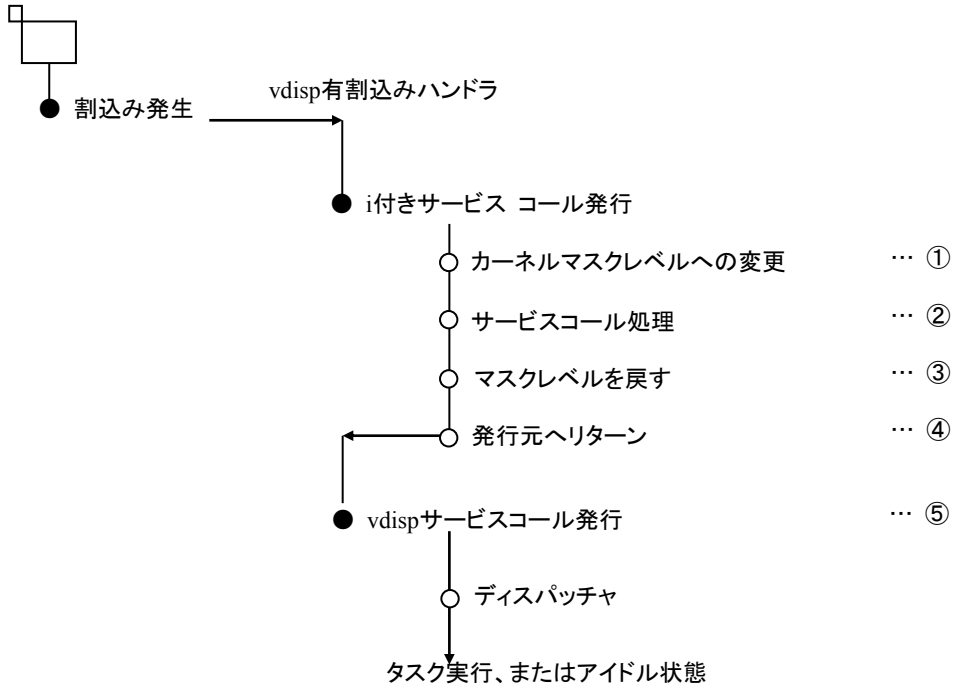

図4-2 i付きサービスコールの基本動作

4.3.4　関数型サービスコールの基本動作

タスクスイッチ型サービスコール、i付きサービスコールに分類されない関数型サービスコールは、通常の関数と同様の動作をします。

表4-6 関数型サービスコール

関数型 サービスコール	can_wup, ican_wup, dis_dsp, clr_flg, iclr_flg, set_tim, iset_tim, get_tim, iget_tim, sta_cyc, ista_cyc, stp_cyc, istp_cyc, vevtflg_init, vsem_init, vdtq_init, vsystim_init, vcyc_init

4.4　サービスコールの説明形式

本節では、サービスコール仕様についての詳細な説明を以下の形式で行っています。

No.　サービスコール名　　機能

【発行可能なシステム状態】

C言語インタフェース

マネージャコール呼出し形式

パラメータ

型	パラメータ	パラメータの意味
・	・	・
・	・	・
・	・	・

リターンパラメータ

型	パラメータ	パラメータの意味
・	・	・
・	・	・

リターン値／エラーコード

リターン値またはニモニック	リターン値またはエラーコードの意味
・	・
・	・
・	・

解　説

・・・・・・・・・・・・

4.5　サービスコールの基本的な振舞い

Smalight の各種サービスコールは、プログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

また、サービスコール発行に伴うシステムまたはタスクの状態に対するエラーチェックを一部を除き行っておりません。システムまたはタスクの状態がサービスコールを発行できる状態でない条件で、サービスコールを発行した場合、動作結果が不定となる場合があります。

4.6　タスク管理

4.6.1　sta_tsk　　タスクの開始

【タスク部】

C言語インタフェース

ER　ercd　=　sta_tsk (ID tid, VP_INT stacd);

パラメータ

ID	tid	起動対象のタスクのID番号
VP_INT	stacd	タスクの起動コード

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了
E_OBJ(H'FFFFFFD7= -47)	対象タスクが休止状態でない

解　説

起動対象タスクをDormant（休止状態）からReady（実行可能状態）に移行します。
パラメータstacdは起動対象タスクの起動時パラメータとして設定します。
起動対象タスクがDormant（休止状態）でない場合は、E_OBJが返ります。

本サービスコールはタスク部のみ発行できます。

パラメータtidにはDormant（休止状態）タスクのタスクIDを指定してください。tidに0(アイドル)や、存在しないタスクのタスクIDを指定しないでください。

4.6.2 ext_tsk 自タスクの終了

【タスク部】

C言語インタフェース

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

パラメータ

無し

リターンパラメータ

無し

解 説

自タスクをRunning（実行状態）からDormant（休止状態）へ移行します。

本サービスコールはタスク部のみ発行できます。

タスクが占有していたセマフォなどの資源を解放する機能はありません。事前に開放してください。
ディスパッチ禁止状態の解除も行われません。事前にディスパッチ許可状態としてください。

4.6.3 slp_tsk タスクの起床待ち

【タスク部】

C言語インタフェース

ER ercd = slp_tsk (void);

パラメータ

無し

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ① (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

自タスクをWaiting状態（起床待ち）に移行します。このタスクの起床待ちは、wup_tskサービスコールにより解除されます。wup_tskサービスコールによるタスク起床要求は255回まで記憶されます(起床要求回数という)。起床要求回数が1以上のとき 本サービスコールを発行した場合は、起床要求回数がデクリメント(-1)され、Waiting状態（起床待ち）には移行せず、そのまま実行を続けます。

リターンパラメータercdには必ずE_OK(0)が返ります。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では発行できません。

4.6.4 tslp_tsk タスクの起床待ち(タイムアウト付き)

【タスク部】

C言語インタフェース

ER ercd = tslp_tsk (TMO tim);

パラメータ

TMO	tim	タイムアウト時間(msec)

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ① (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了
E_TMOUT(H'FFFFFFCE = -50)	ポーリング失敗、または、タイムアウト

解 説

自タスクをタイムアウト有りで Waiting 状態（起床待ち）に移行します。このタスクの起床待ちはタイムアウト、または、wup_tsk サービスコールにより解除されます。wup_tsk サービスコールによるタスク起床要求は 255 回まで記憶されます(起床要求回数という)。起床要求回数が1以上のとき 本サービスコールを発行した場合は、起床要求回数がデクリメント(-1)され、Waiting 状態（起床待ち）には移行せず、そのまま実行を続けます。そのときのリターンパラメータ ercd は E_OK(0)が返ります。

パラメータ tim には、タイムアウト時間(msec)を設定します。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、Waiting 状態（起床待ち）には移行せず、リターンパラメータ ercd は E_OK(0)が返ります。tim に TMO_FEVR(-1)を設定した場合は slp_tsk と同じ動作となります。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態ではtim=TMO_POLのみ発行できます。

4.6.5 wup_tsk, iwup_tsk タスクの起床

【タスク部】wup_tsk
【割込み部】iwup_tsk

C言語インタフェース

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

パラメータ

ID	tid	タスクID

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

slp_tskサービスコール、または、tslp_tskサービスコールによりWaiting状態（起床待ち）になっているタスクの起床待ちを解除します。起床待ちでないタスクを指定した場合、タスク起床要求が記憶されます。タスク起床要求は255回まで記憶されます(起床要求回数という)。

パラメータtidには起床させたいタスクのタスクIDを指定してください。tidには存在しないタスクのタスクIDを指定しないでください。割込み部からiwup_tsk サービスコールを発行する場合、パラメータtidに0(アイドル)を指定しないでください。

Dormant状態のタスクIDを指定しないでください。

wup_tskサービスコールはタスク部から発行します。

iwup_tskサービスコールは割込み部から発行します。

4.6.6 can_wup タスク起床要求のキャンセル

【タスク部】can_wup
【割込み部】ican_wup

C言語インタフェース

```
ER_UINT ercd = can_wup ( ID tid );
ER_UINT ercd = ican_wup ( ID tid );
```

パラメータ

ID	tid	タスクID

リターンパラメータ

ER_UINT	ercd	リターンコード

リターン値／エラーコード

ercd | ① | (4 バイト)

① リターンコード

キャンセルした起床要求回数(0 以上の値となる)	正常終了

解 説

記憶されたタスクの起床要求をキャンセルし、キャンセルした起床要求回数を返します。起床要求は255回まで記憶されます(起床要求回数という)。

パラメータtidには起床要求をキャンセルしたいタスクのタスクIDを指定してください。tidに0(アイドル)をした場合、自タスク対象タスクとします。tidには存在しないタスクのタスクIDを指定しないでください。割込み部からican_wupサービスコールを発行する場合、パラメータtidに0(アイドル)を指定しないでください。

Dormant状態のタスクIDを指定しないでください。

リターンパラメータ ercd にはリターンコード、及び、キャンセルした起床要求の回数が返ります。正常終了時のみ、キャンセルした起床要求の回数が返ります。起床要求が記憶されていない場合は0となります。

can_wupサービスコールはタスク部から発行します。

ican_wupサービスコールは割込み部から発行します。

4.6.7 vrot_rdq, ivrot_rdq タスクのローテーション

【タスク部】vrot_rdq
【割込み部】ivrot_rdq

C言語インタフェース

```
void    vrot_rdq ( void );
void    ivrot_rdq ( void );
```

パラメータ

無し

リターンパラメータ

無し

解 説

ローテーションタスクのレディキューはFCFS(First Come First Service)で管理しています。本サービスコール発行によりローテーションタスクのレディキュー先頭タスクをレディキュー最後尾につなぎかえます。

ローテーションタスクのレディキューは、実行中(Running状態)のローテーションタスクがWaiting状態になるか、本サービスコールを発行しない限り、他のローテーションタスクが実行されません。本サービスコールを用いて、意図的にローテーションタスクのレディキューを操作する必要があります。

本サービスコールをプライオリティタスクが実行中(Running状態)に発行した場合でも、ローテーションタスクのレディキュー操作を行います。そのときローテーションタスクのレディキュー先頭にいたタスクは無条件にレディーキュー最後尾に移動されますので、実行されないまま先頭から最後尾に移動される可能性があります。

vrot_rdqサービスコールはタスク部から発行します。

ivrot_rdqサービスコールは割込み部から発行します。

4.6.8 sus_tsk, isus_tsk 他タスクのサスペンド

【タスク部】sus_tsk
【割込み部】isus_tsk

C言語インタフェース

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

パラメータ

ID	tid	タスクID

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

tidで指定したタスクをSuspended状態に移行させます。Suspended状態は、rsm_tskサービスコールにより解除されます。

サスペンド要求は他の要因でWaiting状態のタスクに対しても有効です。他の要因でWaiting状態のタスクへサスペンド要求を行うと、重複した待ち状態(Waiting + Suspended状態)となります。その場合、両方の要因が解除されないとReady状態になりません。

Suspended状態のタスクに対してサスペンド要求すると、何もせずSuspended状態のままとし、ercdはE_OK(0)が返ります。重複したSuspended状態にならないので、一回のrsm_tskサービスコールでSuspended状態が解除されます。

パラメータtidにはサスペンドさせたいタスクのタスクIDを指定してください。tidに0(アイドル)、自タスクIDや、存在しないタスクのタスクIDを指定しないでください。

Dormant状態のタスクIDを指定しないでください。

割込み部から発行する場合、ディスパッチ禁止状態でRunning状態のタスクIDを指定しないでください。

sus_tskサービスコールはタスク部から発行します

isus_tskサービスコールは割込み部から発行します。

4.6.9 rsm_tsk, irsm_tsk サスペンドの解除

【タスク部】rsm_tsk
【割込み部】irsm_tsk

C言語インタフェース

```
ER ercd = rsm_tsk ( ID tid );
ER ercd = irsm_tsk ( ID tid );
```

パラメータ

ID	tid	タスクID

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ［ ① ］（4 バイト、符号付き）

① リターンコード

E_OK(H'00000000)	正常終了

解 説

sus_tskサービスコールによりSuspended状態になっているタスクのサスペンドを解除します。
Suspended状態でないタスクを指定した場合、その要求は無視され、一切記憶されません。その際も、ercdはE_OK(0)が返ります。

パラメータtidにはサスペンドを解除させたいタスクのタスクIDを指定してください。tidに0(アイドル)や、存在しないタスクのタスクIDを指定しないでください。
Dormant状態のタスクIDを指定しないでください。
また、Running状態のタスクが自タスクへのrsm_tskサービスコールを発行してはいけません。

rsm_tskサービスコールはタスク部から発行します。

irsm_tskサービスコールは割込み部から発行します。

4.6.10　dis_dsp　　ディスパッチの禁止

【タスク部】

C言語インタフェース

ER ercd = dis_dsp (void);

パラメータ

無し

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ［　　①　　］（4 バイト、符号付き）

① リターンコード

E_OK(H'00000000)	正常終了

解　説

ディスパッチ禁止状態に移行します。

ディスパッチ禁止状態の時に呼び出された場合には何もしません。

ディスパッチ禁止状態は重複しないため、一回のena_dspサービスコールでディスパッチ禁止状態が解除されます。

dis_dspサービスコールはタスク部のみ発行できます。

4.6.11　ena_dsp　　ディスパッチの許可

【タスク部】

C言語インタフェース

ER ercd　=　ena_dsp (void);

パラメータ

無し

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ［　①　］（4 バイト、符号付き）

① リターンコード

E_OK(H'00000000)	正常終了

解　説

ディスパッチ禁止状態を解除し、ディスパッチ許可状態に移行します。
ディスパッチ許可状態の時に呼び出された場合には何もしません。

ena_dspサービスコールはタスク部のみ発行できます。

4.7　イベントフラグ

4.7.1　wai_flg　　イベントフラグ待ち

【タスク部】

C言語インタフェース

ER ercd　=　wai_flg (ID fid, FLGPTN ptn, MODE wfmod, FLGPTN *p_ptn);

パラメータ		
ID	fid	イベントフラグID
FLGPTN	ptn	待ちビットパターン(32bit)
MODE	wfmod	待ちモード（TWF_ANDW／TWF_ORW）
FLGPTN	*p_ptn	待ち解除時ビットパターン格納領域(32bit)

