

FR ファミリ

μT-Kernel 仕様準拠

SOFTUNE™ μT-REALOS/FR

ユーザーズガイド

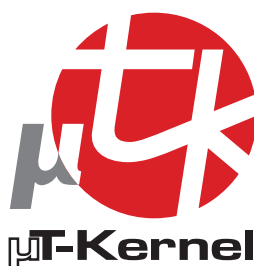


FR ファミリ

μT-Kernel 仕様準拠

SOFTUNE™ μT-REALOS/FR

ユーザーズガイド



富士通マイクロエレクトロニクス株式会社

はじめに

■ 本書の目的と対象読者

本書は、SOFTUNE μ T-REALOS/FR(以降、「 μ T-REALOS」とよびます)を使用して、アプリケーションプログラムを作成する方を対象にしており、 μ T-REALOS の機能全般、アプリケーションプログラムの作成方法、システム構築手順について説明しています。

本書を読むに当たっては、FR のプロセッサに関する基本的な知識とリアルタイム OS に関する基本的な知識が必要です。

また、システムコールインタフェースの詳細は「SOFTUNE μ T-REALOS/FR API リファレンス」(以降、「API リファレンス」とよびます)を、アナライザの詳細は「SOFTUNE μ T-REALOS/FR アナライザガイド」(以降、「アナライザガイド」とよびます)を参照してください。

■ μ T-Kernel について

μ T-Kernel仕様は、T-Engineフォーラムが策定したオープンなリアルタイムOSの仕様です。 μ T-Kernelの仕様書は、T-Engineフォーラムのホームページ(<http://www.t-engine.org/>)から入手できます。 μ T-Kernelの著作権は、坂村健氏に属しています。 μ T-Kernel仕様の著作権は、T-Engineフォーラムに属しています。本製品は、T-Engineフォーラム(www.t-engine.org)の μ T-Licenseに基づき μ T-Kernelソースコードを利用しています。

■ 商標

SOFTUNE は富士通マイクロエレクトロニクス株式会社の商標です。

REALOS は富士通マイクロエレクトロニクス株式会社の商標です。

TRON は、"The Real-time Operating system Nucleus" の略称です。

ITRON は、"Industrial TRON" の略称です。

μ ITRON は、"Micro Industrial TRON" の略称です。

T-Kernel および μ T-Kernel は、コンピュータの仕様に対する名称であり、特定の商品ないしは商品群を指すものではありません。

その他の記載されている社名および製品名などの固有名詞は、各社の商標または登録商標です。

■ 本書の全体構成

本書は、以下に示す 5 つの章と付録から構成されています。

第 1 章 μ T-REALOS の概要

μ T-REALOS の概要について説明します。

第 2 章 μ T-REALOS カーネルの 基本概念

μ T-REALOS カーネルを理解するうえで、あらかじめ理解しておくべき基本的な概念について説明します。

第 3 章 μ T-REALOS の機能

μ T-REALOS がサポートしている機能について説明します。

第 4 章 ユーザプログラムの作成

μ T-REALOS 上でユーザプログラムを作成するときの基本的な事項について説明します。

第 5 章 システム構築方法

ユーザシステムの構築方法について説明します。

付録

コンフィギュレータのエラーメッセージについて説明します。

■ 参考マニュアル

本システムを使用するときには、必要に応じて次に示すマニュアルを参照してください。

『SOFTUNE μ T-REALOS/FR API リファレンス』

『SOFTUNE μ T-REALOS/FR アナライザガイド』

『FR ファミリ SOFTUNE C/C++ コンパイラマニュアル V6』

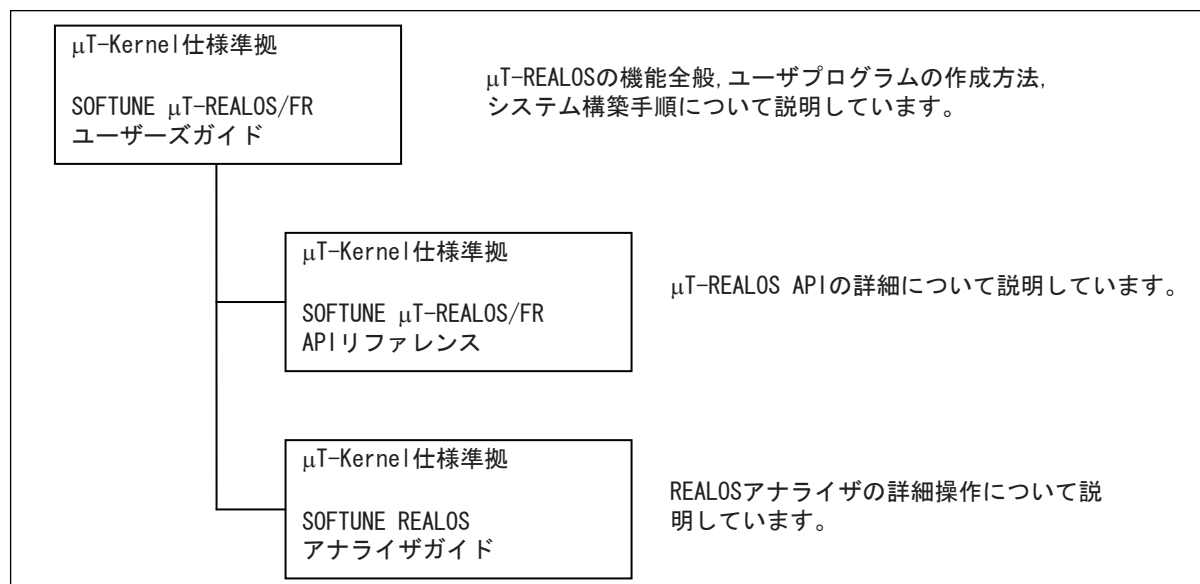
『FR ファミリ SOFTUNE アセンブラマニュアル V6』

『FR ファミリ SOFTUNE リンケージキットマニュアル V6』

■ μT-REALOS のマニュアル体系

μT-REALOS のマニュアルは、以下の 3 分冊になっています。

μT-REALOS をはじめてお使いになる方は、最初に『SOFTUNE μT-REALOS/FR ユーザーズガイド』をお読みください。



■ 本書の読み方

● 用語の説明

以下に本書で使用する用語を説明します。

用語	概要
カーネル	OS 機能を提供するプログラムをカーネルとよびます。
ユーザプログラム	μ T-REALOS の機能を使用したアプリケーションプログラムを指します。ユーザ作成である点を明示するため、本書は以降、ユーザプログラムと表記します。
ユーザシステム	ユーザプログラムと μ T-REALOS をリンクした実行形式のプログラムを指します。
システムコール	OS 機能を実現し、ユーザプログラムから直接呼び出される関数群をシステムコールとよびます。
オブジェクト	カーネルが操作対象とする資源をオブジェクトとよびます。具体的には、タスクや同期・通信機能を実現するセマフォ、メールボックスなどが該当します。
コンフィギュレーション規定マクロ	システムコンフィギュレーションファイル内に記述される、カーネル構築用のパラメタを設定するインタフェースです。
アイドル状態	実行可能タスクが存在しない状態です。

- 本資料の記載内容は、予告なしに変更することがありますので、ご用命の際は営業部門にご確認ください。
- 本資料に記載された動作概要や応用回路例は、半導体デバイスの標準的な動作や使い方を示したもので、実際に使用する機器での動作を保証するものではありません。したがって、これらを使用するにあたってはお客様の責任において機器の設計を行ってください。これらの使用に起因する損害などについては、当社はその責任を負いません。
- 本資料に記載された動作概要・回路図を含む技術情報は、当社もしくは第三者の特許権、著作権等の知的財産権やその他の権利の使用権または実施権の許諾を意味するものではありません。また、これらの使用について、第三者の知的財産権やその他の権利の実施ができることの保証を行うものではありません。したがって、これらの使用に起因する第三者の知的財産権やその他の権利の侵害について、当社はその責任を負いません。
- 本資料に記載された製品は、通常の産業用、一般事務用、パーソナル用、家庭用などの一般的用途に使用されることを意図して設計・製造されています。極めて高度な安全性が要求され、仮に当該安全性が確保されない場合、社会的に重大な影響を与えかつ直接生命・身体に対する重大な危険性を伴う用途（原子力施設における核反応制御、航空機自動飛行制御、航空交通管制、大量輸送システムにおける運行制御、生命維持のための医療機器、兵器システムにおけるミサイル発射制御をいう）、ならびに極めて高い信頼性が要求される用途（海底中継器、宇宙衛星をいう）に使用されるよう設計・製造されたものではありません。したがって、これらの用途にご使用をお考えのお客様は、必ず事前に営業部門までご相談ください。ご相談なく使用されたことにより発生した損害などについては、責任を負いかねますのでご了承ください。
- 半導体デバイスはある確率で故障が発生します。当社半導体デバイスが故障しても、結果的に人身事故、火災事故、社会的な損害を生じさせないよう、お客様は、装置の冗長設計、延焼対策設計、過電流防止対策設計、誤動作防止設計などの安全設計をお願いします。
- 本資料に記載された製品を輸出または提供する場合は、外国為替及び外国貿易法および米国輸出管理関連法規等の規制をご確認の上、必要な手続きをおとりください。
- 本書に記載されている社名および製品名などの固有名詞は、各社の商標または登録商標です。

Copyright© 2008 FUJITSU MICROELECTRONICS LIMITED All rights reserved.

Copyright© 2006 T-Engine Forum. All rights reserved.

このマニュアルは T-Engine フォーラムの許諾を得て μ T-Kernel の仕様書に基づいて作成しています。

目次

第 1 章	μT-REALOS の概要	1
1.1	サポート機能	2
1.2	提供ファイル構成	3
1.3	開発に必要なツール	4
1.4	製品構成	5
第 2 章	μT-REALOS カーネルの基本概念	7
2.1	システムコール	8
2.2	ユーザプログラムの実行単位	9
2.2.1	タスク	10
2.2.2	初期ルーチン	14
2.2.3	割込みハンドラ	15
2.2.4	タイムイベントハンドラ	16
2.2.5	エラールーチン	18
2.2.6	拡張 SVC ハンドラ	19
2.2.7	デバイス処理関数	20
2.3	オブジェクト	21
2.4	システム状態	22
2.5	ディスパッチおよび割込みの禁止 / 許可	24
2.6	タスク / ハンドラの実行優先順位	25
第 3 章	μT-REALOS の機能	27
3.1	μT-REALOS の機能概要	28
3.2	タスク管理機能	29
3.3	タスク付属同期機能	30
3.4	同期・通信機能	31
3.4.1	セマフォ機能	32
3.4.2	イベントフラグ機能	34
3.4.3	メールボックス機能	35
3.5	拡張同期・通信機能	37
3.5.1	ミューテックス機能	38
3.5.2	メッセージバッファ機能	40
3.5.3	ランデブポート機能	42
3.6	メモリプール管理機能	45
3.6.1	固定長メモリプール機能	46
3.6.2	可変長メモリプール機能	47
3.7	時間管理機能	48
3.7.1	システム時刻管理機能	49
3.7.2	周期ハンドラ機能	50
3.7.3	アラームハンドラ機能	52
3.8	割込み管理機能	53
3.9	システム状態管理機能	54
3.10	サブシステム管理機能	55
3.11	デバイス管理機能	56
3.12	省電力機能	58

3.13	コンフィギュレーション機能	59
3.14	デバッグ支援機能	64
第 4 章	ユーザプログラムの作成	73
4.1	ユーザプログラムの構成	74
4.2	起動のながれ	75
4.3	リセットエントリルーチン	76
4.4	初期ルーチン	79
4.5	タスク	81
4.6	周期ハンドラ	85
4.7	アラームハンドラ	86
4.8	割込みハンドラ	87
4.9	エラールーチン	89
4.10	省電力ルーチン	90
4.11	拡張 SVC ハンドラ	91
4.12	デバイスドライバ	92
4.13	ユーザプログラム作成時の注意事項	95
第 5 章	システム構築方法	97
5.1	システム構築手順	98
5.2	μT-REALOS のプロジェクト作成	99
5.3	コンフィギュレーションの設定	102
5.4	リンカオプションの設定	109
5.5	ユーザシステムのビルド	114
付録	115
付録 A	コンフィギュレータのエラーメッセージ	116
索引	129

第1章

μ T-REALOS の概要

μ T-REALOS の概要について説明します。

μ T-REALOS は、32 ビット RISC コントローラ FR ファミリで動作する μ T-Kernel 仕様のリアルタイム OS です。

μ T-REALOS は、 μ T-Kernel 仕様に準拠しています。

- 1.1 サポート機能
- 1.2 提供ファイル構成
- 1.3 開発に必要なツール
- 1.4 製品構成

1.1 サポート機能

μ T-REALOS においてサポートしている機能の概要について説明します。

■ サポート機能

μ T-REALOSでは、ライブラリとヘッダファイルを提供しています。ライブラリは、ユーザプログラムとリンクして対象プロセッサ上で動作可能な実行形式のプログラムを作成するために使用します。ヘッダファイルは、 μ T-Kernel の API を使用するため、ユーザプログラムから参照されます。ライブラリとヘッダファイルを合わせて μ T-REALOS カーネル（以降、「カーネル」とよびます）とよんでいます。カーネルは μ T-Kernel 仕様の以下の機能を実現するプログラムです。

- タスク管理機能
- タスク付属同期機能
- 同期・通信機能（セマフォ、イベントフラグ、メールボックス）
- 拡張同期・通信機能（メッセージバッファ、ミューテックス、ランデブポート）
- メモリプール管理機能（固定長メモリプール、可変長メモリプール）
- 時間管理機能
- 割込み管理機能
- システム構成管理機能
- サブシステム管理機能
- デバイス管理機能
- 省電力機能

上記機能の詳細については、「第 3 章 μ T-REALOS の機能」および「API リファレンス」の「第 3 章 システムコールインタフェース」を参照してください。

また、 μ T-REALOS では、システムのビルド時、デバッグ時に使用する以下の開発ツールを提供しています。これらは、PC 上で動作させる Windows のアプリケーションです。

- SOFTUNE μ T-REALOS コンフィギュレータ
- SOFTUNE μ T-REALOS アナライザ

SOFTUNE μ T-REALOS コンフィギュレータ（以降、コンフィギュレータとよびます）は、ユーザシステムのビルド時に使用して、あらかじめ指定された構成に基づき、カーネルを構築（コンフィギュレーション）します。コンフィギュレータの機能の詳細については、「3.13 コンフィギュレーション機能」を参照してください。

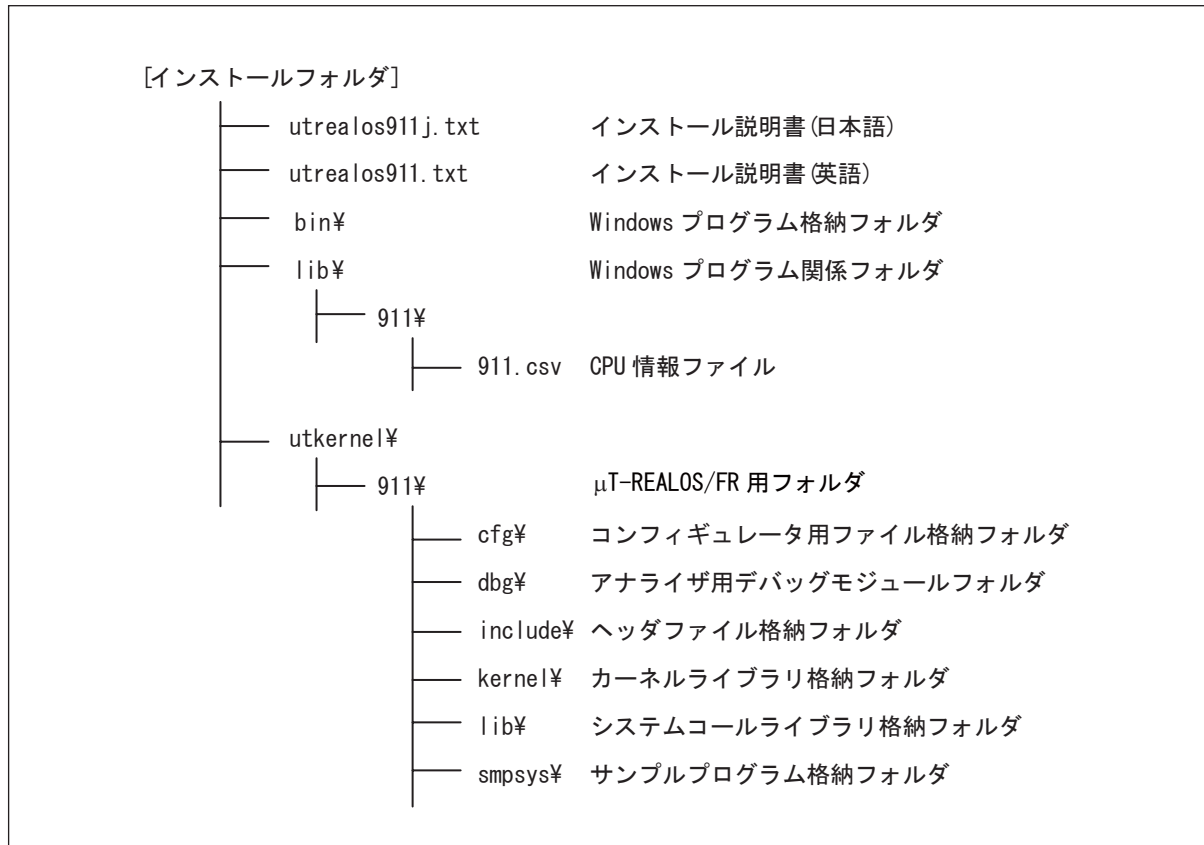
SOFTUNE μ T-REALOS アナライザ（以降、アナライザとよびます）はユーザプログラムのデバッグ時に使用し、デバッグを効率化させるためのさまざまな機能が含まれます。詳細については、「3.14 デバッグ支援機能」および「アナライザガイド」を参照してください。

1.2 提供ファイル構成

μ T-REALOS API の提供ファイル構成について説明します。

■ 提供ファイル構成

μ T-REALOS は以下のファイル構成でインストールされます。



詳しい構成については、「インストール説明書」を参照してください。なおインストール説明書は、製品 CD-ROM またはインストールフォルダに格納されています。

1.3 開発に必要なツール

ユーザシステムの開発に必要なツールについて説明します。

■ 開発に必要なツール

μ T-REALOS のユーザシステムの開発には以下のツールが必要です。

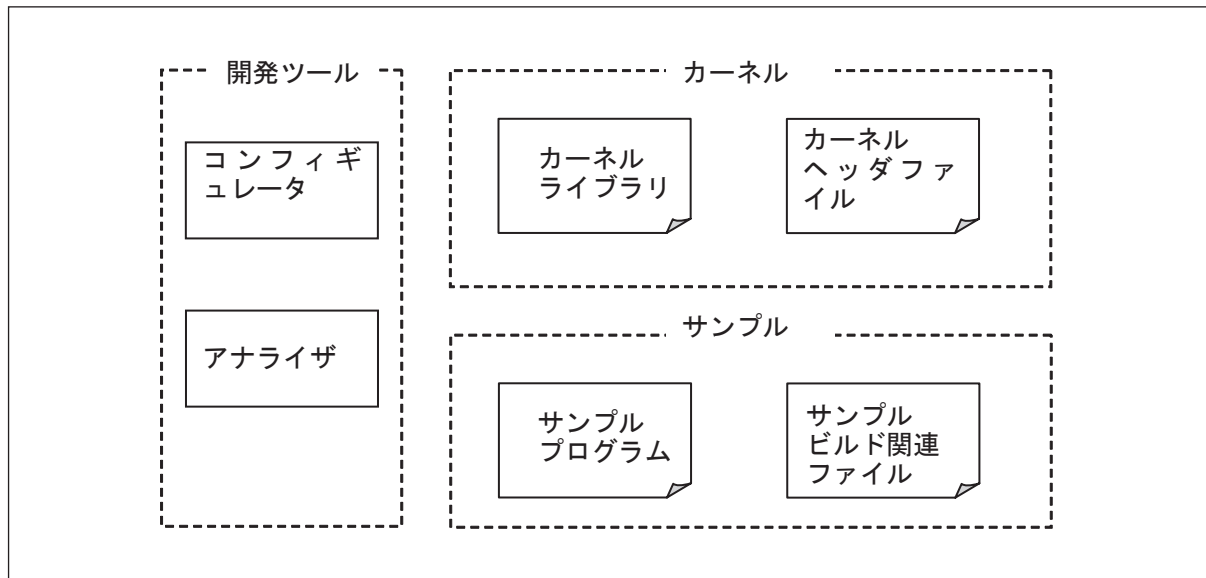
- クロス開発ツール
FR Family SOFTUNE Professional Pack V6
- ICE
富士通 MB2198 シリーズ

1.4 製品構成

製品構成について説明します。

■ 製品構成

μ T-REALOS の構成を以下に示します。



- **コンフィギュレータ**
Windows で動作させるコンフィギュレータのモジュールです。コマンドプロンプトの画面から起動するコマンド形式 (.exe) の実行ファイルと SOFTUNE Workbench にアドインして使用する DLL 形式のファイルを含みます。
- **アナライザ**
SOFTUNE Workbench にアドインして使用する DLL 形式のファイルおよびトレース採取のため、ユーザプログラムとリンクして使用するデバッグオブジェクトファイルです。
- **カーネルライブラリ**
 μ T-REALOS のカーネルのオブジェクトファイルが SOFTUNE のライブラリ形式で含まれています。
- **カーネルヘッダファイル**
ユーザプログラムからインクルードする、システムコールやその引数の型を定義したヘッダファイルです。
- **サンプルプログラム**
リセットエントリルーチン、初期化処理、タイマ割込みハンドラ、タスクのサンプルプログラムです。
- **サンプルビルド関連ファイル**
サンプルプログラム用の SOFTUNE プロジェクトファイル、コンフィギュレーションファイルなどです。

第2章

μ T-REALOS カーネルの 基本概念

μ T-REALOSカーネルを理解するうえで、あらかじめ理解しておくべき基本的な概念について説明します。

- 2.1 システムコール
- 2.2 ユーザプログラムの実行単位
- 2.3 オブジェクト
- 2.4 システム状態
- 2.5 ディスパッチおよび割込みの禁止 / 許可
- 2.6 タスク / ハンドラの実行優先順位

2.1 システムコール

ユーザプログラムから、カーネルの機能を使うためのインタフェースであるシステムコールについて説明します。

■ システムコール

汎用的なデータ型と定数マクロでは、ユーザプログラムから呼び出すカーネル機能のインタフェースをシステムコールとよびます。システムコールは、 μ T-Kernel 仕様に準拠しています。

システムコールの詳細については、「API リファレンス」の「第 3 章 システムコールインタフェース」を参照してください。

2.2 ユーザプログラムの実行単位

ユーザプログラムの実行単位について説明します。

■ ユーザプログラムの実行単位

ユーザプログラムの実行単位は、タスク、初期ルーチン、割込みハンドラ、タイムイベントハンドラ、エラールーチン、拡張 SVC ハンドラ、デバイスドライバ処理関数に大きく分けられます。

- タスク
- 初期ルーチン
- 割込みハンドラ
- タイムイベントハンドラ
- エラールーチン
- 拡張 SVC ハンドラ
- デバイスドライバ処理関数

2.2.1 タスク

タスクについて説明します。

■ タスク

タスクは、ユーザプログラムの処理の基本となるプログラム実行単位です。

μ T-REALOS では、タスクの実行が中断された場合、中断直前の状態(レジスタの値)をタスク単位に保存します。これをタスクコンテキストとよびます。このタスクコンテキストに保存された情報を使用して、中断されたタスクの実行を再開できます。

タスクは実行状態、実行可能状態、待ち状態など、さまざまな状態を持ちます。タスクの状態遷移については、「2.2.1 タスク」の「■ タスクの状態」を参照してください。

■ 自タスクと他タスク

タスクからシステムコールを呼び出した場合に、そのシステムコールを呼び出したタスクを自タスクとよびます。また、自タスク以外のタスクを他タスクとよびます。

■ 優先順位とタスク優先度

プログラム実行単位の実行順序を優先順位とよびます。また、タスクの優先順位を決める値をタスク優先度とよびます。タスク優先度は、値が小さいほど優先度が高くなります。タスク優先度が高いタスクほど優先して実行します。

タスク優先度には、ベース優先度、現在優先度および起動時優先度があります。単にタスク優先度といった場合には、現在優先度を指します。現在優先度は、タスクの実行順序を決めるために使われます。ベース優先度は、タスクの基本となる優先度で、通常は現在優先度と同じ値です。ただし、ミューテックス機能を使った場合、一時的に現在優先度が変更されて、ベース優先度と異なる場合があります。この場合でも、ミューテックス機能の使用が終わったときに、変更された現在優先度はベース優先度に戻します(「3.5.1 ミューテックス機能」参照)。起動時優先度は、タスクの生成時に指定する優先度で、タスク起動時に、そのタスクのベース優先度を起動時優先度の値で初期化します。

■ ディスパッチとプリエンプト

実行状態のタスクが切り換わることをディスパッチとよびます。また、実行状態のタスクの実行権が奪われることをプリエンプトとよびます。なお、ディスパッチを実現するカーネル内の機構をディスパッチャとよびます。

ディスパッチは、実行中のタスクよりも優先度の高いタスクが実行可能状態になった場合に発生します。また、プリエンプトは、タスク実行中にディスパッチが発生した場合や、割り込みハンドラが起動された場合に発生します。

■ タスクの状態

タスクは、以下の状態を持ちます。

● 実行状態 (RUNNING)

タスクが実行中の状態。

ただし、タスク以外のプログラムを実行中は、そのプログラムの実行前に実行していたタスクが実行状態となります。

● 実行可能状態 (READY)

タスクが実行する準備が整っているが、そのタスクより優先順位の高いタスクが実行中のため、実行できない状態。

● 待ち状態 (WAITING)

何らかの条件を待つためのシステムコールを呼び出して実行を中断している状態。
待ち条件により以下の状態に分類されます。

- 起床待ち状態 (tk_slp_tsk による待ち)
- 時間経過待ち状態 (tk_dly_tsk による待ち)
- セマフォ資源の獲得待ち状態 (tk_wai_sem による待ち)
- イベントフラグ待ち状態 (tk_wai_flg による待ち)
- メールボックスからの受信待ち状態 (tk_rcv_mbx による待ち)
- ミューテックスのロック待ち状態 (tk_loc_mtx による待ち)
- メッセージバッファへの送信待ち状態 (tk_snd_mbf による待ち)
- メッセージバッファからの受信待ち状態 (tk_rcv_mbf による待ち)
- 固定長メモリブロックの獲得待ち状態 (tk_get_mpf による待ち)
- 可変長メモリブロックの獲得待ち状態 (tk_get_mpl による待ち)
- ランデブの呼出し / 終了待ち状態 (tk_cal_por による待ち)
- ランデブの受付け待ち状態 (tk_acp_por による待ち)

● 強制待ち状態 (SUSPEDED)

ほかのタスクから実行を強制的に中断されている状態。

● 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。

● 休止状態 (DORMANT)

タスクが起動されていない状態。または、タスクが終了した状態。

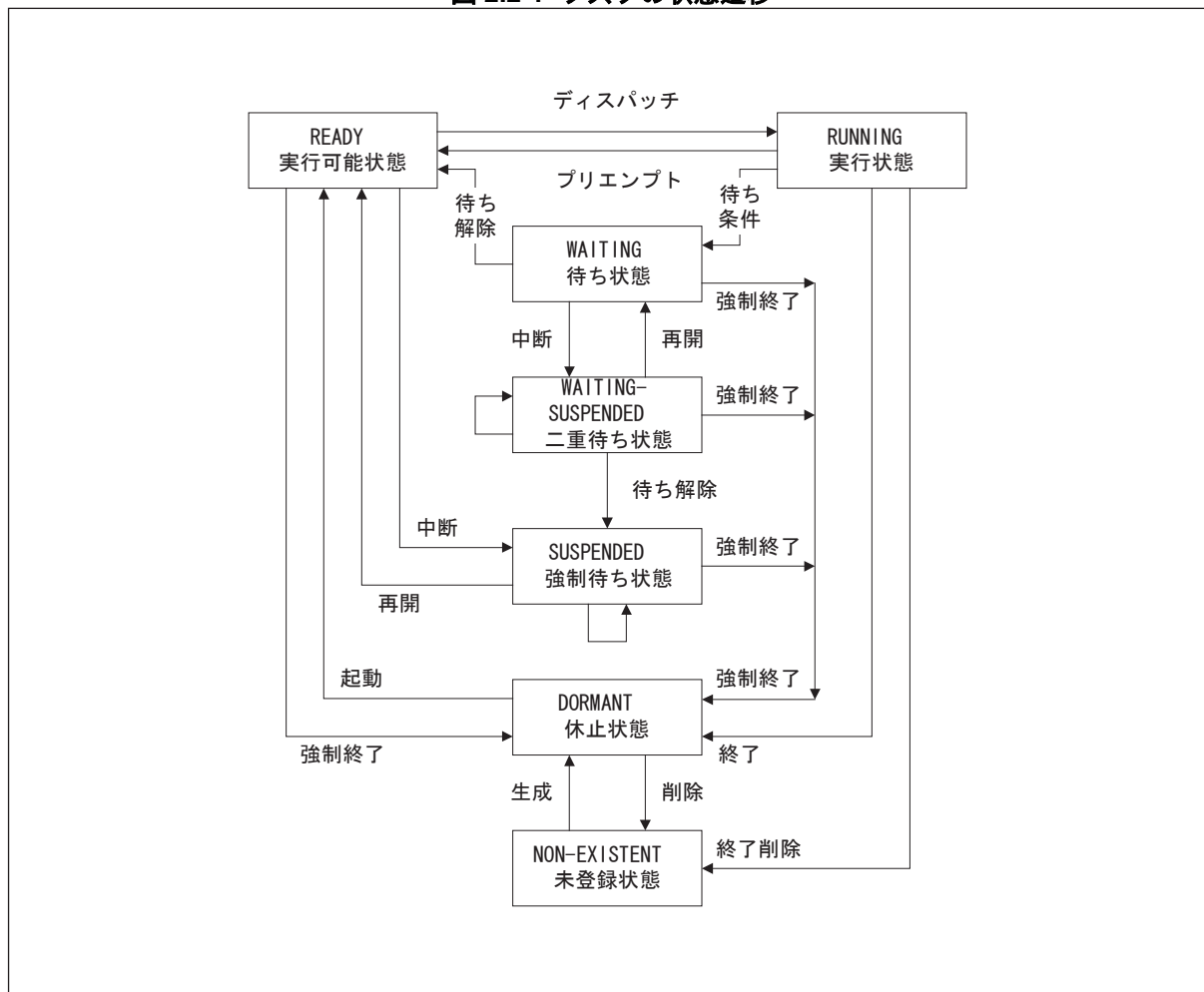
● 未登録状態 (NON-EXISTENT)

