

Cortex-M 対応 MULCOS-MK

(MULti Core Operating System – Multilayer Kernel)

μ ITRON4.0 仕様版共通部制限ユーザーズガイド



第1版 2025年1月

改訂履歴

版数	発行日	改訂内容
第 1 版	2025 年 1 月	Cortex-M 対応 MULCOS-MK μ ITRON4.0 仕様版 共通部制限ユーザーズガイドの初版を発行

はじめに

MULCOS-MK (MULTi Core Operating System - Multilayer Kernel) とは、セキュリティレベルの異なる階層化したリアルタイム・オペレーティング・システム (Real Time Operating System、以降 RTOS と省略) の実行環境を管理する多層化カーネル (以降では、本製品を単に本カーネルと呼ぶ) です。

μ ITRON4.0 仕様版の RTOS (以降では、単に本 RTOS と呼ぶ) は、最も多く使われている仕様の 1 つで、当時の社団法人トロン協会が策定し公開した「 μ ITRON4.0 仕様」(現在は、トロンフォーラムが公開) に準拠しています。

本書は、Cortex-M 対応 MULCOS-MK の μ ITRON4.0 仕様版共通部制限ユーザーズガイドで、非セキュアのみを対象とする開発向けに用意したドキュメントです。その他、各メーカー別のデバイス依存部ユーザーズガイドが用意されています。

(μ ITRON とは Micro Industrial TRON、TRON とは The Realtime Operating system Nucleus の略称)

参考資料

- ・社団法人トロン協会発行「 μ ITRON4.0 仕様 Ver.4.03.00」
- ・Arm Limited 社発行「Arm[®]v8-M Architecture Reference Manual」

目次

改訂履歴	0
はじめに	1
目次	2
第 1 章 MULCOS-MK とは	7
1-1. Cortex-M 対応 MULCOS-MK とは	7
1-2. Cortex-M 対応 MULCOS-MK の μ ITRON4.0 仕様版とは	8
第 2 章 用語の定義	12
2-1. カーネル用語の定義	12
2-1-1. セキュア層と非セキュア層	12
2-1-2. タスク	12
2-1-3. コンテキスト	12
2-1-4. 移行タスク	12
2-1-5. ディスパッチとプリエンプション	13
2-1-6. システム時刻とタイムチェック	13
2-2. RTOS 用語の定義	14
2-2-1. タスクコンテキストと非タスクコンテキスト	14
2-2-2. 特権モードと非特権モード	14
2-2-3. オブジェクトと ID 番号	14
2-2-4. タスク優先度と優先順位	14
2-2-5. 相対時間と時間監視	15
2-2-6. 待ち行列	15
2-2-7. レディキュー	15
2-2-8. ディスパッチ保留状態	15
2-2-9. キューイング	16
2-2-10. エラーコード	16
第 3 章 RTOS の基本機能	17
3-1. タスクの状態とスケジューリング	17
3-1-1. タスクの状態	17
3-1-2. スケジューリング規則	19
3-2. セマフォ同期・通信機能	21
3-2-1. セマフォを排他制御に用いる場合	21
3-2-2. セマフォを資源の通知に用いる場合	23
3-3. イベントフラグ同期・通信機能	25
3-3-1. イベントフラグをクリアせず読み出す場合	27
3-3-2. イベントフラグを読み出し後にクリアする場合	29
3-4. データキュー同期・通信機能	31