リターンパラメータ		
ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] （4 バイト、符号付き）

① リターンコード

E_OK(H'00000000)　　正常終了

解　説

fid で指定されるイベントフラグが、ptn で指定される待ちビットパターンの待ち解除条件を満たすのを待ちます。既に待ち解除要因を満たす場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、32 ビットの待ちビットパターンを設定します。

パラメータ wfmode には、イベントフラグの待ち解除条件を指定します。TWF_ANDW（=H'00）が指定された場合の解除条件は、ptn にて指定された bit が全て設定された場合となります。TWF_ORW（=H'01）が指定された場合の解除条件は、ptn にて指定された bit のいずれかが設定された場合となります。

パラメータ p_ptn には、待ち解除時のビットパターンを格納する領域を指定します。正常に待ちが解除された場合には、指定した領域に待ち解除時のビットパターンが格納されます。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では発行できません。

4.7.2 twai_flg イベントフラグ待ち(タイムアウト付き)

【タスク部】

C言語インタフェース

ER ercd = twai_flg (ID fid, FLGPTN ptn, MODE wfmod, FLGPTN *p_ptn , TMO tim);

パラメータ

ID	fid	イベントフラグID
FLGPTN	ptn	待ちビットパターン(32bit)
MODE	wfmod	待ちモード（TWF_ANDW／TWF_ORW）
FLGPTN	*p_ptn	待ち解除時ビットパターン格納領域(32bit)
TMO	tim	タイムアウト時間(msec)

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ① （4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了
E_TMOUT(H'FFFFFFCE= -50)	ポーリング失敗、または、タイムアウト

解 説

fid で指定されるイベントフラグが、ptn で指定される待ちビットパターンの待ち解除条件を満たすか、タイムアウトするのを待ちます。既に待ち解除要因を満たす場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、32 ビットの待ちビットパターンを設定します。

パラメータ wfmode には、イベントフラグの待ち解除条件を指定します。TWF_ANDW（=H'00）が指定された場合の解除条件は、ptn にて指定された bit が全て設定された場合となります。TWF_ORW（=H'01）が指定された場合の解除条件は、ptn にて指定された bit のいずれかが設定された場合となります。

パラメータ p_ptn には、待ち解除時のビットパターンを格納する領域を指定します。正常に待ちが解除された場合には、指定した領域に待ち解除時のビットパターンが格納されます。

パラメータ tim には、タイムアウト時間(msec)を設定します。待ちビットパターンの待ち解除条件を満たさない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、イベントフラグをポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は wai_flg と同じ動作となります。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では tim=TMO_POL のみ発行できます。

4.7.3 set_flg , iset_flg イベントフラグの設定

【タスク部】set_flg
【割込み部】iset_flg

C言語インタフェース

```
ER ercd = set_flg (ID fid, FLGPTN ptn);
ER ercd = iset_flg (ID fid, FLGPTN ptn);
```

パラメータ

ID	fid	イベントフラグID
FLGPTN	ptn	セットするビットパターン(32bit)

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ① (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

fid で指定されるイベントフラグを、ptn で指定された値との論理和(OR)で更新します。更新の結果、待ちタスクの解除条件を満たせば、待ちタスクを Waiting 状態から Ready 状態に移行します。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、32 ビットのセットするビットパターンを設定します。

set_flg はタスク部から発行できます。

iset_flgは割込み部から発行できます。

4.7.4 clr_flg イベントフラグのクリア

【タスク部】clr_flg
【割込み部】iclr_flg

C言語インタフェース

```
ER ercd = clr_flg (ID fid, FLGPTN ptn);
ER ercd = iclr_flg (ID fid, FLGPTN ptn);
```

パラメータ

ID	fid	イベントフラグID
FLGPTN	ptn	クリアするビットパターン(32bit)

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ① (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

fid で指定されるイベントフラグを、ptn で示された値との論理積(AND)に更新します。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、32 ビットのクリアするビットパターンを設定します。

clr_flgサービスコールはタスク部から発行します。

iclr_flgサービスコールは割込み部から発行します。

4.7.5 vevtflg_init イベントフラグの初期化

【システム初期化部】

C言語インタフェース

```
void    vevtflg_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解 説

イベントフラグに関する待ちキューなどの初期化を行います。イベントフラグを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.7.6　VEVTFLG_ATTR　イベントフラグ属性の設定(マクロ)

【システム初期化部】

C言語インタフェース

void　VEVTFLG_ATTR (ID fid, UB attr);

パラメータ

ID	fid	イベントフラグID
UB	attr	イベントフラグ属性

リターンパラメータ

無し

解　説

fid で指定されるイベントフラグの属性を設定します。イベントフラグ ID 毎に属性設定が可能です。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ attr でイベントフラグ属性を設定します。設定できる属性は以下の通りです。

・ イベントフラグ待ちの待ちキュー設定
　VEVFLG_TA_TFIFO(FIFO 順、H'00)、または、 VEVFLG_TA_TPRI(優先度順、H'01)

・ 待ち解除時のイベントフラグクリア属性
　VEVFLG_TA_CLR(H'02)

イベントフラグ初期化したとき、イベントフラグ属性の初期値は(VEVFLG_TA_TFIFO)です。本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックにて vevtflg_init サービスコール後に発行してください。

4.8　セマフォ

4.8.1　wai_sem　　セマフォの獲得

【タスク部】

C言語インタフェース

ER ercd = wai_sem (ID sid);

パラメータ

ID	sid	セマフォID

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解　説

sid で指定されるセマフォを獲得します。セマフォの獲得ができない場合 Waiting 状態となり、セマフォが返却されるのを待ちます。セマフォを獲得できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では発行できません。

4.8.2 twai_sem セマフォの獲得(タイムアウト付き)

【タスク部】

C言語インタフェース

ER ercd = twai_sem (ID sid, TMO tim);

パラメータ

ID	sid	セマフォID
TMO	tim	タイムアウト時間(msec)

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ① (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了
E_TMOUT(H'FFFFFFCE = -50)	ポーリング失敗、または、タイムアウト

解 説

sid で指定されるセマフォを獲得します。セマフォの獲得ができない場合 Waiting 状態となり、セマフォが返却されるか、タイムアウトするのを待ちます。セマフォを獲得できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

パラメータ tim には、タイムアウト時間(msec)を設定します。セマフォの獲得ができない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、セマフォ獲得をポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は wai_sem と同じ動作となります。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では tim=TMO_POL のみ発行できます。

4.8.3 sig_sem , isig_sem セマフォの返却

【タスク部】sig_sem
【割込み部】isig_sem

C言語インタフェース

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

パラメータ

UB	sid	セマフォID

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

sid で指定されるセマフォを返却します。セマフォ待ちのタスクがあれば、Waiting 状態から Ready 状態に移行します(セマフォ待ちタスクのセマフォ獲得順序は、セマフォ属性により決定します)。獲得していないセマフォを返却してはいけません。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

sig_sem はタスク部から発行できます。

isig_sem は割込み部から発行できます。

4.8.4 vsem_init セマフォの初期化

【システム初期化部】

C言語インタフェース

```
void    vsem_init (void);
```

パラメータ

無し

リターンパラメータ

無し

解 説

セマフォに関する待ちキューなどの初期化を行います。セマフォを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.8.5 VSEM_ATTR セマフォ属性の設定(マクロ)

【システム初期化部】

C言語インタフェース

void VSEM_ATTR (ID sid, UB semcnt, UB attr);

パラメータ

ID	sid	セマフォID
UB	semcnt	セマフォ資源数の初期値
UB	attr	セマフォ属性

リターンパラメータ

無し

解 説

sid で指定されるセマフォの属性を設定します。セマフォ ID 毎に属性設定が可能です。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

パラメータ semcnt でセマフォ資源数の初期値を設定します。0～127 の値を設定してください。

パラメータ attr でセマフォ属性を設定します。設定できる属性は以下の通りです。

・ セマフォ待ちの待ちキュー設定
VSEM_TA_TFIFO(FIFO 順、H'00)、または、 VSEM_TA_TPRI(優先度順、H'01)

セマフォ初期化したとき、セマフォ属性の初期値は、セマフォ資源数が 1、セマフォ待ちの待ちキュー設定が VSEM_TA_TFIFO(FIFO 順)となります。本サービスコールはシステム初期化部からのみ発行できます。uinit コールバックにて vsem_init サービスコール後に発行してください。

4.9 データキュー

4.9.1 rcv_dtq　　データキューからの受信

【タスク部】

C言語インタフェース

```
ER ercd  =  rcv_dtq (ID qid, VP_INT *data);
```

パラメータ

ID	qid	データキューID
VP_INT	*data	受信データを格納する領域のポインタ

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

qid で指定されるデータキューからデータを受信し、data に返します。データキューからのデータ受信した結果、送信待ちタスクがいれば、データキューへの送信を行い Waiting 状態から Ready 状態に移行します。

データキューにデータがない場合 Waiting 状態となり、データキューにデータが送信されるのを待ちます。データキューからデータを受信できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ*data には受信データを格納する領域のポインタを指定します。受信したデータは*data で指定した領域に格納されます。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では発行できません。

4.9.2 trcv_dtq データキューからの受信(タイムアウト付き)

【タスク部】

C言語インタフェース

ER ercd = trcv_dtq (ID qid, VP_INT *data, TMO tim);

パラメータ

ID	qid	データキューID
VP_INT	*data	受信データを格納する領域のポインタ
TMO	tim	タイムアウト時間(msec)

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了
E_TMOUT(H'FFFFFFCE = -50)	ポーリング失敗、または、タイムアウト

解 説

qid で指定されるデータキューからデータを受信し、data に返します。データキューからのデータ受信した結果、送信待ちタスクがいれば、データキューへの送信を行い Waiting 状態から Ready 状態に移行します。

データキューにデータがない場合 Waiting 状態となり、データキューにデータが送信されるのを待ちます。データキューからデータを受信できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ*data には受信データを格納する領域のポインタを指定します。受信したデータは*data で指定した領域に格納されます。

パラメータ tim には、タイムアウト時間(msec)を設定します。データキューからの受信ができない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、データキュー受信をポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は rcv_dtq と同じ動作となります。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では tim=TMO_POL のみ発行できます。

4.9.3 snd_dtq , isnd_dtq データキューへの送信

【タスク部】snd_dtq
【割込み部】isnd_dtq

C言語インタフェース

```
ER ercd  =  snd_dtq (ID qid, VP_INT data);
ER ercd  =  isnd_dtq (ID qid, VP_INT data);
```

パラメータ

ID	qid	データキューID
VP_INT	data	送信データ

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000) 正常終了

E_TMOUT(H'FFFFFFCE = -50) ポーリング失敗、または、タイムアウト

解 説

qid で指定されるデータキューに、data で指定されるデータを送信します。データキューへデータ送信した結果、受信待ちタスクがいれば、受信待ちタスクの先頭のタスクにデータを渡して、Waiting 状態から Ready 状態に移行します。

データキューに空きがない場合、Waiting 状態となり、データキューに空きができるまで待ちます。データキューへの送信できた場合は、Waiting 状態にならずそのまま実行を続けます。

(但し、isnd_dtq でデータキューに空きがない場合、待ちには入らずリターンパラメタ ercd に E_TMOUT(H'CE)が返ります)

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ data には送信データを指定します。

snd_dtq はタスク部から発行できます。
ディスパッチ禁止状態では発行できません。

isnd_dtq は割込み部から発行できます。

4.9.4 tsnd_dtq データキューへの送信(タイムアウト付き)

【タスク部】

C言語インタフェース

ER ercd = tsnd_dtq (ID qid, VP_INT data, TMO tim);

パラメータ

ID	qid	データキューID
VP_INT	data	受信データ
TMO	tim	タイムアウト時間(msec)

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了
E_TMOUT(H'FFFFFFCE = -50)	ポーリング失敗、または、タイムアウト

解 説

qid で指定されるデータキューに、data で指定されるデータを送信します。データキューへデータ送信した結果、受信待ちタスクがいれば、受信待ちタスクの先頭のタスクにデータを渡して、Waiting 状態から Ready 状態に移行します。

データキューに空きがない場合、Waiting 状態となり、データキューに空きができるまで待ちます。 データキューへの送信できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ data には送信データを指定します。

パラメータ tim には、タイムアウト時間(msec)を設定します。データの送信ができない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、データキュー受信をポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は snd_dtq と同じ動作となります。

本サービスコールはタスク部のみ発行できます。
ディスパッチ禁止状態では tim=TMO_POL のみ発行できます。

4.9.5 fsnd_dtq , ifsnd_dtq データキューへの強制送信

【タスク部】fsnd_dtq
【割込み部】ifsnd_dtq

C言語インタフェース

```
ER ercd  =  fsnd_dtq (ID qid, VP_INT data);
ER ercd  =  ifsnd_dtq (ID qid, VP_INT data);
```

パラメータ

ID	qid	データキューID
VP_INT	data	送信データ

リターンパラメータ

ER	ercd	リターンコード

リターン値／エラーコード

ercd ［ ① ］（4 バイト、符号付き）

① リターンコード

E_OK(H'00000000)	正常終了
E_ILUSE(H'FFFFFFE4 = -28)	データ数が 0 のデータキューを指定した

解 説

qid で指定されるデータキューに、data で指定されるデータを強制送信します。データキューへデータ送信した結果、受信待ちタスクがいれば、受信待ちタスクの先頭のタスクにデータを渡して、Waiting 状態から Ready 状態に移行します。

データキューに空きがない場合には、データキューに格納される一番古いデータを抹消し空き領域を確保してから、データキューへの送信を行います。指定したデータキューのデータ数が 0 の場合、リターンパラメータ ercd に E_ILUSE(H'E4)を返します。

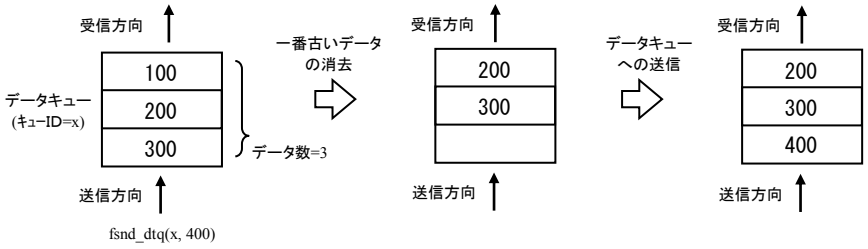

図4-3 データキューへの強制送信時、空きがない場合の動作(データ数=3)

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ data には送信データを指定します。

fsnd_dtq はタスク部から発行できます。

ifsnd_dtq は割込み部から発行できます。

4.9.6 vdtq_init データキューの初期化

【システム初期化部】

C言語インタフェース