タスクが生成されていない状態。または、タスクが削除された状態。

■ タスクの状態遷移

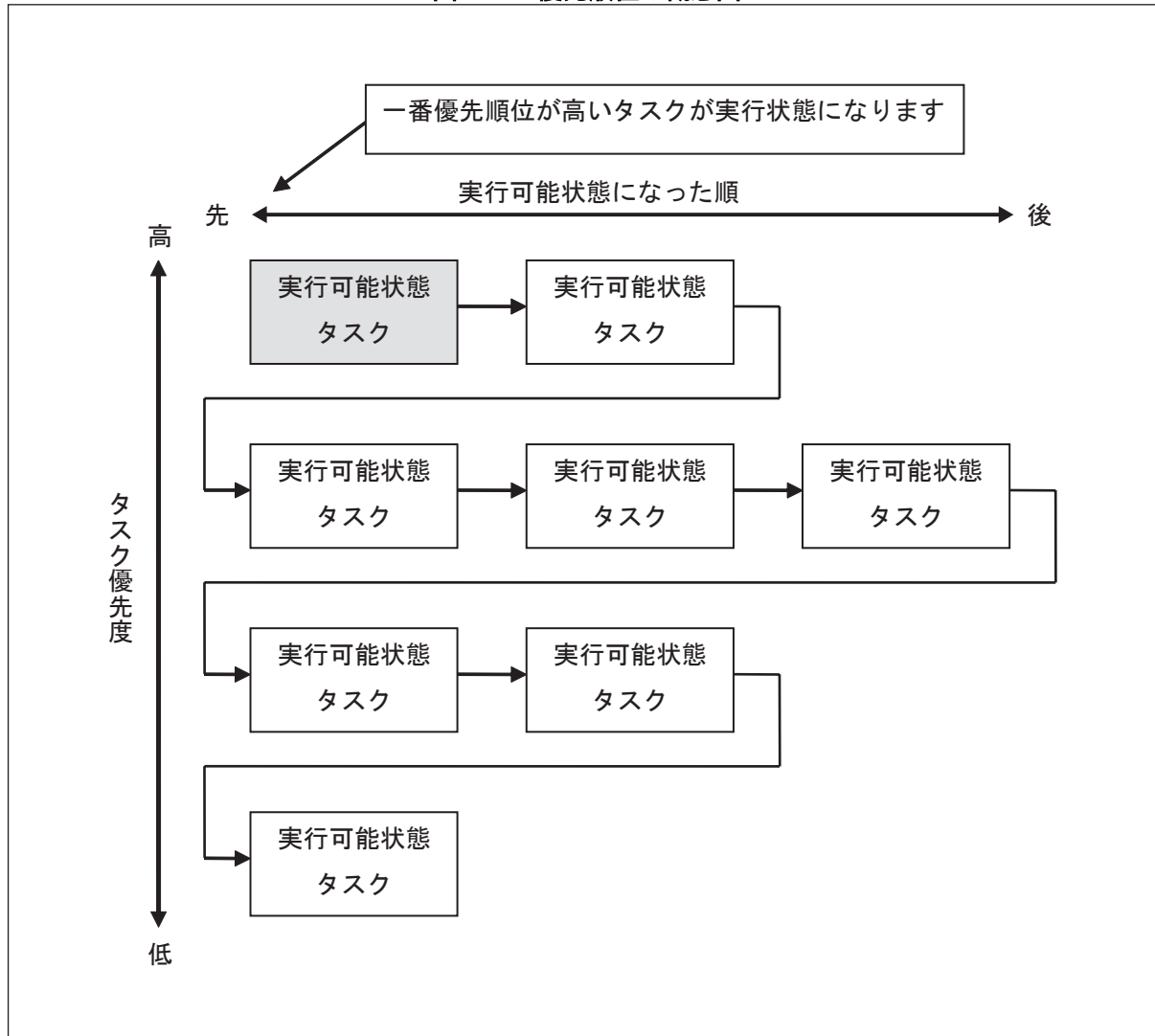
タスクの状態は、以下のように遷移します。

図 2.2-1 タスクの状態遷移



複数の実行可能状態タスクは、タスク優先度によってスケジューリング（実行順序を制御）します。実行可能状態のタスクの中で最も優先順位が高いタスクが実行状態となり、タスクを実行します。タスクの優先順位は、高いタスク優先度を持つタスクが、高い優先順位になります。同じタスク優先度のタスク間では、先に実行可能状態になったタスクが、高い優先順位になります。

図 2.2-2 優先順位の概念図



2.2.2 初期ルーチン

初期ルーチンについて説明します。

■ 初期ルーチン

初期ルーチンはユーザプログラム独自の初期化処理を行うプログラムで、一般的にタスク、セマフォなど、オブジェクトの生成、割込みハンドラ、デバイスの登録をして、ユーザプログラムが動作可能な環境を整えます。

カーネル初期化時は、初期タスクが自動的に生成されて、カーネル内部の初期化処理を行います。この初期タスクで、ユーザプログラムからあらかじめ定義された初期ルーチン呼び出します。このため、初期ルーチンはタスクとして実行されます。

2.2.3 割込みハンドラ

タスク独立部について説明します。

■ 割込みハンドラ

割込みハンドラは、周辺ハードウェアの割込み要因、CPU 例外またはソフトウェア割込み命令に同期して起動するプログラムです。割込み要因にそれぞれ定義できます。

タスク実行中に割込みが発生すると、カーネルはタスクのプログラムの実行を一時中断して、発生した割込み要因に対応する割込みハンドラを実行します。この場合、スタックは割込み処理実行用に用意されたスタック（システムスタック）に切り換わります。このため、割込みハンドラは、これまで実行していたタスクのコンテキストではなく、独立したコンテキストで実行します。

また、すべての割込みハンドラはタスクより優先して実行するため、割込みハンドラが終了するまでタスクは実行しません。割込みハンドラが多重に起動している場合も、すべての割込みハンドラの処理が終了するまでタスクは実行しません。このため、割込みハンドラからディスパッチを伴うシステムコール（優先度の高いタスクの起動など）を呼び出しても、実際にタスクにディスパッチするのは、すべての割込みハンドラの処理の終了後になります。この動作を「遅延ディスパッチ」とよびます。

割込みハンドラの詳細については、「3.8 割込み管理機能」を参照してください。

2.2.4 タイムイベントハンドラ

タイムイベントハンドラについて説明します。

■ タイムイベントハンドラ

周期ハンドラとアラームハンドラをあわせて、タイムイベントハンドラとよびます。

■ 周期ハンドラ

周期ハンドラは、指定した時間間隔で起動するプログラムです。周期的に実行するプログラムを周期ハンドラとして定義して、その実行や停止を制御できます。

タスク実行中に指定した周期になると、タスクの実行を一時中断して、対応する周期ハンドラを起動します。この場合、周期ハンドラは、これまで実行していたタスクのコンテキストではなく、独立したコンテキストで実行します。

また、周期ハンドラは、タスクより優先して実行するため、周期ハンドラが終了するまで、タスクは実行しません。このため、周期ハンドラからディスパッチを伴うシステムコール（優先度の高いタスクの起動など）を呼び出しても、実際にタスクにディスパッチするのは、周期ハンドラの処理の終了後になります。

なお、 μ T-REALOS の周期ハンドラは、システムクロック用のタイマ割込みハンドラから呼び出された `isig_tim` 内から起動します。このため、周期ハンドラはタイマ割込みハンドラの一部として動作します。このようにタイマ割込みハンドラから起動する時刻関係のハンドラを「タイムイベントハンドラ」とよびます。 μ T-REALOS では、周期ハンドラと次に説明するアラームハンドラを併せてタイムイベントハンドラとよんでいます。前に説明したとおり、周期ハンドラは、タイマ割込みハンドラの一部として実行するため、周期ハンドラ実行中に別のタイムイベントハンドラの処理が割り込むことはありません。

周期ハンドラの初回の起動時刻は、周期ハンドラを生成または起動した時刻の次のタイムティックを基準に計算します。ただし、タイムイベントハンドラで周期ハンドラを生成または起動した場合には、タイムイベントハンドラが起動した時刻を基準として計算します。初回以降の起動時刻は、周期ハンドラが起動した時刻を基準に計算します。

周期ハンドラ機能の詳細については、「3.7.2 周期ハンドラ機能」を参照してください。

■ アラームハンドラ

アラームハンドラは、指定した時間に起動するプログラムです。指定した時間に行うプログラムをアラームハンドラとして生成して、その実行や停止を制御できます。

タスク実行中に指定した時間になると、タスクのプログラムの実行を一時中断して、対応するアラームハンドラを実行します。この場合、アラームハンドラは、これまで実行していたタスクのコンテキストではなく、独立したコンテキストで実行します。

また、アラームハンドラは、タスクより優先して実行するため、アラームハンドラが終了するまで、タスクは実行しません。このため、アラームハンドラからディスパッチを伴うシステムコール（優先度の高いタスクの起動など）を呼び出しても、実際にタスクにディスパッチするのは、アラームハンドラの処理の終了後になります。

なお、 μ T-REALOS のアラームハンドラは、システムクロック用の割込みハンドラの一部として起動します。このため、アラームハンドラが実行中に別のタイムイベントハンドラの処理が割り込むことはありません。

アラームハンドラの起動時刻は、アラームハンドラを起動した時刻の次のタイムティックを基準に計算します。ただし、タイムイベントハンドラ内でアラームハンドラを起動した場合には、タイムイベントハンドラが起動した時刻を基準として計算します。

アラームハンドラ機能の詳細については、「3.7.3 アラームハンドラ機能」を参照してください。

2.2.5 エラールーチン

エラールーチンについて説明します。

■ エラールーチン

エラールーチンは、カーネルが何らかのエラーを検出したときに起動するプログラムです。

エラールーチンは、以下の条件で起動します。

- システムダウン
カーネル内部の矛盾を検出
- 初期設定エラー
カーネル初期化でエラー発生
- 未定義の割込み
割込みハンドラが定義されていない割込みが発生

エラールーチンは、ユーザプログラムのデバッグを目的として使用します。エラールーチンから、復帰できません。このため、エラールーチンが呼び出された場合、エラー原因を取り除き、システムを再起動してください。

エラールーチンが初期設定エラーで呼ばれた場合、タスクとして動作します。それ以外は、タスク独立部として動作します。

2.2.6 拡張 SVC ハンドラ

拡張 SVC ハンドラについて説明します。

■ 拡張 SVC ハンドラ

拡張 SVC ハンドラは、サブシステムに対する要求の受け口となるハンドラです。タスクから呼び出された場合には、準タスク部として動作します。タスク独立部から呼び出された場合には、タスク独立部として動作します。サブシステムについては、「3.10 サブシステム管理機能」を参照してください。

2.2.7 デバイス処理関数

デバイス処理関数について説明します。

■ デバイス処理関数

デバイス処理関数は、デバイス管理機能から呼び出されるデバイスドライバの関数です。デバイス処理関数は、タスクの延長で呼び出された場合には、タスク部として動作します。タスク独立部の延長で呼び出された場合には、タスク独立部として動作します。デバイス処理関数の詳細については、「3.11 デバイス管理機能」を参照してください。

2.3 オブジェクト

オブジェクトについて説明します。

■ オブジェクト

μT-REALOSでは、タスク間の同期/通信、排他制御、メモリ領域の獲得/解放などのさまざまな機能をサポートしています。これらの機能をユーザプログラムから使用するために、システムコールで操作対象となる資源をオブジェクトとよびます。

μT-REALOSでは、以下のオブジェクトが用意されています。各オブジェクトについては、表 2.3-1 の各説明を参照してください。

表 2.3-1 オブジェクト一覧

オブジェクト	概要
タスク	ユーザプログラムを構成する最も基本的な単位のオブジェクトです。
セマフォ	セマフォは、使用されていない資源の有無や数量を数値で表現することにより、その資源を使用する際の排他制御や同期を行うためのオブジェクトです。
イベントフラグ	イベントフラグは、イベントの有無をビットごとのフラグで表現することにより、同期を行うためのオブジェクトです。
メールボックス	メールボックスは、メモリ上に置かれたメッセージの受渡しにより、同期と通信を行うためのオブジェクトです。
ミューテックス	ミューテックスは、共有資源を使用するときにタスク間で排他制御を行うためのオブジェクトです。
メッセージバッファ	メッセージバッファは、可変長のメッセージの受渡しにより、同期と通信を行うためのオブジェクトです。
ランデブポート	ランデブポート機能は、タスク間で同期通信を行うための機能で、あるタスクから別のタスクへの処理依頼と、後者のタスクから前者のタスクへの処理結果の返却を、一連の手順としてサポートします。双方のタスクが待ち合わせするためのオブジェクトを、ランデブポートとよびます。
固定長メモリプール	固定長メモリプールは、固定されたサイズのメモリブロックを動的に管理するためのオブジェクトです。
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管理するためのオブジェクトです。
周期ハンドラ	周期ハンドラは、一定周期で起動するタイムイベントハンドラです。
アラームハンドラ	アラームハンドラは、指定した時刻に起動するタイムイベントハンドラです。

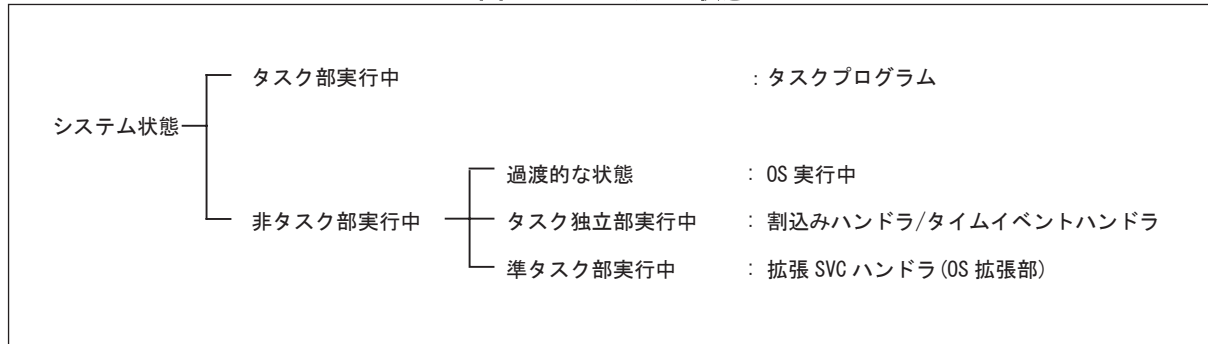
2.4 システム状態

システム状態について説明します。

■ システム状態

μ T-REALOS のシステム状態は、以下のように分類されます。

図 2.4-1 システム状態



■ タスク部実行中

「タスク部実行中」は、タスクのプログラムを実行している状態です。後述の「非タスク部実行中」に含まれる、OS(システムコール)を実行している状態やハンドラを実行している状態を含みません。

■ 非タスク部実行中

「非タスク部実行中」の状態は、さらに「過渡的な状態」、「タスク独立部実行中」、「準タスク部実行中」の 3 つの状態に分けられます。

(1) 「過渡的な状態」

「過渡的な状態」とは、 μ T-REALOS がシステムコールの処理を実行している状態です。

(2) 「タスク独立部実行中」

「タスク独立部実行中」の状態は、割込みハンドラやタイムイベントハンドラを実行している状態です。

(3) 「準タスク部実行中」

タスクから拡張 SVC ハンドラやデバイスドライバインタフェース関数を実行している状態です。

■ 呼び出すことのできるシステムコール

「タスク部」「準タスク部」では、`tk_ret_int` や `isig_tim` のような「タスク独立部」から呼び出されることを前提としたシステムコールを除き、すべてのシステムコールを呼び出せます。

それに対して、「タスク独立部」はタスクとは独立したコンテキストで実行され、タスクの概念がありません。このため、以下のようなシステムコールは呼び出すことができません。

- 明示的に自タスクを指定するシステムコール (`tskid` に `"TSK_SELF"` のマクロを指定するもの)
- 暗黙に自タスクを指定するシステムコール (待ち状態に移行するもの)

各システム状態で呼出し可能なシステムコールの詳細については、「API リファレンス」の「3.1 システムコール一覧」を参照してください。

■ ユーザプログラムとシステム状態

ユーザプログラムとシステム状態の関係を表 2.4-1 に示します。

表 2.4-1 各ユーザプログラムのシステム状態

ユーザプログラム名	システム状態
タスク	タスク部
拡張 SVC ハンドラ	非タスク部 (準タスク部, タスク独立部)
デバイスドライバ	非タスク部 (準タスク部)
周期ハンドラ	非タスク部 (タスク独立部)
アラームハンドラ	非タスク部 (タスク独立部)
割込みハンドラ	非タスク部 (タスク独立部)
エラールーチン	タスク部, 非タスク部 (タスク独立部)

< 注意事項 >

`isig_tim` およびエラールーチンは、μT-Kernel 仕様では規定されていません。μT-REALOS 独自の拡張機能です。

2.5 ディスパッチおよび割込みの禁止 / 許可

ディスパッチ禁止 / 許可状態および割込み禁止 / 許可状態について説明します。

■ ディスパッチ禁止 / 許可状態

ユーザプログラム実行中は、ディスパッチ禁止状態またはディスパッチ許可状態のいずれかの状態になっています。システム初期化後、最初のタスクが実行を開始した時点では、ディスパッチ許可状態になっています。

ディスパッチ禁止状態では、実行状態のタスクが切り換わることはありません（ディスパッチが起こりません）。ディスパッチ禁止状態で、実行中のタスクが待ち状態になる可能性のあるシステムコールを呼び出した場合には、エラー（E_CTX）となります。ただし、割込みハンドラや周期ハンドラ、アラームハンドラは起動します。

ディスパッチ禁止 / 許可状態は、ユーザプログラムから以下のシステムコールを呼び出すことにより制御できます。

- tk_dis_dsp : ディスパッチ禁止状態にする（ディスパッチを禁止する）
- tk_ena_dsp : ディスパッチ許可状態にする（ディスパッチを許可する）

■ 割込み禁止 / 許可状態

ユーザプログラム実行中は、割込み禁止状態または割込み許可状態のいずれかの状態になっています。システム初期化後、最初のタスクが実行を開始した時点では、割込み許可状態になっています。

割込み禁止状態では、PSレジスタのIフラグが"0"となり、外部割込みがすべて禁止され、ハードウェアで割込みが発生しても、割込みハンドラに制御が渡りません。また、ディスパッチも禁止されて、実行状態のタスクが切り換わることはありません（ディスパッチが起こりません）。割込み禁止状態で、実行中のタスクが待ち状態になる可能性のあるシステムコールを呼び出した場合には、エラー（E_CTX）となります。割込み禁止状態で、ディスパッチ禁止 / 許可のシステムコール（tk_dis_dsp/tk_ena_dsp）を呼び出した場合、エラー（E_CTX）となります。

割込み禁止 / 許可状態は、ユーザプログラムから以下のマクロを呼び出すことにより制御できます。

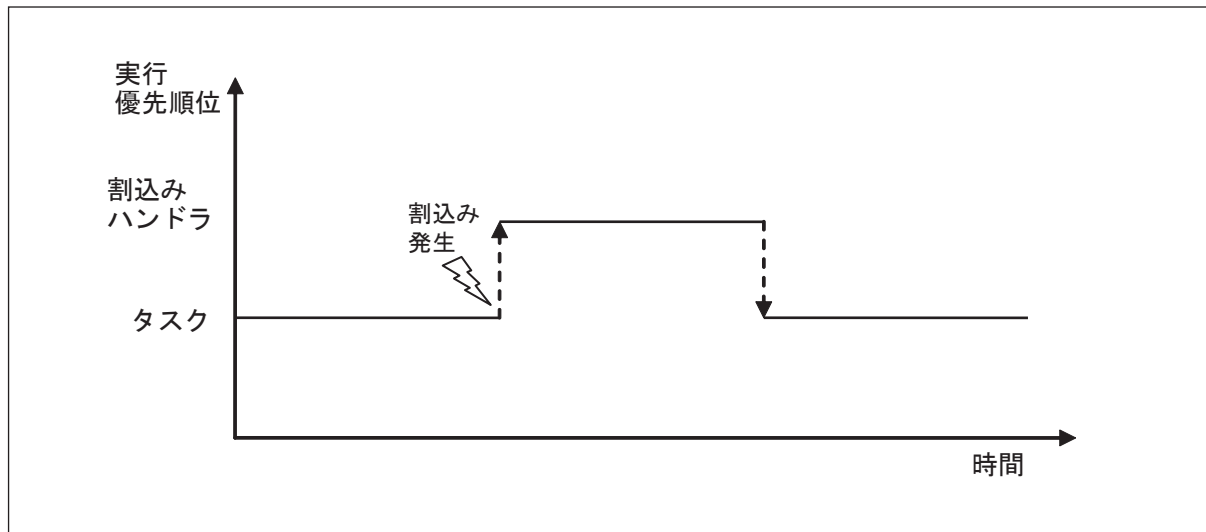
- DI : 割込み禁止状態にする（割込みを禁止する）
- EI : 割込み許可状態にする（割込みを許可する）

2.6 タスク / ハンドラの実行優先順位

タスクやハンドラの実行優先順位について説明します。

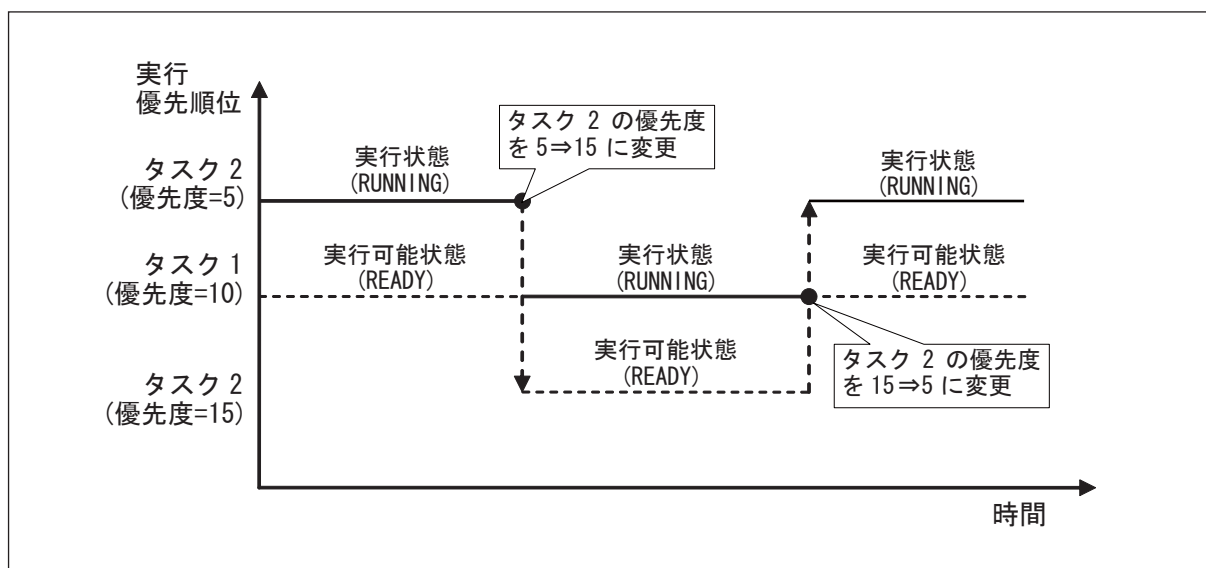
■ 実行優先順位 (タスク対割込みハンドラ, タイムイベントハンドラ)

ハンドラはタスクよりも優先して実行します。たとえば、タスク実行中にハードウェアの割込みが発生した場合、タスクの実行は中断されて、その割込みに対応した割込みハンドラを実行します。割込みハンドラの実行が終了すると、タスクが中断された場所から再実行します。



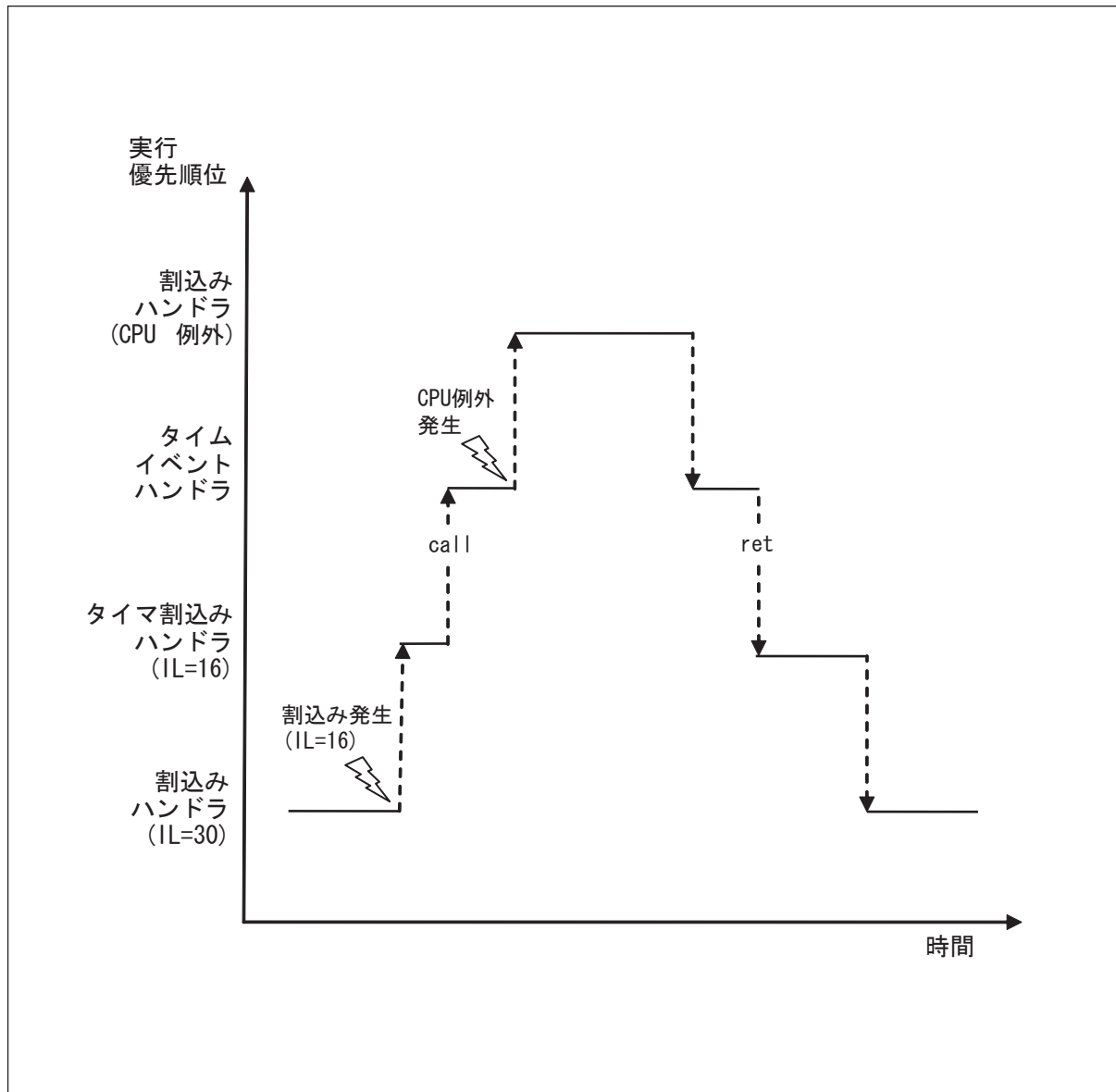
■ 実行優先順位 (タスク対タスク)

タスクの実行優先順位では、優先度の高いタスクを優先して実行します。実行中のタスクよりも優先度が高いタスクが実行可能状態 (READY) になった場合、実行中のタスクの実行は中断されて、優先度が高いタスクを実行します。



■ 実行優先順位 (ハンドラ対ハンドラ)

ハンドラの実行優先順位は, CPU例外要因の割込みハンドラを一番優先して実行します。ほかの割込みハンドラの実行優先順位は, ハードウェアの割込みレベル (Interrupt Level = IL) に依存して, 割込みレベルが高い (割込みレベルの値が小さい) 割込みに対応した割込みハンドラの実行を優先します。タイムイベントハンドラは, タイマ割込みハンドラの延長で実行するため, 実行優先順位はタイマ割込みの割込みレベルに依存します。



第3章

μ T-REALOS の機能

μ T-REALOS がサポートしている機能について説明します。

- 3.1 μ T-REALOS の機能概要
- 3.2 タスク管理機能
- 3.3 タスク付属同期機能
- 3.4 同期・通信機能
- 3.5 拡張同期・通信機能
- 3.6 メモリプール管理機能
- 3.7 時間管理機能
- 3.8 割込み管理機能
- 3.9 システム状態管理機能
- 3.10 サブシステム管理機能
- 3.11 デバイス管理機能
- 3.12 省電力機能
- 3.13 コンフィギュレーション機能
- 3.14 デバッグ支援機能