3-4-1. データキューへ送信する場合	31
3-4-2. データキューから受信する場合	33
3-4-3. データキューへ強制送信する場合	34
3-4-4. データキューの同期メッセージ機能の場合	35
3-5. メールボックス同期・通信機能	36
3-6. ミューテックス拡張同期・通信機能	39
3-7. メッセージバッファ拡張同期・通信機能	42
3-7-1. メッセージバッファへ送信する場合	43
3-7-2. メッセージバッファから受信する場合	45
3-7-3. メッセージバッファの同期通信機能の場合	47
3-8. 固定長メモリプール機能	48
3-9. 可変長メモリプール機能	50
3-10. 周期ハンドラ機能	53
3-10-1. 起動位相を保存する場合	53
3-10-2. 起動位相を保存しない場合	53
3-10-3. 起動位相とタイムチックの関係	54
第4章 カーネルと RTOS の特徴と注意事項	55
4-1. 時間の管理方法	55
4-1-1. システム時刻の更新	55
4-1-2. タイムイベントハンドラの実行	55
4-2. 割込み禁止／許可と割込みマスク優先度の変更	55
4-3. タスクのスタック領域	56
4-4. アイドル関数	56
第5章 RTOS のコンフィグレーションと起動	57
5-1. コンフィグレーションのデータ作成	57
5-2. RTOS が必要とするメモリ領域	59
5-3. 初期化から起動までの手順	60
5-3-1. RTOS の初期化	61
5-3-2. RTOS の起動	61
第6章 各種管理機能とシステムコール	62
6-1. タスク管理機能	62
6-1-1. タスクの生成	64
6-1-2. タスクの削除	65
6-1-3. タスクの起動	66
6-1-4. タスク起動要求のキャンセル	66
6-1-5. タスクの起動（起動コード指定）	67
6-1-6. 自タスクの終了	67

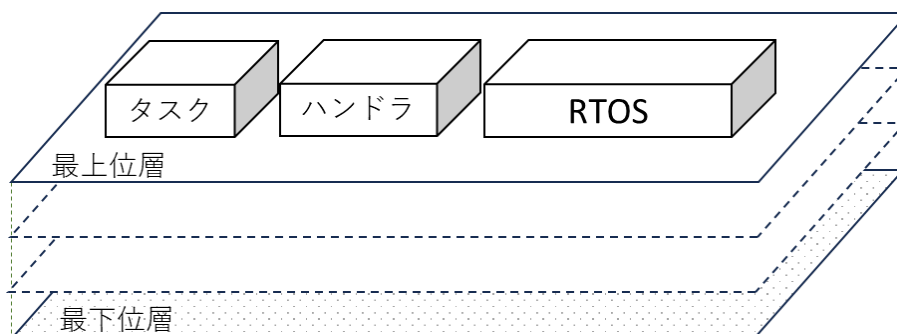
6-1-7. 自タスクの終了と削除	68
6-1-8. タスクの強制終了	68
6-1-9. タスク優先度の変更	69
6-1-10. タスク優先度の参照	69
6-1-11. タスクの状態参照	70
6-1-12. タスクの状態参照（簡易版）	71
6-1-13. セキュアタスクの非セキュアスタック定義	72
6-2. タスク付属同期機能	73
6-2-1. 起床待ち	75
6-2-2. タスクの起床	75
6-2-3. タスク起床要求のキャンセル	76
6-2-4. 待ち状態の強制解除	76
6-2-5. 強制待ち状態への移行	77
6-2-6. 強制待ち状態からの再開	77
6-2-7. 強制待ち状態からの強制再開	78
6-2-8. 自タスクの遅延	78
6-3. セマフォ同期・通信機能	79
6-3-1. セマフォの生成	81
6-3-2. セマフォの削除	82
6-3-3. セマフォ資源の返却	82
6-3-4. セマフォ資源の獲得	83
6-3-5. セマフォの状態参照	84
6-4. イベントフラグ同期・通信機能	85
6-4-1. イベントフラグの生成	87
6-4-2. イベントフラグの削除	88
6-4-3. イベントフラグのセット	88
6-4-4. イベントフラグのクリア	89
6-4-5. イベントフラグ待ち	90
6-4-6. イベントフラグの状態参照	91
6-5. データキュー同期・通信機能	93
6-5-1. データキューの生成	95
6-5-2. データキューの削除	96
6-5-3. データキューへの送信	96
6-5-4. データキューへの強制送信	97
6-5-5. データキューからの受信	98
6-5-6. データキューの状態参照	99