```
void    vdtq_init (void);
```

パラメータ

無し

リターンパラメータ

無し

解 説

データキューに関する待ちキューなどの初期化を行います。データキューを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.9.7 VDTQ_ATTR データキューの属性設定

【システム初期化部】

C言語インタフェース

void VDTQ_ATTR (ID qid, UB attr, B cnt, W *dtq);

パラメータ

ID	qid	データキューID
UB	attr	データキュー属性
B	cnt	データ数
W	dtq	データキュー領域の先頭番地

リターンパラメータ

無し

解 説

qid で指定されるデータキューの属性を設定します。データキューID 毎に属性設定が可能です。

パラメータ qid にはデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ attr でデータキュー属性を設定します。設定できる属性は以下の通りです。

・ データキュー待ち(データ送信時)の待ちキュー設定
VDTQ_TA_TFIFO(FIFO 順、H'00)、または、 VDTQ_TA_TPRI(優先度順、H'01)

※ データ受信時の待ちキューは、常に FIFO 順でキューイングされます。

パラメータ cnt でデータの数を指定します。0～127 の値を設定してください。

パラメータ dtq でデータキューの先頭番地を指定します。データ数分の 32bit 配列を用意し、本パラメータに指定してください。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.10　時間管理

4.10.1　set_tim　　システム時刻の設定

【タスク部】set_tim
【割込み部】iset_tim

C言語インタフェース

```
ER ercd  =  set_tim (SYSTIM *tim);
ER ercd  =  iset_tim (SYSTIM *tim);
```

パラメータ

SYSTIM　　*tim　　システム時刻(msec)を格納した領域のポインタ

```
typedef struct _systim {
    UH htim;        /* システム時刻 上位16ビット */
    UW ltim;        /* システム時刻 下位32ビット */
} SYSTIM;
```

リターン値／エラーコード

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)　　正常終了

解 説

システム時刻を設定します。単位は msec です。

システム時刻は 48bit(符号無し)で管理しています。

set_timサービスコールはタスク部から発行します。

iset_timサービスコールは割込み部から発行します。

4.10.2 get_tim システム時刻の取得

【タスク部】get_tim
【割込み部】iget_tim

C言語インタフェース

```
ER ercd  =  get_tim (SYSTIM *tim);
ER ercd  =  iget_tim (SYSTIM *tim);
```

パラメータ

SYSTIM	*tim	システム時刻(msec)を格納する領域のポインタ

```
typedef struct _systim {
    UH htim;        /* システム時刻 上位16ビット */
    UW ltim;        /* システム時刻 下位32ビット */
} SYSTIM;
```

リターン値／エラーコード

ercd ［ ① ］（4 バイト、符号付き）

① リターンコード

E_OK(H'00000000)	正常終了

解 説

システム時刻を取得します。単位は msec です。

システム時刻は 48bit(符号無し)で管理しています。

get_timサービスコールはタスク部から発行します。

iget_timサービスコールは割込み部から発行します。

4.10.3 vsystim_init 時間管理の初期化

【システム初期化部】

C言語インタフェース

```
void    vsystim_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解 説

時間管理に関する待ちキューなどの初期化を行います。時間管理、周期ハンドラを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinitコールバックから発行してください。

4.10.4 ivsig_tim タイムティックの供給（周期タイマ処理）

【割込み部】

C言語インタフェース

```
void    ivsig_tim ( void );
```

パラメータ

無し

リターンパラメータ

無し

解 説

周期タイマハンドラ(割込み部)から本サービスコールを発行します。時間管理の周期タイマ処理を実行します。

4.11　周期ハンドラ

4.11.1　sta_cyc　　周期ハンドラの開始

【タスク部】sta_cyc
【割込み部】ista_cyc

C言語インタフェース

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

パラメータ

ID	cid	周期ハンドラID

リターンパラメータ

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解　説

cid で指定される周期ハンドラの動作を開始します。動作開始により、周期時間のカウントが開始され、周期時間が経過したタイミングで周期ハンドラ関数が実行されます。既に周期ハンドラが開始されている場合は何も行いません。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

sta_cycサービスコールはタスク部から発行します。

ista_cycサービスコールは割込み部から発行します。

4.11.2 stp_cyc 周期ハンドラの停止

【タスク部】stp_cyc
【割込み部】istp_cyc

C言語インタフェース

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

パラメータ

ID	cid	周期ハンドラID

リターンパラメータ

ercd [①] (4 バイト、符号付き)

① リターンコード

E_OK(H'00000000)	正常終了

解 説

cid で指定される周期ハンドラの動作を停止します。周期ハンドラ停止中の場合は何も行いません。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

stp_cycサービスコールはタスク部から発行します。

istp_cycサービスコールは割込み部から発行します。

4.11.3 vcyc_init 周期ハンドラの初期化

【システム初期化部】

C言語インタフェース

```
void    vcyc_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解 説

周期ハンドラに関する管理領域などの初期化を行います。周期ハンドラを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinitコールバックから発行してください。

4.11.4 VCYC_ATTR 周期ハンドラの属性設定

【システム初期化部】

C言語インタフェース

void VCYC_ATTR (ID cid, UB attr, void (*hdr)(ID), W cyc);

パラメータ

ID	cid	周期ハンドラID
UB	attr	周期ハンドラ属性
void (*)(ID)	hdr	周期ハンドラ関数
W	cyc	周期時間(msec)

リターンパラメータ

無し

解 説

cid で指定される周期ハンドラの属性を設定します。周期ハンドラ ID 毎に属性設定が可能です。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

パラメータ attr で周期ハンドラ属性を設定します。設定できる属性は以下の通りです。

・ 周期ハンドラの初期動作設定
VCYC_TA_NON (初期状態で周期ハンドラ動作が停止、H'00)
または
VCYC_TA_STA (初期状態で周期ハンドラ動作が開始、H'01)

パラメータ hdr で周期ハンドラ関数を指定します。

パラメータ cyc で周期時間(msec)を指定します。設定できる時間の最大値は H'7fffffff です。周期時間が 0 に設定されている場合は起動しても、開始されません。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.11.5　VCYC_CHG　　周期ハンドラの起動周期変更

【タスク部】【割込み部】【システム初期化部】

C言語インタフェース

```
void　VCYC_CHG (ID cid, W cyc);
```

パラメータ

ID	cid	周期ハンドラID
W	cyc	周期時間(msec)

リターンパラメータ

無し

解　説

cid で指定される周期ハンドラの起動周期を変更します。μTRON4.0 仕様「周期ハンドラの起動位相」の実装や、起動周期の動的変更が可能です。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

パラメータ cyc で周期時間(msec)を指定します。設定できる時間の最大値は H'7fffffff です。周期時間が 0 に設定されている場合は起動しても、開始されません。

本サービスコールは割込み部（周期ハンドラ実行中）から発行できます。

4.11.6 *callback_cychdr* 周期ハンドラ本体(コールバック)

【－】

C言語インタフェース

void *callback_cychdr* (ID cid);

パラメータ

ID	cid	周期ハンドラID

リターンパラメータ

無し

解 説

周期ハンドラ本体（コールバック）です。関数名はユーザ任意の名称です。

周期ハンドラの動作を開始し、指定した周期時間が経過した時点で、本コールバックが呼び出されます。任意の周期ハンドラ処理を記載してください。本コールバックが実行されるとき、システム状態は割込み部です。

パラメータ cid には周期ハンドラ ID が設定されます。

使用方法の詳細は「3.5.3 周期ハンドラ」を参照ください。

4.12　割込み

4.12.1　vdisp　　vdisp 有割込みハンドラをディスパッチして終了

【割込み部】

C言語インタフェース

void　vdisp (H mode);

パラメータ

H	mode	多重割込みフラグ

リターンパラメータ

無し

解　説

vdisp有割込みハンドラをディスパッチして終了します。本サービスコールは割込み部からのみ発行できます。

パラメータ mode には、多重割込みフラグを指定します。mode が 0以外のとき多重割込み中であることを示します。 mode は、callback_int のパラメータ mode をそのまま指定してください。

多重割込み中にvdispサービスコールを発行すると発行元に一度戻りますが、vdispサービスコールの発行前後でレジスタ状態が保証されていない為、必ず、vdisp有割込みハンドラ本体(callback_int)の最後で実行してください。

使用方法の詳細は「5.4 割込みハンドラ」を参照ください。

4.12.2 *callback_int* 割込みハンドラ本体(コールバック)

【－】

C言語インタフェース

void *callback_int* (UW irqid);
void *callback_int* (UW irqid, H mode);

パラメータ

UW	irqid	割込みID
H	mode	多重割込みフラグ

リターンパラメータ

無し

解 説

割込みハンドラ受付けから呼び出されるコールバックです。関数名はユーザ任意の名称です。

割込み処理および割込み終了処理など必要な処理を行ってください。本コールバックが実行されるとき、システム状態は割込み部です。

vdisp無割込みハンドラには、パラメータ irqid が渡されます。

vdisp有割込みハンドラには、パラメータ irqid、mode が渡されます。

パラメータ irqid は、発生した割込みID情報です。割込み終了処理で使用します。

パラメータ mode は、多重割込みフラグが設定されて呼び出されます。mode が 0以外のとき多重割込み中であることを示します。vdispサービスコール発行時の引数としても使用されますのでパラメータmodeを書き換えないでください。

使用方法の詳細は「5.4 割込みハンドラ」を参照ください。

4.13　その他の初期化

4.13.1　vslos_init　　OS の起動

【システム初期化部】

C言語インタフェース

```
void    vslos_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解　説

Smalight OSを起動します。リセット割込みで呼び出される処理などで、割込み禁止のまま vslos_init サービスコールを発行してください。呼び出し元へは戻りません。詳細は「5.2.2 初期化処理」を参照ください。

4.13.2 kinit OS 初期化処理(コールバック)

【－】

C言語インタフェース

```
void    kinit ( void );
```

パラメータ

無し

リターンパラメータ

無し

解 説

vslos_init（OS初期化処理）から呼び出されるコールバックです。イベントフラグやセマフォの初期化・属性設定などのOS初期化を行ってください。本コールバックが実行されるとき、システム状態はシステム初期化部です。

本コールバックルーチンの詳細は「5.2.2 初期化処理」を参照ください。呼び出されたとき、スタックはOSスタックを使用しています。

4.13.3 uinit ユーザ初期化処理(コールバック)

【－】

C言語インタフェース

```
void    uinit ( void );
```

パラメータ

無し

リターンパラメータ

無し

解 説

vslos_init（OS初期化処理）から呼び出されるコールバックです。ユーザシステムに応じた初期化を行ってください。本コールバックが実行されるとき、システム状態はシステム初期化部です。

呼び出されたとき、スタックはOSスタックを使用しています。必要に応じてOSスタックエリアサイズを大きくしてください。設定方法は「5.2.2 初期化処理」を参照ください。

4.13.4 stack_init タスクスタックの初期化(コールバック)

【－】

C言語インタフェース

```
void    stack_init ( ID tid, TSTACK *sp );
```

パラメータ

ID	tid	タスクID
TSTACK	*sp	スタックポインタ

リターンパラメータ

無し

解 説

OS初期化処理で呼び出されるコールバックです。各タスクの開始アドレス等の情報を各タスクのスタックに格納する処理を行います。登録したタスク数分だけ発行されます。本コールバックが実行されるとき、システム状態はシステム初期化部です。

4.14　その他

4.14.1　idle　　アイドル処理(コールバック)