3.1 μ T-REALOS の機能概要

μ T-REALOS の機能概要について説明します。

■ μ T-REALOS の機能概要

μ T-REALOS では以下の機能をサポートしています。

- カーネル
 - タスク管理機能
 - タスク付属同期機能
 - 同期・通信機能 (セマフォ, イベントフラグ, メールボックス)
 - 拡張同期・通信機能 (ミューテックス, メッセージバッファ, ランデブポート)
 - メモリプール管理機能 (固定長メモリプール, 可変長メモリプール)
 - 時間管理機能 (システム時刻, 周期ハンドラ, アラームハンドラ)
 - 割込み管理機能
 - システム状態管理機能
 - サブシステム管理機能
 - デバイス管理機能
 - 省電力機能
- コンフィギュレータ
 - コンフィギュレーション機能
- アナライザ
 - デバッグ支援機能

なお, 本章の中で記載されているシステムコールの詳細については, 「API リファレンス」の「第 3 章 システムコールインタフェース」を参照してください。

3.2 タスク管理機能

タスク管理機能について説明します。

■ タスク管理機能

タスク管理機能は、タスクの状態を直接的に操作 / 参照するための機能です。タスクを生成 / 削除する機能、タスクを起動 / 終了する機能、タスクの優先度を変更する機能、タスクの状態を参照する機能が含まれます。

タスクは各タスクで一意に決まる ID 番号で識別されます。タスクの ID 番号をタスク ID とよびます。

カーネルは、タスクの終了時に、タスクが獲得した資源（セマフォ資源、メモリブロックなど）を解放しません。ただし、ミューテックス（「3.5.1 ミューテックス機能」参照）のロックは解除します。タスク終了時、ユーザプログラムはタスクが獲得した資源を解放してください。

タスク管理機能では、対応するシステムコールにより以下の機能が含まれます。

- タスクの生成 / 削除

タスクの生成	:tk_cre_tsk
タスクの削除	:tk_del_tsk(休止状態のタスクの削除)
	:tk_ext_tsk(タスクの終了 + 削除)

- タスクの起動 / 終了

タスクの起動	:tk_sta_tsk
タスクの終了	:tk_ext_tsk(自タスクの終了)
	:tk_ext_tsk(自タスクの終了 + 削除)
	:tk_ter_tsk(他タスクの強制終了)

- タスク優先度の変更

:tk_chg_pri

- タスク状態の参照

:tk_ref_tsk

- タスクレジスタの設定 / 参照

タスクレジスタの設定	:tk_set_reg
タスクレジスタの取得	:tk_get_reg

3.3 タスク付属同期機能

タスク付属同期機能機能について説明します。

■ タスク付属同期機能

タスク付属同期機能は、タスクの状態を直接的に操作することによって同期を行うための機能です。

タスクを起床待ちにする機能とそこから起床する機能、タスクの起床要求をキャンセルする機能、タスクの待ち状態を強制解除する機能、タスクを強制待ち状態へ移行する機能とそこから再開する機能、自タスクの実行を遅延する機能が含まれます。

カーネルでは、タスクに対する起床要求をキューイングします。すなわち、起床待ち状態でないタスクを起床しようとする、そのタスクを起床しようとした記録が残り、後でそのタスクが起床待ちに移行しようとしたときに、タスクを起床待ち状態にしません。タスクに対する起床要求のキューイングを実現するために、カーネルではタスク単位にキューイングされた起床要求の数を保持しています。これを、「起床要求キューイング数」とよびます。タスクの起床要求キューイング数は、タスクの起動時に 0 にクリアします。

また、カーネルでは、タスクに対する強制待ち要求をネストします。すなわち、すでに強制待ち状態（二重待ち状態を含む）になっているタスクを、再度強制待ち状態に移行させようとする、そのタスクを強制待ち状態に移行させようとした記録が残り、後でそのタスクを強制待ち状態（二重待ち状態を含む）から再開させようとした時に、強制待ちからの再開を行いません。強制待ち要求のネストを実現するために、カーネルではタスク単位にネストされた強制待ち要求の回数を保持しています。これを「強制待ち要求ネスト数」とよびます。タスクの強制待ち要求ネスト数は、タスクの起動時に 0 にクリアします。

タスク付属同期機能では、対応するシステムコールにより以下の機能が含まれます。

- タスクの起床待ち / 起床

タスクの起床待ち	:tk_slp_tsk
タスクの起床	:tk_wup_tsk
- 起床要求のキャンセル :tk_can_wup
- タスクの待ち状態の強制解除 :tk_rel_wai
- タスクの強制待ち / 再開

強制待ち要求	:tk_sus_tsk
強制待ち再開	:tk_rsm_tsk
	:tk_frsm_tsk(強制再開)
- タスクの実行遅延 :tk_dly_tsk

tskstat が TTS_WAI(TTS_WAS を含む) 以外の状態の場合には、tskwait, wid はともに "0" です。また、休止状態のタスクでは、wupcnt, suscnt はすべて "0" です。

タスク状態を返すパケットアドレス (pk_rtsk) が不正な場合でも、エラーチェックを行わず、動作は保証されません。

3.4 同期・通信機能

同期・通信機能について説明します。

■ 同期・通信機能

同期・通信機能は、タスクとは独立したオブジェクトにより、タスク間の同期・通信を行うための機能です。

同期・通信機能では、以下のオブジェクトをサポートしています。

- セマフォ
- イベントフラグ
- メールボックス

3.4.1 セマフォ機能

セマフォ機能について説明します。

■ セマフォ機能

セマフォは、使用されていない資源の有無や数量を数値で表現する（これをセマフォカウントとよびます）ことにより、その資源を使用する際の排他制御や同期を行うためのオブジェクトです。セマフォ機能には、セマフォを生成 / 削除する機能、セマフォの資源を獲得 / 返却する機能、セマフォの状態を参照する機能が含まれます。

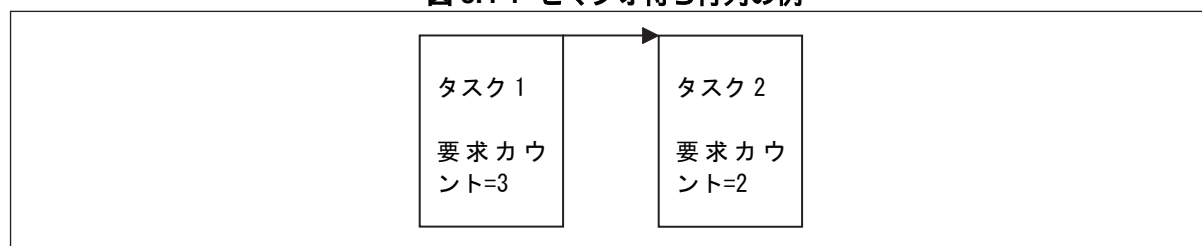
セマフォは ID 番号で識別されるオブジェクトです。セマフォの ID 番号をセマフォ ID とよびます。

セマフォは、セマフォカウント、資源の獲得を待つタスクの待ち行列を持ちます。資源を m 個返却する側（イベントを知らせる側）では、セマフォカウントを m 個増やします。一方、資源を n 個獲得する側（イベントを待つ側）では、セマフォカウントを n 個減らします。セマフォの資源数が足りなくなった場合（具体的には、セマフォカウントを減らすと負になる場合）、資源を獲得しようとしたタスクは、次に資源が返却されるまでセマフォ資源の獲得待ち状態となります。セマフォ資源の獲得待ち状態になったタスクは、そのセマフォの待ち行列につながれます。また、セマフォに資源が返却され過ぎるのを防ぐために、セマフォごとに最大資源数を設定できます。最大セマフォカウントを超える資源がセマフォに返却されようとした場合（具体的には、セマフォカウントを増やすと最大セマフォカウントを超える場合）には、エラーを報告します。

待ち行列の順番は、FIFO 順 (TA_TFIFO)、タスクの優先度順 (TA_TPRI) の 2 種類から選択できます。また、資源獲得の優先順位は、待ち行列先頭のタスクを優先する場合 (TA_FIRST) と要求数の少ないタスクを優先する場合 (TA_CNT) の 2 種類から選択できます。これらは、セマフォ生成時にセマフォの属性として指定します。

図 3.4-1 で、セマフォカウントが 1 から 2 に変化した場合、TA_CNT 属性のセマフォでは、タスク 1 を追い越して、タスク 2 に資源が割り当てられます。

図 3.4-1 セマフォ待ち行列の例



セマフォカウントの最大値は、セマフォ生成時に指定します。セマフォカウントの最大値の上限値は 0x7FFFFFFF です。詳細については、「API リファレンス」の「3.5.1.1 tk_cre_sem」を参照してください。

セマフォ機能では、対応するシステムコールにより以下の機能が含まれます。

- セマフォの生成 / 削除

セマフォの生成 :tk_cre_sem

セマフォの削除 :tk_del_sem

- セマフォ資源の獲得 / 返却
セマフォ資源の返却 :tk_sig_sem
セマフォ資源の獲得 :tk_wai_sem
- セマフォの状態を参照 :tk_ref_sem

3.4.2 イベントフラグ機能

イベントフラグ機能について説明します。

■ イベントフラグ機能

イベントフラグは、イベントの有無をビットごとのフラグで表現することにより、同期を行うためのオブジェクトです。イベントフラグ機能には、イベントフラグを生成 / 削除する機能、イベントフラグをセット / クリアする機能、イベントフラグで待つ機能、イベントフラグの状態を参照する機能が含まれます。

イベントフラグは ID 番号で識別されるオブジェクトです。イベントフラグの ID 番号をイベントフラグ ID とよびます。

イベントフラグは、対応するイベントの有無をビットごとに表現するビットパターンと、そのイベントフラグで待つタスクの待ち行列を持ちます。イベントフラグのビットパターンを、単にイベントフラグとよぶ場合もあります。イベントを知らせる側では、イベントフラグのビットパターンの指定したビットをセットないしはクリアが可能です。一方、イベントを待つ側では、イベントフラグのビットパターンの指定したビットのすべて、またはいずれかがセットされるまで、タスクをイベントフラグ待ち状態にできません。イベントフラグ待ち状態になったタスクは、そのイベントフラグの待ち行列につながれます。

イベントフラグの待ち解除条件として、AND 待ちと OR 待ちの 2 種類の属性が指定できます。これは、複数のイベントを待つ場合の待ち解除の動作を指定するものです。AND 待ちの場合、すべてのイベントが通知されないと待ちが解除されないのに対して、OR 待ちの場合、待っているイベントのうち、1 つでも通知されると待ちを解除します。また、待ち解除時にビットをクリアするか否かについても指定可能で、この場合、全ビットのクリアと一致したビットのクリアが選択できます。

μ T-REALOS では、イベントの発生を 32 ビットのビットパターンで管理しています。

イベントフラグ機能では、システムコールにより以下の機能が含まれます。

- イベントフラグの生成 / 削除

イベントフラグの生成	:tk_cre_flg
------------	-------------

イベントフラグの削除	:tk_del_flg
------------	-------------

- イベントフラグのセット / クリア

イベントフラグのセット	:tk_set_flg
-------------	-------------

イベントフラグのクリア	:tk_clr_flg
-------------	-------------

- イベントフラグの待ち

	:tk_wai_flg
--	-------------

- イベントフラグの状態を参照

	:tk_ref_flg
--	-------------

3.4.3 メールボックス機能

メールボックス機能について説明します。

■ メールボックス機能

メールボックスは、メモリ上に置かれたメッセージの受渡しにより、同期と通信を行うためのオブジェクトです。メールボックス機能には、メールボックスを生成 / 削除する機能、メールボックスにメッセージを送信 / 受信する機能、メールボックスの状態を参照する機能が含まれます。

メールボックスは ID 番号で識別されるオブジェクトです。メールボックスの ID 番号をメールボックス ID とよびます。

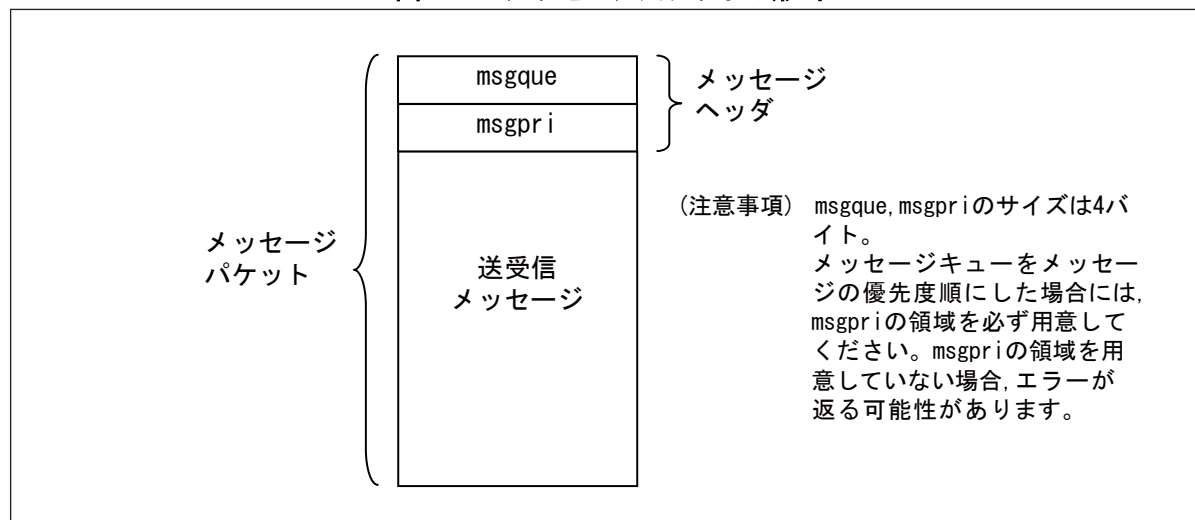
メールボックスは、送信されたメッセージを入れるためのメッセージキューと、メッセージの受信を待つタスクの待ち行列を持ちます。メッセージを送信する側（イベントを知らせる側）では、送信したいメッセージをメッセージキューに入れます。一方、メッセージを受信する側（イベントを待つ側）では、メッセージキューに入っているメッセージを 1 つ取り出します。

メッセージキューにメッセージが入っていない場合には、次にメッセージが送られてくるまでメールボックスからの受信待ち状態になります。メールボックスからの受信待ち状態になったタスクは、そのメールボックスの待ち行列につながれます。

メールボックスによって実際に送受信するのは、メモリ上に置かれたメッセージの先頭番地のみです。すなわち、送受信するメッセージの内容のコピーは行いません。カーネルは、メッセージキューに入っているメッセージを、リンクリストにより管理します。

図 3.4-2 にメッセージの優先度順におけるメッセージパケットの形式を示します。ユーザプログラムは、送信するメッセージの先頭に、カーネルがリンクリストに用いるための領域 (msgque) を確保してください。この領域をメッセージヘッダとよびます。また、メッセージキューをメッセージの優先度順にする場合には、メッセージの優先度を入れるための領域 (msgpri) も、メッセージヘッダ中に持つ必要があります。メッセージヘッダと、それに続くアプリケーションがメッセージを入れるための領域をあわせて、メッセージパケットとよびます。メールボックスへメッセージを送信するシステムコールは、メッセージパケットの先頭番地 (pk_msg) をパラメータとします。また、メールボックスからメッセージを受信するシステムコールは、メッセージパケットの先頭番地を復帰値として返します。

図 3.4-2 メッセージパケットの形式



メールボックス機能では、システムコールにより、以下の機能が含まれます。

- メールボックスの生成 / 削除
 - メールボックスの生成 :tk_cre_mbx
 - メールボックスの削除 :tk_del_mbx
- メールボックスのセット / クリア
 - メールボックスへの送信 :tk_snd_mbx
 - メールボックスからの受信 :tk_rcv_mbx
- メールボックスの状態を参照 :tk_ref_mbx

■ 補足事項

メールボックス機能では、メッセージヘッダの領域をユーザプログラムで確保しているため、メッセージキューに入れるメッセージの数には上限がありません。また、メッセージを送信するシステムコールで待ち状態になることもありません。メッセージパケットとしては、固定長メモリプールまたは可変長メモリプールから動的に確保したメモリブロックを用いることも、静的に確保した領域を用いることも可能です。一般的な使い方としては、送信側のタスクがメモリプールからメモリブロックを確保して、それをメッセージパケットとして送信し、受信側のタスクはメッセージの内容を取り出した後にそのメモリブロックを直接メモリプールに返却する手順をとります。

3.5 拡張同期・通信機能

拡張同期・通信機能について説明します。

■ 拡張同期・通信機能

拡張同期・通信機能は、タスクとは独立したオブジェクトにより、タスク間の高度な同期・通信を行うための機能です。ミューテックス、メッセージバッファ、ランデブポートの各機能が含まれます。

以下のオブジェクトをサポートしています。

- ミューテックス
- メッセージバッファ
- ランデブポート

3.5.1 ミューテックス機能

ミューテックス機能について説明します。

■ ミューテックス機能

ミューテックスは、共有資源を使用するときにタスク間で排他制御を行うためのオブジェクトです。

ミューテックスは、排他制御に伴う上限のない優先度逆転を防ぐための機構として、優先度継承プロトコル(priority inheritance protocol)と優先度上限プロトコル(priority ceiling protocol)をサポートします。ミューテックス機能には、ミューテックスを生成 / 削除する機能、ミューテックスをロック / ロック解除する機能、ミューテックスの状態を参照する機能が含まれます。

ミューテックスは ID 番号で識別されるオブジェクトです。ミューテックスの ID 番号をミューテックス ID とよびます。

ミューテックスは、ロックされているかどうかの状態と、ロックを待つタスクの待ち行列を持ちます。また、カーネルは以下を管理します。

- 各ミューテックスをロックしているタスク
- 各タスクがロックしているミューテックス

タスクは、資源を使用する前に、ミューテックスをロックします。ミューテックスがほかのタスクにロックされていた場合には、ミューテックスのロックを解除するまで、ミューテックスのロック待ち状態となります。ミューテックスのロック待ち状態になったタスクは、そのミューテックスの待ち行列につながれます。タスクは、資源の使用を終了すると、ミューテックスのロックを解除します。

ミューテックスは、ミューテックス属性に TA_INHERIT(= 0x02) を指定することにより優先度継承プロトコルをサポートします。また TA_CEILING(= 0x03) を指定することにより優先度上限プロトコルをサポートします。

TA_CEILING 属性のミューテックスには、そのミューテックスをロックする可能性のあるタスクの中で最も高いベース優先度を持つタスクのベース優先度を、ミューテックス生成時に上限優先度として設定します。TA_CEILING 属性のミューテックスを、その上限優先度よりも高いベース優先度を持つタスクがロックしようとした場合、E_ILUSE エラーとなります。また、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タスクのベース優先度を、tk_chg_pri によってそのミューテックスの上限優先度よりも高く設定しようとした場合、tk_chg_pri が E_ILUSE エラーを返します。

これらのプロトコルを用いた場合、上限のない優先度逆転を防ぐために、ミューテックスの操作に伴ってタスクの現在優先度を変更します。ミューテックスをロックしているタスクの現在優先度は、次に挙げる優先度の最高値に常に一致するようにカーネルで変更します。

- ミューテックスをロックしているタスクのベース優先度
- タスクが TA_INHERIT 属性のミューテックスをロックしている場合、それらのミューテックスのロックを待っているタスクの中で、最も高い現在優先度を持つタスクの現在優先度

- 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、ロックされているミューテックス中で、最も高い上限優先度を持つミューテックスの上限優先度ここで、TA_INHERIT 属性のミューテックスを待っているタスクの現在優先度が、ミューテックス操作がtk_chg_pri によるベース優先度の変更に伴って変更された場合、そのミューテックスをロックしているタスクの現在優先度の変更が必要になる場合があります。これを推移的な優先度継承とよびます。さらにそのタスクが、別のTA_INHERIT 属性のミューテックスを待っていた場合には、そのミューテックスをロックしているタスクに推移的な優先度継承の処理が必要になる場合があります。ミューテックスの操作に伴ってタスクの現在優先度を変更した場合には、以下の処理を行います。

- 優先度を変更されたタスクが実行できる状態である場合、タスクの優先順位を、変更後の優先度にしたがって変化させます（変更後の優先度と同じ優先度を持つタスクの間では最低の優先順位になるものとします）。
- 優先度を変更されたタスクが何らかのタスク優先度順の待ち行列につながれている場合にも、その待ち行列の中での順序を、変更後の優先度にしたがって変化させます（変更後の優先度と同じ優先度を持つタスクの間では最低の順位になるものとします）。
- タスク終了時に、そのタスクがロックしているミューテックスが残っている場合には、それらのミューテックスをすべてロック解除します。ロックしているミューテックスが複数ある場合には、それらは後に確保したものから順に解除します。

ロック解除の具体的な処理内容については、「API リファレンス」の「3.6.1.4 tk_unl_mtx」を参照してください。

ミューテックス機能では、システムコールにより以下の機能が含まれます。

- ミューテックスの生成 / 削除

ミューテックスの生成	:tk_cre_mtx
ミューテックスの削除	:tk_del_mtx
- ミューテックスのロック / ロック解除

ミューテックスのロック	:tk_loc_mtx
ミューテックスのロック解除	:tk_unl_mtx
- ミューテックスの状態を参照 :tk_ref_mtx

■ 補足事項

TA_TFIFO 属性または TA_TPRI 属性のミューテックスは、最大資源数が1のセマフォ（バイナリセマフォ）と同等の機能を持ちます。ただしミューテックスは、ロックしたタスク以外はロック解除できない、タスク終了時に自動的にロック解除する、という相違点があります。

3.5.2 メッセージバッファ機能

メッセージバッファ機能について説明します。

■ メッセージバッファ機能

メッセージバッファは、可変長のメッセージの受渡しにより、同期と通信を行うためのオブジェクトです。メッセージバッファ機能には、メッセージバッファを生成 / 削除する機能、メッセージバッファにメッセージを送信 / 受信する機能、メッセージバッファの状態を参照する機能が含まれます。

メッセージバッファは ID 番号で識別されるオブジェクトです。メッセージバッファの ID 番号をメッセージバッファ ID とよびます。

メッセージバッファは、メッセージの送信を待つタスクの待ち行列（送信待ち行列）とメッセージの受信を待つタスクの待ち行列（受信待ち行列）を持ちます。また、送信されたメッセージを格納するためのメッセージバッファ領域を持ちます。

メッセージを送信する側（イベントを知らせる側）では、送信したいメッセージをメッセージバッファにコピーします。メッセージバッファ領域の空き領域が足りなくなった場合、メッセージバッファ領域に十分な空きができるまでメッセージバッファへの送信待ち状態になります。メッセージバッファへの送信待ち状態になったタスクは、そのメッセージバッファの送信待ち行列につながれます。

一方、メッセージを受信する側（イベントを待つ側）では、メッセージバッファに入っているメッセージを 1 つ取り出します。メッセージバッファにメッセージが入っていない場合には、次にメッセージが送られてくるまでメッセージバッファからの受信待ち状態になります。メッセージバッファからの受信待ち状態になったタスクは、そのメッセージバッファの受信待ち行列につながれます。

メッセージバッファ領域のサイズを 0 にすることで、同期メッセージ機能を実現できます。すなわち、送信側のタスクと受信側のタスクが、それぞれ相手のタスクがシステムコールを呼び出すのを待ち合わせ、両者がシステムコールを呼び出した時点で、メッセージの受渡しが行われます。

メッセージバッファ機能では、システムコールにより以下の機能が含まれます。

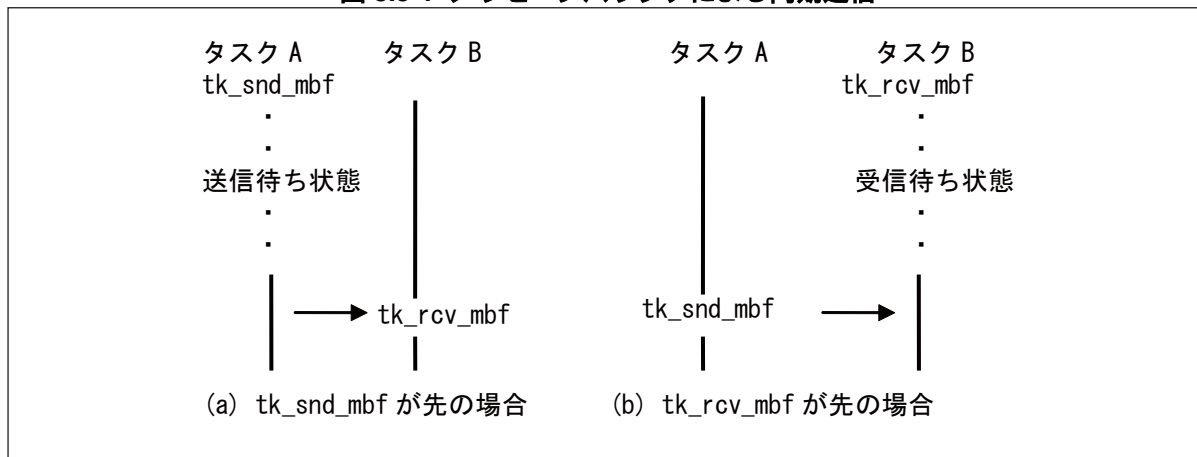
- メッセージバッファの生成 / 削除
 - メッセージバッファの生成 :tk_cre_mbf
 - メッセージバッファの削除 :tk_del_mbf
- メッセージバッファのメッセージ送信 / 受信
 - メッセージバッファへの送信 :tk_snd_mbf
 - メッセージバッファからの受信 :tk_rcv_mbf
- メッセージバッファの状態を参照 :tk_ref_mbf

■ 補足事項

メッセージバッファ領域のサイズを 0 にした場合の、メッセージバッファの動作を図 3.5-1 の例を使用して説明します。この図で、タスク A とタスク B は非同期に実行しているものとします。

- もしタスク A が先に tk_snd_mbf を呼び出した場合には、タスク B が tk_rcv_mbf を呼び出すまでタスク A は待ち状態となります。この場合タスク A は、メッセージバッファへの送信待ち状態になっています ((図 3.5-1 (a))。
- 逆にタスク B が先に tk_rcv_mbf を呼び出した場合には、タスク A が tk_snd_mbf を呼び出すまでタスク B は待ち状態となります。この場合タスク B は、メッセージバッファからの受信待ち状態になっています (図 3.5-1 (b))。
- タスク A が tk_snd_mbf を呼び出し、タスク B が tk_rcv_mbf を呼び出した時点で、タスク A からタスク B へメッセージの受渡しが行われます。その後は、両タスクとも実行できる状態となります。

図 3.5-1 メッセージバッファによる同期通信



メッセージバッファへの送信を待っているタスクは、待ち行列につながれている順序でメッセージを送信します。たとえば、あるメッセージバッファに 40 バイトのメッセージを送信しようとしているタスク A と、10 バイトのメッセージを送信しようとしているタスク B が、この順で待ち行列につながれているときに、別のタスクによるメッセージの受信により 20 バイトの空き領域ができたとします。この場合でも、タスク A がメッセージを送信するまで、タスク B はメッセージを送信できません。

メッセージバッファは、メールボックスと違い、可変長のメッセージをコピーして受渡しします。

3.5.3 ランデブポート機能

ランデブポート機能について説明します。

■ ランデブポート機能

ランデブポート機能は、タスク間で同期通信を行うための機能で、あるタスクから別のタスクへの処理依頼と、後者のタスクから前者のタスクへの処理結果の返却を、一連の手順としてサポートします。双方のタスクが待ち合わせるためのオブジェクトを、ランデブポートとよびます。ランデブポート機能は、典型的にはクライアント/サーバモデルのタスク間通信を実現するために用いられますが、クライアント/サーバモデルよりも柔軟な同期通信モデルを提供するものです。

ランデブポート機能には、ランデブポートを生成/削除する機能、ランデブポートに処理の依頼を行う機能（ランデブの呼出し）、ランデブポートで処理依頼を受け付ける機能（ランデブの受け付け）、処理結果を返す機能（ランデブの終了）、受け付けた処理依頼をほかのランデブポートに回送する機能（ランデブの回送）、ランデブポートおよびランデブの状態を参照する機能が含まれます。

ランデブポートは ID 番号で識別されるオブジェクトです。ランデブポートの ID 番号をランデブポート ID とよびます。

ランデブポートに処理依頼を行う側のタスク（クライアント側のタスク）は、ランデブポートとランデブ条件、依頼する処理に関する情報を入れたメッセージ（これを呼出しメッセージとよぶ）を指定して、ランデブの呼出しを行います。一方、ランデブポートで処理依頼を受け付ける側のタスク（サーバ側のタスク）は、ランデブポートとランデブ条件を指定して、ランデブの受け付けを行います。

ランデブ条件は、ビットパターンで指定します。あるランデブポートに、呼び出したタスクのランデブ条件のビットパターンと、受け付けたタスクのランデブ条件のビットパターンをビットごとに論理積をとり、結果が 0 以外の場合にランデブが成立します。ランデブを呼び出したタスクは、ランデブが成立するまでランデブ呼出し待ち状態となります。逆に、ランデブを受け付けるタスクは、ランデブが成立するまでランデブ受け付け待ち状態となります。

ランデブが成立すると、ランデブを呼び出したタスクから受け付けたタスクへ、呼出しメッセージを渡します。ランデブを呼び出したタスクはランデブ終了待ち状態へ移行して、依頼した処理が完了するのを待ちます。一方、ランデブを受け付けたタスクは待ち解除され、依頼された処理を行います。ランデブを受け付けたタスクが依頼された処理を完了すると、処理結果を返答メッセージの形で呼び出したタスクに渡して、ランデブを終了します。この時点で、ランデブを呼び出したタスクのランデブ終了待ち状態を解除します。