6-6. メールボックス同期・通信機能	100
6-6-1. メールボックスの生成	102
6-6-2. メールボックスの削除	103
6-6-3. メールボックスへの送信	104
6-6-4. メールボックスからの受信	105
6-6-5. メールボックスの状態参照	106
6-7. ミューテックス拡張同期・通信機能	107
6-7-1. ミューテックスの生成	108
6-7-2. ミューテックスの削除	109
6-7-3. ミューテックスのロック	109
6-7-4. ミューテックスのロック解除	110
6-7-5. ミューテックスの状態参照	111
6-8. メッセージバッファ拡張同期・通信機能	112
6-8-1. メッセージバッファの生成	114
6-8-2. メッセージバッファの削除	115
6-8-3. メッセージバッファへの送信	116
6-8-4. メッセージバッファからの受信	117
6-8-5. メッセージバッファの状態参照	118
6-9. 固定長メモリプール管理機能	119
6-9-1. 固定長メモリプールの生成	121
6-9-2. 固定長メモリプールの削除	122
6-9-3. 固定長メモリブロックの獲得	123
6-9-4. 固定長メモリブロックの返却	124
6-9-5. 固定長メモリプールの状態参照	125
6-10. 可変長メモリプール管理機能	126
6-10-1. 可変長メモリプールの生成	128
6-10-2. 可変長メモリプールの削除	129
6-10-3. 可変長メモリブロックの獲得	130
6-10-4. 可変長メモリブロックの返却	131
6-10-5. 可変長メモリプールの状態参照	132
6-11. システム時刻管理	133
6-11-1. システム時刻の設定	134
6-11-2. システム時刻の参照	134
6-12. 周期ハンドラ機能	135
6-12-1. 周期ハンドラの生成	137
6-12-2. 周期ハンドラの削除	138

6-12-3. 周期ハンドラの動作開始	138
6-12-4. 周期ハンドラの動作停止	138
6-12-5. 周期ハンドラの状態参照	139
6-13. システム状態管理機能	140
6-13-1. タスクの優先順位の回転	141
6-13-2. 実行状態のタスク ID の参照	141
6-13-3. CPU ロック状態への移行	142
6-13-4. CPU ロック状態の解除	142
6-13-5. ディスパッチの禁止	142
6-13-6. ディスパッチの許可	143
6-13-7. コンテキストの参照	143
6-13-8. CPU ロック状態の参照	143
6-13-9. ディスパッチ禁止状態の参照	143
6-13-10. ディスパッチ保留状態の参照	144
6-13-11. システムの状態参照	144
6-14. 割込み管理機能	145
6-14-1. 割込みハンドラの定義	147
6-14-2. 割込みサービスルーチンの生成	148
6-14-3. 割込みサービスルーチンの削除	149
6-14-4. 割込みサービスルーチンの状態参照	150
6-14-5. 割込みの禁止	150
6-14-6. 割込みの許可	151
6-14-7. 割込みマスクの変更	151
6-14-8. 割込みマスクの参照	151
6-15. システム構成管理機能	152
6-15-1. コンフィグレーション情報の参照	152
6-15-2. バージョン情報の参照	153
第 7 章 付録	154
7-1. データの型	154
7-2. パケット形式	156
7-3. 定数とマクロ	164
7-4. 構成定数とマクロ	166
7-5. エラーコード	167

第1章 MULCOS-MK とは

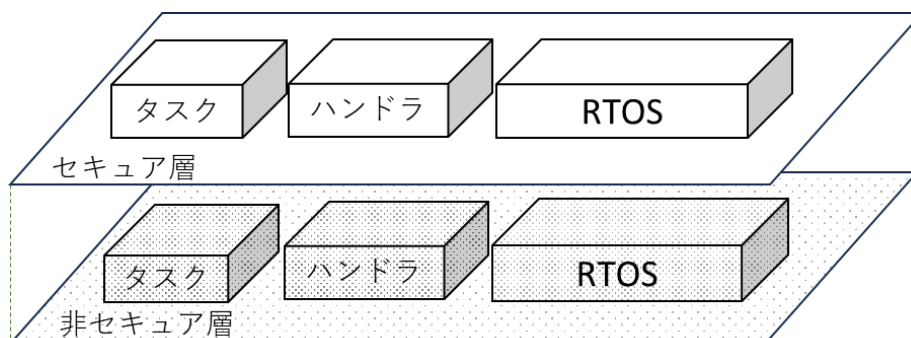
MULCOS-MK は、複数の RTOS 実行環境を管理し、実行環境間の調停を行います。この実行環境は優先度の高い上位層（閉じられた RTOS 実行環境を本書では層と表現します）、優先度の低い下位層から成り立ち、同一優先度の層はありません。また、層の数はマイコンやプロセッサのアーキテクチャに依存します。