【－】

C言語インタフェース

```
void    idle ( void );
```

パラメータ

無し

リターンパラメータ

無し

解　説

アイドル状態(実行可能なタスクが存在しない)のとき、実行されるコールバックルーチンです。割込みマスクを空けて、ループ処理には入ります。

省電力モードへの対応など、必要に応じて本コールバックルーチンを変更してください。

呼び出されたとき、スタックはOSスタックを使用しています。必要に応じてOSスタックエリアサイズを大きくしてください。

5. アプリケーションプログラムの作成

5.1 作成の手順

以下にアプリケーション作成フローを示します。

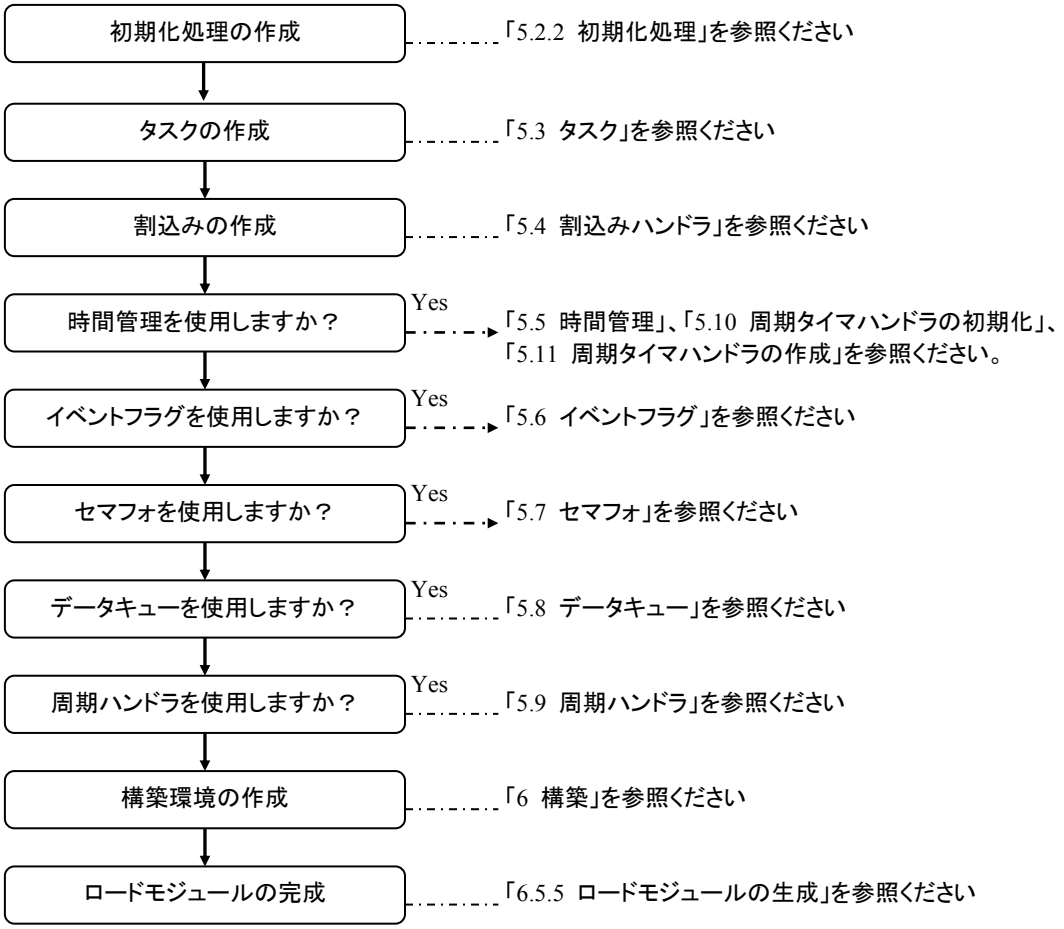

図5-1 アプリケーション作成フロー

5.2　システムの起動処理

5.2.1　システム起動時の処理フロー

システム起動時のフローを以下に示します。

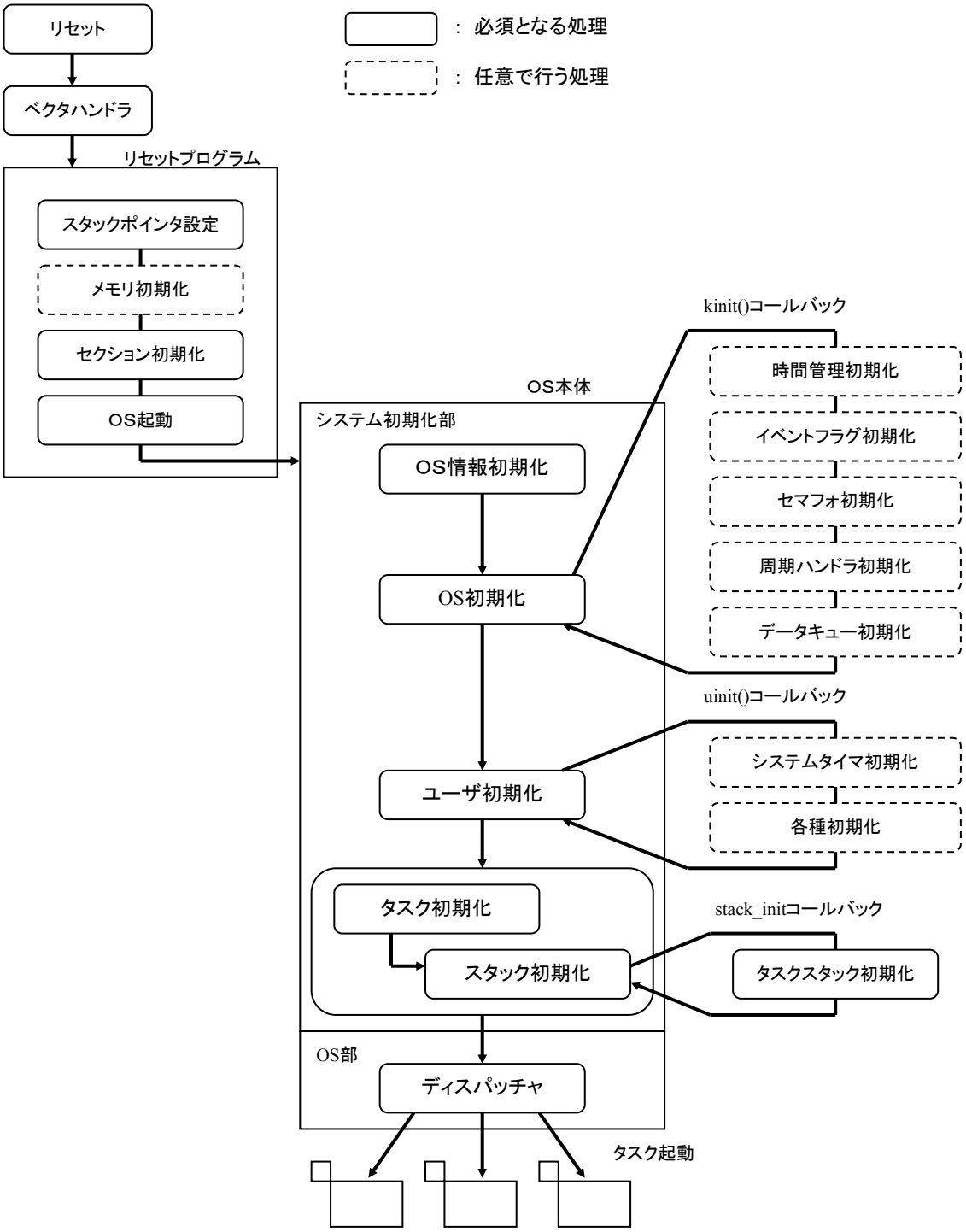

図5-2 システム起動時のフロー

5.2.2　初期化処理

(1) リセットプログラム

リセットプログラム(vhandlerRES.s)ではプロセッサの初期化処理を行い vslos_init()によって OS を起動します。OS 内部からの kinit コールバック、uinit コールバックは OS スタックを使用して実行されます。OS スタック領域(osstack.s)は必要に応じてスタックエリアサイズを大きくしてください。

【解説】

① OS 内部では NEON/VFP レジスタをアクセスしますので、NEON/VFP を有効化します。

② プロセッサモードをシステムモードに設定します。smalight OS はタスク、割込み、OS 全てシステムモードでの動作です。

② C コンパイラ標準モジュールをコールします。標準モジュールではスキャッタローディング機能によるメモリ領域の初期化(RW セクションのコピー、ZI セクションのクリアなど)が行われます。C コンパイラ標準モジュール処理の詳細については、コンパイラツールの各種ドキュメントを参照してください。

リスト5-1 リセットプログラム(vhandlerRES.s)

```
    EXPORT  PowerOnReset_Handler
PowerOnReset_Handler

            :

    ; NEON,VFP enable
    mrc     p15, 0, r0, c1, c0, 2                       … ①
    orr     r0, r0, #0x00F00000
    mcr     p15, 0, r0, c1, c0, 2
    isb
    mov     r0, #0x40000000
    vmsr    FPEXC, r0

    ; Transition to SYSTEM mode
    mov     r0, #MODE_SYS:OR:F_BIT                      … ②
    msr     cpsr_c, r0

    IMPORT  __main                                      … ③
    b       __main

    END
```

【解説】

① slos.h をインクルードしてください。

② main は C コンパイラ標準モジュールよりコールされます。

③ OS を起動します。

リスト5-2 リセットプログラム(main.c)

```
#include "slos.h"                                       … ①

            :

int main(void)                                          … ②
{
    vslos_init();                                       … ③
    while(1){}
}
```

(2) kinit コールバック

kinit コールバック(OS 初期化処理)では以下の処理を行ってください。

【解説】

① slos.h をインクルードしてください。

② 時間管理の初期化

時間管理を使用する場合には初期化を行ってください。

時間管理の詳細は「5.5 時間管理」を参照ください。

③ イベントフラグの初期化

イベントフラグを使用する場合には初期化を行ってください。

イベントフラグの詳細は「5.6 イベントフラグ」を参照ください。

④ セマフォの初期化

セマフォを使用する場合には初期化を行ってください。

セマフォの詳細は「5.7 セマフォ」を参照ください。

⑤ 周期ハンドラの初期化

周期ハンドラを使用する場合には初期化を行ってください。

周期ハンドラの初期化の詳細は「5.9 周期ハンドラ」を参照ください。

⑥ データキューの初期化

時間管理を使用する場合には初期化を行ってください。

データキューの初期化の詳細は「5.8 データキュー」を参照ください。

リスト5-3 kinitコールバック(kinit.c)

```
#include "slos.h"                                                    … ①

          :

void kinit(void)
{
    /* initialize */

    /* systime initialize */                                        … ②
/*  vsystim_init(); */

    /* event flag initialize */                                     … ③
/*  vevtflg_init();  */
/*  VEVTFLG_ATTR((ID)1u, (VEVFLG_TA_CLR | VEVFLG_TA_TPRI));  */

    /* semaphore initialize */
/*  vsem_init();                    */                              … ④
/*  VSEM_ATTR((ID)1u, 1, VSEM_TA_TPRI);  */

    /* cyclic handler initialize  */
/*  vcyc_init(); */
/*  VCYC_ATTR((ID)1u, (VCYC_TA_STA), cychdr1, 1000L); */        … ⑤
/*  VCYC_ATTR((ID)2u, (VCYC_TA_NON), cychdr2, 5000L); */

    /* data-que initialize  */
/*  vdtq_init(); */
/*  VDTQ_ATTR((ID)1u,(UB)VDTQ_TA_TFIFO,(B)DTQ1_SIZE, knl_dtq1); */  … ⑥
/*  VDTQ_ATTR((ID)2u,(UB)VDTQ_TA_TPRI, (B)DTQ2_SIZE, knl_dtq2); */
}
```

(3) uinit コールバック

uinit コールバック(ユーザ初期化処理)では以下の処理を行ってください。

【解説】

① slos.h をインクルードしてください。

② その他初期化。

周期タイマハンドラの初期化、ポート、システム資源などのその他初期化を行ってください(ユーザ任意)。

リスト5-4 uinitコールバック(user.c)

```
#include "slos.h"                                          … ①

          :

void uinit(void)
{
    /* initialize */                                       … ②

}
```

(4) stack_init コールバック

OS 初期化処理で呼び出さるコールバックです。stack_init コールバック(ユーザ初期化処理)では各タスクの開始アドレス等の情報を各タスクのスタックに格納する処理を行います。登録したタスク数分だけ発行されます。

5.3　タスク

タスクは C 言語で記述します。タスクの記述方法を以下に示します。

【解説】

① slos.h をインクルードしてください。

② 通常の関数同様、タスクの処理を記述します。sta_tsk サービスコールまたは stack_init コールバックで指定したパラメータが stacd として渡されます。

③ タスク関数を終了する場合は、ext_tsk サービスコールを発行して Dormant 状態にします。ext_tsk サービスコールを発行せずにタスク関数を終了した場合の動作は保証されません。

リスト5-5 タスク記述方法

```
#include "slos.h"                                   … ①

       :

void tsk01(VP_INT stacd)                            … ②
{
    /* タスクの処理 */
    …
    ext_tsk();                                      … ③
}
```

④ タスク起動時の stacd(第 1 引数)を指定します。タスク起動時の第 2~4 引数を指定する場合は、r[1]~r[3]に指定します。

⑤ タスク起動時の fpscr 値を指定します。

⑥ タスク起動時の割込みマスクレベルを指定します。最低の割込みマスクレベルを指定します。

⑦ タスク開始アドレスを設定します。

⑧ タスクのプロセッサモードにシステムモードを指定します。

リスト5-6 タスク起動状態の設定方法(stackini.c)

```
void stack_init(ID tid, TSTACK *sp)
{
    extern const TCBADDR knl_tcbAddrInit[];

    sp->r[0]   = 0x12345678L;                   … ④
    sp->fpscr  = 0x00c00000L;                   … ⑤
    sp->imask  = 0x000000f8L;                   … ⑥
    sp->pc     = (W)knl_tcbAddrInit[tid-1u].tsk;    … ⑦
    sp->cpsr   = 0x0000001fL;                   … ⑧
}
```

5.4　割込みハンドラ

5.4.1　ベクタハンドラ

リセット、例外、割込み要因に応じてベクタハンドラが実行されます。
リセット(PowerOnReset_Handler)、IRQ 割込み(InterruptIRQ_Handler)についてのみ処理を登録しています。
リセット、IRQ 割込み以外の例外(データアボート、未定例外など)についての処理は未定義です。必要に応じて処理を登録します。

リスト5-7 ベクタテーブル定義(vector.s)

```
; ********************************************************************
; * vector table
; ********************************************************************
  VectorsTable
      LDR     pc,=PowerOnReset_Handler
      LDR     pc,=UndefinedException_Handler
      LDR     pc,=SuperVisorCall_Handler
      LDR     pc,=PrefetchAbort_Handler
      LDR     pc,=DataAbort_Handler
      nop
      LDR     pc,=InterruptIRQ_Handler
      LDR     pc,=InterruptFIQ_Handler
```

5.4.2　割込みベスタ ID テーブルと割込みスタック

割込み ID 毎の割込みハンドラの登録は vectbl.c にて行います。
割込みスタックの定義は istack.c にて行います。

割込みベクタ ID テーブルの記述方法を以下に示します。
以下の例では、ID 番号 32 の IRQ0 に int2(vdisp 有割込)を登録しております。

【解説】

① slos.h をインクルードしてください。
② 登録する割込みハンドラ関数の外部参照定義を行います。
③ ベクタ ID テーブル個数を定義します。
④ IRQ0 割込みレベルを定義します。
⑤ IRQ0 割込みハンドラ、割込みレベルを登録します。

リスト5-8 ベクタIDテーブル(vectbl.c)

```
#include "slos.h"                                          … ①

extern void int2(void);                                    … ②

/* ------------ Define ------------------------------ */
#define VECT_MAX           1024                            … ③

#define IMSK_IRQ0          16                              … ④

/* ------------ Vector ------------------------------ */
#pragma arm section rodata = "Cvectbl"

const INTTBL vectorIDtable[VECT_MAX] = {
                                    /* IRQ Function, intlevel  */
/* SW0              *//*    0 */    { DummyIrqFunc, IMSK_UNDEF },
/* SW1              *//*    1 */    { DummyIrqFunc, IMSK_UNDEF },
       :
/* Reserved         *//*   31 */    { DummyIrqFunc, IMSK_UNDEF },
/* IRQ0             *//*   32 */    { int2,         IMSK_IRQ0  }, … ⑤
/* IRQ1             *//*   33 */    { DummyIrqFunc, IMSK_UNDEF },
       :
```