ランデブポートは、ランデブ呼出し待ち状態のタスクをつなぐための呼出し待ち行列と、ランデブ受け付け待ち状態のタスクをつなぐための受け付け待ち行列を持ちます。それに、ランデブが成立した後は、ランデブした双方のタスクをランデブポートから切り離します。すなわち、ランデブポートは、ランデブ終了待ち状態のタスクをつなぐための待ち行列は持ちません。また、ランデブを受け付け、依頼された処理を実行しているタスクに関する情報も持ちません。

カーネルは、同時に成立しているランデブを識別するために、オブジェクト番号を付与します。ランデブのオブジェクト番号をランデブ番号とよびます。ランデブ番号は、上位 16 ビットがランデブを呼び出したタスクのタスク ID、下位 16 ビットはランデブ受けごとに +1 増加するシーケンシャルな番号です。このため、同じタスクが呼び出したランデブであっても、1 回目のランデブと 2 回目のランデブで異なるランデブ番号を付与します。

ランデブポート機能では、システムコールにより以下の機能が含まれます。

- ランデブポートの生成 / 削除

ランデブポートの生成	:tk_cre_por
ランデブポートの削除	:tk_del_por
- ランデブポートへの処理の依頼 / 受付 / 返答

ランデブポートへの処理の依頼	:tk_cal_por
ランデブポートへの処理の受付	:tk_acp_por
ランデブポートへの処理の返答	:tk_rpl_rdv
- ランデブポートの回送

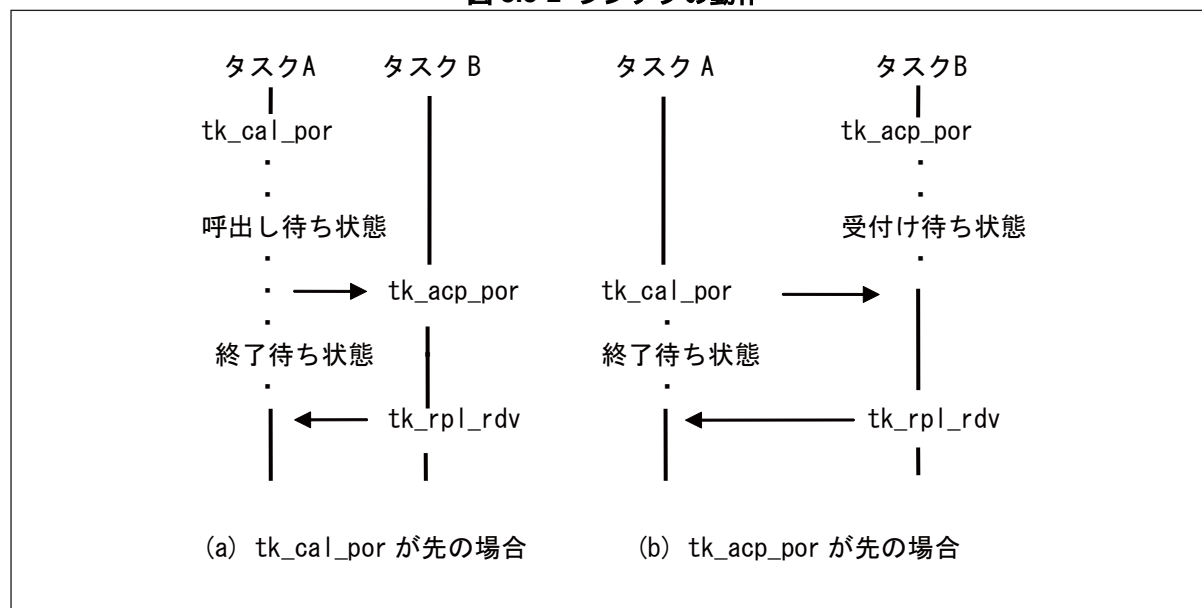
	:tk_fwd_por
--	-------------

■ 補足事項

ランデブの動作を図 3.5-2 の例を使用して説明します。この図で、タスク A とタスク B は非同期に実行しているものとします。

- もしタスク A が先に tk_cal_por を呼び出した場合には、タスク B が tk_acp_por を呼び出すまでタスク A は待ち状態となります。この場合タスク A は、ランデブの呼出し待ち状態になっています (図 3.5-2 (a))。
- 逆にタスク B が先に tk_acp_por を呼び出した場合には、タスク A が tk_cal_por を呼び出すまでタスク B は待ち状態となります。この場合タスク B は、ランデブの受け待ち状態になっています (図 3.5-2 (b))。
- タスク A が tk_cal_por を呼び出し、タスク B が tk_acp_por を呼び出した時点でランデブが成立し、タスク A を待ち状態としたまま、タスク B の待ち状態を解除します。この場合タスク A は、ランデブの終了待ち状態になっています。
- タスク B が tk_rpl_rdv を呼び出した時点で、タスク A の待ち状態を解除します。その後は、両タスクとも実行できる状態となります。

図 3.5-2 ランデブの動作



3.6 メモリプール管理機能

メモリプール管理機能について説明します。

■ メモリプール管理機能

「メモリプール管理機能」は、メモリプールの管理やユーザプログラムで使用するメモリ領域（メモリブロック）の割当てを行う機能です。

メモリプールには、固定長メモリプールと可変長メモリプールがあります。両者は別のオブジェクトであり、操作のためのシステムコールが異なります。固定長メモリプールから獲得するメモリブロックはサイズが固定されていますが、可変長メモリプールから獲得するメモリブロックは、任意のサイズを指定できます。

メモリプール管理機能では、以下のメモリプールをサポートしています。

- 固定長メモリプール
- 可変長メモリプール

3.6.1 固定長メモリプール機能

固定長メモリプール機能について説明します。

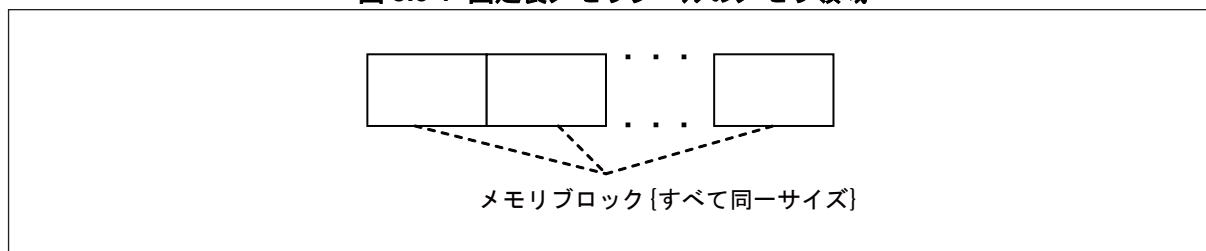
■ 固定長メモリプール機能

固定長メモリプールは、固定されたサイズのメモリブロックを動的に管理するためのオブジェクトです。固定長メモリプール機能には、固定長メモリプールを生成 / 削除する機能、固定長メモリプールにメモリブロックを獲得 / 返却する機能、固定長メモリプールの状態を参照する機能が含まれます。

固定長メモリプールは ID 番号で識別されるオブジェクトです。固定長メモリプールの ID 番号を固定長メモリプール ID とよびます。

固定長メモリプールは、固定長メモリプールとして使用するメモリ領域（これを固定長メモリプール領域、または単にメモリプール領域とよびます）と、メモリブロックの獲得を待つタスクの待ち行列を持ちます。固定長メモリプールからメモリブロックを獲得するタスクは、メモリプール領域に空きがなくなった場合、次にメモリブロックが返却されるまで固定長メモリプールの獲得待ち状態となります。固定長メモリプールの獲得待ち状態になったタスクは、その固定長メモリプールの待ち行列につながれます。

図 3.6-1 固定長メモリプールのメモリ領域



固定長メモリプール機能では、システムコールにより以下の機能が含まれます。

- 固定長メモリプールの生成 / 削除

固定長メモリプールの生成	:tk_cre_mpf
固定長メモリプールの削除	:tk_del_mpf
- 固定長メモリブロックの獲得 / 返却

固定長メモリブロックの獲得	:tk_get_mpf
固定長メモリブロックの返却	:tk_rel_mpf
- 固定長メモリブロックの状態を参照

	:tk_ref_mpf
--	-------------

3.6.2 可変長メモリプール機能

可変長メモリプール機能について説明します。

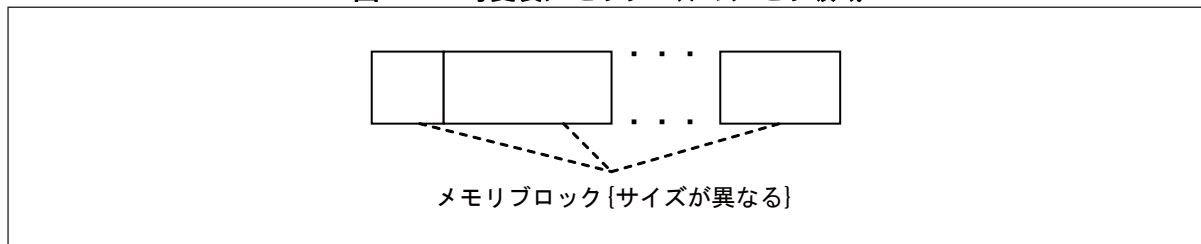
■ 可変長メモリプール機能

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管理するためのオブジェクトです。可変長メモリプール機能には、可変長メモリプールを生成 / 削除する機能、可変長メモリプールにメモリブロックを獲得 / 返却する機能、可変長メモリプールの状態を参照する機能が含まれます。

可変長メモリプールは ID 番号で識別されるオブジェクトです。可変長メモリプールの ID 番号を可変長メモリプール ID とよびます。

可変長メモリプールは、可変長メモリプールとして使用するメモリ領域（これを可変長メモリプール領域、または単にメモリプール領域とよびます）と、メモリブロックの獲得を待つタスクの待ち行列を持ちます。可変長メモリプールからメモリブロックを獲得するタスクは、メモリプール領域の空き領域が足りなくなった場合、十分なサイズのメモリブロックが返却されるまで可変長メモリプールの獲得待ち状態となります。可変長メモリプールの獲得待ち状態になったタスクは、その可変長メモリプールの待ち行列につながれます。

図 3.6-2 可変長メモリプールのメモリ領域



可変長メモリプール機能では、システムコールにより以下の機能が含まれます。

- 可変長メモリプールの生成 / 削除

可変長メモリプールの生成	:tk_cre_mpl
可変長メモリプールの削除	:tk_del_mpl
- 可変長メモリブロックの獲得 / 返却

可変長メモリブロックの獲得	:tk_get_mpl
可変長メモリブロックの返却	:tk_rel_mpl
- 可変長メモリブロックの状態を参照

	:tk_ref_mpl
--	-------------

3.7 時間管理機能

時間管理機能について説明します。

■ 時間管理機能

時間管理機能は、時間に依存した処理を行うための機能です。システム時刻管理、周期ハンドラ、アラームハンドラの各機能が含まれます。周期ハンドラとアラームハンドラを総称して、タイムイベントハンドラとよびます。

以下の機能をサポートしています。

- システム時刻管理
- 周期ハンドラ
- アラームハンドラ

3.7.1 システム時刻管理機能

システム時刻管理機能について説明します。

■ システム時刻

システム時刻は、1985 年 1 月 1 日 (GMT) からの通算のミリ秒数であらわします。たとえば、システム時刻の値が 0 の場合、1985 年 1 月 1 日 0 時 0 分 0 秒 (GMT) をあらわします。システム時刻の値が 1000 の場合、1985 年 1 月 1 日 0 時 0 分 1 秒 (GMT) をあらわします。μT-REALOS では、システム起動時に自動的に現在の時刻を設定する機能を持たないため、ユーザプログラムで現在の時刻を設定してください。

■ システム時刻の更新

μT-REALOS では、ユーザプログラムがシステム時刻を更新してください。このため、μT-REALOS では、`isig_tim` が用意されています。このシステムコールを呼び出すことにより、システム時刻を 1 増やします。`isig_tim` は、μT-REALOS の独自システムコールです。システム時刻の分解能は 1ms であるため、通常はインターバルタイマで 1ms 周期の割込みを発生させ、その割込みハンドラから `isig_tim` を呼び出して、システム時刻を更新します。

- システム時刻の更新 `:isig_tim`

■ システム時刻の設定 / 参照

μT-REALOS では、システム時刻の設定/参照のため、以下のシステムコールが用意されています。

- システム時刻の設定 / 参照

システム時刻の設定	<code>:tk_set_tim</code>
システム時刻の参照	<code>:tk_get_tim</code>

■ システム稼働時間の参照

システム起動時からの経過時間をシステム稼働時間とよびます。システム稼働時間は以下のシステムコールによって参照可能です。

- システム稼働時間の参照 `:tk_get_otm`

システム稼働時間はシステム時刻と異なり、`tk_set_tim` によるシステム時刻設定に影響しません。

3.7.2 周期ハンドラ機能

周期ハンドラ機能について説明します。

■ 周期ハンドラ機能

周期ハンドラは、一定周期で起動するタイムイベントハンドラです。周期ハンドラ機能には、周期ハンドラを生成 / 削除する機能、周期ハンドラの動作を開始 / 停止する機能、周期ハンドラの状態を参照する機能が含まれます。

周期ハンドラは ID 番号で識別されるオブジェクトです。周期ハンドラの ID 番号を周期ハンドラ ID とよびます。

周期ハンドラは、動作している状態か動作していない状態かのいずれかの状態をとります。周期ハンドラが動作していない状態では、周期ハンドラを起動すべき時刻となっても周期ハンドラを起動せず、次に起動すべき時刻の決定のみを行います。周期ハンドラの動作を開始するシステムコール (tk_sta_cyc) が呼び出されると、周期ハンドラを動作している状態に移行させ、必要なら周期ハンドラを次に起動すべき時刻を決定しなおします。周期ハンドラの動作を停止するシステムコール (tk_stp_cyc) が呼び出されると、周期ハンドラを動作していない状態に移行させます。周期ハンドラを生成した後にどちらの状態になるかは、周期ハンドラ属性によって決められます。

周期ハンドラの起動位相は、周期ハンドラを生成するシステムコールが呼び出された時刻を基準に、周期ハンドラを最初に起動する時刻を指定する相対時間です。周期ハンドラの起動周期は、周期ハンドラを (起動した時刻ではなく) 起動すべきであった時刻を基準に、周期ハンドラを次に起動する時刻を指定する相対時間です。

周期ハンドラの起動周期と起動位相は、周期ハンドラの生成時に、周期ハンドラごとに設定できます。カーネルは、周期ハンドラの操作時に、設定された起動周期と起動位相から、周期ハンドラを次に起動すべき時刻を決定します。周期ハンドラの生成時には、周期ハンドラを生成した時刻に起動位相を加えた時刻を、次に起動すべき時刻とします。周期ハンドラを起動すべき時刻になると、その周期ハンドラの拡張情報 (exinf) をパラメータとして、周期ハンドラを起動します。またこの場合、周期ハンドラの起動すべき時刻に起動周期を加えた時刻を、次に起動すべき時刻とします。また、周期ハンドラの動作を開始するときに、次に起動すべき時刻を決定しなおす場合があります。

起動位相に起動周期よりも長い時間が指定された場合、周期ハンドラは指定した起動位相分だけ時間が経過してから起動します。たとえば、起動周期 100 ms 起動位相 200 ms の場合、周期ハンドラが最初に起動するのは 200 ms より後になり、それ以降は $200+100 \times (n-1)$ ms 経った後に起動します。

周期ハンドラ機能では、システムコールにより以下の機能が含まれます。

- 周期ハンドラの生成 / 削除

周期ハンドラの生成 :tk_cre_cyc

周期ハンドラの削除 :tk_del_cyc

- 周期ハンドラの動作の動作開始 / 停止

周期ハンドラの動作開始 :tk_sta_cyc

: tk_cre_cyc (TA_STA 指定で、生成 + 動作開始)

周期ハンドラの動作停止 :tk_stp_cyc

- 周期ハンドラの状態を参照 :tk_ref_cyc

なお , 周期ハンドラの手式については , 「4.6 周期ハンドラ」を参照してください。

3.7.3 アラームハンドラ機能

アラームハンドラ機能について説明します。

■ アラームハンドラ機能

アラームハンドラは、指定した時刻に起動するタイムイベントハンドラです。アラームハンドラ機能には、アラームハンドラを生成 / 削除する機能、アラームハンドラの動作を開始 / 停止する機能、アラームハンドラの状態を参照する機能が含まれます。

アラームハンドラは ID 番号で識別されるオブジェクトです。アラームハンドラの ID 番号をアラームハンドラ ID とよびます。

アラームハンドラを起動する時刻（これをアラームハンドラの起動時刻とよぶ）は、アラームハンドラごとに設定できます。アラームハンドラの起動時刻になると、そのアラームハンドラの拡張情報 (exinf) をパラメータとして、アラームハンドラを起動します。

アラームハンドラの生成直後には、アラームハンドラの起動時刻は設定されておらず、アラームハンドラの動作は停止しています。アラームハンドラの動作を開始するシステムコール (tk_sta_alm) が呼び出されると、指定された相対時間後にアラームハンドラを起動します。アラームハンドラの動作を停止するシステムコール (tk_stp_alm) が呼び出されると、アラームハンドラの起動時刻の設定を解除します。また、アラームハンドラを起動するときにも、アラームハンドラの起動時刻の設定を解除して、アラームハンドラの動作を停止します。

アラームハンドラ機能では、システムコールにより以下の機能が含まれます。

- アラームハンドラの生成 / 削除
 - アラームハンドラの生成 :tk_cre_alm
 - アラームハンドラの削除 :tk_del_alm
- アラームハンドラの動作の動作開始 / 停止
 - アラームハンドラの動作開始 :tk_sta_alm
 - アラームハンドラの動作停止 :tk_stp_alm
- アラームハンドラの状態を参照 :tk_ref_alm

なお、アラームハンドラの手式については、「4.7 アラームハンドラ」を参照してください。

3.8 割込み管理機能

割込み管理機能について説明します。

■ 割込み管理機能

割込み管理機能は、外部割込みおよび CPU 例外に対するハンドラの定義や割込み制御などの操作を行う機能です。

割込み管理機能では、システムコールにより以下の機能が含まれます。

- 割込みハンドラ管理

割込みハンドラの定義 :tk_def_int

割込みハンドラから復帰 :tk_ret_int

割込みハンドラは、タスク独立部として扱われます。タスク独立部でも、タスク部と同じ形式でシステムコールを呼出し可能ですが、タスク独立部で呼び出せるシステムコールには以下の制限が付きます。

- 自タスクを指定するシステムコールや、内部で待ち状態に遷移するシステムコールは呼び出せず、エラーとなります。

タスク独立部の実行中は、システムコールの処理でディスパッチの要求が出されても、タスク独立部を抜けるまでディスパッチが遅れます。これを遅延ディスパッチとよびます。

なお、割込みハンドラの書式については、「4.8 割込みハンドラ」を参照してください。

- CPU 割込み制御

すべての外部割込みを禁止 :DI

すべての外部割込みを許可 :EI

DI 呼出し前の割込み禁止状態の取得 :isDI

CPU のレジスタを操作して、割込みの許可 / 禁止の設定を行います。DI, EI, isDI は、タスク独立部およびディスパッチ禁止・割込み禁止の状態から呼び出せます。

3.9 システム状態管理機能

システム状態管理機能について説明します。

■ システム状態管理機能

システム状態管理機能は、システムの状態を変更 / 参照するための機能です。タスクの優先順位を回転する機能、実行状態のタスク ID を参照する機能、タスクディスパッチを禁止 / 許可する機能、コンテキストやシステム状態を参照する機能、カーネルのバージョンを参照する機能が含まれます。

システム状態管理機能では、システムコールにより以下の機能が含まれます。

- タスクの優先順位の回転 :tk_rot_rdq
- 実行状態のタスク ID の参照 :tk_get_tid
- ディスパッチの禁止 / 許可
 - ディスパッチの禁止 :tk_dis_dsp
 - ディスパッチの許可 :tk_ena_dsp
- システム状態の参照 :tk_ref_sys
- カーネルバージョンの参照 :tk_ref_ver

3.10 サブシステム管理機能

サブシステム管理機能について説明します。

■ サブシステム管理機能

サブシステム管理機能では、要求を受け付けるための拡張 SVC ハンドラだけから構成します。サブシステム管理機能では、システムコールにより以下の機能が含まれます。

- サブシステムの定義 `:tk_def_ssy`
- サブシステム定義情報の参照 `:tk_ref_ssy`

3.11 デバイス管理機能

デバイス管理機能について説明します。

■ デバイス管理機能

デバイス管理機能は、デバイスの登録、削除、デバイスへのデータのアクセス、デバイス情報の取得など、デバイスに関する操作を、デバイスが異なっても共通の API で扱えるようにした機能です。

デバイス管理機能では、システムコールにより以下の機能が含まれます。

- システムコール

デバイスの登録	:tk_def_dev
デバイス初期情報の取得	:tk_ref_idv
デバイスのオープン	:tk_opn_dev
デバイスのクローズ	:tk_cls_dev
デバイスの読み込み開始	:tk_rea_dev
デバイスの同期読み込み	:tk_srea_dev
デバイスの書き込み開始	:tk_wri_dev
デバイスの同期書き込み	:tk_swri_dev
デバイスの要求完了待ち	:tk_wai_dev
デバイスのサスペンド	:tk_sus_dev
デバイス名取得	:tk_get_dev
デバイス情報取得	:tk_ref_dev
	:tk_oref_dev
登録済みデバイス一覧の取得	:tk_lst_dev
デバイスにドライバ要求イベントを送信	:tk_evt_dev

デバイスの登録とオープンの関係で呼出し可能なシステムコールを以下に示します。

デバイスの登録	デバイスのオープン	呼出し可能なシステムコール
なし	-	tk_def_dev
あり	なし	tk_opn_dev, tk_ref_idv, tk_get_dev, tk_ref_dev, tk_lst_dev, tk_sus_dev, tk_def_dev
	あり	デバイス管理機能のすべてのシステムコール

また、デバイスドライバとカーネル間のインタフェースの仕様として、デバイスドライバインタフェースを規定しています。デバイスドライバインタフェースでは、カーネルから呼び出すデバイス処理関数、カーネルとデバイスドライバ間で受け渡すデータの型などを規定しています。

デバイスドライバインタフェースは、 μ T-Kernel 仕様の OS すべてがサポートしているため、デバイスドライバインタフェースに準じて作成したデバイスドライバは、 μ T-Kernel 仕様の OS 間での移植性が向上します。

デバイスドライバインタフェースの詳細については、「API リファレンス」の「付録 C デバイスドライバインタフェース」を参照してください。

3.12 省電力機能

省電力機能について説明します。

■ 省電力機能

μ T-REALOS では、省電力機能として、すべてのタスクが実行されていないアイドル状態に遷移した場合、カーネルからユーザが定義した省電力ルーチンを呼び出す機能があります。これを省電力機能とよびます。

省電力ルーチンの処理は、ユーザがターゲットハードウェアに合わせて、自由に記述できます。この省電力ルーチンは、コンフィギュレータの静的 API を使って定義することにより有効となります。省電力ルーチンの定義方法については、「3.13 コンフィギュレーション機能」を、省電力ルーチンの書式については、「4.10 省電力ルーチン」を参照してください。

3.13 コンフィギュレーション機能

コンフィギュレーション機能について説明します。

■ コンフィギュレーション機能

コンフィギュレーション機能は、カーネルが使用する資源数の上限値などのカーネルの構成をユーザが定義して、それに基づいて、カーネル内部の管理データを構築（コンフィギュレーション）する機能です。カーネルの構成をユーザプログラムに合わせて最適化することにより、カーネルが使用するメモリを削減できます。また、割込みハンドラや初期ルーチンなどのユーザプログラムのモジュールを静的に登録する機能があります。

コンフィギュレーションを実行する μ T-RELAOS の開発ツールを「コンフィギュレータ」とよびます。また、カーネルの構成を定義するために用意されているマクロを「コンフィギュレーションマクロ」とよび、ユーザプログラムのモジュールに登録するための定義文を「静的 API」とよびます。

コンフィギュレーション実行時には、コンフィギュレーション規定マクロや静的 API を、「コンフィギュレーションファイル」とよばれるテキスト形式のファイルに記述して、これを入力ファイルとしてコンフィギュレータを実行します。

コンフィギュレータは、コンフィギュレーションファイルを入力とし、カーネル構成ファイルを SOFTUNE 言語ツールのライブラリ形式で出力します。カーネル構成ファイルは、実行形式のオブジェクト作成時にユーザプログラムとリンクします。

通常は、コンフィギュレーションの設定は、SOFTUNE Workbench の GUI 画面を介して行い、コンフィギュレーションファイルは自動生成されるため、コンフィギュレーション規定マクロや静的 API の書式を意識する必要はありません。また、カーネルコンフィギュレーションは、SOFTUNE Workbench を用いたビルド環境において、自動的に実行します。GUI 画面を介したコンフィギュレーションの設定については、「5.3 コンフィギュレーションの設定」を参照してください。

■ コンフィギュレーション規定マクロ

コンフィギュレーション規定マクロは、カーネルの構成をユーザ定義するために用意されているマクロです。以下にコンフィギュレーション規定マクロの一覧を示します。

表 3.13-1 コンフィギュレーション規定マクロ一覧

機能種別	名前	意味	値の範囲 (太字がデフォルト値)
優先度定義	_KERNEL_MAX_TSKPRI	最大タスク優先度	1 ~ 1024
	_KERNEL_INIT_TSKPRI	初期タスク優先度	1 ~ 1024
	_KERNEL_MAX_SSPRI	最大サブシステム優先度	1 ~ 16
機能選択	_KERNEL_USE_TKDEFINT	tk_def_int の使用 / 不使用 (1 で使用)	0 または 1
	_KERNEL_USE_IMALLOC	ヒープ領域の使用 / 不使用 (1 で使用)	0 または 1
	_KERNEL_REALMEMSZ	ヒープ領域のサイズ	任意
各オブジェクトの最大数	_KERNEL_MAX_TSK	最大タスク数	1 ~ 32767
	_KERNEL_MAX_SEM	最大セマフォ数	0 ~ 32767
	_KERNEL_MAX_FLG	最大イベントフラグ数	0 ~ 32767
	_KERNEL_MAX_MBX	最大メールボックス数	0 ~ 32767
	_KERNEL_MAX_MTX	最大ミューテックス数	0 ~ 32767
	_KERNEL_MAX_MBF	最大メッセージバッファ数	0 ~ 32767
	_KERNEL_MAX_POR	最大ランデブポート数	0 ~ 32767
	_KERNEL_MAX_MPF	最大固定長メモリプール数	0 ~ 32767
	_KERNEL_MAX_MPL	最大可変長メモリプール数	0 ~ 32767
	_KERNEL_MAX_CYC	最大周期ハンドラ数	0 ~ 32767
	_KERNEL_MAX_ALM	最大アラームハンドラ数	0 ~ 32767
	_KERNEL_MAX_SSY	最大サブシステム数	0 ~ 255
	_KERNEL_MAX_REGDEV	最大デバイス登録数	0 ~ 255
	_KERNEL_MAX_OPNDEV	最大デバイスオープン数	0 ~ 255
	_KERNEL_MAX_REQDEV	最大デバイスリクエスト数	0 ~ 255
サイズ指定	_KERNEL_SYS_STKSIZE	システムスタックサイズ	128 ~ 4294967292
	_KERNEL_INIT_TSKSTKSZ	初期タスクスタックサイズ	128 ~ 4294967292

"_KERNEL_MAX_TSKPRI" の値と "_KERNEL_INIT_TSKPRI" の値は、以下の条件を満たす必要があります。

_KERNEL_MAX_TSKPRI _KERNEL_INIT_TSKPRI

_KERNEL_REALMEMSZ はヒープ領域のサイズを指定します。ヒープ領域は、タスク、メッセージバッファ、メモリプールの生成時に TA_USERBUF の属性を指定しないことにより、タスクスタック、メッセージバッファ領域、メモリプール領域をヒープ領域から自動的に獲得することができます。

コンフィギュレーション規定マクロの各定義を省略した場合、とりうる値の最小値をデフォルト値として選択します。たとえば、"_KERNEL_MAX_TSKPRI" の定義を省略した場合、タスク優先度の最大値は "1" になります。また、"_KERNEL_MAX_SEM" の定義を省略した場合、セマフォ最大数は "0" となります。

オブジェクトの最大数が "0" の場合、そのオブジェクトは使用できません。たとえば、セマフォの最大数を "0" に設定して、ユーザプログラム内にセマフォに関連したシステムコールを記述した場合、ユーザシステムのビルド時にセマフォ関連のシステムコール関数が未定義エラーとなります。

コンフィギュレーション規定マクロは以下の書式でコンフィギュレーションファイルに記述します。ただし、通常はこれらの値の操作は、SOFTUNE Workbench のプロジェクトウィンドウの "CFG" タブから行います（「5.3 コンフィギュレーションの設定」参照）。

【コンフィギュレーション規定マクロの書式】

コンフィギュレーション規定マクロ 定義する値

例)

_KERNEL_MAX_TSK	256
_KERNEL_INIT_TSKSTKSZ	0x1000

< 注意事項 >

"_KERNEL_MAX_TSK" で定義するタスク最大数は、カーネル内部で生成する初期タスクを含みます。このため、ユーザプログラムで生成するタスク数が N 個の場合、(N+1) 個以上の値を最大タスク数に定義してください。

また、デバイス管理機能では、以下のオブジェクトを使用するため、オブジェクトの最大値に「ユーザプログラムで使用する数 + デバイス管理で使用する数」の値を設定してください。

- ・セマフォ : デバイスオープンごとに 1 つ使用
 - ・メッセージバッファ : デバイス管理機能全体で 1 つ使用
 - ・イベントフラグ : デバイス管理機能全体で 1 つ使用
-