MULCOS-MK のコード自体は最上位層に配置され、下位層からは分離されているため、最上位層以外には RTOS の実体はなく仮想化されています。そのため、制御データなどの RTOS のみがアクセスするメモリ領域も最上位層に配置され、下位層からのアクセスが制限されるため、セキュリティの面からもメモリ保護機能が備わり優位性があります。

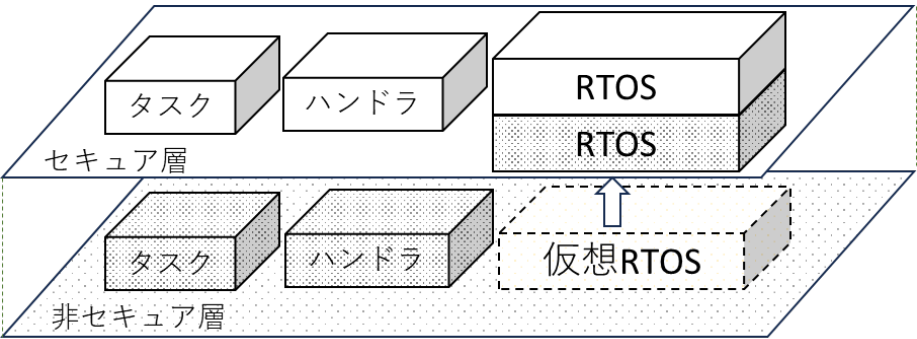
1-1. Cortex-M 対応 MULCOS-MK とは

Cortex-M 対応 MULCOS-MK では、ARM®v8-M アーキテクチャの TrustZone®に対応し、上位層であるセキュア層と、下位層である非セキュア層とで、2つの RTOS 実行環境を管理します。ただし、ARMv8-M アーキテクチャのベースラインには対応していません。



セキュア層と非セキュア層とで独立した実行環境でありながら、TrustZone の特徴であるセキュアゲートウェイやコールバックに対応し、タスクまたはハンドラは、セキュアへの移動も非セキュアへの移動も容易に行えます。

RTOS のコードと RTOS のみがアクセスするメモリは、セキュア層内に配置され、非セキュア層では仮想化された RTOS を使用することになります。



また、ARM®v7-M アーキテクチャを含めた TrustZone 未実装のデバイスでは、単一の実行環境として構築することができ、一般的な RTOS として機能することができます。

1-2. Cortex-M 対応 MULCOS-MK の μ ITRON4.0 仕様版とは

Cortex-M 対応 MULCOS-MK の RTOS には、μ ITRON4.0 仕様の RTOS を採用しています。

MULCOS-MK の μ ITRON4.0 仕様版は、フルセットではなく、以下のようなサブセットを実装しています。仕様に定められる静的 API とコンフィグレータには対応せず、独自のコンフィグレーションを採用し、C 言語 API のみに対応しています。また、各システムコール機能の対応／非対応の詳細に関しては「第 6 章 RTOS の機能とシステムコール」を参照してください。

○：対応、×：非対応

タスク管理機能			
タスクの生成	○	タスクの削除	○
タスクの起動	○	タスクの起動キャンセル	○
タスクの起動（起動コード指定）	○	自タスクの終了	○
自タスクの終了と削除	○	タスクの強制終了	○
タスク優先度の変更	○	タスク優先度の参照	○
タスクの状態参照	○	タスクの状態参照（簡易版）	○
タスク付属同期機能			
起床待ち	○	起床待ち（時間監視）	○
タスクの起床	○	タスク起床要求キャンセル	○
待ち状態の強制解除	○	強制待ち状態への移行	○
強制待ち状態からの再開	○	強制待ち状態からの強制再開	○
自タスクの遅延	○		