割込みスタックの記述方法を以下に示します。
以下の例では、割込みレベル 16 の割込みスタックを定義しております。

【解説】

① slos.h をインクルードしてください。
② 割込みレベル 16 のスタックサイズを定義します。

リスト5-9 割込みスタック定義(istack.c)

```
#include "slos.h"                                          … ①

/* ---------- interrupt stack size -------------------------- */
#define INTLV00_SP_SIZE     0
       :
#define INTLV16_SP_SIZE     1024                           … ②
       :
```

5.4.3 vdisp 無割込みハンドラ

vdisp 無割込みハンドラのフローを以下に示します。

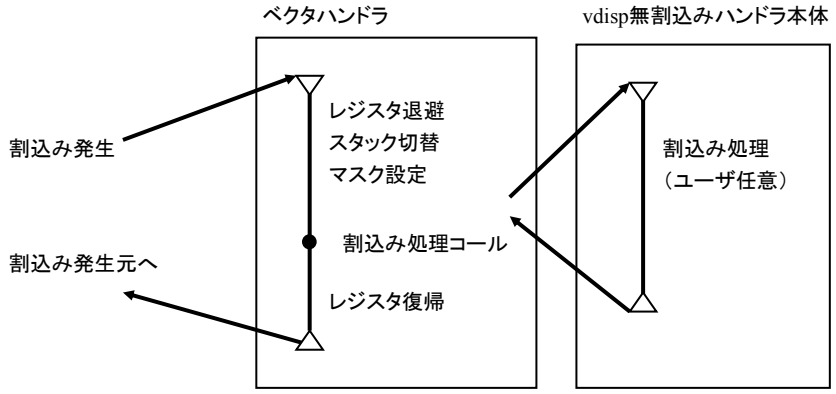

図5-3 vdisp無割込みハンドラのフロー

(1) vdisp 無割込みハンドラ本体

vdisp 無割込みハンドラ本体は C 言語で記述します。記述方法を以下に示します。

【解説】

① slos.h をインクルードしてください。

② vdisp 無割込みハンドラ本体の関数(intProc1)は callback_int コールバックです。関数名称はユーザ任意です。引数 irqid は割込み ID 情報です。割込み終了処理で使用します。通常の関数同様、割込みハンドラの処理を記述します。

③ vdisp 有割込みハンドラ本体処理を記述します。vdisp 無割込みハンドラでは、OS サービスコールを発行してはいけません。

④ 割込み終了処理を行います。

リスト5-10 vdisp無割込みハンドラ本体

```
#include "slos.h"                                          … ①

/*
 * Normal Interrupt sample
 */
void intProc1(UW irqid)                                    … ②
{
    /* interrupt process */                                … ③

    /* interrupt end */
    INTC_ICCEOIR_ADDR = irqid;                             … ④
}
```

(2) ベクタテーブル

作成した割込み関数をベクタIDテーブルに登録します。詳細は「5.4.2 割込みベスタIDテーブル」を参照ください。

5.4.4 vdisp 有割込みハンドラ

vdisp 有割込みハンドラのフローを以下に示します。図中の点線(……)処理は、多重割込みの時、または、割込み発生元タスクの割込みマスクレベルがマスクありの時に実行されます。その場合、vdisp サービスコール発行によりディスパッチャへ制御が遷移せずに、割込み発生元に戻ります。

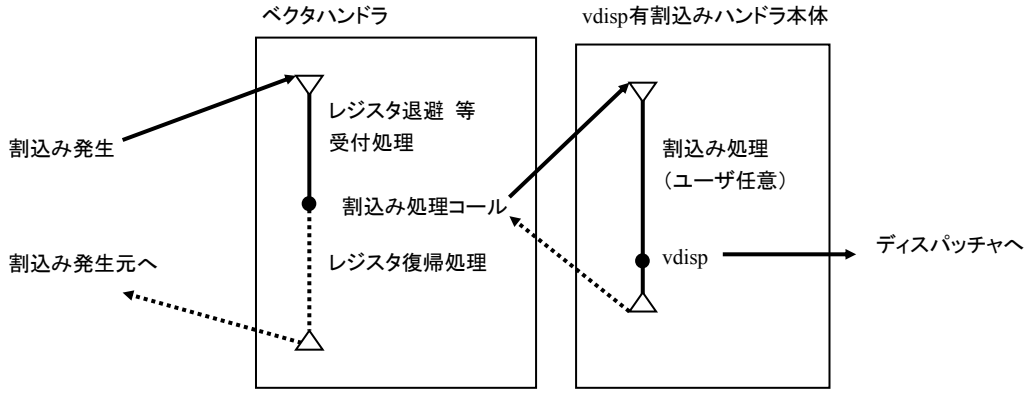

図5-4 vdisp有割込みハンドラのフロー

(1) vdisp 有割込みハンドラ本体

vdisp 有割込みハンドラ本体は C 言語で記述します。記述方法を以下に示します。

【解説】

① slos.h をインクルードしてください。

② vdisp 有割込みハンドラ本体の関数(intProc2)は callback_int コールバックです。関数名称はユーザ任意です。引数 irqid は割込み ID 情報です。割込み終了処理で使用します。引数 mode は多重割込みフラグです。多重割込みフラグが 0 以外のとき、多重割込みであることを示します。

③ vdisp 有割込みハンドラ本体処理を記述します。

④ 割込み終了処理を行います。

⑤ vdisp 有割込みハンドラ本体の最後で、vdisp サービスコールを発行します。それによりディスパッチャへ遷移します(但し、多重割込みの場合は割込み発生元に戻ります)。

リスト5-11 vdisp有割込みハンドラ本体

```
#include "slos.h"                                          … ①

       :

/*
 * return to dispatch
 */
void intProc2(UW irqid , H mode)                           … ②
{
    /* interrupt process */                                … ③

    /* interrupt end */
    INTC_ICCEOIR_ADDR = irqid;                             … ④

    vdisp(mode);                                           … ⑤
}
```

5.4.5　割込みスタックについて

同一レベルの割込みは、共通のスタックを使用しても問題ありません(vdisp 無割込み、vdisp 有割込み関係なく)。異なる割込みレベルの割込みスタックは割込みレベル毎に割込みスタックを確保する必要があります。割込みスタックの計算方法については「6.6.2 割込みスタック」を参照ください。

5.5　時間管理

5.5.1　時間管理の対象サービスコール

対象サービスコールは以下の通りです。

表5-1 時間管理関連サービスコール一覧

区分	サービスコール名称	説明
タスク管理関連	tslp_tsk	タスクの起床待ち(タイムアウト付き)
イベントフラグ	twai_flg(*1)	イベントフラグ待ち(タイムアウト付き)
セマフォ	twai_sem(*2)	セマフォの獲得(タイムアウト付き)
周期ハンドラ	sta_cyc (*3)	周期ハンドラの開始
	stp_cyc (*3)	周期ハンドラの停止
データキュー	trcv_dtq(*4)	データキューからの受信(タイムアウト付き)
	tsnd_dtq(*4)	データキューへの送信(タイムアウト付き)
時間管理	set_tim	システム時刻の設定
	get_tim	システム時刻の取得
	vsystim_init	時間管理の初期化
	ivsig_tim	周期タイマ処理

(*1) イベントフラグの設定が必要となります。
(*2) セマフォの設定が必要となります。
(*3) 周期ハンドラの設定が必要となります。
(*4) データキューの設定が必要となります。

5.5.2　時間管理の初期化

（1）OS 定義ファイルの ”#define SYSTIME ” 行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-12 時間管理の有効設定 (slosdef.h)

```
/*------ configuration define  ------------------*/
       :
#define SYSTIME
       :
```

（2）kinit コールバックに初期化関数 systim_init サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されております)。

リスト5-13 時間管理の初期化(kinit.c)

```
void kinit(void)
{
    /* initialize */

    /* systime initialize */
    vsystim_init();
           :

}
```

（3） ユーザ任意の周期タイマハンドラにて、ivsig_tim サービスコールを追加します。詳細は「5.10 周期タイマハンドラの初期化」を参照ください。

（4） ユーザプログラムにて"slos.h"をインクルードすることで、サービスコールの発行が可能です。

5.6　イベントフラグ

5.6.1　イベントフラグの対象サービスコール

対象サービスコールは以下の通りです。

表5-2 イベントフラグ関連サービスコール一覧

区分	サービスコール名称	説明
イベントフラグ	wai_flg	イベントフラグ待ち
	twai_flg(*1)	イベントフラグ待ち(タイムアウト付き)
	set_flg, iset_flg	イベントフラグの設定
	clr_flg	イベントフラグのクリア
	vevtflg_init	イベントフラグの初期化
	VEVTFLG_ATTR(*2)	イベントフラグの属性設定

(*1) 時間管理機能の設定が必要となります。
(*2) マクロです。

5.6.2　イベントフラグの初期化

（1）OS 定義ファイルの ”#define EVENTFLG” 行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-14 イベントフラグの有効設定 (slosdef.h)

```
/*------ configuration define  -----------------*/
       :
#define EVENTFLG
       :
```

（2）kinit コールバックに初期化関数 vevtflg_init サービスコール、VEVTFLG_ATTR サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されております)。

リスト5-15 イベントフラグの初期化(kinit.c)

```
void kinit(void)
{
          :
   /* event flag initialize */
    vevtflg_init();
    VEVTFLG_ATTR((ID)1u, (VEVFLG_TA_CLR | VEVFLG_TA_TPRI));
          :
}
```

（3）ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.7　セマフォ

5.7.1　セマフォの対象サービスコール

対象サービスコールは以下の通りです。

表5-3 セマフォ関連サービスコール一覧

区分	サービスコール名称	説明
セマフォ	wai_sem	セマフォの獲得
	twai_sem(*1)	セマフォの獲得(タイムアウト付き)
	sig_sem, isig_sem	セマフォの返却
	vsem_init	セマフォの初期化
	VSEM_ATTR(*2)	セマフォの属性設定

(*1) 時間管理機能の設定が必要となります。
(*2) マクロです。

5.7.2　セマフォの初期化

（1）OS 定義ファイルの ”#define SEMAPHORE” 行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-16 セマフォの有効設定 (slosdef.h)

```
/*------ configuration define  -----------------*/
       :
#define SEMAPHORE
       :
```

（2）kinit コールバックに初期化関数 vsem_init サービスコール、VSEM_ATTR サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されております)。

リスト5-17 セマフォの初期化(kinit.c)

```
void kinit(void)
{
          :
    /* semaphore initialize */
    vsem_init();
    VSEM_ATTR((ID)1u, 1, VSEM_TA_TPRI);
          :
}
```

（3）ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.8　データキュー

5.8.1　データキューの対象サービスコール

対象サービスコールは以下の通りです。

表5-4 データキュー関連サービスコール一覧

区分	サービスコール名称	説明
データキュー	rcv_dtq	データキューからの受信
	trcv_dtq(*1)	データキューからの受信(タイムアウト付き)
	snd_dtq, isnd_dtq	データキューへの送信
	tsnd_dtq(*1)	データキューへの送信(タイムアウト付き)
	fsnd_dtq, ifsnd_dtq	データキューへの強制送信
	vdtq_init	データキューの初期化
	VDTQ_ATTR(*2)	データキューの属性設定

(*1) 時間管理機能の設定が必要となります。
(*2) マクロです。

5.8.2　データキューの初期化

（1）OS 定義ファイルの" #define DATAQUE"行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-18 データキューの有効設定 (slosdef.h)

```
/*------ configuration define  ------------------*/
       :
#define DATAQUE
       :
```

（2）kinit コールバックに初期化関数 vdtq_init サービスコール、VDTQ_ATTR サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されております)。

リスト5-19 データキューの初期化(kinit.c)

```
/* -------- data que --------------- */
#define DTQ1_SIZE  1
#define DTQ2_SIZE  2
W knl_dtq1[DTQ1_SIZE];
W knl_dtq2[DTQ2_SIZE];

void kinit(void)
{
          :
    /* data-que initialize  */
    vdtq_init();
    VDTQ_ATTR((ID)1u,(UB)VDTQ_TA_TFIFO,(B)DTQ1_SIZE, knl_dtq1);
    VDTQ_ATTR((ID)2u,(UB)VDTQ_TA_TPRI, (B)DTQ2_SIZE, knl_dtq2);
          :
}
```

（3）ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.9 周期ハンドラ

5.9.1 周期ハンドラの対象サービスコール

対象サービスコールは以下の通りです。

表5-5 周期ハンドラ関連サービスコール一覧

区分	サービスコール名称	説明
周期ハンドラ	sta_cyc (*1)	周期ハンドラの開始
	stp_cyc (*1)	周期ハンドラの停止
	vcyc_init(*1)	周期ハンドラの初期化
	VCYC_ATTR(*1) (*2)	周期ハンドラの属性設定
	VCYC_CHG(*1) (*2)	周期ハンドラの起動周期変更

(*1) 時間管理機能の設定が必要となります。
(*2) マクロです。

5.9.2 周期ハンドラの作成

周期ハンドラの記述方法を以下に示します。

【解説】
① 必ず slos.h をインクルードしてください。
② 周期ハンドラ本体の関数(cychdr1)は callback_cychdr コールバックです。関数名称はユーザ任意です。引数 cid は周期ハンドラ ID です。どの周期ハンドラ ID に割り当てられた周期ハンドラ関数かを判断することができます。
③ 周期ハンドラ処理を記述します。周期ハンドラ内では割込み部から発行可能なサービスコールを使用できます。

リスト5-20 周期ハンドラ記述方法

```
#include "slos.h"                              … ①

        :

void cychdr1(UB cid)                           … ②
{
        /* 周期ハンドラの処理 */                … ③
}
```

5.9.3　周期ハンドラの初期化

（1）OS 定義ファイルの” #define CYC_HDR”行を有効にします(無効にする場合は、コメントアウトしてください)。周期ハンドラを使用する場合、時間管理の初期化が必須となります(初期化の詳細は「5.5.2　時間管理の初期化」を参照ください)。