■ 静的 API

コンフィギュレータにより，ユーザプログラムのモジュールを静的に定義するインタフェースを静的 API とよびます。

静的 API では，初期ルーチン，割込みハンドラ，エラールーチンおよび省電力ルーチンの登録が可能です。このうち，初期ルーチン，エラールーチンおよび省電力ルーチンは静的 API でのみ登録可能ですが，割込みハンドラは，tk_def_int により，ユーザプログラムからも登録可能です。

表 3.13-2 に静的 API の一覧を示します。

表 3.13-2 静的 API 一覧

名称	機能
ATT_INI	初期ルーチンの定義
DEF_INH	割込みハンドラの定義
VATT_ERR	エラールーチンの定義
VDEF_PSR	省電力ルーチンの定義

静的 API は以下の書式でコンフィギュレーションファイルに記述します。ただし，通常はこれらの定義の操作は，SOFTUNE Workbench のプロジェクトウィンドウの "CFG" タブから行います（「5.3 コンフィギュレーションの設定」参照）。

【静的 API の書式】

- ATT_INI
ATT_INI({ 属性, 拡張情報, エントリ });
- DEF_INH
DEF_INH(割込み番号, { 属性, エントリ });
- VATT_ERR
VATT_ERR({ 属性, エントリ });
- VDEF_PSR
VDEF_PSR({ 属性, エントリ });

例)

```
ATT_INI({ TA_HLNG, 0, uint});
DEF_INH(35, { TA_HLNG, inthdr});
VATT_ERR({TA_HLNG, uerr});
VDEF_PSR({TA_HLNG, pow_down});
```

■ コンフィギュレータの起動

コンフィギュレータは SOFTUNE Workbench のプロジェクトから、ビルド、またはメイクのメニューを選択すると自動的に実行します。

コンフィギュレータは、 μ T-REALOS をインストールしたフォルダ配下の "bin" フォルダに、"ftcfs.exe" の名前で配置されます。コンフィギュレータを手動やバッチプロシジャから起動する場合には、以下の書式で起動します。

```
ftcfs -f file_name -cpu cpu_name -out path [ -V ] [ -g ] [ -cif cif_name ]
```

-f file_name	コンフィギュレーションファイル名を file_name で指定します。 指定は省略できません。
-cpu cpu_name	ターゲットの CPU を指定します。 指定は省略できません。
-out path	コンフィギュレータが最終的に出力するカーネル構成ファイルの出力フォルダを "path" で指定します。 指定は省略できません。
-V	コンフィギュレータの起動メッセージを出力します。 コンフィギュレータで呼び出されるツールにも適用されます。
-g	デバッグ情報を出力します。
-cif cif_name	CPU 情報ファイル名を "cif_name" で指定します。 指定が省略された場合には、"[SOFTUNE インストールフォルダ]\lib\911\911.csv" を CPU 情報ファイルとして使用します。

- 起動例

```
ftcfs -f C:\smphys\system.tcf -cpu MB91403 -out C:\smphys
```

< 注意事項 >

[] 内の要素は、記述が省略可能であることを示します。

3.14 デバッグ支援機能

μ T-REALOS で、ユーザプログラムのデバッグ支援のために提供しているアナライザについて説明します。

■ デバッグ支援機能の概要

ユーザプログラムのデバッグは、SOFTUNE Workbench デバッガ（以下、SOFTUNE デバッガとよびます）を使用して、Windows の GUI 画面から操作します。 μ T-REALOS では、SOFTUNE デバッガのプラグインツールとして、アナライザを提供しています。ユーザプログラムに対するデバッグを支援するため、アナライザは以下の機能が含まれています。

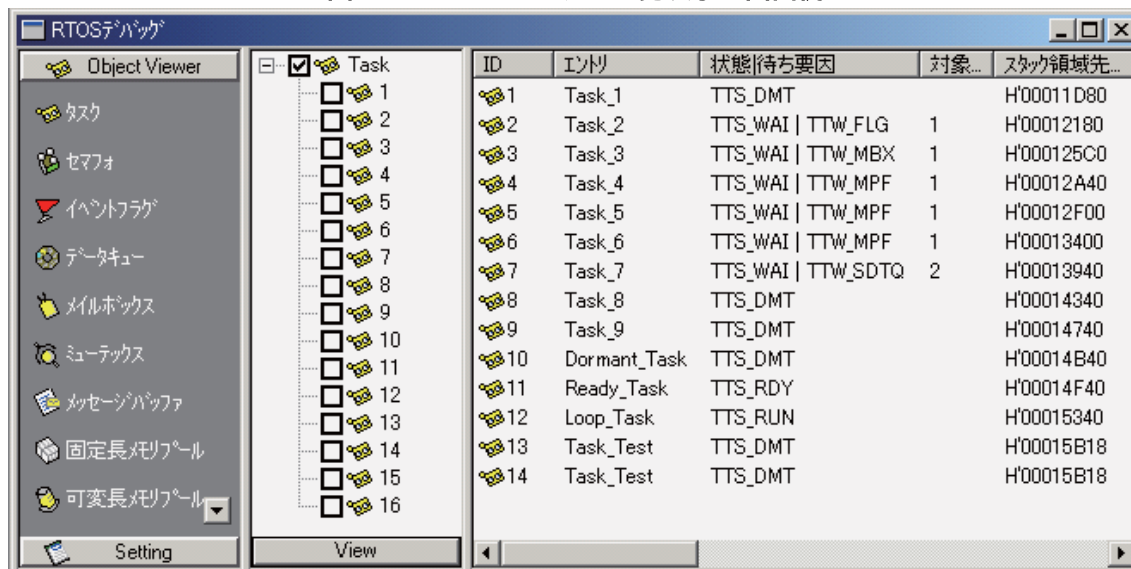
- オブジェクト一覧表示
- OS ブレーク
- ログ
- システムコール発行
- スタック情報
- タスクコンテキスト表示

以降、これらの機能について説明します。これらの機能の詳細および操作方法については、「アナライザガイド」を参照してください。

■ オブジェクト一覧表示

ユーザプログラムで生成したオブジェクトを、種類別にその ID 番号および状態を一覧形式で表示します。図 3.14-1 にオブジェクト一覧表示の画面例を示します。

図 3.14-1 オブジェクト一覧表示の画面例



■ OS ブレーク

OS ブレークはタスク単位にブレークポイントを設定できる機能です。複数のタスクで同一のコードを共有している場合（共通関数など）、特定のタスクがそのコードを実行したときにブレークできます。たとえば、タスク 1、タスク 2、タスク 3 が、共通関数、`"comm_func()"` を呼び出している場合に、`"comm_func()"` がタスク 2 から呼ばれたときだけ、ブレークできます。

また、タスク単位で、以下の条件でもブレークできます。

- タスクが指定したデータをアクセスしたとき。
- タスクの実行権が与えられた、または奪われたとき。
- タスクから呼び出したシステムコールの入り口、または出口。

■ ログ

ユーザプログラム実行中に、その動作ログを取得して、さまざまな表示フォーマットでログの内容を時系列に表示できます。これにより、ユーザプログラムの動作を容易に解析できます。

ログに採取する情報は、以下のとおりで、これらの情報はユーザ定義により、採取の要否を指定できます。

- 割込みハンドラの開始 / 終了
- タイマ割込みハンドラの開始 / 終了
- ディスパッチの開始 / 終了
- システムコールの開始 / 終了

ログの表示形式として、以下の形式があります。

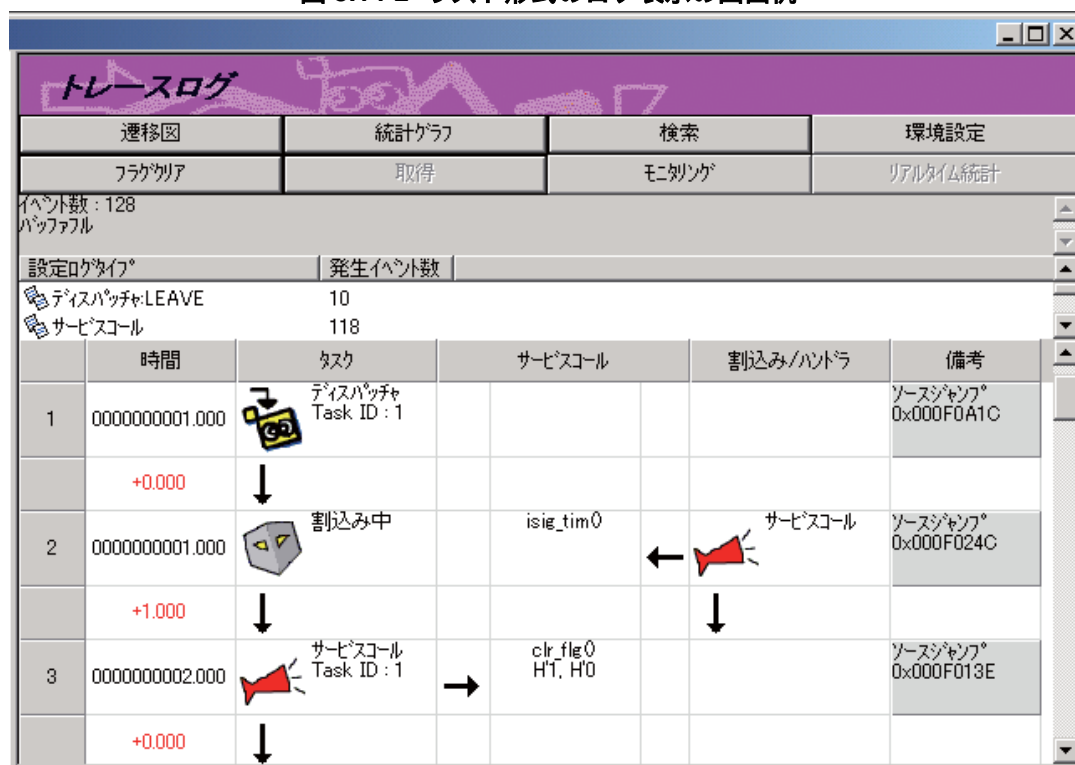
- リスト形式
- 状態遷移図
- 統計形式

また、「状態遷移図」をリアルタイムで表示させる機能として、「モニタリング」があります。

● リスト形式

図 3.14-2 にリスト形式でのログ表示の画面例を示します。

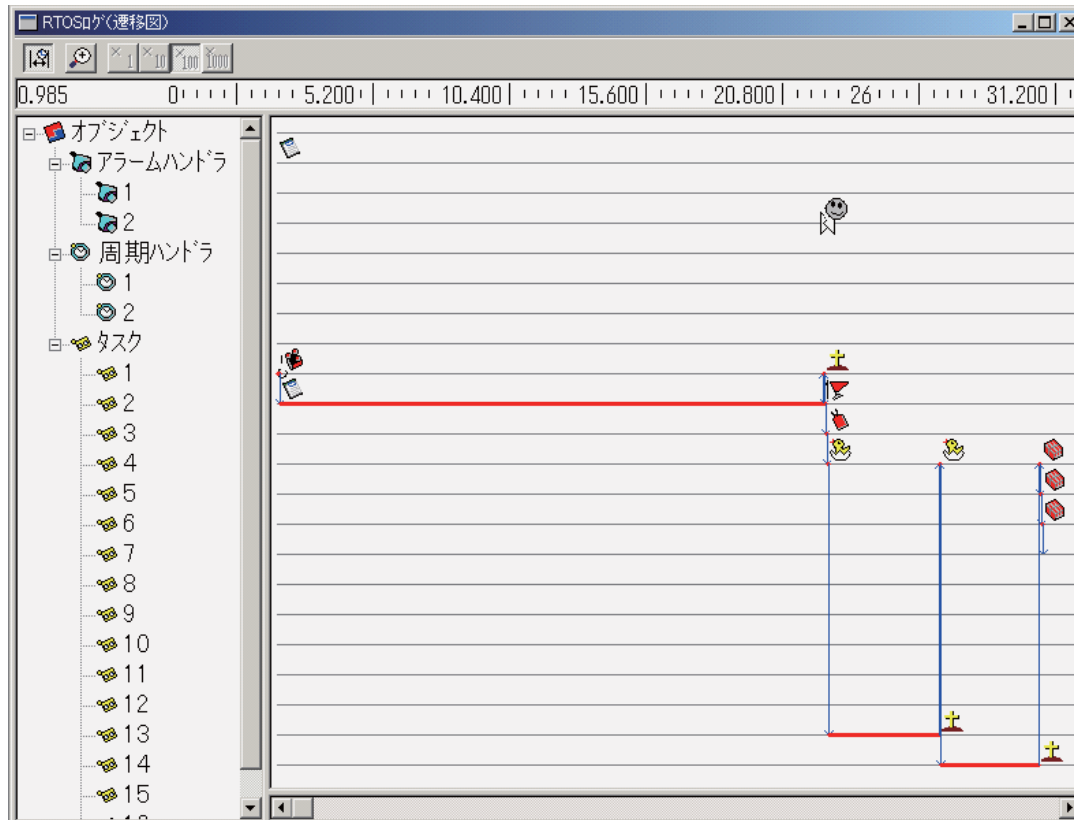
図 3.14-2 リスト形式のログ表示の画面例



● 状態遷移図

図 3.14-3 に状態遷移図形式でのログ表示の画面例を示します。状態遷移図では、タスクのディスパッチの様子が一目で分かります。

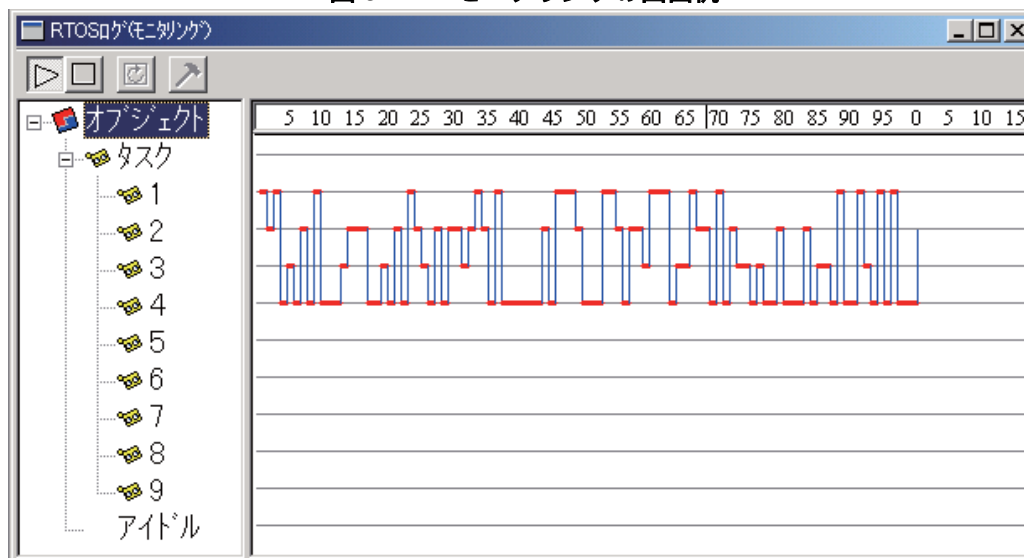
図 3.14-3 状態遷移図の画面例



● モニタリング

図 3.14-3 では、ユーザプログラムを停止させて、停止直前までのログの情報をもとに、状態遷移図を表示しています。これにたいして、ユーザプログラムを動作させながら状態遷移図を表示させることもできます。この機能を「モニタリング」とよんでいます。図 3.14-4 にモニタリングの画面例を示します。

図 3.14-4 モニタリングの画面例



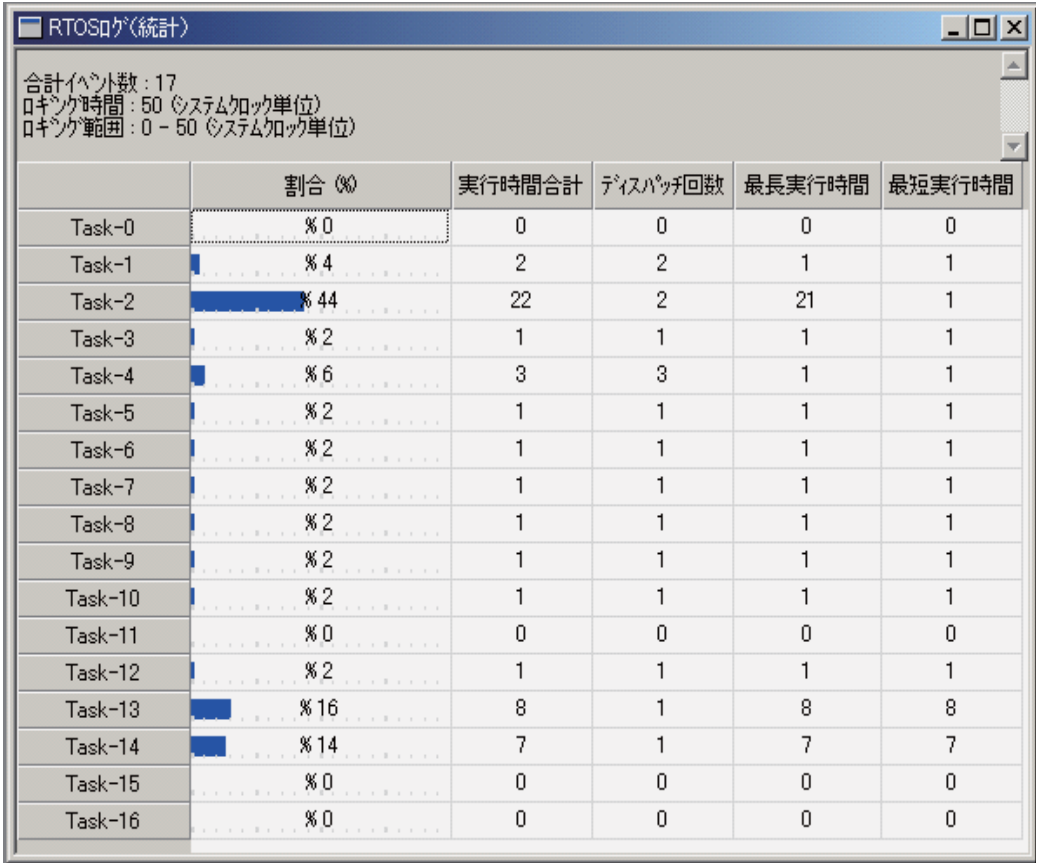
< 注意事項 >

モニタリングの機能は、CPU が DSU4 のデバッグユニットを内蔵していないと使用できません。詳細については「アナライザガイド」を参照してください。

● 統計形式

図 3.14-5 に統計形式でのログ表示の画面例を示します。統計形式では、各タスクの実行時間の割合、ディスパッチ回数などの情報が得られます。

図 3.14-5 統計形式の画面例



■ システムコール発行

ユーザプログラム停止中に、選択したシステムコールの関数を実行できます。

■ スタック情報

タスクスタックの現在の使用量, 最大使用量をグラフで表示します。図 3.14-6 にスタック情報の画面例を示します。

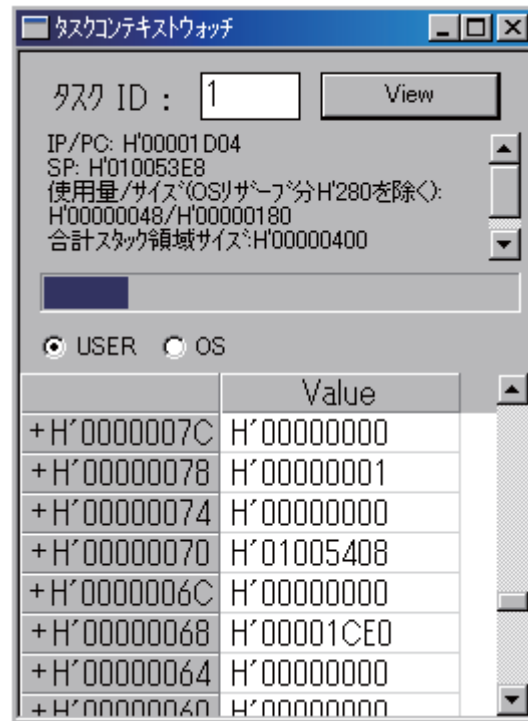
図 3.14-6 スタック情報の画面例



■ タスクコンテキスト表示

指定したタスクのコンテキストの内容を表示します。図 3.14-7 にタスクコンテキスト表示の画面例を示します。

図 3.14-7 タスクコンテキスト表示の画面例



第4章

ユーザプログラムの作成

μT-REALOS 上でユーザプログラムを作成するときの基本的な事項について説明します。

- 4.1 ユーザプログラムの構成
- 4.2 起動のながれ
- 4.3 リセットエントリルーチン
- 4.4 初期ルーチン
- 4.5 タスク
- 4.6 周期ハンドラ
- 4.7 アラームハンドラ
- 4.8 割込みハンドラ
- 4.9 エラールーチン
- 4.10 省電力ルーチン
- 4.11 拡張 SVC ハンドラ
- 4.12 デバイスドライバ
- 4.13 ユーザプログラム作成時の注意事項

4.1 ユーザプログラムの構成

ユーザプログラムの構成について説明します。

■ ユーザプログラムの構成

ユーザプログラムは表 4.1-1 のモジュールで構成されます。ユーザシステムに合わせて必要なモジュールを作成して、ユーザシステムを構築します。

表 4.1-1 ユーザプログラムの構成要素

モジュール名	処理概要	必要性
リセットエントリ ルーチン	ハードウェアリセットにより、最初に起動するルーチン。ハードウェアの初期化などをして、カーネルを起動します。	必須
初期ルーチン	カーネルの初期化処理から呼ばれるルーチン。ユーザプログラムの動作環境を整えます。	必須
タスク	ユーザプログラムのメインの処理を行う。	必須
周期ハンドラ	一定時間ごとに定期的に行う処理がある場合に作成します。	任意
アラームハンドラ	一定時間後に行う処理がある場合に作成します。	任意
割込みハンドラ	ハードウェアの割込みを扱う場合に作成します。システム時刻を使用する場合には、タイマ割込みハンドラの作成が必要です。	任意
エラールーチン	カーネルのエラーをユーザプログラムから扱う場合に作成します。	任意
省電力ルーチン	ユーザプログラムで省電力処理を行う場合に作成します。	任意
拡張 SVC ハンドラ	拡張 SVC ハンドラでユーザ関数を定義する場合に作成します。	任意
デバイスドライバ	デバイスドライバをデバイス管理の API 経由で制御する場合に作成します。	任意

これらについては、本書の「4.3 リセットエントリルーチン」から「4.12 デバイスドライバ」をそれぞれ参照してください。

4.2 起動のながれ

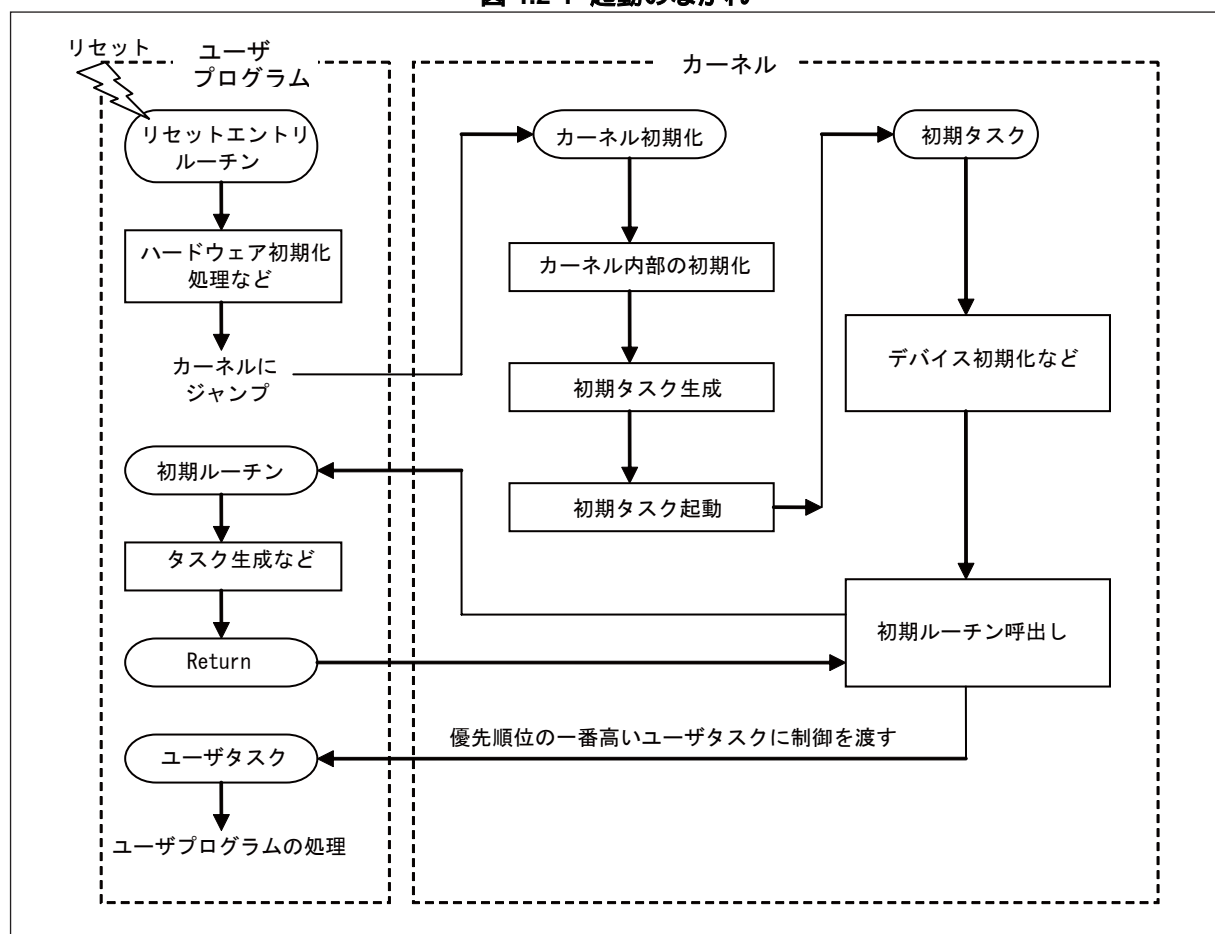
ハードウェアリセットが発生してから，ユーザプログラムのタスクに制御が渡るまでの処理のながれについて説明します。

■ ユーザプログラムの起動

図 4.2-1 に，ハードウェアリセット発生後の処理のながれを示します。

初期タスクでユーザプログラムの初期ルーチンの呼出し完了後に，ユーザプログラムの一番優先順位の高いタスクに制御が渡ります。

図 4.2-1 起動のながれ



4.3 リセットエントリルーチン

リセットエントリルーチンの作成方法について説明します。

■ リセットエントリルーチン

リセットエントリルーチンは、リセットにより起動されて、プロセッサの初期化や、リセット時に初期設定が必要な周辺デバイスの初期化を行います。その後、 μ T-REALOS へ制御を移します。

■ リセットエントリルーチンの処理

リセットエントリルーチンでは、一般的に以下の処理を行います。

- スタックポインタ (SP) の設定 (必須)
SP にリセットエントリルーチン動作中に使用するスタックポインタのアドレスを設定します。
- ハードウェアの初期設定 (任意)
メモリコントローラ、CPU クロック、割込みコントローラ、CPU キャッシュなど、カーネル起動前に設定が必要なハードウェアを初期設定します。
- ROM 内 INIT セクションの RAM へのコピー (任意)
ユーザプログラムを ROM 化した場合、INIT セクション (オブジェクトプログラム内で、初期値付きグローバル変数が格納される領域) は ROM 内に配置されています。これを読み込み / 書込みが可能な RAM 領域にコピーします。
- ユーザプログラムの DATA セクションの初期化 (必須)
ターゲットハードウェアのメモリ上にロードされたユーザシステムの内、DATA セクション (オブジェクトプログラムの中で初期値なしのグローバル変数が格納される領域) の内容を 0 クリアします。
- カーネルの起動 (必須)
リセットエントリルーチンの最後に、カーネルを起動します。カーネルの起動は、システムスタックの設定を行った後に、アセンブラ言語の JMP 命令により、"`__kernel_start`" のラベルにジャンプします。

■ リセットエントリルーチンの具体例

図 4.3-1 から図 4.3-3 にリセットエントリルーチンの各処理の記述例を示します。このコードは、 μ T-REALOS のサンプルプログラム (icrt0.asm) として、製品に添付されています。

図 4.3-1 INIT セクションコピーの記述例

- ・定数RAM_INIT, ROM_INIT, INITはリンカで定義されるため、ユーザプログラムで定義する必要はありません。RAM_INITはRAM上のINITセクションの先頭アドレス, ROM_INITは、ROM上のINITセクションの先頭アドレス, INITはINITセクションの大きさ(バイト数)をあらわします。
- ・以下の記述例では, "copy_rom1"で1バイトずつ, "copy_rom2"で4バイト単位のコピーを行っています。

```

/*
 *      Initialization of 'data' area (ROM startup)
 */
        ldi    #_RAM_INIT, r0
        ldi    #_ROM_INIT, r1
        ldi    #sizeof(INIT), r2
        cmp    #0, r2
        beq:d  copy_rom_end
        ldi    #3, r12
        and    r2, r12
        beq:d  copy_rom2
        mov    r2, r13
        mov    r2, r3
        sub    r12, r3
copy_rom1:
        add    #-1, r13
        ldub   @(r13, r1), r12
        cmp    r3, r13
        bhi:d  copy_rom1
        stb    r12, @(r13, r0)
        cmp    #0, r3
        beq:d  copy_rom_end
copy_rom2:
        add    #-4, r13
        ld     @(r13, r1), r12
        bgt:d  copy_rom2
        st     r12, @(r13, r0)
copy_rom_end:

```

図 4.3-2 DATA セクション 0 クリアの記述例

- ・ 定数 DATA, sizeof DATA はリンカで定義されるため、ユーザプログラムで定義する必要はありません。DATA は DATA セクション領域の先頭アドレスです。sizeof DATA は、DATA セクションの大きさ(バイト数)です。
- ・ 以下の記述例では、"clear_ram0" で 4 バイトずつ 0 クリアを行い、領域最終の 4 バイトに満たない領域を 1 バイトずつ "clear_ram2" で 0 クリアしています。

```

/*
 *      Clear 'bss' area
 */
        ldi:8    #0, r0
        ldi      #sizeof DATA & ~0x3, r1
        ldi      #DATA, r13
        cmp      #0, r1
        beq      clear_ram1
clear_ram0:
        add2     #-4, r1
        bne:d    clear_ram0
        st       r0, @(r13, r1)
clear_ram1:
        ldi:8    #sizeof DATA & 0x3, r1
        ldi      #DATA + (sizeof DATA & ~0x3), r13
        cmp      #0, r1
        beq      clear_ramE
clear_ram2:
        add2     #-1, r1
        bne:d    clear_ram2
        stb      r0, @(r13, r1)
clear_ramE:

```

図 4.3-3 カーネル起動の記述例

- ・ カーネルスタックを SP レジスタに設定します。
- ・ "`__kernel_start`" のラベルにジャンプして、カーネルを起動します。

```

        ldi:32   #__kernel_sstack_end, sp // Set SP(SSP)
        ldi:32   #__kernel_start, r0     // System startup
        jmp      @r0

```

4.4 初期ルーチン

初期ルーチンの作成方法について説明します。

■ 初期ルーチンの処理

初期ルーチンは、カーネルの初期化処理で生成する初期タスクから呼び出します。初期ルーチンは、ユーザプログラムに合わせて自由に処理を記述できますが、一般的には、以下の処理を行います。

- タスク、セマフォ、タイムイベントハンドラなど、ユーザプログラムの動作に必要なオブジェクトの生成、起動
 - デバイスドライバの初期化、登録
 - ハードウェアの初期化
 - タイマ割込みの起動
-

< 注意事項 >

初期ルーチンは割込み許可状態で実行されます。

■ 初期ルーチンの記述形式

初期ルーチンは以下のように記述します。

図 4.4-1 初期ルーチンの記述形式

```
void sample_init(void)
{
    /*
     初期ルーチンの処理
    */
    return;
}
```

■ 初期ルーチンの記述例

図 4.4-2 に初期ルーチンの記述例を示します。このコードは、 μ T-REALOS のサンプルプログラム (init_task.c) として、製品に添付されています。

図 4.4-2 初期ルーチンの記述例

```

・ システムクロック用のタイマ起動 (START_TIMER0()) 後、セマフォと 3 つのタスクを
  (タスク ID=tsk1, tsk2, tsk3) を生成しています。その後、生成した 3 つのタスクを
  起動しています。タスク ID が tsk1 のタスクと tsk2 のタスクは同じ関数 (task1) を
  使用しています。

static UB task1_stack[0x400], task2_stack[0x400], task3_stack[0x400];
void uinit_task(void)
{
    ID tsk1, tsk2, tsk3;
    T_CTSK ctsk;
    T_CSEM csem;

    START_TIMER0(); /* Start Timer0 for System clock */

    csem.sematr = TA_TFIFO | TA_FIRST;
    csem.isemcnt = 0;
    csem.maxsem = 1;
    sem1 = tk_cre_sem(&csem); /* Create semaphore */

    ctsk.exinf = (VP)1;
    ctsk.tskatr = TA_HLNG | TA_RNGO | TA_USERBUF;
    ctsk.task = task1;
    ctsk.itstkpri = 1;
    ctsk.stksz = sizeof(task1_stack);
    ctsk.bufptr = task1_stack;
    tsk1 = tk_cre_tsk(&ctsk); /* Create task1 */

    ctsk.exinf = (VP)2;
    ctsk.tskatr = TA_HLNG | TA_RNGO | TA_USERBUF;
    ctsk.task = task1;
    ctsk.itstkpri = 2;
    ctsk.stksz = sizeof(task2_stack);
    ctsk.bufptr = task2_stack;
    tsk2 = tk_cre_tsk(&ctsk); /* Create task2 */

    ctsk.exinf = (VP)3;
    ctsk.tskatr = TA_HLNG | TA_RNGO | TA_USERBUF;
    ctsk.task = task2;
    ctsk.itstkpri = 3;
    ctsk.stksz = sizeof(task3_stack);
    ctsk.bufptr = task3_stack;
    tsk3 = tk_cre_tsk(&ctsk); /* Create task3 */

    tk_sta_tsk(tsk1, 1); /* Start task1 */
    tk_sta_tsk(tsk2, 2); /* Start task2 */
    tk_sta_tsk(tsk3, 3); /* Start task3 */
}

```

4.5 タスク

タスクの作成方法について説明します。

■ タスクの記述形式

タスクは、以下のように記述します。

図 4.5-1 タスクの記述形式

```
void task(INT stacd, VP exinf)
{
    /*
     * タスクプログラム本体の処理
     */
    tk_ext_tsk(); または tk_exd_tsk(); /* タスクの終了 */
}
```

stacd には、タスク起動時 (tk_sta_tsk 呼出し時) に指定したタスク起動コード (stacd) を渡します。exinf には、タスク生成時 (tk_cre_tsk) に指定した拡張情報 (exinf) を渡します。関数 (タスク) は、単純な return で終了できません。必ず tk_ext_tsk または tk_exd_tsk で終了してください。

■ タスクの生成

タスクを生成する場合、tk_cre_tsk を呼び出します。以下に例を示します。この例では、関数 "task1" をタスク優先度 1 で生成しています。tk_cre_tsk が正常に終了した場合、タスク ID を復帰値として返します。

図 4.5-2 タスク生成の記述例

```
ID tid1; /* task1のタスクID */
T_CTSK ctsk; /* tk_cre_tskの入力パラメータ */
INT task1_stack[256]; /* タスクのスタック領域 */

ctsk.exinf = (VP)1; /* 拡張情報=1 */
ctsk.tskatr = TA_HLNG | TA_RNG0 | TA_USERBUF; /* 属性 */
ctsk.task = task1; /* タスクの先頭アドレス */
ctsk.itskpri = 1; /* タスク優先度 */
ctsk.stksz = sizeof(task1_stack); /* スタックサイズ */
ctsk.bufptr = task1_stack; /* スタックの先頭アドレス */
tid1 = tk_cre_tsk(&ctsk); /* タスクの生成 */
```

tk_cre_tsk の詳細については、「API リファレンス」の「3.3.1 tk_cre_tsk」を参照してください。

■ タスクの起動

tk_cre_tsk で生成されたタスクの初期状態は休止状態になります。このため、tk_sta_tsk を呼び出してこのタスクを動作させます。以下の例では、タスク ID が tid1 のタスクを起動しています。

図 4.5-3 タスク起動の記述例

```
tk_sta_tsk(tid1, 1); /* タスクIDがtid1のタスクを起動 */
```

tk_sta_tsk で起動されたタスクの状態は、実行可能状態となります。ほかの実行状態および実行可能状態のタスクよりも優先度が高い場合、このタスクが実行状態になります。

tk_sta_tsk の詳細については、「API リファレンス」の「3.3.3 tk_sta_tsk」を参照してください。

■ タスクの具体例

図 4.5-4 にタスクの記述例を、図 4.5-5 に記述例のプログラムの動作図を示します。このコードは、μT-REALOS のサンプルプログラム (init_task.c) として、製品に添付されています。また、本具体例のタスクは、図 4.4-2 に挙げた初期ルーチンで生成 / 起動されています。

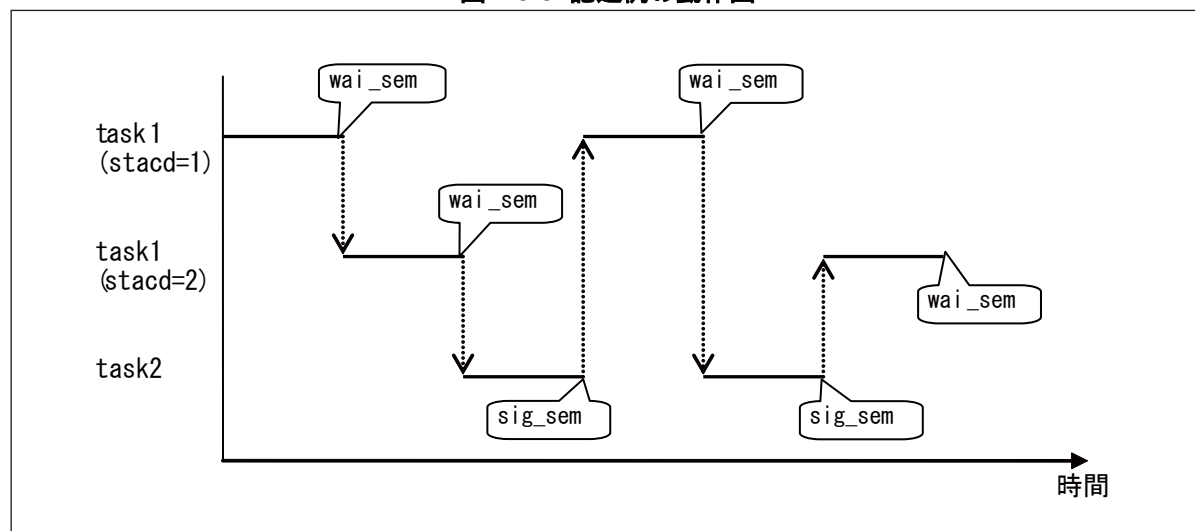
図 4.5-4 タスクの記述例

- ・ 関数task1はstacd=1で起動されたタスクと、stacd=2で起動されたタスクの2つのタスクで動作します。各タスクはtk_wai_semシステムコールにより、sem1のセマフォ待ち状態になります。
- ・ task2は、tk_sig_semシステムコールにより、sem1のセマフォ資源を解放します。これにより、task1が待ち状態から解除されます。
- ・ タスクの優先順位は、task1(stacd=1) > task1(stacd=2) > task2です。

```
static void task1( INT stacd, VP exinf )
{
    if(stacd == 1){
        while (1) {
            tk_wai_sem(sem1, 1, TMO_FEVR);
        }
    }
    else if (stacd == 2){
        while (1) {
            tk_wai_sem(sem1, 1, TMO_FEVR);
        }
    }
    else{
        tk_ext_tsk();    /* Exit task */
    }
}

static void task2( INT stacd, VP exinf )
{
    if (stacd == 3){
        while (1) {
            tk_sig_sem(sem1, 1);
        }
    }
    else{
        tk_ext_tsk();    /* Exit task */
    }
}
```

図 4.5-5 記述例の動作図



4.6 周期ハンドラ

周期ハンドラの作成方法について説明します。

■ 周期ハンドラの記述形式

周期ハンドラは、以下のように記述します。

図 4.6-1 周期ハンドラの記述例

```
void cychdr1(VP exinf)
{
    /* 周期ハンドラの処理 */
    return; /* 周期ハンドラの終了 */
}
```

■ 周期ハンドラの生成

周期ハンドラを生成する場合、tk_cre_cyc を呼び出します。以下に例を示します。この例では、関数 "cychdr1" を起動周期 1 秒 (1000ms) の周期ハンドラとして生成しています。周期ハンドラの生成が正常に終了した場合、周期ハンドラ ID を復帰値として返します。

図 4.6-2 周期ハンドラ生成の記述例

```
ID cycid1; /* 周期ハンドラID */
T_CCYC ccyc; /* tk_cre_cyc の入力パラメータ */
ccyc.exinf = (VP)1; /* 拡張情報=1 */
ccyc.cycatr = TA_HLNG | TA_RNGO | TA_STA; /* 属性 */
ccyc.cychdr = cychdr1; /* 周期ハンドラの先頭アドレス */
ccyc.cyctim = 1000; /* 起動周期 */
ccyc.cycphs = 0; /* 起動位相 */
cycid1 = tk_cre_cyc(&ccyc); /* 周期ハンドラの生成 */
```

■ 周期ハンドラの起動

停止状態の周期ハンドラを動作状態にするには、tk_sta_cyc を呼び出します。以下の例では、周期ハンドラ ID が cycid1 の周期ハンドラを起動しています。

図 4.6-3 周期ハンドラ起動の記述例

```
tk_sta_cyc(cycid1); /* IDがcycid1の周期ハンドラを起動 */
```

4.7 アラームハンドラ

アラームハンドラの作成方法について説明します。

■ アラームハンドラの記述形式

アラームハンドラは、以下のように記述します。

図 4.7-1 アラームハンドラの記述例

```
void almhdr1(VP exinf)
{
    /* アラームハンドラの処理 */
    return; /* アラームハンドラの終了 */
}
```

exinf には、アラームハンドラ生成 (tk_cre_alm) 時に指定した拡張情報を渡します。

■ アラームハンドラの生成

アラームハンドラを生成する場合、tk_cre_alm を呼び出します。以下に例を示します。この例では、関数 "almhdr1" をアラームハンドラとして生成しています。アラームハンドラの生成が正常に終了した場合、アラームハンドラ ID を復帰値として返します。アラームハンドラは生成後、停止状態となります。

図 4.7-2 アラームハンドラ生成の記述例

```
ID almid1;          /* アラームハンドラ ID */
T_CALM calm;        /* tk_cre_alm の入力パラメータ */

calm.exinf = (VP)1; /* 拡張情報=1 */
calm.almatr = TA_HLNG | TA_RNGO; /* 属性 */
calm.almhdr = almhdr1; /* アラームハンドラの実行アドレス */
almid1 = tk_cre_alm(&calm); /* アラームハンドラの生成 */
```

■ アラームハンドラの起動

停止状態のアラームハンドラを動作状態にするには、tk_sta_alm を呼び出します。以下の例では、アラームハンドラ ID が almid1 のアラームハンドラを、タイムアウト時間 100ms で起動しています。

図 4.7-3 アラームハンドラ起動の記述例

```
tk_sta_alm(almid1, 100); /* IDがalmid1のアラームハンドラを
                           タイムアウト時間 100msで起動 */
```

4.8 割込みハンドラ

割込みハンドラの作成方法について説明します。

■ 割込みハンドラの記述形式

割込みハンドラは以下のように記述します。

図 4.8-1 割込みハンドラの記述例

```
void sample_inthdr(void)
{
    /* 割込みハンドラ本体 */
}
```

割込みハンドラは、タスク独立部で実行します。また、割込み許可状態で起動します。このため、割込みハンドラ実行中に、多重に割込みハンドラが起動する場合があります。詳細については、「FR ファミリ インストラクションマニュアル」の「第 4 章 リセット, EIT 処理」を参照してください。

■ 割込みハンドラの登録

割込みハンドラの登録は、静的に登録する方法と動的に登録する方法があります。静的に登録する場合には、「5.3 コンフィギュレーションの設定」を参照してください。動的に登録する場合には、ユーザプログラムから、tk_def_int を呼び出します。

• 例

ベクタ番号 24 のタイマ割込みのハンドラ, "timer" を割込みハンドラとしてユーザプログラムで登録する場合。

図 4.8-2 ユーザプログラムによる割込みハンドラの登録例

```
T_DINT dint;
ER err;

dint.intatr = TA_HLNG|TA_RNGO;      /* 属性 */
dint.inthdr = timer;               /* 割込みハンドラの実体アドレス */
err = tk_def_int(&dint);            /* 割込みハンドラの登録 */
```

割込みハンドラをユーザプログラムで登録する場合には、コンフィギュレータ規定マクロの "_KERNEL_USE_TKDEFINT" を 1 に設定して、コンフィギュレーションを実行してください。また、ベクタテーブルを ROM に配置して動作させる場合には、割込みハンドラを静的 API で登録してください。

■ タイマ割込みハンドラ

タイムイベントハンドラ、タイムアウトおよびシステム時刻の機能を使う場合、1ms の周期でタイマ割込みを発生させ、その割込みハンドラによりシステム時刻を更新してください。システム時刻は、isig_tim を呼び出すことにより更新します。

以下にタイマ割込みの例を示します。

図 4.8-3 タイマ割込みハンドラの記述例

```
void timer(void)
{
    /*
     タイマ割込み要因のクリア
    */
    isig_tim();
}
```

< 注意事項 >

割込みハンドラをアセンブラで記述する場合には、以下の点に注意してください。

- 割込みハンドラの呼出し、および割込みハンドラからの復帰は、OS を介しません。
 - OS でレジスタの退避・復元、スタックの設定は行わないため、割込みハンドラ側でこれらの処理を行ってください。
-

4.9 エラールーチン

エラールーチンの作成方法について説明します。

■ エラールーチンの記述形式

エラールーチンの記述形式について、以下に示します。

図 4.9-1 エラールーチンの記述形式

```
void sample_errrtn (UINT errrtn, INT errinf1, INT errinf2)
{
    /* エラールーチン本体 */
}
```

errrtn, errinf1, errinf2 には以下の情報を渡します。

- errrtn : エラー要因
 - = _KERNEL_ERR_SYS_DOWN (0x01) : システムダウン
 - = _KERNEL_ERR_INI_ERR (0x02) : 初期設定エラー
 - = _KERNEL_ERR_EIT_DOWN (0x04) : 未定義の割込み
- errinf1 : エラー情報 1
 - 【_KERNEL_ERR_SYS_DOWN の場合】
 - = 0x1 : タスク独立部から tk_ext_tsk がよばれた。
 - = 0x2 : タスク独立部から tk_exd_tsk がよばれた。
 - = 0x3 : ディスパッチ禁止状態で tk_ext_tsk がよばれた。
 - = 0x4 : ディスパッチ禁止状態で tk_exd_tsk がよばれた。
 - 【_KERNEL_ERR_INI_ERR の場合】
 - 初期設定エラー情報
 - = 0x1 : ヒープ領域割当てエラー
 - = 0x2 : システムスタートエラー
 - = 0x3 : 初期タスク起動エラー
 - = 0x4 : モジュール初期化エラー
 - = 0x5 : 電源オフ処理時エラー
 - 【_KERNEL_ERR_EIT_DOWN の場合】
 - 不定値
- errinf2 : エラー情報 2
 - 未使用です。将来の拡張のために予約します。

■ エラールーチンの登録

エラールーチンの登録については、「5.3 コンフィギュレーションの設定」を参照してください。

4.10 省電力ルーチン

省電力ルーチンの作成方法について説明します。

■ 省電力ルーチンの記述形式

省電力ルーチンは、カーネル内でアイドル状態になったときに呼び出される処理です。
省電力モードに移行する処理を関数内に記述します。

図 4.10-1 省電力ルーチンの記述例

```
void usr_low_pow( void )
{
    /* 省電力モードに移行する処理を記述 */
}
```

■ 省電力ルーチンの登録

省電力ルーチンの登録については、「5.3 コンフィギュレーションの設定」を参照してください。

< 注意事項 >

省電力ルーチン内でシステムコールを呼び出した場合には、その動作は保証しません。

4.11 拡張 SVC ハンドラ

拡張 SVC ハンドラの作成方法および呼出し方法について説明します。

■ 拡張 SVC ハンドラの記述形式

拡張 SVC ハンドラの記述形式について、以下に示します。

図 4.11-1 拡張 SVC ハンドラの記述形式

```
INT svchdr (VP pk_para, FN fncd)
{
    /*
     * fncdにより、分岐して処理
     */
    return retcode; /* 拡張SVCハンドラの終了 */
}
```

pk_para は呼出し元から渡されたパラメータをパケット形式にしたものです。パケットの形式はサブシステムで任意に決められます。

fncd は機能コードは、下位 8 ビットにサブシステム ID が入ります。残りの上位ビットは、サブシステム側で任意に決められます。

■ 拡張 SVC ハンドラの呼出し形式

拡張 SVC ハンドラは、fncd の値を r0 レジスタに設定して、割り込み番号 64 のソフトウェア割り込みでユーザプログラムから呼び出します。このため、拡張 SVC ハンドラの呼出し部はアセンブラでユーザプログラムとして記述してください。

図 4.11-2 拡張 SVC ハンドラの呼出し形式

```
#define FUNC1_FNCD    0x10A    /* ssyid = 10 */
INT func1(int arg1, int arg2)
{
    __asm( "        ldi:32 #FUNC1_FNCD, r0" );
    __asm( "        int      #64" );
}
```

図 4.11-2 の例では、サブシステム ID(ssyid) が 10 の SVC ハンドラを呼び出します。呼び出された SVC ハンドラでは、pk_para に arg1, arg2 を格納したパケットの先頭アドレスが、fncd に "0x10A" が渡されます。

4.12 デバイスドライバ

デバイスドライバの作成方法について説明します。

■ デバイスドライバインタフェース

μT-Kernel仕様では、デバイス管理機能により、デバイスドライバのインタフェースを統一して、デバイスドライバの移植性を高めています。以降、デバイスドライバインタフェースにもとづいたドライバの作成方法について説明します。なお、デバイスドライバの詳細については、「API リファレンス」の「付録 C デバイスドライバインタフェース」を参照してください。

■ デバイス名の決定

デバイス名はデバイスの種別単位に付与する名前です。最大 8 バイトの文字列です。

デバイス名は以下の形式です。

種別	ユニット	サブユニット
----	------	--------

デバイス名は、以下の要素により構成されます。

種別 : デバイスの種別を示す名前です。使用可能な文字は a ~ z, A ~ Z です。

ユニット : 物理的なデバイスを示す番号です。使用可能な文字は a ~ z です。1 文字で指定します。ユニットごとに a から順に割り当てます。

サブユニット : 論理的なデバイスを示す番号です。使用可能な文字は 0 ~ 254 の数字で最大 3 文字です。サブユニットごとに 0 から順に割り当てます。

■ オープン関数 (openfn) の作成

オープン関数は、tk_opn_dev がユーザプログラムから呼び出されたときに、カーネルから呼び出されます。オープン処理関数では、デバイスのデータにアクセスできるよう、準備を行います。

オープン処理関数の詳細については、「API リファレンス」の「付録 C デバイスドライバインタフェース」の「デバイス処理関数 オープン処理関数 (openfn)」を参照してください。

```

ER ercd = openfn(ID devid, UINT omode, VP exinf)
{
    /*
     デバイスのオープン処理
    */
}

```


■ クローズ関数 (closefn) の作成

クローズ関数は、tk_cls_dev がユーザプログラムから呼び出されたときに、カーネルから呼び出されます。

クローズ処理関数が呼び出された場合、デバイスへのアクセスが終了したことを意味し、ドライバでは、必要に応じてデバイスの終了処理を行います。

```
ER ercd = closefn(ID devid, UINT option, VP exinf)
{
    /*
     * デバイスの終了処理
     */
}
```

■ 処理開始関数 (execfn) の作成

処理開始関数は、tk_rea_dev, tk_srea_dev, tk_wri_dev, tk_swri_dev がユーザプログラムから呼び出されたときに、カーネルから呼び出されます。

処理開始関数では、処理すべきデータが、引数に設定されて呼び出されますが、これらのデータの処理が完了してから復帰するのではなく、処理を受け付けた時点で復帰します。たとえば、tk_wri_dev 経由で、デバイスに書き込むデータが渡ってきた場合、デバイスに対して書き込みの開始指示が完了した時点で復帰し、デバイスの書き込み処理の完了を待つ必要はありません。

処理開始関数の詳細については、「API リファレンス」の「付録 C デバイスドライバインタフェース」の「デバイス処理関数 処理開始関数 (execfn)」を参照してください。

```
ER ercd = execfn(T_DEVREQ *devreq, TMO tmout, VP exinf)
{
    /*
     * デバイスの処理開始
     */
}
```

■ 完了待ち関数 (waitfn) の作成

完了待ち関数は、tk_wai_dev, tk_srea_dev, tk_swri_dev がユーザプログラムから呼び出されたときに、カーネルから呼び出されます。

完了待ち関数では、処理開始関数で受け付けた I/O 要求の完了を待ちます。このため、完了待ち関数では、待ち状態に入るシステムコール (tk_slp_tsk など) を使用する場合があります。

完了待ち関数の詳細については、「API リファレンス」の「付録 C デバイスドライバインタフェース」の「デバイス処理関数 完了待ち関数 (waitfn)」を参照してください。

```
INT pktno = waitfn(T_DEVREQ *devreq, INT nreq, TMO tmout, VP exinf)
{
    /*
     * デバイスの処理完了待ち
     */
}
```

■ 処理中止関数 (abortfn) の作成

処理中止関数は、tk_cls_dev により、ユーザプログラムからデバイスのクローズ支持が出された時点で、そのデバイスに対して、未完了の I/O 要求が残っている場合に、カーネルから呼び出されます。

処理中止関数では、引数で指定された I/O 要求の中止処理を行います。デバイスに対して、I/O を中止させ、タスクが I/O 完了待ち状態に入っている場合には、その待ち状態を解除します。

処理中止関数の詳細については、「API リファレンス」の「付録 C デバイスドライバインタフェース」の「デバイス処理関数 処理中止関数 (abortfn)」を参照してください。

```
INT pktno = waitfn(T_DEVREQ *devreq, INT nreq, TMO tmout, VP exinf)
{
    /*
     * デバイスの中止処理
     */
}
```

■ イベント関数 (eventfn)

イベント関数は、tk_sus_dev, tk_evt_dev がユーザプログラムから呼び出されたときに、カーネルから呼び出されます。ユーザプログラムやカーネルから、デバイスに対してなんらかのイベントを伝えたい場合に呼び出されます。

イベント関数では、イベントの種別が引数に渡されますので、ドライバでは、そのイベントに応じた処理を行います。

```
INT rtnod = eventfn(INT evttyp, INT evtinf, VP exinf)
{
    /*
     * デバイスのイベント処理
     */
}
```

4.13 ユーザプログラム作成時の注意事項

μT-REALOS のユーザプログラム作成時の注意事項について説明します。

■ プログラム全般に関する注意事項

- `"_KERNEL"`, `"_kernel"`, `"tk_"`, `"tm_"` および `"knl_"` で始まる内部識別子
μT-REALOS のカーネルは、上記で始まるシンボルやマクロを使用しています。ユーザプログラムでこれらシンボルやマクロを使用すると二重定義になるため使用しないでください。
- μT-REALOS の管理レジスタ
PS レジスタの ILM フィールドは μT-REALOS が使用します。このフィールドはユーザプログラムで変更しないでください。
- カーネルのインクルードファイル
システムコールを使用するユーザプログラムは、`"[SOFTUNE インストールディレクトリ]\utkernel\911\include\tk\tkernel.h"` をインクルードしてください。

■ システムコール全般に関する注意事項

- タスク独立部、およびディスパッチ禁止状態から呼出し可能なシステムコール
呼出しが許可されていない場合、`E_CTX` エラーもしくは例外が発生します。呼出しの可否は「API リファレンス」の「3.1 システムコール一覧」を参照してください。また、`isig_tim`, `tk_ret_int` をタスク部から発行した場合の動作は保証しません。
- システムコールのエラーチェックの省略
エントリアドレスやパケットアドレスのチェックは行っていません。不正なアドレスを指定すると、異常動作となる場合があります。

■ タスクに関する注意事項

- スタック定義
スタック領域は、先頭アドレスが 4 バイト境界になるように領域を確保してください。
- ディスパッチが保留されている状態の実行中タスクの状態遷移
ディスパッチが保留されている状態で、タスク独立部から `tk_ter_tsk` や `tk_sus_tsk` を呼び出し、実行中のタスクを強制待ち状態や休止状態へ移行させる場合、ディスパッチが起こる状態になるまで状態遷移は遅延します。この場合の実行中のタスクは、実行状態を継続しますが、`tk_ref_tsk` でタスクの状態を参照した場合、強制待ち状態や休止状態になります。

■ 割込みに関する注意事項

- 割込みハンドラとタイムイベントハンドラの実行優先順位

それぞれのハンドラは、定義した割込みのレベルに応じて実行順位が決まります。詳細は、「FR ファミリ インストラクションマニュアル」の「第 4 章 リセット, EIT 処理」を参照してください。