タスク例外処理機能			
タスク例外処理ルーチンの定義	×	タスク例外処理の要求	×
タスク例外処理の禁止	×	タスク例外処理の許可	×
タスク例外処理の禁止状態の参照	×	タスク例外処理の状態参照	×
同期・通信機能（セマフォ）			
セマフォの生成	○	セマフォの削除	○
セマフォ資源の返却	○	セマフォ資源の獲得	○
セマフォ資源の獲得（ポーリング）	○	セマフォ資源の獲得（時間監視）	○
セマフォの状態参照	○		
同期・通信機能（イベントフラグ）			
イベントフラグの生成	○	イベントフラグの削除	○
イベントフラグのセット	○	イベントフラグのクリア	○
イベントフラグ待ち	○	イベントフラグ待ち（ポーリング）	○
イベントフラグ待ち（時間監視）	○	イベントフラグの状態参照	○
同期・通信機能（データキュー）			
データキューの生成	○	データキューの削除	○
データキューへの送信	○	データキューへの送信（ポーリング）	○
データキューへの送信（時間監視）	○	データキューへの強制送信	○
データキューからの受信	○	データキューからの受信（ポーリング）	○
データキューからの受信（時間監視）	○	データキューの状態参照	○
同期・通信機能（メールボックス）			
メールボックスの生成	○	メールボックスの削除	○
メールボックスへの送信	○	メールボックスからの受信	○
メールボックスからの受信（ポーリング）	○	メールボックスからの受信（時間監視）	○
メールボックスの状態参照	○		
拡張同期・通信機能（ミューテックス）			
ミューテックスの生成	○	ミューテックスの削除	○
ミューテックスのロック	○	ミューテックスのロック（ポーリング）	○
ミューテックスのロック（時間監視）	○	ミューテックスのロック解除	○
ミューテックスの状態参照	○		
拡張同期・通信機能（メッセージバッファ）			
メッセージバッファの生成	○	メッセージバッファの削除	○
メッセージバッファへの送信	○	メッセージバッファへの送信（ポーリング）	○
メッセージバッファへの送信（時間監視）	○	メッセージバッファからの受信	○
メッセージバッファからの受信（ポーリング）	○	メッセージバッファからの受信（時間監視）	○
メッセージバッファの状態参照	○		

拡張同期・通信機能（ランデブ）			
ランデブポートの生成	×	ランデブポートの削除	×
ランデブの呼出し	×	ランデブの呼出し（時間監視）	×
ランデブの受け付け	×	ランデブの受け付け（ポーリング）	×
ランデブの受け付け（時間監視）	×	ランデブの回送	×
ランデブの終了	×	ランデブポートの状態参照	×
メモリプール管理機能（固定長メモリプール）			
固定長メモリプールの生成	○	固定長メモリプールの削除	○
固定長メモリブロックの獲得	○	固定長メモリブロックの獲得（ポーリング）	○
固定長メモリブロックの獲得（時間監視）	○	固定長メモリブロックの返却	○
固定長メモリプールの状態参照	○		
メモリプール管理機能（可変長メモリプール）			
可変長メモリプールの生成	○	可変長メモリプールの削除	○
可変長メモリブロックの獲得	○	可変長メモリブロックの獲得（ポーリング）	○
可変長メモリブロックの獲得（時間監視）	○	可変長メモリブロックの返却	○
可変長メモリプールの状態参照	○		
時間管理機能（システム時刻管理）			
システム時刻の設定	○	システム時刻の参照	○
タイムチェックの供給	○		
時間管理機能（周期ハンドラ）			
周期ハンドラの生成	○	周期ハンドラの削除	○
周期ハンドラの動作開始	○	周期ハンドラの動作停止	○
周期ハンドラの状態参照	○		
時間管理機能（アラームハンドラ）			
アラームハンドラの生成	×	アラームハンドラの生成	×
アラームハンドラの動作開始	×	アラームハンドラの動作停止	×
アラームハンドラの状態参照	×		
時間管理機能（オーバランハンドラ）			
オーバランハンドラの定義	×	オーバランハンドラの動作開始	×
オーバランハンドラの動作停止	×	オーバランハンドラの状態参照	×
システム状態管理機能			
タスク優先順位の回転	○	実行状態のタスク ID の参照	○
CPU ロック状態への移行	○	CPU ロック状態の解除	○
ディスパッチの禁止	○	ディスパッチの許可	○
コンテキストの参照	○	CPU ロック状態の参照	○
ディスパッチ禁止状態の参照	○	ディスパッチ保留状態の参照	○