リスト5-21 周期ハンドラの有効設定 (slosdef.h)

```
/*------ configuration define  -----------------*/
       :
#define CYC_HDR
       :
```

（2）kinit コールバックに初期化関数 cyc_init サービスコール、VCYC_ATTR サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されております)。周期ハンドラを使用する場合、時間管理の初期化が必須となります(初期化の詳細は「5.5.2　時間管理の初期化」を参照ください)。

リスト5-22 周期ハンドラの初期化(kinit.c)

```
/* -------- cyclic handler -------- */
extern void cychdr1(UB cid);

void kinit(void)
{
          :
    /* cyclic handler initialize  */
    vcyc_init();
    VCYC_ATTR((ID)1u, (VCYC_TA_NON), cychdr1, 1000L);
          :
}
```

（3）ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.10　周期タイマハンドラの初期化

uinit コールバックに周期タイマハンドラに利用するタイマモジュールを初期化します。使用するタイマモジュールはユーザ任意です。初期化にて一定周期のタイマ割込みを発生させてください。

作成した周期タイマハンドラの周期時間をコンフィギュレーションファイルにて設定する必要があります。詳細は「6.4.3 周期タイマハンドラ周期時間の設定」を参照ください。

リスト5-23 周期タイマハンドラの初期化(user.c)

```
void uinit(void)
{
          :

    /* cyclic timer initialize */

          :
}
```

5.11　周期タイマハンドラの作成

「5.10 周期タイマハンドラの初期化」にて初期化したタイマモジュールの周期割込みを、vdisp 有割込みハンドラとして登録します。vdisp 有割込みハンドラの詳細は「5.4.4 vdisp 有割込みハンドラ」を参照ください。作成した vdisp 有割込みハンドラにて、ivsig_tim サービスコールを発行します。割込み終了処理等が必要となります。

リスト5-24 周期タイマハンドラの例(user.c)

```
/*
 * timer handler
 */
void intTim(UW irqid, H mode)
{
    ivsig_tim();

    INTC_ICCEOIR_ADDR = irqid;

    vdisp(mode);
}
```

※ 本サンプルをコメントアウトした状態で user.c に記載しております。

6. 構築

6.1 ファイル･ディレクトリ構成と操作

以下に Smalight OS インストール後の主要なファイル構成およびユーザによる編集可否を示します。インストールパスはデフォルト("<システムドライブ>:¥smalight")を想定しています。

表6-1 ファイル・ディレクトリ構成

ディレクトリ／ファイル	説明	ユーザ編集可否
〈Smalight 〉	—	－
├ 〈os〉	OS 格納	－
│ └ 〈RZA_v*vvv*/〉	RZ/A 用	－
│ ├ 〈armcc〉	ARM コンパイラ(armcc)用	－
│ ├ ├ 〈smalight〉	OS ライブラリ格納	－
│ │ │ └ smalight-os.a	OS ライブラリ	不可
│ │ ├ 〈smalight-os〉	OS 格納	－
│ │ │ ├ .cproject, .project	DS-5 ファイル(*1)	可
│ │ │ ├ 〈inc〉	OS ヘッダ・定義格納	－
│ │ │ │ ├ slos.h	OS ヘッダ〈C〉	不可
│ │ │ │ ├ slos.inc	OS ヘッダ〈ASM〉	不可
│ │ │ │ └ slosdef.h	OS 定義	可
│ │ │ ├ 〈sys〉	OS 本体ソース格納(*1)	－
│ │ │ └ 〈Release〉	OS ライブラリ生成・格納(*1)	－
│ │ └ 〈u-ap〉	ユーザアプリケーション格納	－
│ │ ├ .cproject, .project	DS-5 ファイル	可
│ │ ├ config.c	コンフィギュレーションファイル	可
│ │ ├ idle.c	アイドル処理	可
│ │ ├ kinit.c	OS 初期化処理	可
│ │ ├ main.c	OS 初期化コール処理	可
│ │ ├ stackini.c	タスクスタック初期化処理	可
│ │ ├ istack.c	割込みスタック定義	可
│ │ ├ osstack.s	OS スタック定義	可
│ │ ├ vcttbl.c	割込み ID テーブル定義	可
│ │ ├ vectors.s	ベクタテーブル	可
│ │ ├ vhandlerIRQ.s	割込みベクタハンドラ	不可
│ │ ├ vhandlerRES.s	リセットベクタハンドラ	可
│ │ ├ vhandler.s	ベクタハンドラ	可
│ │ ├ scatter.scat	スキャッタファイル	可
│ │ └ 〈src〉	ユーザアプリケーション格納	－
│ │ └ user.c	ユーザプログラム	可
│ └ 〈sample〉	サンプルプログラム格納	可
└ 〈doc〉	マニュアル格納	－

vvv: バージョン／リビジョン
l: 提供形態(s:ソース付属あり、r:ソース付属なし)
(*1) 提供形態によっては存在しない場合があります

6.2　構築手順

構築手順を以下に示します。

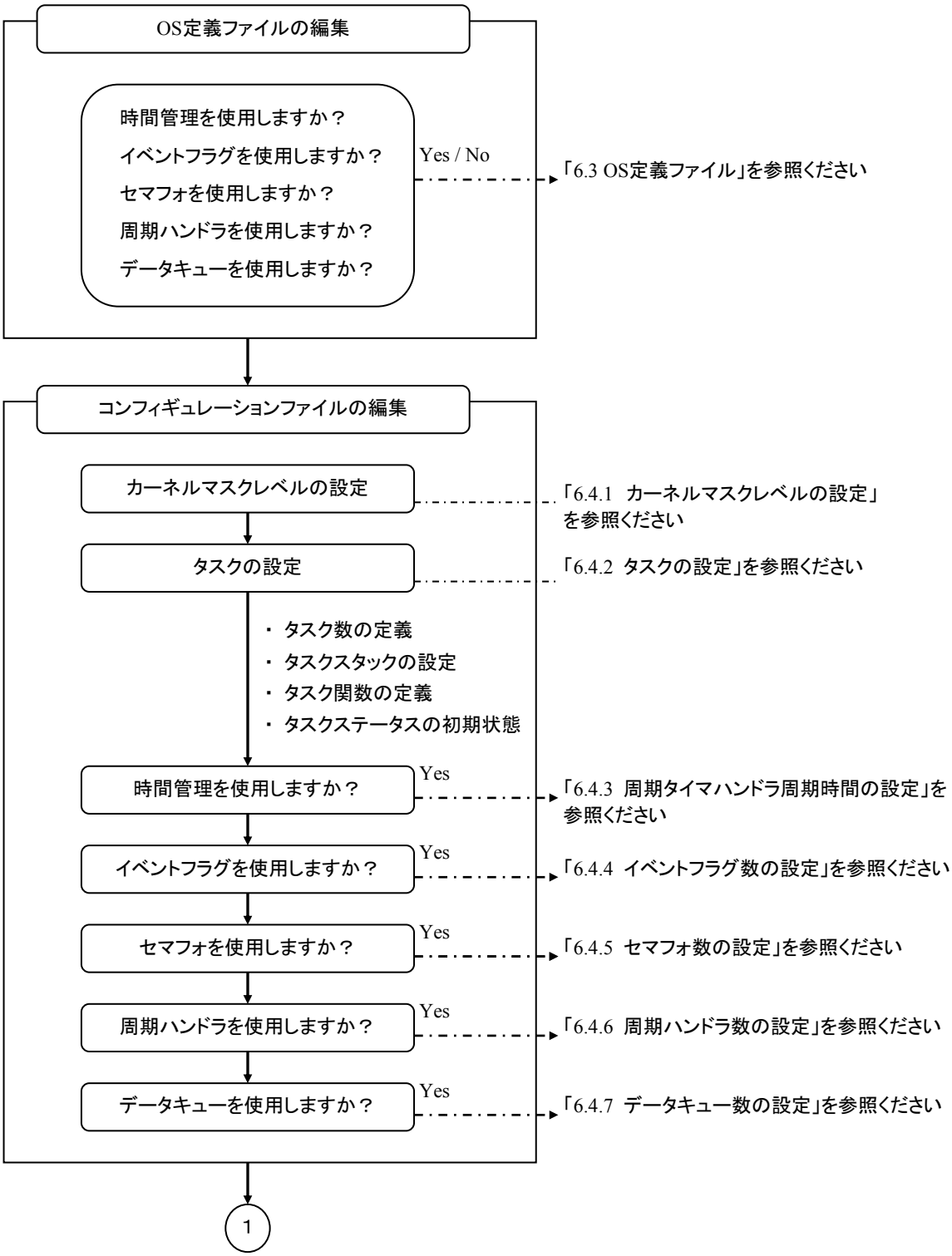

図6-1 構築手順①

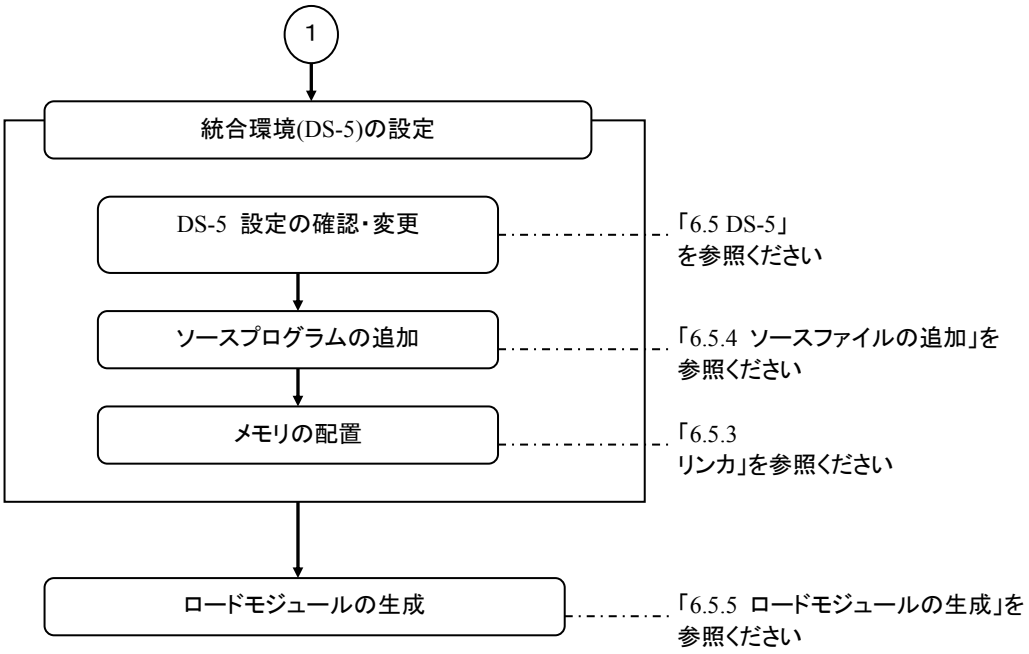
1
統合環境(DS-5)の設定
DS-5 設定の確認・変更
ソースプログラムの追加
メモリの配置
ロードモジュールの生成
「6.5 DS-5」
を参照ください
「6.5.4 ソースファイルの追加」を
参照ください
「6.5.3
リンカ」を参照ください
「6.5.5 ロードモジュールの生成」を
参照ください

図6-2 構築手順②

6.3　OS 定義ファイル

OS 定義ファイルでは、各機能の有効、無効設定を行います。設定可能な機能は、イベントフラグ、セマフォ、時間管理、周期ハンドラ、データキューです。無効に設定した場合、その機能は一切使用できなくなります。

無効にした場合、コンフィギュレーションファイル(config.c)の設定内容は無効となります。また、OS 初期化処理(kinit.c)に無効にした機能の初期化処理を記載してはいけません。その場合、ビルドでリンクエラーが発生します。

【解説】

① #define EVENTFLG
有効にするとイベントフラグの機能が有効となります。使用しない場合、コメントアウトしてください。

② #define SEMAPHORE
有効にするとセマフォが有効となります。使用しない場合、コメントアウトしてください。

③ #define SYSTIME
有効にすると時間管理が有効となります。使用しない場合、コメントアウトしてください。

④ #define CYC_HDR
有効にすると周期ハンドラが有効となります。使用しない場合、コメントアウトしてください。

⑤ #define DATAQUE
有効にするとデータキューが有効となります。使用しない場合、コメントアウトしてください。

リスト6-1 OS定義ファイルの設定(slosdef.h)

```
/*------ configuration define  ------------------*/
//#define EVENTFLG                                          … ①
//#define SEMAPHORE                                         … ②
//#define SYSTIME                                           … ③
//#define CYC_HDR                                           … ④
//#define DATAQUE                                           … ⑤
```

6.4　コンフィギュレーションファイル

6.4.1　カーネルマスクレベルの設定

コンフィギュレーションファイルでカーネルマスクレベルを設定してください。

【解説】

① カーネルマスクレベル:KNL_MSK_LEVEL
　カーネルマスクレベルを設定してください。設定したカーネルマスクレベルによりOS動作中の割込みマスクが決定します。

リスト6-2 カーネルマスクレベルの設定(config.c)

```
/*------ Kernel Mask ----------------------------------------*/
#define KNL_MSK_LEVEL       14u     /* kernel mask level(30-0) */          … ①
```

表6-2 カーネルマスクレベル設定①

カーネルマスクレベル	設定内容
0	OS動作中の割込みマスクを0に設定します(ICCPMRレジスタの7-3ビットを00000に設定)
1	OS動作中の割込みマスクを1に設定します(ICCPMRレジスタの7-3ビットを00001に設定)
…	…
29	OS動作中の割込みマスクを29に設定します(ICCPMRレジスタの7-3ビットを11101に設定)
30	OS動作中の割込みマスクを30に設定します(ICCPMRレジスタの7-3ビットを11110に設定)

6.4.2　タスクの設定

コンフィギュレーションファイルでタスクの設定をしてください。設定項目を以下に示します。

- タスク数の定義
- タスクのスタックサイズの定義
- タスクスタックの実体の定義
- タスクスタックの終端アドレスの定義
- タスク関数の定義
- タスクステータスの初期状態