タイムイベントハンドラは、isig_tim を呼び出すシステムクロック用割込みハンドラの割込みレベルで実行します。

システムスタックサイズ的设计時には、これらのハンドラの多重起動を考慮して、1 レベルあたりの割込みに 80 バイトを加えてください。

第5章

システム構築方法

ユーザシステムの構築方法について説明します。

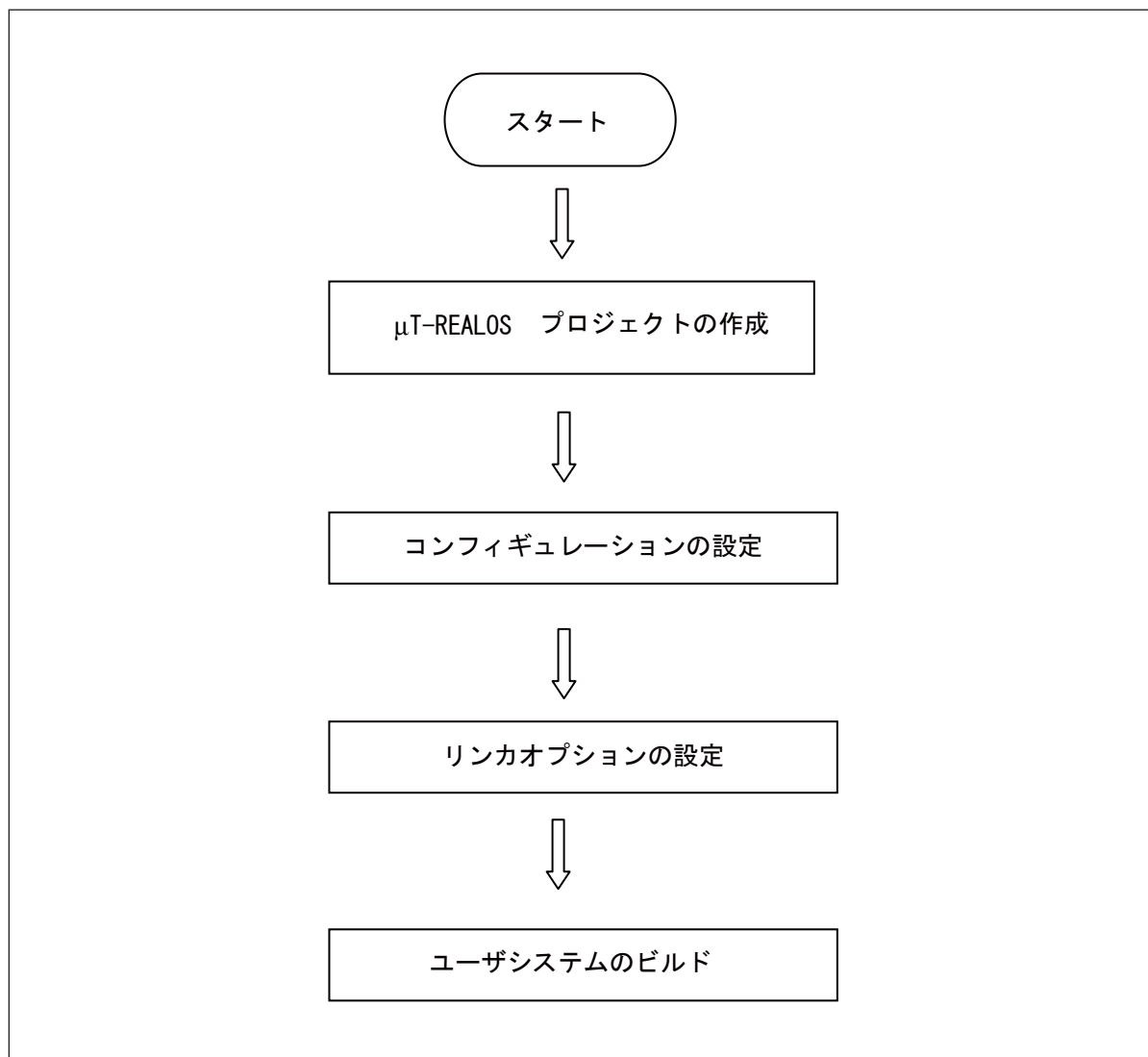
- 5.1 システム構築手順
- 5.2 μ T-REALOS のプロジェクト作成
- 5.3 コンフィギュレーションの設定
- 5.4 リンカオプションの設定
- 5.5 ユーザシステムのビルド

5.1 システム構築手順

μT-REALOS用のユーザプログラムのコンパイル, コンフィギュレーション, リンクなどのシステム構築手順について説明します。

■ システム構築手順

以下の手順で μT-REALOS のユーザシステムの構築を行います。



5.2 μ T-REALOS のプロジェクト作成

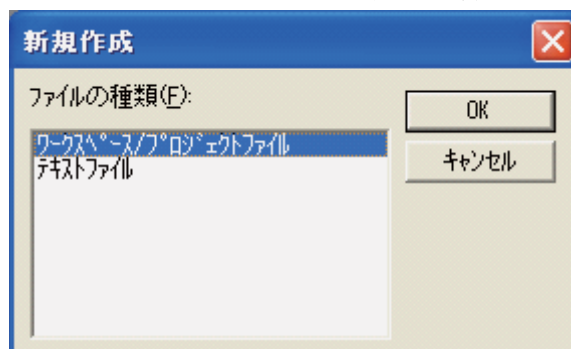
SOFTUNE Workbench で μ T-REALOS プロジェクトを作成する方法について説明します。

■ μ T-REALOS プロジェクトの作成

μ T-REALOS プロジェクトは、以下の手順で作成します。

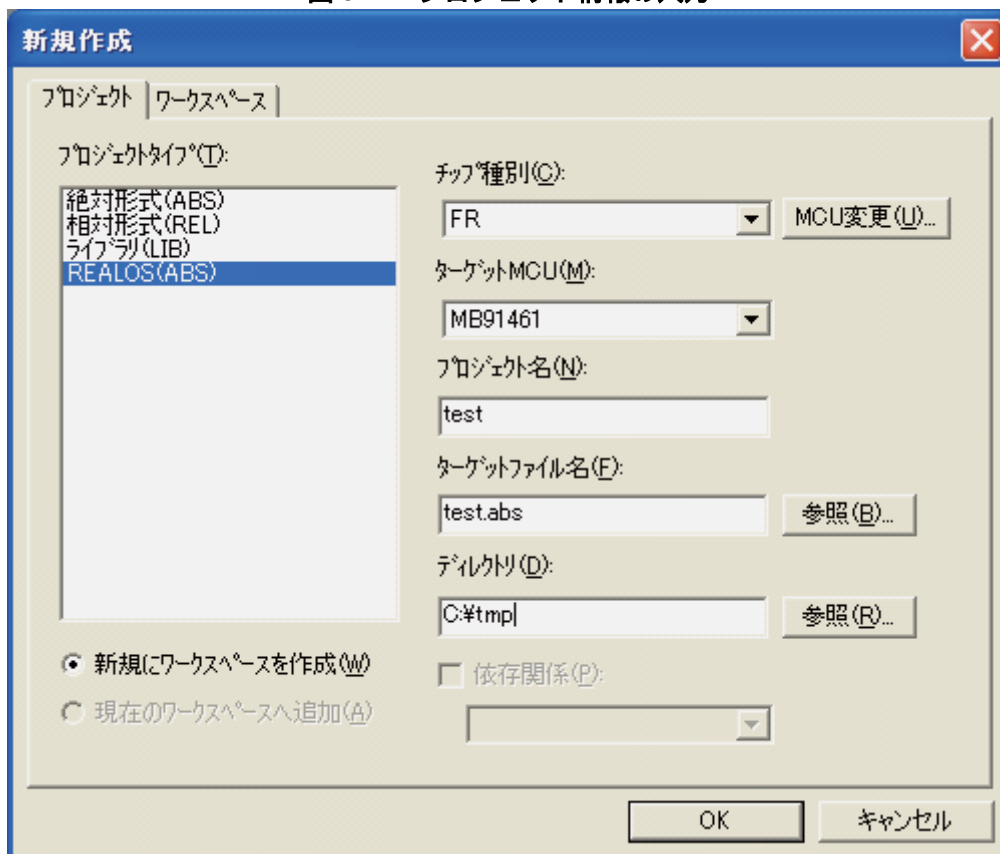
- SOFTUNE Workbench 上で [ファイル]-[新規作成] メニューを選択します。オープンした「新規作成」のダイアログで、ファイルの種類から「ワークスペース / プロジェクトファイル」を選択して、「OK」ボタンをクリックします。

図 5.2-1 ファイル種別の選択



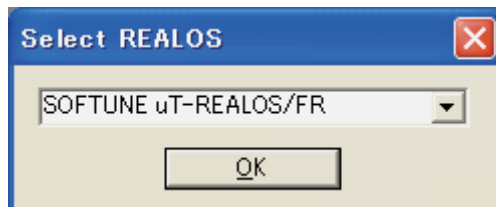
- 「新規作成」のダイアログのプロジェクトタブを選択して、プロジェクトタイプから「REALOS(ABS)」を選択します。「チップ種別」、「ターゲット MCU」および「プロジェクト名」などを入力して、「OK」ボタンをクリックします。

図 5.2-2 プロジェクト情報の入力



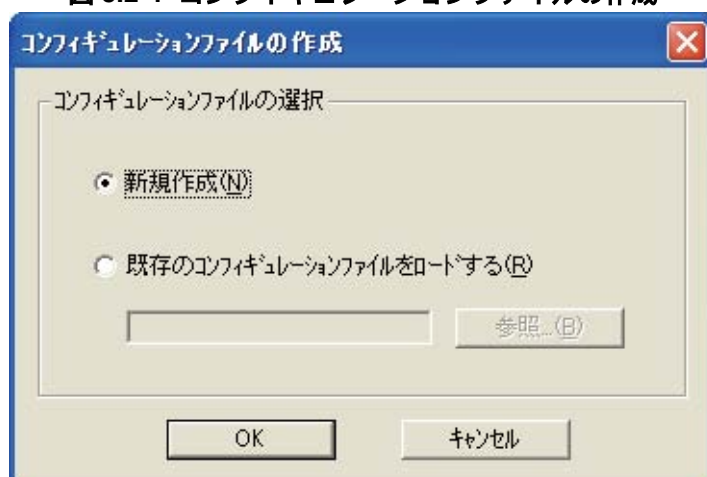
- 「Select REALOS」のダイアログが表示されます。ここで、「SOFTUNE uT-REALOS/FR」を選択して、「OK」ボタンをクリックします。

図 5.2-3 REALOS の種別選択



- コンフィギュレーションファイルの作成ダイアログ上で、以下の事項のいずれかを選択します。
 - 新規作成
 - 既存のコンフィギュレーションファイルをロードする

図 5.2-4 コンフィギュレーションファイルの作成



5.3 コンフィギュレーションの設定

コンフィギュレーションの設定について説明します。

■ コンフィギュレーションの設定

コンフィギュレーションでは、以下を定義します。

- オブジェクト数の最大値の定義

ユーザプログラムで使用しているオブジェクトについて、その数の最大値を設定します。ユーザプログラムでは、この最大値までオブジェクトを作成して、使用できます。このため、最大値はユーザプログラムで使用しているオブジェクト数以上の値で定義します。

たとえば、ユーザプログラムで3個のセマフォを使っている場合、コンフィギュレーション規定マクロの "_KERNEL_MAX_SEM" を "3" で定義します。この場合、"10" を定義しても問題なく動作しますが、カーネル内部で10個のセマフォの管理領域を確保するため、使用されない7個分の管理領域が無駄となります。このため、メモリの使用効率を最適化するには、アプリケーションで使用しているオブジェクト数を最大数で定義してください。

ユーザプログラムで使用しないオブジェクトは、定義する必要はありません。

表 5.3-1 にオブジェクト 1 個あたりのカーネルで消費するメモリのバイト数を示します。

表 5.3-1 オブジェクト管理ブロックの消費メモリ

オブジェクト名	コンフィギュレーション 規定マクロ	管理領域の 大きさ (バイト)
タスク	_KERNEL_MAX_TSK	119
セマフォ	_KERNEL_MAX_SEM	28
イベントフラグ	_KERNEL_MAX_FLG	24
メールボックス	_KERNEL_MAX_MBX	28
ミューテックス	_KERNEL_MAX_MTX	32
メッセージバッファ	_KERNEL_MAX_MBF	52
ランデブポート	_KERNEL_MAX_POR	36
固定長メモリプール	_KERNEL_MAX_MPF	56
可変長メモリプール	_KERNEL_MAX_MPL	56
周期ハンドラ	_KERNEL_MAX_CYC	44
アラームハンドラ	_KERNEL_MAX_ALM	40
デバイス	_KERNEL_MAX_REGDEV	1328

オブジェクト数の最大値の定義方法は、本節の「■ コンフィギュレーションの設定操作」を参照してください。

- 優先度最大値の定義

タスクの優先度とサブシステムの優先度の最大値を定義します。タスクの優先度は、オブジェクト最大値と同様に、定義する値を小さくすれば、カーネルの消費メモリを削減できます。タスク優先度がPの場合の消費メモリは、以下の計算式により、算出できます。

$$\text{消費メモリ (バイト)} = (8 \times P) + 4 \times (P/32)$$

- システムスタックサイズおよび初期タスクのスタックサイズの定義

システムスタックのサイズと初期タスクのスタックのサイズを指定します。これら、スタックサイズの指定方法は、本節の「■ コンフィギュレーションの設定操作」を参照してください。

- 初期ルーチン、エラールーチンおよび省電力ルーチンの登録

初期ルーチン、エラールーチンおよび省電力ルーチンを使用する場合、静的 API により、これらのルーチンを登録します。これらのルーチンの登録方法は、本節の「■ コンフィギュレーションの設定操作」、「4.4 初期ルーチン」、「4.9 エラールーチン」および「4.10 省電力ルーチン」を参照してください。

- 割込みハンドラの登録

割込みハンドラを使用する場合、静的 API により登録します。割込みハンドラは、tk_def_int で、システム起動後に動的に登録もできます。動的に登録する場合、静的 API で登録する必要はありません。また、この場合、コンフィギュレーションマクロの "_KERNEL_USE_TKDEFINT" を "1" で定義してください。

割込みベクタテーブルを ROM 領域に配置して使用する場合、"_KERNEL_USE_TKDEFINT" を "0" に設定することにより、カーネルがベクタテーブルを ROM から RAM にコピーする処理をキャンセルします。そのため、カーネルが使用するメモリを削減できます。

割込みハンドラを静的 API で登録するか、tk_def_int で登録するかは、ユーザプログラムの処理に合わせて自由に選択できます。静的 API で登録した場合、tk_def_int で登録するための処理部分のコードが削減できるメリットがあります。

■ コンフィギュレーションの設定操作

以下の手順でコンフィギュレーションのパラメータを設定します。

- コンフィギュレータのプロジェクトウィンドウ

SOFTUNE Workbench のウィンドウ左下の "CFG" タブをクリックします。これにより、図 5.3-1 を表示します。

図 5.3-1 コンフィギュレータのプロジェクトウィンドウ



- コンフィギュレーション定義

プロジェクトウィンドウで、「コンフィギュレーション定義」をダブルクリックすると図 5.3-2 を表示します。ここで、表 3.13-1 の「優先度定義」、「各オブジェクトの最大数」を設定します。

各コンフィギュレーションウィンドウで入力後、「再計算」ボタンをクリックすると、入力したオブジェクト数に対応した RAM 領域使用量および RAM 領域総使用量を表示します。

図 5.3-2 コンフィギュレーション定義ウィンドウ

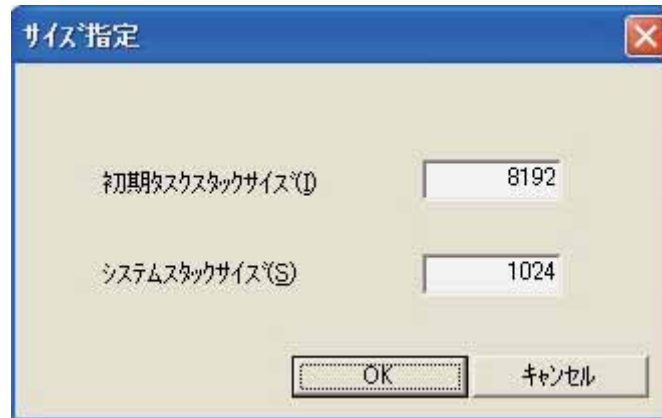
コンフィギュレーション定義		RAM領域使用量		RAM領域使用量	
最大タスク数(T)	32	2432 bytes	最大周期ハンドラ数(D)	8	224 bytes
最大タスク優先度(P)	140		最大アラームハンドラ数(A)	8	192 bytes
最大セマフォ数(S)	16	160 bytes	最大ランタイムポート数(Z)	8	96 bytes
最大イベントフラグ数(E)	16	192 bytes	最大サブシステム数(U)	8	144 bytes
最大メールボックス数(M)	8	128 bytes	最大サブシステム優先度(I)	16	
最大キュータスク数(X)	8	112 bytes	最大デバイス登録数(G)	8	160 bytes
最大メッセージバッファ数(B)	8	288 bytes	最大デバイスオフセット数(N)	16	352 bytes
最大固定長メモリアル数(F)	8	192 bytes	最大デバイスリスト数(Q)	16	128 bytes
最大可変長メモリアル数(V)	8	256 bytes	初期タスク優先度(Y)	138	
RAM領域総使用量		5056 bytes			

OK キャンセル 再計算(R) 一覧表示(L)

- サイズ指定

プロジェクトウィンドウで、「サイズ指定」をダブルクリックすると図 5.3-3 を表示します。ここで、表 3.13-1 の「サイズ指定」の項目を設定します。

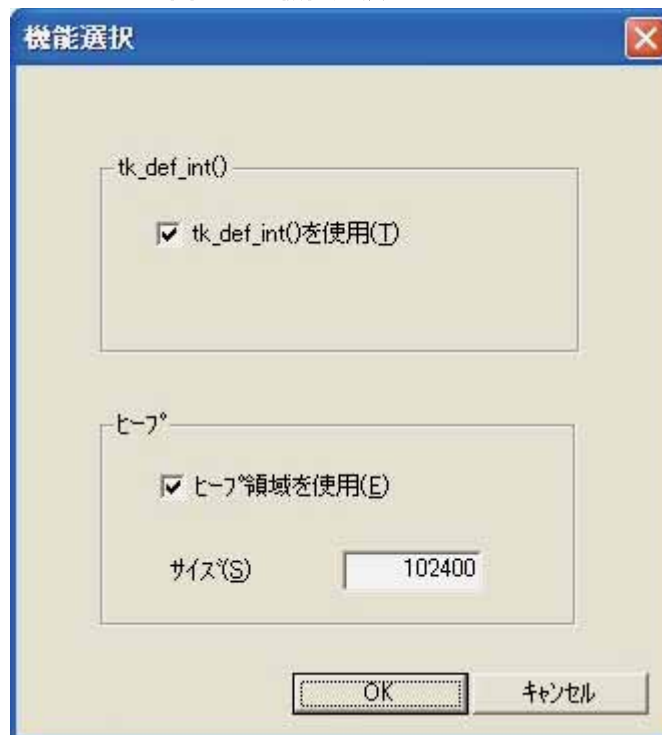
図 5.3-3 サイズ指定ウィンドウ



- 機能選択

プロジェクトウィンドウで、「機能選択」をダブルクリックすると図 5.3-4 を表示します。ここで、表 3.13-1 の「機能選択」の項目を設定します。

図 5.3-4 機能選択ウィンドウ



- エントリ登録

初期ルーチン、エラールーチン、割込みハンドラ、省電力ルーチン（省電力機能）の登録を行います。

プロジェクトウィンドウで、「初期ルーチン」、「エラールーチン」、「省電力機能」をダブルクリックすると、それぞれ図 5.3-5、図 5.3-6 および図 5.3-7 を表示します。ここで、初期ルーチン、エラールーチンおよび省電力ルーチンに使用する関数のエントリ名を設定します。

図 5.3-5 初期ルーチンの編集ウィンドウ



図 5.3-6 エラールーチンの編集ウィンドウ



図 5.3-7 省電力機能の編集ウィンドウ



割込みハンドラの新規登録では、最初に図 5.3-8 を表示します。このウィンドウで、登録するハンドラに対応した割込み番号を選択します。

図 5.3-8 割込み番号の選択ウィンドウ



図 5.3-8 で「OK」ボタンをクリックすると、図 5.3-9 を表示します。ここで、割込みハンドラに使用する関数のエントリ名を設定します。

図 5.3-9 割り込みハンドラの編集ウィンドウ



- デバッグ

μT-REALOS アナライザのログ機能でデバッグモジュールを使用する場合(モジュール使用ログ), そのデバッグモジュールの種類を指定します。コンフィギュレータのプロジェクトウィンドウで, 「type1」, 「type2」, 「type3」および「type5」の名前の部分をダブルクリック, または右クリックして, 「設定」メニューを選択すると選択したデバッグモジュールを設定します。

モジュール使用ログについては, 「アナライザガイド」の「2.4 ログ」および「第 3 章 タスク解析モジュール」を参照してください。

■ コンフィギュレーションの実行

コンフィギュレーションはプロジェクトのビルド時に自動的に実行します。

5.4 リンカオプションの設定

リンカオプションの設定方法について説明します

■ ROM と RAM のメモリマップ設定

ROM/RAM 領域として使用するメモリ領域をユーザシステムに合わせて設定します。サンプルプログラムでは以下のように ROM/RAM 領域に、それぞれ 1MB, 4MB を割り当てています。

表 5.4-1 サンプルプログラムにおける ROM/RAM 領域のメモリマップ

領域属性	領域名	スタートアドレス	エンドアドレス
ROM イメージ	_ROM_	0x00800000	0x008FFFFFFF
RAM イメージ	_RAM_	0x80000000	0x803FFFFFFF

■ カーネル固有のセクション

以下にカーネルで使用しているセクション名について説明します。

表 5.4-2 固有セクション一覧

種別	セクション名	意味
CONST	INTVECT	ベクタテーブル
DATA	SYSINFO	オブジェクト管理テーブル
STACK	_KERNEL_STACK_SC	システムスタック領域
DATA	HEAP	カーネルヒープ領域

INTVECTはROM領域に、SYSINFOと_KERNEL_STACK_SCはRAM領域に配置します。上記表で、「種別」の詳細については、「SOFTUNE リンケージキットマニュアル」(以降、「リンケージキットマニュアル」とよびます)の「5.3 セクションの種類」を参照してください。

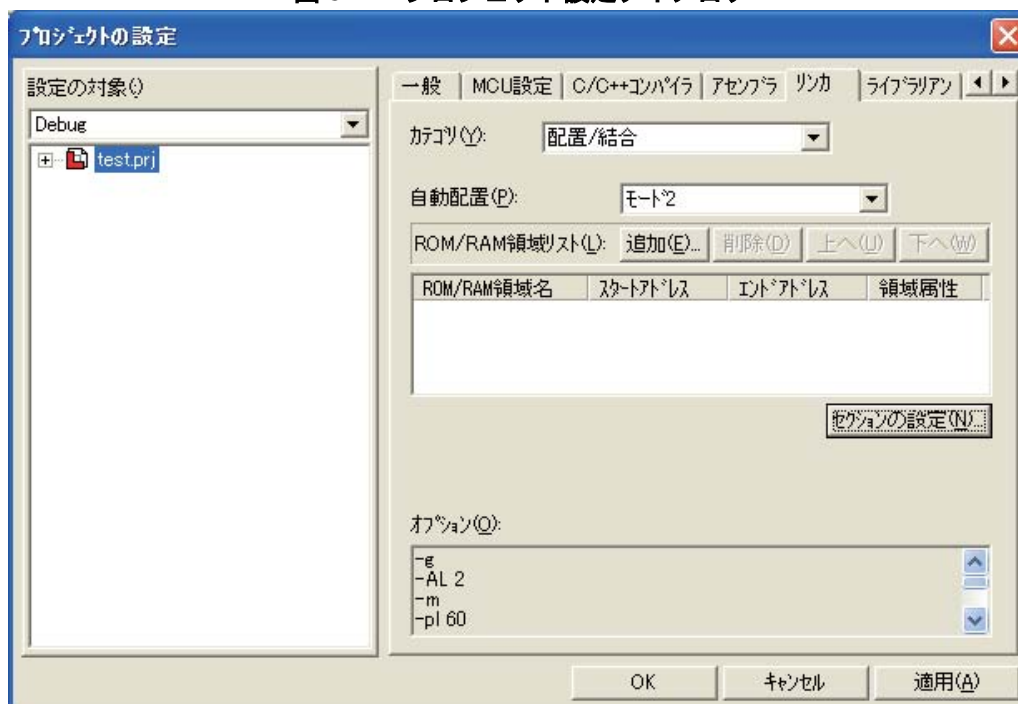
ベクタテーブルの先頭アドレスは、CPU のテーブル・ベース・レジスタ (TBR) に設定します。このため、INTVECT セクションは通常、固定アドレスをリンク時に指定します。

■ メモリマップの指定方法

メモリマップは、ユーザシステムのリンク時に、リンカのオプションとして指定します。以下にその操作方法について説明します。リンカの詳細については、「リンケージキットマニュアル」の「第 2 部 リンカ編」を参照してください。

- SOFTUNE Workbench 上で [プロジェクト]-[プロジェクトの設定] メニューを選択します。「リンカ」のタブをクリックして、プロジェクトの設定ダイアログの「配置 / 結合」のカテゴリで、「ROM/RAM 領域リスト」の「追加」ボタンをクリックします。

図 5.4-1 プロジェクト設定ダイアログ



- RAM 領域, ROM 領域の属性, スタートアドレス, エンドアドレスを入力して, 「OK」ボタンをクリックします。

図 5.4-2 RAM 領域の指定

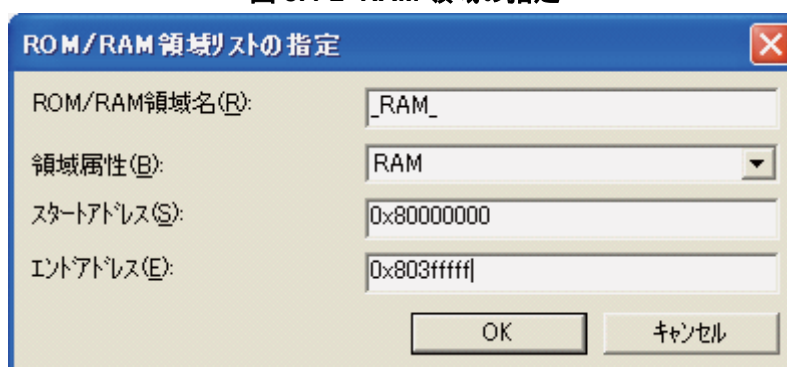
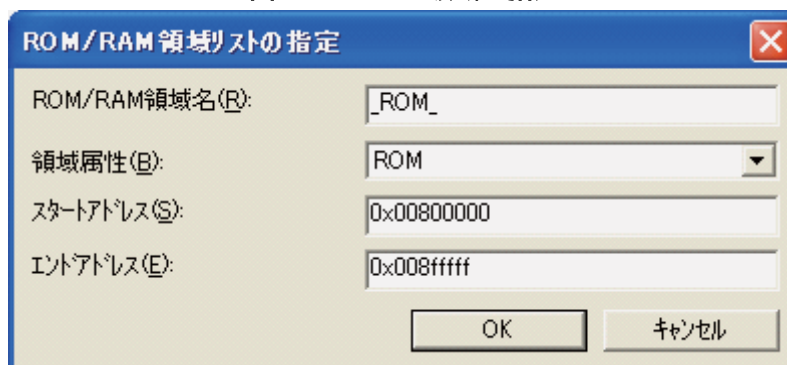
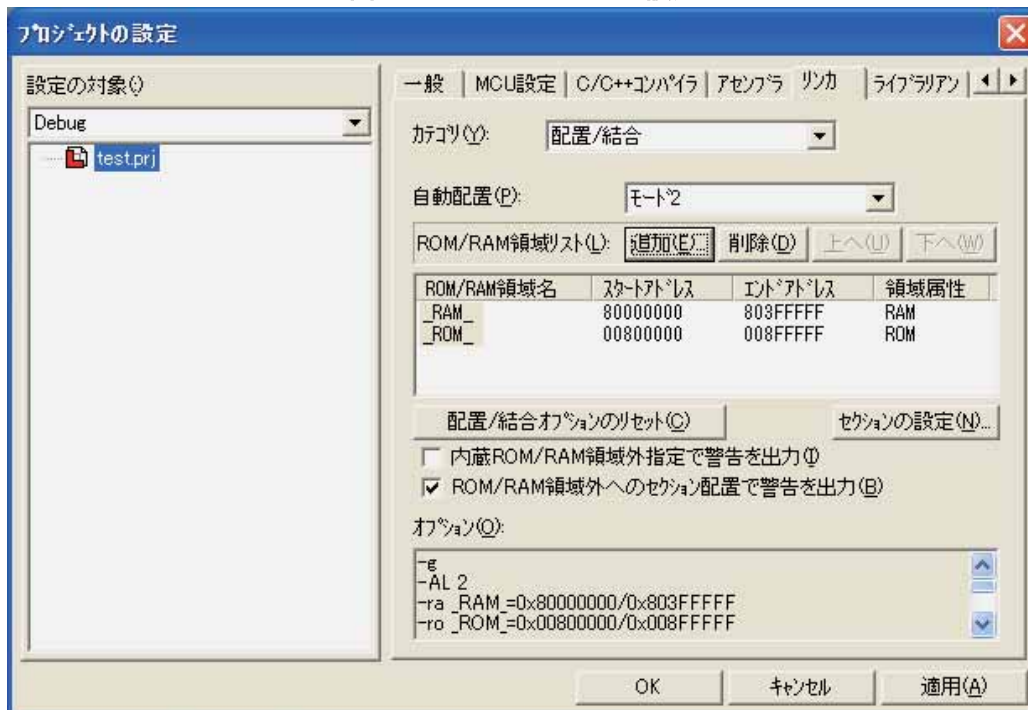


図 5.4-3 ROM 領域の指定



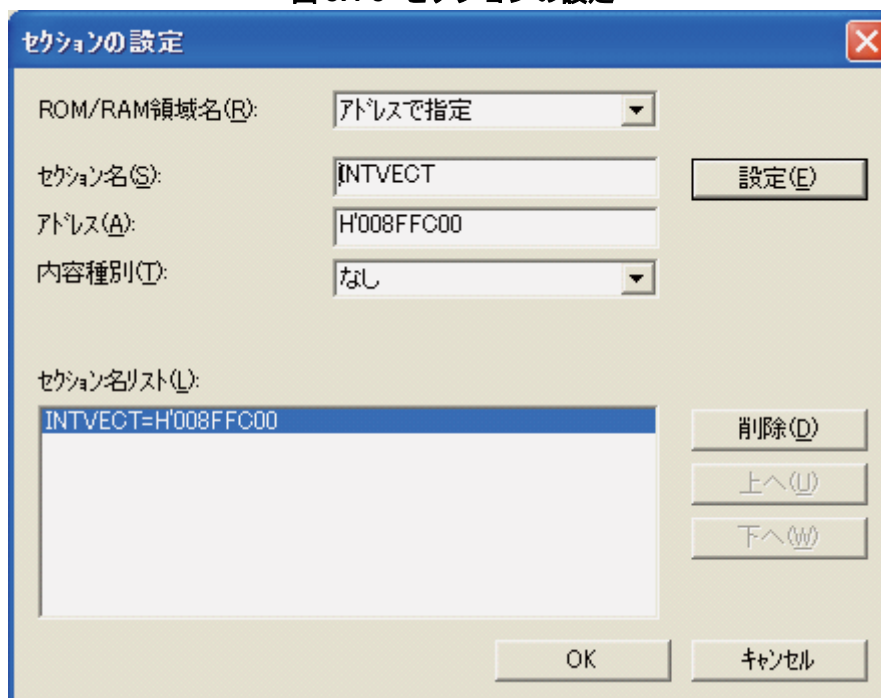
- 次に INTVECT セクションのアドレスを設定します。「プロジェクトの設定」ダイアログで、「セクションの設定」ボタンをクリックします。