【解説】

② タスク数:KNL_TCB_NUM
タスク数を設定してください。設定値は 1～127 まで指定できます。

③ プライオリティタスクの数:KNL_TCB_PRI_NUM
プライオリティタスクの数を設定してください。設定値は 0～KNL_TCB_NUM です。
ローテーションタスクの数は、KNL_TCB_NUM － KNL_TCB_PRI_NUM で決定します。

④ タスク毎のスタックサイズ:TCBx_SIZE
タスク毎のスタックサイズを設定してください。タスク数用意してください。タスクのスタックサイズ算出方法は「6.6.1 タスクスタック」を参照ください。

⑤ タスク毎のスタック領域:tcb_stackx
タスク毎のスタック領域の実態が用意されます。タスク数用意してください。

⑥ タスク毎のスタックポインタ初期値:TCBspInit
タスク毎のスタックポインタ初期値を設定します。タスク数用意してください。

リスト6-3 タスクの設定①(config.c)

```
/*------ TCB Number ------------------------------------------*/
#define KNL_TCB_NUM         3                                          … ②
#ifdef KNL_BB_PRIORITY
#define KNL_TCB_PRI_NUM     1                                          … ③
#endif
/*------ TCB Stack Size --------------------------------------*/
#define TCB1_SIZE       0x180u                                         … ④
#define TCB2_SIZE       0x180u
#define TCB3_SIZE       0x180u

#pragma section ustack
W tcb_stack1[(TCB1_SIZE/(UH)sizeof(W))];                               … ⑤
W tcb_stack2[(TCB2_SIZE/(UH)sizeof(W))];
W tcb_stack3[(TCB3_SIZE/(UH)sizeof(W))];

#pragma section
/*------ TCB Stack addr --------------------------------------*/
const TCBSP TCBspInit[KNL_TCB_NUM] = {                                 … ⑥
  {  &tcb_stack1[TCB1_SIZE/(UH)sizeof(W)]  },       /* TCB1 */
  {  &tcb_stack2[TCB2_SIZE/(UH)sizeof(W)]  },       /* TCB2 */
  {  &tcb_stack3[TCB3_SIZE/(UH)sizeof(W)]  }        /* TCB3 */
};
```

⑦ タスク毎の開始アドレス:TCBaddrInit
タスク毎の開始アドレスを設定します。関数の場合、外部参照の定義が必要です。タスク数用意してください。

⑧ タスク毎の初期タスク状態:TCBstInit
タスク毎の初期タスク状態を設定します。設定できる初期状態は、Ready 状態(CTCB_ST_RDY)、Dormant 状態(CTCB_ST_DMT)及び、Waiting 状態(起床待ち:CTCB_ST_SLP)です。タスク数用意してください。Dormant 状態で初期登録したタスクは、sta_tsk サービスコールで起動します。Waiting 状態で初期登録したタスクは wup_tsk、または iwup_tsk サービスコールにより起床させます。

リスト6-4 タスクの設定②(config.c)

```
/*------ TCB Start addr Init ---------------------------------*/
/*** Refer to exterior ***/
        extern void tsk01(void);                    /* TCB1   */
        extern void tsk02(void);                    /* TCB2   */
        extern void tsk03(void);                    /* TCB3   */
/*** Table for start addr init ***/

const TCBADDR TCBaddrInit[KNL_TCB_NUM] = {                          … ⑦
  { tsk01    },                                     /* TCB1   */
  { tsk02    },                                     /* TCB2   */
  { tsk03    }                                      /* TCB3   */
};

/*------ TCB Status Init -------------------------------------*/
/*
 * Task status Init : CTCB_ST_RDY | CTCB_ST_SLP | ...
 *                   ("slos.h" Refer to for details.)
 */
const UB TCBstInit[KNL_TCB_NUM] = {                                 … ⑧
    CTCB_ST_RDY,                                    /* TCB1   */
    CTCB_ST_RDY,                                    /* TCB2   */
    CTCB_ST_RDY                                     /* TCB3   */
};
```

6.4.3　周期タイマハンドラ周期時間の設定

コンフィギュレーションファイルで周期タイマハンドラ周期時間を設定してください。

【解説】

⑨ 周期タイマハンドラ周期時間:SYSTIM_CYCLIC_TIM
単位は msec です。周期タイマハンドラはユーザ任意作成する必要があります。詳細は「5.10 周期タイマハンドラの初期化」「5.11 周期タイマハンドラの作成」を参照ください。

リスト6-5 周期タイマハンドラ周期時間の設定例(config.c)

```
/*------ SYSTEM TIME ----------------------------------------*/
#ifdef SYSTIME
#define SYSTIM_CYCLIC_TIM   250uL        /* systim time(msec) */          … ⑨
#endif
```

6.4.4　イベントフラグ数の設定

コンフィギュレーションファイルでイベントフラグ数を設定してください。

【解説】

⑩ イベントフラグの数:EVFLG_NUM
イベントフラグ数を設定ください。設定値は 1～127 まで指定できます。イベントフラグ ID は 1,2,...n までとなります。

リスト6-6 イベントフラグ数の設定(config.c)

```
/*------ EVENTFLG -------------------------------------------*/
#ifdef EVENTFLG
#define EVFLG_NUM      1                 /* FLG Number        */          … ⑩
#endif
```

6.4.5　セマフォ数の設定

コンフィギュレーションファイルでセマフォ数を設定してください。

【解説】

⑪ セマフォの数:SEM_NUM
セマフォ数を設定ください。設定値は 1～127 まで指定できます。セマフォ ID は 1,2,...n までとなります。

リスト6-7 セマフォ数の設定(config.c)

```
/*------ SEMAPHORE ----------------------------------------*/
#ifdef SEMAPHORE
#define SEM_NUM         1                   /* Semaphore Number  */          … ⑪
#endif
```

6.4.6　周期ハンドラ数の設定

コンフィギュレーションファイルで周期ハンドラ数を設定してください。

【解説】

⑫ 周期ハンドラの数:CYC_NUM
周期ハンドラ数を設定ください。設定値は 1～127 まで指定できます。周期ハンドラ ID は 1,2,...n までとなります。

リスト6-8 周期ハンドラ数の設定(config.c)

```
/*------ CYCLIC HANDLER -----------------------------------*/
#ifdef CYC_HDR
#define CYC_NUM         1                   /* Cyclic Handler    */          … ⑫
#endif
```

6.4.7　データキュー数の設定

コンフィギュレーションファイルでデータキュー数を設定してください。

【解説】

⑬ データキューの数:DTQ_NUM
データキュー数を設定ください。設定値は 1～127 まで指定できます。データキューID は 1,2,...n までとなります。

リスト6-9 データキュー数の設定(config.c)

```
/*------ DATAQUE ------------------------------------------*/
#ifdef DATAQUE
#define DTQ_NUM         1               /* DTQ Number       */         … ⑬
#endif
```

6.5 DS-5

本 OS は ARM Development Studio 5(DS-5)用のプロジェクトで提供されます。
以下、コンパイル、アセンブルおよびリンク時の主要な設定について記載します。また、説明されない詳細な設定については、提供されるプロジェクトおよび関連するマニュアルを参照ください。

6.5.1 C コンパイラ

以下が設定済みです。記載のない項目はデフォルトです。

表6-3 Cコンパイラの設定

項目	設定
ターゲットCPU	Cortex-A9
バイト順序	リトルエンディアン
命令セット	ARM
ターゲットFPU	vfpv3_fp16
インクルードパス	"${workspace_loc}/smalight-os/inc"
最適化レベル	最少
デバッグ形式	DWARF 2

6.5.2 アセンブラ

以下が設定済みです。記載のない項目はデフォルトです。

表6-4 アセンブラの設定

項目	設定
ターゲットCPU	Cortex-A9
バイト順序	リトルエンディアン
命令セット	ARM
ターゲットFPU	vfpv3_fp16
インクルードパス	"${workspace_loc}/smalight-os/inc"
デバッグ形式	DWARF 2

6.5.3 リンカ

(1) リンカ設定

以下が設定済みです。記載のない項目はデフォルトです。

表6-5 リンカの設定

項目	設定
ターゲットCPU	Cortex-A9
ターゲットFPU	vfpv3_fp16
ライブラリ	"${workspace_loc}/smalight/smalight-os.a"

(2) メモリ属性と配置

デフォルトで以下が設定されております(scatter.scat)。デバイスに合わせて配置アドレスを設定してください。また、配置名に変更がある場合は、ユーザの責任で設定を変更してください。

表6-6 属性と配置名

属性	配置名	説明
RAM	Bistack	割込みスタック
	Bustack	ユーザタスクスタック
	Boss	OSスタック
	BsmalightosW	OSワーク領域
	BsmalightosH	
	BsmalightosB	
	RW, ZI	ユーザプログラムワーク領域
ROM	VECTORS	ベクタテーブル
	Cvectbl	割込みIDテーブル情報
	Cistack	割込みスタック情報
	HANDLERS	ベクタハンドラ
	Pvectbl	デフォルト割込みハンドラ
	Cosver	OSバージョン情報
	Psmalightos	OSプログラム
	Csmalightos	
	RO	ユーザプログラム

6.5.4 ソースファイルの追加

作成したアプリケーションのソースファイル（C ソース、ASM ソース）を、プロジェクトに追加します。

6.5.5 ロードモジュールの生成

プロジェクトをビルドしてください。

6.6　スタック使用量の算出

6.6.1　タスクスタック

タスクのスタックサイズの計算方法について説明します。スタックサイズは各関数のスタックサイズと関数のネストが関係します。下記に関数のスタック計算方法を示します。

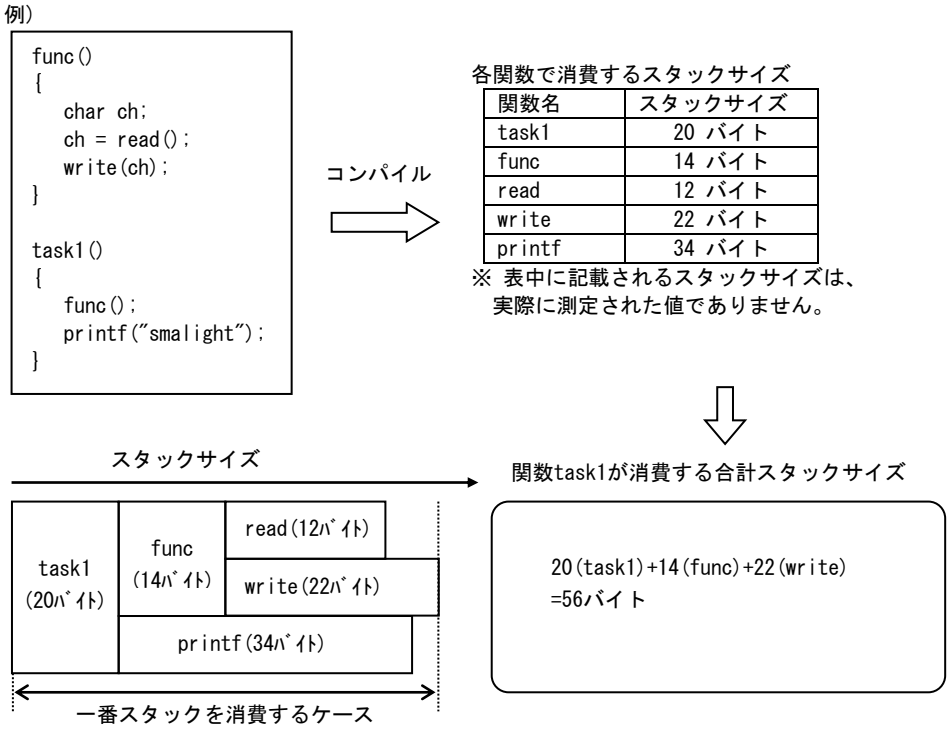

図6-3 関数のスタック計算方法

そこに OS 使用時のタスク最低スタックサイズを加算したものが、タスクスタックとして必要なサイズとなります。タスク最低スタックサイズとは、タスク実行中の割込み受け付けに必要なスタックサイズ、及び、サービスコール発行に必要なスタックサイズを考慮したものです。タスク最低スタックサイズは 400 バイト(サービスコールによるレジスタ退避分 200 バイト＋割込みベクタハンドラによるスタック使用 200 バイト)です。

各関数で消費するスタックサイズは、コンパイラオプション(--callgraph --info=stack など)を指定することで確認することができます。

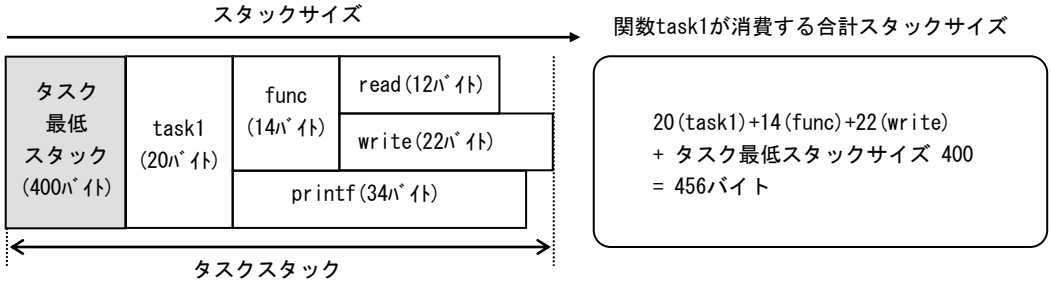

図6-4 タスクのスタック計算方法

6.6.2　割込みスタック

(1) vdisp 無割込みハンドラ

vdisp 無割込みハンドラのスタックについて説明します。下記は、vdisp 無割込みのスタック切り替えタイミングです。

① 割込み発生時のレジスタ退避は、割込み発生元のタスクスタック(アイドル状態の時は OS スタック、多重割込み時は発生元の割込みスタック)に退避されます。ベクタハンドラでレジスタ退避、割込みスタックへのスタック切り替えを行います。使用されるスタックは以下となります。

割込み発生元スタック ： 200 バイト
割込みスタック　　： 4 バイト

② vdisp 無割込みハンドラ本体は、割込みスタックを使用して処理されます。
③ vdisp 無割込みハンドラ本体の処理が完了すると、ベクタハンドラにリターンし、タスクスタック(アイドル状態の時は OS スタック、多重割込み時は発生元の割込みスタック)にスタック切り替えし、割込み発生元に戻ります。

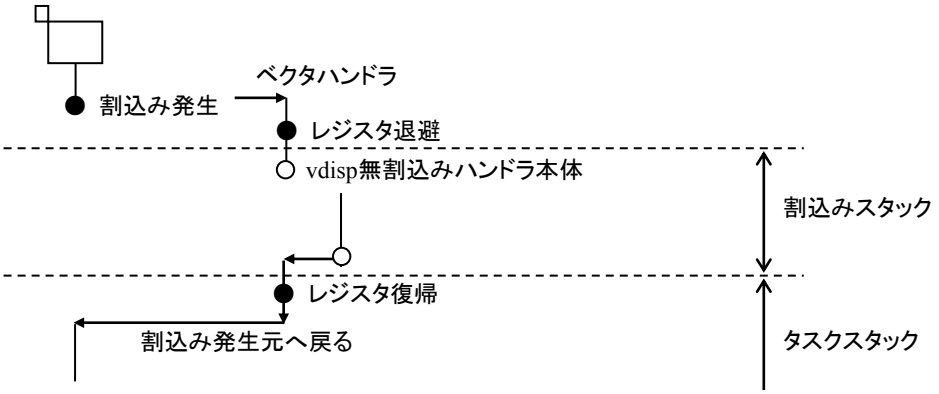

図6-5 vdisp無割込みハンドラのスタック切り替えタイミング

vdisp 無割込みスタックのスタックサイズは、vdisp 無割込みハンドラ本体で使用するスタックサイズ＋4 バイトとなります。

また、vdisp 無割込みハンドラ処理中に上位レベルの割込みを受け付ける場合は、その分の割込みスタック(200 バイト)を加味して確保してください。

各関数で消費するスタックサイズは、コンパイラオブション(--callgraph --info=stack など)を指定することで確認することができます。

(2) vdisp 有割込みハンドラ(周期タイマハンドラ以外)

vdisp 有割込みハンドラのスタックについて説明します。下記は、vdisp 有割込み発生からディスパッチャへ制御を移すまでのスタック切り替えタイミングです。

① 割込み発生時のレジスタ退避は、割込み発生元のタスクスタック(アイドル状態の時は OS スタック、多重割込み時は発生元の割込みスタック)に退避されます。ベクタハンドラでレジスタ退避、割込みスタックへのスタック切り替えを行います。使用されるスタックは以下となります。

割込み発生元スタック ： 200 バイト
割込みスタック ： 4 バイト

② vdisp 有割込みハンドラ本体(callback_int)は、割込みスタックを使用して処理されます。この割込みスタックは、vdisp 有割込みハンドラから発行される i 付きサービスコール処理でも使用されます。

③ vdisp サービスコールを発行すると OS スタックにスタック切り替えします。vdisp サービスコールを発行しない場合は、vdisp 有割込みハンドラ本体(callback_int)から、ベクタハンドラにリターンし、タスクスタック(アイドル状態の時は OS スタック、多重割込み時は発生元の割込みスタック)にスタック切り替えし、割込み発生元に戻ります。

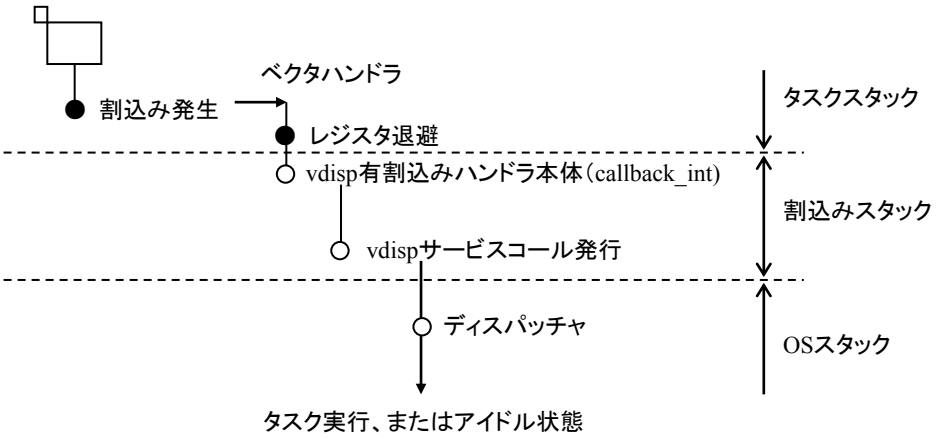

図6-6 vdisp有割込みハンドラのスタック切り替えタイミング

vdisp 有割込みスタックのスタックサイズは、vdisp 有割込みハンドラ本体(callback_int)で使用するスタックサイズ＋128(i 付きサービスコール使用分)＋4 バイトとなります。

また、vdisp 有割込みハンドラ処理中に処理中に上位レベルの割込みを受け付ける場合は、その分の割込みスタック(200 バイト)を加味して確保してください。

各関数で消費するスタックサイズは、コンパイラオブション(--callgraph --info=stack など)を指定することで確認することができます。

(3) 周期タイマハンドラ

周期タイマハンドラは vdisp 有割込みハンドラです。

周期タイマハンドラのスタックについて説明します。下記は、周期タイマハンドラからディスパッチャへ制御を移すまでのスタック切り替えタイミングです。

① 割込み発生時のレジスタ退避は、割込み発生元のタスクスタック(アイドル状態の時は OS スタック、多重割込み時は発生元の割込みスタック)に退避されます。ベクタハンドラでレジスタ退避、割込みスタックへのスタック切り替えを行います。使用されるスタックは以下となります。

割込み発生元スタック : 200 バイト
割込みスタック : 4 バイト

② 周期タイマハンドラ本体は、割込みスタックを使用して処理されます。この割込みスタックは、visig_tim サービスコール、ivsig_tim サービスコールによって OS からコールバックされる周期ハンドラ(callback_cychdr)および、周期ハンドラから発行される i 付きサービスコール処理でも使用されます。

③ vdisp サービスコールを発行すると OS スタックにスタック切り替えします。vdisp サービスコールを発行しない場合は、周期ハンドラ本体から、ベクタハンドラにリターンし、タスクスタック(アイドル状態の時は OS スタック、多重割込み時は発生元の割込みスタック)にスタック切り替えし、割込み発生元に戻ります。

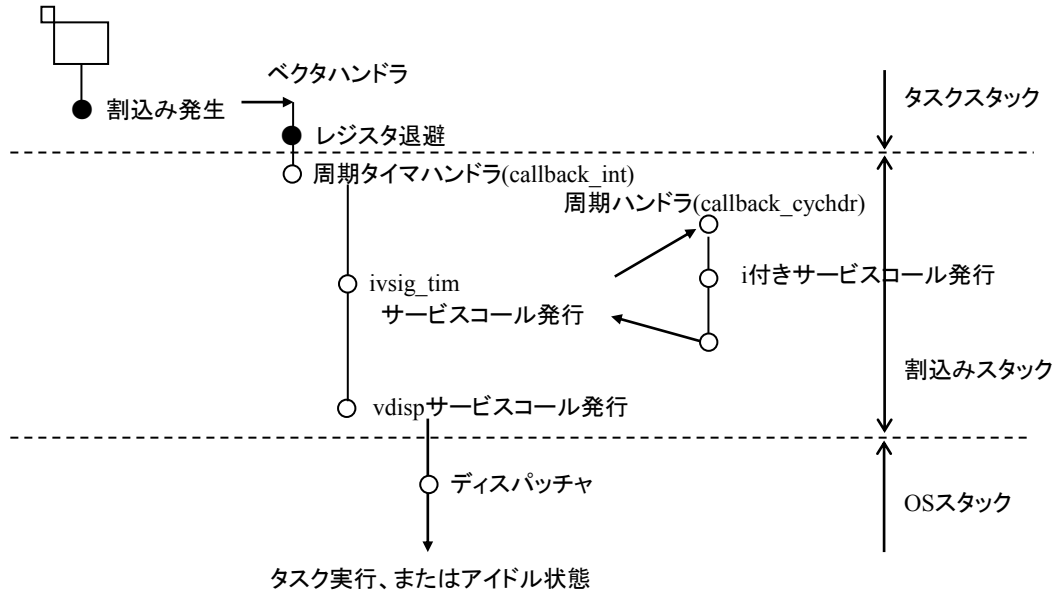

図6-7 周期タイマハンドラのスタック切り替えタイミング

周期タイマハンドラのスタックサイズは、周期タイマハンドラ本体で使用するスタックサイズ＋128(ivsig_tim サービスコール使用分)＋4 バイトに、周期ハンドラ(callback_cychdr)ID 毎に使用するサイズの内、最も大きいサイズを加算した値となります。なお、周期ハンドラ ID 毎のスタック使用サイズ算出の際、周期ハンドラで i 付きサービスコールを発行する場合には、128(i 付きサービスコール使用分)バイトを加算します。

また、vdisp 有割込みハンドラ処理中に処理中に上位レベルの割込みを受け付ける場合は、その分の割込みスタック(200 バイト)を加味して確保してください。

各関数で消費するスタックサイズは、コンパイラオプション(--callgraph --info=stack など)を指定することで確認することができます。

6.6.3 OS スタック

OS スタックは OS 動作中に使用されます。osstack.s にてデフォルトで確保されています。

kinit コールバックルーチン、uinit コールバックルーチン、stack_init コールバックルーチン、idle コールバックルーチンは、OS スタックで呼び出されます。その処理中に設定値以上のスタックを使用する場合は、OS スタック確保サイズを変更する必要があります。

7. サンプルプログラム

各種機能を使用したシンプルなサンプルプログラムを用意しました。各種機能の動作確認や、サービスコールのパラメータ指定(コーディング方法)の確認などにご利用ください。

表7-1 サンプルプログラム一覧

No	ディレクトリ名	内容	備考
1	<smalight>¥os¥RZA_v*vvv*/¥sample¥flg	イベントフラグの使用例	
2	<smalight>¥os¥RZA_v*vvv*/¥sample¥sem	セマフォの使用例	
3	<smalight>¥os¥RZA_v*vvv*/¥sample¥dtq	データキューの使用例	
4	<smalight>¥os¥RZA_v*vvv*/¥sample¥cyc	周期ハンドラの使用例	(*1)

vvv : バージョン／リビジョン
l : 提供形態(s:ソース付属あり、r:ソース付属なし)
(*1) 周期タイマハンドラを用意する必要があります(ユーザ任意)。

付録A　　変更履歴

【初版】
　　V4.00 をリリース

付録B　　制限事項

（1）Smalight OS ではプログラムサイズを抑える工夫のひとつとして、サービスコールのパラメータに対するエラーチェックなど OS 内部のチェックを最小限に留めております。そのため、構築の設定ミスやサービスコールにて不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。構築の設定やサービスコールのパラメータなど、使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

付録C　　NEON/VFP に関する注意事項

C.1　　NEON/VFPレジスタの有効化

タスクまたは割込みコンテキストとして NEON/VFP レジスタを退避/復帰のためにアクセスしますので、NEON/VFP レジスタはリセット処理などで有効化する必要があります。

C.2　　fpscrレジスタの初期化

fpscr レジスタの初期化は行われませんので、リセットから明示的に初期化するまで fpscr レジスタは不定値です。タスク初期起動時の fpscr レジスタの値は stackini.c にて設定します。

付録D　　スタック仕様

Ready, Dormant, Waiting, Suspended, Waiting + Suspended 状態のタスク及び、割込み発生時のユーザスタックは以下に示すスタック仕様で格納されています。

SP(R13)

	+0	+1	+2	+3
+H'00	D0			
+H'04				
+H'08	D1			
+H'0C				
+H'10	D2			
+H'14				
+H'18	D3			
+H'1C				
+H'20	D4			
+H'24				
+H'28	D5			
+H'2C				
+H'30	D6			
+H'34				
+H'38	D7			
+H'3C				
+H'40	D8			
+H'44				
+H'48	D9			
+H'4C				
+H'50	D10			
+H'54				
+H'58	D11			
+H'5C				
+H'60	D12			
+H'64				
+H'68	D13			
+H'6C				
+H'70	D14			
+H'74				
+H'78	D15			
+H'7C				
+H'80	fpscr			
+H'84	ICCPMR			

右上(↗)

	+0	+1	+2	+3
+H'88	R0			
+H'8C	R1			
+H'90	R2			
+H'94	R3			
+H'98	R4			
+H'9C	R5			
+H'A0	R6			
+H'A4	R7			
+H'A8	R8			
+H'AC	R9			
+H'B0	R10			
+H'B4	R11			
+H'B8	R12			
+H'BC	R14(lr)			
+H'C0	R15(pc)			
+H'C4	cpsr			

図D-1 スタック仕様

付録E　　データ型、リターンコード

Smalight で規定するデータ型の一覧を以下に示します。

表E-1 データ型一覧

型	意味
B	符号付き8ビット整数
H	符号付き16ビット整数
W	符号付き32ビット整数
UB	符号無し8ビット整数
UH	符号無し16ビット整数
UW	符号無し32ビット整数
FP	void型関数へのポインタ
VB	符号付き8ビット整数
VH	符号付き16ビット整数
VW	符号付き32ビット整数
ER	符号付き32ビット整数
ID	符号無し8ビット整数
FLGPTN	符号無し32ビット整数
MODE	符号無し32ビット整数
TMO	符号付き32ビット整数
VP_INT	符号付き32ビット整数
ER_UINT	符号付き32ビット整数

サービスコールのリターンコード一覧を以下に示します。

表E-2 リターンコード一覧

シンボル	値	内容
E_OK	0 (H'00000000)	正常終了
E_ID	-18 (H' FFFFFFEE)	不正ID番号
E_ILUSE	-28 (H'FFFFFFE4)	サービスコール不正使用
E_OBJ	-41（H'FFFFFFD7）	状態不正
E_TMOUT	-50 (H' FFFFFFCE)	ポーリング失敗またはタイムアウト

Smalight®
Smalight® OS V4 リファレンスマニュアル RZ/A版

発行年月　　2015年10月　　初版
発　　行　　ルネサス セミコンダクタ パッケージ&テスト ソリューションズ株式会社
編　　集　　ルネサス セミコンダクタ パッケージ&テスト ソリューションズ株式会社