図 5.4-4 プロジェクトの設定



- 「セクションの設定」のダイアログで、ROM/RAM 領域名に「アドレスで指定」を選択します。セクション名に「INTVECT」を入力して、INTVECT セクションのスタートアドレスを「アドレス」に入力し、「OK」ボタンをクリックします。

図 5.4-5 セクションの設定



■ リンク対象のオブジェクト

表 5.4-3 にユーザシステムを作成するために必要なオブジェクトファイルについて、サンプルプログラムを例として、一覧で示します。これらのファイルは、REALOS プロジェクト作成時およびユーザプログラム登録時に、自動的にリンクのオプションとして設定されるため、ユーザが設定する必要はありません。

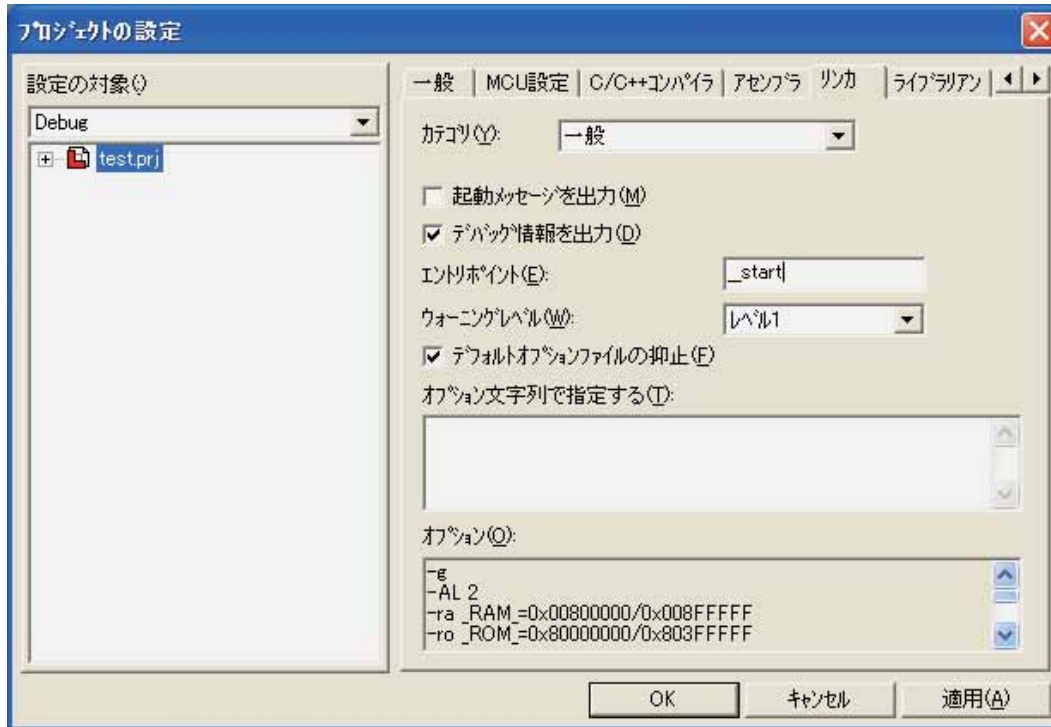
表 5.4-3 リンク対象のオブジェクト一覧

分類	ファイル名	備考
ユーザプログラム (サンプルプログラム)	icrt0.obj	リセットエントリルーチン
	init_task.obj	タスク、タイマ割込みハンドラ
カーネル構成ファイル	config.lib	ファイル名はコンフィギュレーション起動時に指定
カーネルライブラリ	libtm.lib	ファイル名は固定
	libstdlib.lib	
	libstr.lib	
	libtk.lib	
	libtkernel.lib	

■ リセットエントリの設定

プロジェクトの設定ダイアログの「一般」カテゴリで、リセットエントリのアドレスを「エントリポイント」の欄に設定します。以下の例では、"_start" の名前のシンボルのアドレスをリセットエントリに設定しています。

図 5.4-6 リセットエントリの設定



5.5 ユーザシステムのビルド

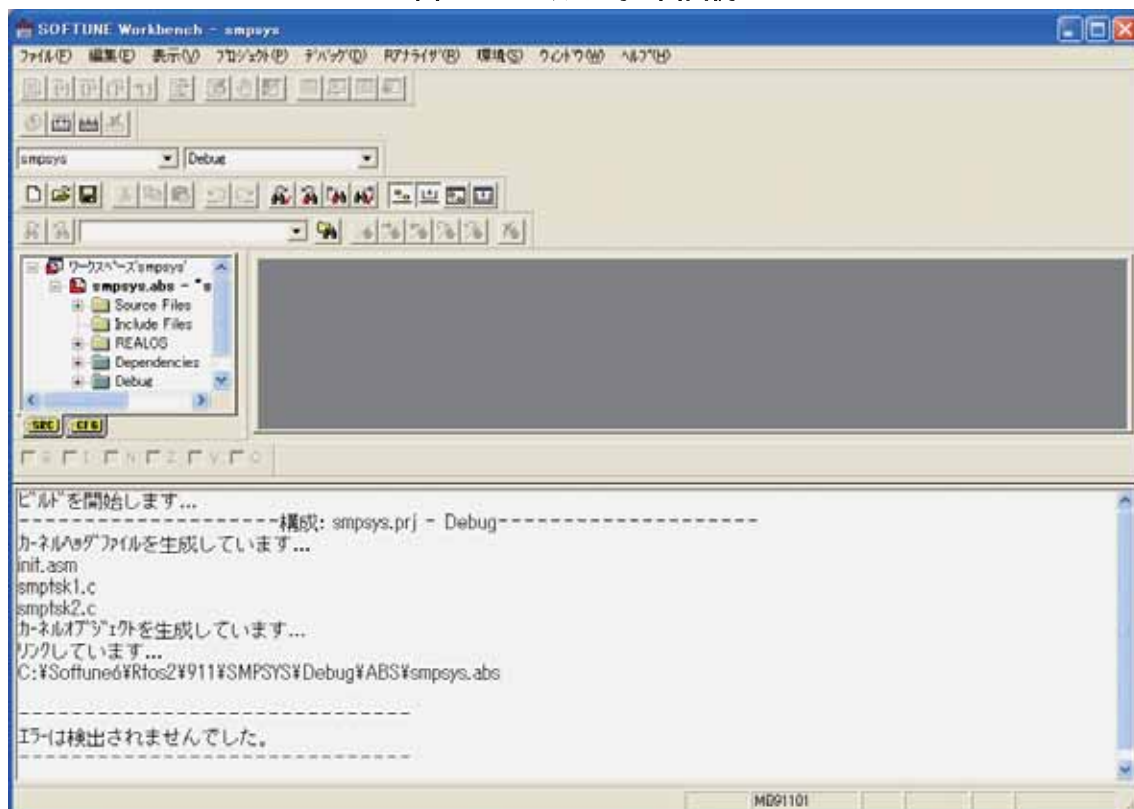
ユーザシステムのビルド方法について説明します。

■ ユーザシステムのビルド

μT-REALOSを含んだ実行形式のオブジェクトを作成するため、ビルドを実行します。ビルドにより、ユーザプログラムのコンパイル、コンフィギュレーション、μT-REALOSとのリンクを自動的に実行します。

- SOFTUNE Workbench 上で [プロジェクト]-[メイク] または [ビルド] を選択します。SOFTUNE Workbench ウィンドウ下部のアウトプットウィンドウにビルドの結果を表示します。

図 5.5-1 ビルド時の画面例



コンフィギュレータのエラーメッセージについて
説明します。

付録 A コンフィギュレータのエラーメッセージ

付録 A コンフィギュレータのエラーメッセージ

コンフィギュレーションでエラーが発生した場合に出力されるメッセージの分類と表示形式およびその意味について説明します。

■ コンフィギュレータのエラーメッセージ分類

コンフィギュレーションを実行したときにコンフィギュレータが出力するエラーメッセージは、重要度に応じて以下の3つのレベルに分類されています。

- 警告メッセージ

下に説明するエラーより軽微であり、出力結果はほとんど問題なく使用できます。

場合により、ユーザの意図と異なる処理が行われる可能性があります。

メッセージ内容を確認したうえで、出力結果が使用可能か否かを判断してください。

- エラーメッセージ

処理は続行しますが、コンフィギュレーションは行いません。エラーとなった原因を取り除いて、再実行してください。

主に、ファイルの読み込み時に発生します。

- 致命的なエラーメッセージ

処理の続行が不可能なエラーです。ユーザの誤指定が原因のほか、実行環境の問題で発生します。

このほかにも、コンフィギュレータが内部で実行するコンパイラ、アセンブラ、リンカが出力するエラーなどのメッセージがあります。コンパイラ、アセンブラ、リンカが出力するメッセージについては、それぞれのマニュアルを参照してください。

■ コンフィギュレータのエラーメッセージの表示形式

エラーメッセージは以下の形式で出力します。

```
*** ファイル名 (行番号)XnnnnT: メッセージ文 (補助メッセージ)
```

各部	説明
ファイル名 (行番号)	エラーの発生したコンフィギュレーションファイル名と、行番号 コンフィギュレーションファイルの読み込みでエラーが発生した場合に出力します。
X	エラーのレベルを以下の 3 つの英字 1 文字であらわします。 W...警告メッセージ E...エラーメッセージ F...致命的なエラーメッセージ
nnnn	エラー番号 エラー番号と、エラーのレベルは以下の対応関係があります。 1000 ~ 1999...W 4000 ~ 4999...E 9000 ~ 9999...F
T	ツール識別を以下の英字 1 文字であらわします。 M...コンフィギュレータ C...コンパイラ A...アセンブラ L...リンカ
メッセージ文	エラーメッセージ本文 (日本語 / 英語の選択が可能)
補助メッセージ	エラーについてのより詳細な情報 エラーの発生原因となったシンボル名などを表示します。 エラーメッセージ本文中に出力されることもあります。

< 注意事項 >

コンフィギュレータが起動するコンパイラ、アセンブラ、リンカでエラーが発生する場合があります (ツール識別にそれぞれ "C", "A", "L" を出力します)。

この場合のエラーの詳細については、それぞれのマニュアルを参照してください。

■ メッセージ表記の説明

以降、コンフィギュレータの警告メッセージ、エラーメッセージ、致命的なエラーメッセージについて、以下の形式で説明します。

エラーコード	日本語メッセージ
	英語メッセージ

メッセージ内で可変となる文字列は下線付きで記載しています。

■ 警告メッセージ

W1130M	重複指定定義されました (定義名)
	Multiple definition (定義名)

定義名で表示された定義が重複定義されました。

この定義は、後に指定した定義により上書きされました。

定義名に、数値の表示がある場合、その数値で示される ID において後に指定した定義により上書きされました。

W1401M	最大領域定義が見つかりませんでした (定義名)
	Not found maximum area definition (定義名)

定義名で表示された最大領域定義名がありません。

このエラーが出力された場合、自動的に最大サイズが割り当てられます。

W1405M	EIT ベクタ番号 番号はシステムでリザーブされています
	EIT vector No. 番号 is system reserve

番号 で指定した割込み番号はシステムでリザーブされているため、使えません。

"ATT_INI" で定義した割込みハンドラの割込み番号がシステムでリザーブされている場合、このエラーメッセージが出力します。

■ エラーメッセージ

E4024M	パラメータの指定に不正な文字があります (<u>パラメータ</u>)
	Illegal character (<u>パラメータ</u>)

パラメータで表示されたパラメータに使用できない文字が含まれています。
このエラーは数値を指定するパラメータに文字が指定された場合、ラベルを指定しなければならないパラメータに数値が指定された場合などに発生します。

E4026M	値が範囲外です (<u>パラメータ</u>)
	Specified value is out of range (<u>パラメータ</u>)

パラメータで表示された値が指定できる範囲外です。
このエラーはオブジェクトIDに32767を超える数値を指定した場合などに発生します。

E4110M	API 名の指定に誤りがあります (<u>文字列</u>)
	Unknown API name (<u>文字列</u>)

文字列で表示された定義は使用できません。
このエラーはサポートされていない API 名を記述した場合などに発生します。

E4111M	一行の長さが長過ぎます (MAX <u>値</u>)
	Too long line (MAX <u>値</u>)

一行の長さは MAX 値で表示された長さを超えて記述できません。
MAX 値で表示された長さに収まるように記述してください。

E4112M	パラメータの記述が不正です
	Illegal parameter expression

行番号で示された場所の表記が不正です。
API の記述構文や定義方法が不正な場合に出力します。

E4120M	<u>パラメータ</u> が長過ぎます
	<u>パラメータ</u> is too long

パラメータで表示したパラメータの長さが長過ぎます。
シンボルが規定以上の長さで記述されている場合などに出力されます。

E4121M	定義名は値個より多く定義できません
	Too many 定義名 (MAX 値)

定義名で表示された定義が、値で表示された数以上定義されています。
このエラーは API を規定以上定義しようとした場合などに出力します。

E4123M	パラメータ数が多過ぎます (<u>パラメータ</u>)
	Too many parameters (<u>パラメータ</u>)

パラメータで表示された以降で記述されたパラメータは不必要です。

E4125M	パラメータが足りません
	Short of parameter

定義名のパラメータが不足しています。

E4130M	重複して定義されました (<u>定義名</u>)
	Multiple definition (<u>定義名</u>)

二重定義ができない定義が重複して定義されました。
このエラーは、API の ID が重複している場合などに出力します。

E4131M	パラメータが指定されていません (<u>パラメータ名</u>)
	Parameter not defined (<u>パラメータ名</u>)

省略することができないパラメータが省略されました。
パラメータ名で表示されたパラメータは省略することができません。

E4132M	不正なパラメータが指定されました (<u>パラメータ</u>)
	Illegal parameter (<u>パラメータ</u>)

指定できないパラメータが指定されました。

このエラーは、主に選択項目に選択できない文字列が指定された場合に表示します。

E4133M	既に定義されているシンボルが指定されました (<u>シンボル名</u>)
	Symbol is already defined (<u>シンボル名</u>)

既に定義されているシンボルが再定義されました。

このエラーは、主に API で指定したタスク、イベントフラグ名などが重複している場合に発生します。

E4136M	サイズまたはアドレスが不正です (<u>値</u>)
	Illegal size or address (<u>値</u>)

指定されたサイズまたはアドレスが正しくありません。

E4142M	デバイスオープン数がセマフォ数より大きいです (MAX <u>値</u>)
	Device open count is bigger than semaphore count (MAX <u>値</u>)

デバイスオープン数にセマフォ数より大きな値が指定されました。

デバイスオープン数を小さくするか、セマフォ数を大きくしてください。

E4402M	API ID が最大領域定義数を超過しました (<u>パラメータ</u>)
	API ID exceed maximum area definition (<u>パラメータ</u>)

パラメータで表示される API が、最大領域定義された値より多く定義されました。

最大領域定義を行わずに、API 定義を行った場合にもこのエラーを出力します。

■ 致命的なエラーメッセージ

F9000M	環境変数が定義されていません (環境変数名)
	Environment variable not found (環境変数名)

環境変数名で表示された環境変数が定義されていません。

F9001M	メモリが足りません
	Insufficient memory

プログラム実行のためのメモリが不足しました。

F9002M	コンフィギュレーションの実行が行われませんでした
	Not configured

コンフィギュレーションの実行が行われませんでした。このエラーは、コンフィギュレーション中にエラーが発生したために、実行が中断された場合に出力します。

F9011M	ファイルが見つかりません (ファイル名)
	Input file is not found (ファイル名)

指定された入力ファイルが見つかりません。

F9016M	ファイルからの読み込みができません (ファイル名)
	File read error (ファイル名)

ファイルが読み出し許可のないファイルであるか、ハードウェアの問題が考えられます。

F9017M	ファイルへの書き込みができません (ファイル名)
	File write error (ファイル名)

ファイルが書き込み許可のないファイルである、同名のディレクトリが存在するまたはディスクに空き容量がないことなどが考えられます。

F9022M	オプション名の指定に誤りがあります (<u>オプション</u>)
	Unknown option name (<u>オプション</u>)

指定できないオプションが指定されました。

F9023M	パラメータの指定に誤りがあります (<u>パラメータ</u>)
	Illegal option parameter (<u>パラメータ</u>)

パラメータで表示されたパラメータの指定方法に誤りがあります。

F9024M	パラメータの指定がありません (<u>オプション</u>)
	Option parameter not specified (<u>オプション</u>)

オプションで表示されたオプションにパラメータの指定がありません。

F9030M	入力ファイル名の指定がありません
	Missing input file name

コンフィギュレーションファイルが指定されませんでした。

F9033M	ファイルの形式が正しくありません (<u>パラメータ</u>)
	Illegal file format (<u>パラメータ</u>)

CPU 情報ファイルなどのファイルの形式が正しくない場合に発生します。

F9405M	初期タスク優先度が最大タスク優先度より高いです (MAX <u>値</u>)
	Initial task priority is higher than maximum task priority (MAX <u>値</u>)

初期タスクの優先度がタスクの最大優先度より高いです。

初期タスク優先度を小さくするか、最大タスク優先度を大きくしてください。

F9501M	CPU 情報ファイルが見つかりません
	Not found CPU information file

指定場所に CPU 情報ファイルがありません。

F9502M	CPU 情報が見つかりません。(MB 番号がありません)
	Not found CPU information

-cpu オプションで指定された MB 番号が、CPU 情報ファイルに登録されていない場合に発生します。

-cpu オプションにて指定された MB 番号をご確認ください。

F9801M	定義名が定義されていません
	定義名 is not defined

定義名で表示された定義内容が定義されていません。

このエラーは、省略できない定義またはオプションが指定されなかった場合に出力します。

F9805M	EIT ベクタ番号 番号 はシステムでリザーブされています
	EIT vector No. 番号 is system reserve

番号で表示されたベクタ番号はシステムでリザーブされているため、使用できません。

F9895M	コンパイラでエラーが発生しました (ファイル名)
	Error in Compiler (ファイル名)

表示されたファイルをコンパイル中にエラーが発生しました。

F9897M	アセンブラでエラーが発生しました (ファイル名)
	Error in Assembler (ファイル名)

表示されたファイルをアセンブル中にエラーが発生しました。

F9898M	リンカでエラーが発生しました
	Error in Linker

リンカでエラーが発生しました。

F9899M	<u>ツール名</u> が見つかりません
	<u>ツール名</u> is not found

コンパイラ、アセンブラまたはリンカが、環境変数 "PATH" からみつけることができませんでした。

環境変数 "PATH" にツールが格納されているパスを定義してください。

F9990M	ファイルの入出力でエラーが発生しました (<u>ファイル名</u> , 情報)
	File I/O error (<u>ファイル名</u> , 情報)

ファイルの入出力で何らかのエラーが発生しました。

F9993M	テンポラリディレクトリを作成できません (<u>ディレクトリ名</u>)
	Can not create directory (<u>ディレクトリ名</u>)

ディレクトリ名で表示されたディレクトリの作成に失敗しました。

ディレクトリに書き込み許可がない、同名のファイル名が存在する、ディスクの空き容量がないなどが考えられます。

F9994M	テンポラリファイルを作成できません (<u>ファイル名</u>)
	Can not create file (<u>ファイル名</u>)

ファイル名で表示されたファイルの作成に失敗しました。

ファイルに書き込み許可がない、同名のディレクトリ名が存在する、ディスクの空き容量がないなどが考えられます。

F9995M	ファイルをクローズできません (<u>ファイル名</u>)
	Can not close file (<u>ファイル名</u>)

ファイル名で表示されたファイルのクローズに失敗しました。

ファイルに書き込み許可がない、同名のディレクトリ名が存在する、ディスクの空き容量がないなどが考えられます。

F9996M	ファイルをオープンできません (ファイル名)
	Can not open file (ファイル名)

ファイル名で表示されたファイルのオープンに失敗しました。
 ファイルに書き込み許可がない、同名のディレクトリ名が存在する、ディスクの空き容量がないなどが考えられます。

F9999M	プログラム内部エラーが発生しました (識別情報)
	Internal error (識別情報)

このエラーが出力された場合、直ちに営業部門までご連絡願います。

索引

A

abortfn	
処理中止関数 (abortfn) の作成	94
API	
静的 API	62

C

closefn	
クローズ関数 (closefn) の作成	93

E

eventfn	
イベント関数 (eventfn)	94
execfn	
処理開始関数 (execfn) の作成	93

M

μT-REALOS	
μT-REALOS の機能概要	28
μT-REALOS プロジェクトの作成	99

O

openfn	
オープン関数 (openfn) の作成	92
OS ブレーク	
OS ブレーク	65

R

RAM	
ROM と RAM のメモリマップ設定	109
REALOS	
μT-REALOS の機能概要	28
μT-REALOS プロジェクトの作成	99
ROM	
ROM と RAM のメモリマップ設定	109

S

SVC	
拡張 SVC ハンドラ	19
拡張 SVC ハンドラの記述形式	91
拡張 SVC ハンドラの呼出し形式	91

W

waitfn	
完了待ち関数 (waitfn) の作成	93

あ

アラームハンドラ	
アラームハンドラ	17
アラームハンドラの記述形式	86
アラームハンドラの起動	86
アラームハンドラの生成	86
アラームハンドラ機能	
アラームハンドラ機能	52

い

イベント関数	
イベント関数 (eventfn)	94
イベントフラグ	
イベントフラグ機能	34

え

エラールーチン	
エラールーチン	18
エラールーチンの記述形式	89
エラールーチンの登録	89

お

オープン関数	
オープン関数 (openfn) の作成	92
オブジェクト	
オブジェクト	21
リンク対象のオブジェクト	112
オブジェクト一覧表示	
オブジェクト一覧表示	64

か

カーネル	
カーネル固有のセクション	109
拡張 SVC ハンドラ	
拡張 SVC ハンドラ	19
拡張 SVC ハンドラの記述形式	91
拡張 SVC ハンドラの呼出し形式	91
拡張同期・通信機能	
拡張同期・通信機能	37
可変長メモリプール機能	
可変長メモリプール機能	47
関数	
イベント関数 (eventfn)	94
完了待ち関数 (waitfn) の作成	93
クローズ関数 (closefn) の作成	93
処理開始関数 (execfn) の作成	93
処理中止関数 (abortfn) の作成	94
デバイス処理関数	20
完了待ち関数	
完了待ち関数 (waitfn) の作成	93

く

クローズ関数	
クローズ関数 (closefn) の作成	93

こ

固定長メモリプール機能	
固定長メモリプール機能	46
コンフィギュレーション	
コンフィギュレーションの実行	108
コンフィギュレーションの設定	102
コンフィギュレーションの設定操作	103
コンフィギュレーション規定マクロ	
コンフィギュレーション規定マクロ	60
コンフィギュレーション機能	
コンフィギュレーション機能	59
コンフィギュレータ	
コンフィギュレータの起動	63

さ

サブシステム管理機能	
サブシステム管理機能	55
サポート機能	
サポート機能	2

し

時間管理機能	
時間管理機能	48
システム稼動時間	
システム稼動時間の参照	49
システム構築手順	
システム構築手順	98
システムコール	
システムコール	8
システムコール発行	69
呼び出すことのできるシステムコール	23
システムコール全般	
システムコール全般に関する注意事項	95
システム時刻	
システム時刻	49
システム時刻の更新	49
システム時刻の設定 / 参照	49
システム状態	
システム状態	22
ユーザプログラムとシステム状態	23
システム状態管理機能	
システム状態管理機能	54
自タスク	
自タスクと他タスク	10
実行優先順位	
実行優先順位 (タスク対タスク)	25
実行優先順位 (タスク対割込みハンドラ, タイムイベントハンドラ)	25
実行優先順位 (ハンドラ対ハンドラ)	26
周期ハンドラ	
周期ハンドラ	16
周期ハンドラの記述形式	85
周期ハンドラの起動	85

周期ハンドラの生成	85
周期ハンドラ機能	
周期ハンドラ機能	50
状態遷移	
タスクの状態遷移	12
省電力機能	
省電力機能	58
省電力ルーチン	
省電力ルーチンの記述形式	90
省電力ルーチンの登録	90
初期ルーチン	
初期ルーチン	14
初期ルーチンの記述形式	79
初期ルーチンの記述例	80
初期ルーチンの処理	79
処理開始関数	
処理開始関数 (execfn) の作成	93
処理中止関数	
処理中止関数 (abortfn) の作成	94

す

スタック	
スタック情報	70

せ

静的 API	
静的 API	62
製品構成	
製品構成	5
セクション	
カーネル固有のセクション	109
セマフォ	
セマフォ機能	32

た

タイマ割込みハンドラ	
タイマ割込みハンドラ	88
タイムイベントハンドラ	
実行優先順位 (タスク対割込みハンドラ, タイムイベントハンドラ)	25
タイムイベントハンドラ	16
タスク	
自タスクと他タスク	10
実行優先順位 (タスク対タスク)	25
実行優先順位 (タスク対割込みハンドラ, タイムイベントハンドラ)	25
タスク	10
タスク管理機能	29
タスクに関する注意事項	95
タスクの記述形式	81
タスクの起動	82
タスクの具体例	82
タスクの状態	11
タスクの状態遷移	12
タスクの生成	81
タスク部実行中	22
タスク付属同期機能	30
非タスク部実行中	22

優先順位とタスク優先度	10
タスク管理機能	
タスク管理機能	29
タスクコンテキスト表示	
タスクコンテキスト表示	71
タスク付属同期機能	
タスク付属同期機能	30
タスク優先度	
優先順位とタスク優先度	10
他タスク	
自タスクと他タスク	10

ち

注意事項	
システムコール全般に関する注意事項	95
タスクに関する注意事項	95
プログラム全般に関する注意事項	95
割込みに関する注意事項	96

つ

通信機能	
拡張同期・通信機能	37
同期・通信機能	31
ツール	
開発に必要なツール	4

て

提供ファイル	
提供ファイル構成	3
ディスパッチ	
ディスパッチ禁止 / 許可状態	24
ディスパッチとプリエンプト	10
デバイス管理機能	
デバイス管理機能	56
デバイス処理関数	
デバイス処理関数	20
デバイスドライバ	
デバイスドライバインタフェース	92
デバイス名	
デバイス名の決定	92
デバッグ支援機能	
デバッグ支援機能の概要	64

と

同期・通信機能	
同期・通信機能	31

は

ハンドラ	
アラームハンドラ	17
拡張 SVC ハンドラ	19
実行優先順位 (ハンドラ対ハンドラ)	26
周期ハンドラ	16
タイマ割込みハンドラ	88
タイムイベントハンドラ	16
割込みハンドラ	15

ひ

非タスク	
非タスク部実行中	22

ふ

付属同期	
タスク付属同期機能	30
プリエンプト	
ディスパッチとプリエンプト	10
ブレーク	
OS ブレーク	65
プログラム全般	
プログラム全般に関する注意事項	95
プロジェクト	
μT-REALOS プロジェクトの作成	99

ほ

補足事項	
補足事項	36, 39, 41, 43

ま

マクロ	
コンフィギュレーション規定マクロ	60

み

ミューテックス	
ミューテックス機能	38

め

メールボックス	
メールボックス機能	35
メッセージバッファ	
メッセージバッファ機能	40
メモリプール	
可変長メモリプール機能	47
固定長メモリプール機能	46
メモリプール管理機能	45
メモリプール管理機能	
メモリプール管理機能	45
メモリマップ	
ROM と RAM のメモリマップ設定	109
メモリマップの指定方法	109

ゆ

ユーザシステム	
ユーザシステムのビルド	114
ユーザプログラム	
ユーザプログラムとシステム状態	23
ユーザプログラムの起動	75
ユーザプログラムの構成	74
ユーザプログラムの実行単位	9
優先順位	
実行優先順位 (タスク対タスク)	25

索引

実行優先順位 (タスク対割込みハンドラ, タイム イベントハンドラ)	25
実行優先順位 (ハンドラ対ハンドラ)	26
優先順位とタスク優先度	10
優先度	
優先順位とタスク優先度	10

ら

ランデブポート機能	
ランデブポート機能	42

り

リセットエントリ	
リセットエントリの設定	113
リセットエントリルーチン	
リセットエントリルーチン	76
リセットエントリルーチンの具体例	77
リセットエントリルーチンの処理	76
リンク対象	
リンク対象のオブジェクト	112

ろ

ログ	
ログ	65

わ

割込み

実行優先順位 (タスク対割込みハンドラ, タイム イベントハンドラ)	25
タイマ割込みハンドラ	88
割込み管理機能	53
割込み禁止 / 許可状態	24
割込みに関する注意事項	96
割込みハンドラ	15

割込み管理機能

割込み管理機能	53
---------------	----

割込みハンドラ

実行優先順位 (タスク対割込みハンドラ, タイム イベントハンドラ)	25
割込みハンドラ	15
割込みハンドラの記述形式	87
割込みハンドラの登録	87

CM81-00322-1

富士通マイクロエレクトロニクス・CONTROLLER MANUAL

FR ファミリ

μT-Kernel 仕様準拠

SOFTUNE™ μT-REALOS/FR

ユーザーズガイド

2008 年 6 月 初版発行

発行

富士通マイクロエレクトロニクス株式会社

編集

マーケティング統括部 ビジネス推進部