システム状態の参照	○	
割込み管理機能		
割込みハンドラの定義	○	割込みサービスルーチンの生成 ○
割込みサービスルーチンの削除	○	割込みサービスルーチンの状態参照 ○
割込みの禁止	○	割込みの許可 ○
割込みマスクの変更	○	割込みマスクの参照 ○
サービスコール管理機能		
拡張サービスコールの定義	×	サービスコールの呼出し ×
システム構成管理機能		
CPU 例外ハンドラの定義	×	コンフィグレーション情報の参照 ○
バージョン情報の参照	○	

第 2 章 用語の定義

本カーネルと RTOS とで使用される用語を説明します。

2-1. カーネル用語の定義

本カーネルで使用する基本的な用語の定義を説明します。また、本 RTOS は μ ITRON 仕様のため、カーネル自体も μ ITRON 仕様に基づいた用語を使用しています。

2-1-1. セキュア層と非セキュア層

TrustZone を実装した本カーネルでは、セキュア層と非セキュア層の 2 つの RTOS 実行環境があり、これらは独立した実行環境として階層化されています。また、セキュア層から非セキュア層へのメモリアクセスは可能ですが、逆の非セキュア層からセキュア層へのアクセスは禁止されます。

ARMv7-M アーキテクチャを含めた TrustZone 未実装のデバイスでは単一の RTOS 実行環境となるため、セキュア層／非セキュア層の区別はありません。通常、この場合は非セキュア層と見なされます。

2-1-2. タスク

プログラムの実行単位で、RTOS がスケジューリング（後述）する単位を「タスク」と呼びます（一般的な RTOS には、スレッドやプロセスなどと呼ぶものもあります）。また、セキュア層と非セキュア層のタスクを区別する場合は、セキュア層で生成されたタスクをセキュアタスク、非セキュア層で生成されたタスクを非セキュアタスクと呼びます。

2-1-3. コンテキスト

プログラムが実行される環境を「コンテキスト」と呼びます。一般的に、同一のコンテキストとは、プロセッサコアのレジスタやスタック領域が継承されていることを言います。

2-1-4. 移行タスク

Cortex-M の TrustZone 仕様ではセキュア／非セキュア間の移動が定められ、タスクは RTOS を介在せずに自由に移動できます。そのため、生成された実行環境とは異なる実行環境へ移動したタスクを本カーネルでは移行タスクと呼び、非セキュアに移動中のセキュアタスクを非セキュア移行タスク、セキュアに移動中の非セキュアタスクをセキュア移行タスクと呼びます。

2-1-5. ディスパッチとプリエンプション

RTOS が実行するタスクを切り替える動作を「ディスパッチ」と呼び、その機構を「ディスパッチャ」と呼びます。ディスパッチは RTOS 内部で行われ、通常、プログラミング上で意識する必要はありません。

システムコールの処理により、RTOS が実行中のタスクよりも優先順位の高いタスクを実行可能状態にすると判断した際に、実行中タスクを中断させる動作を「プリエンプション」、優先順位の高いタスクにディスパッチすることを「コンテキストスイッチ」と呼びます。

割込みの発生により、実行中のタスクが中断して、例外処理に遷移することを「プリエンプト」と呼びます。プリエンプトによって実行された例外処理から呼出したシステムコールにより、タスクに戻る際に必要に応じてディスパッチが発生する場合があります。この一連の動作を「プリエンプション」と呼び、この機能を持つことを「プリエンティブな機構」と表現します。

2-1-6. システム時刻とタイムチック

RTOS 起動時を起点の 0 とする時刻を「システム時刻」と呼び、これを基準に時間管理を行っています。そして、このシステム時刻をカウントアップする刻み時間を「チック」と呼び、このチックの処理を行うきっかけ（一般的にはタイマ割込み）を「タイムチック」と呼びます。本カーネルの内部では、マイクロ秒単位のシステム時刻を管理しています。

2-2. RTOS 用語の定義

本 RTOS の μ ITRON4.0 仕様の基本的な用語の定義を説明します。

2-2-1. タスクコンテキストと非タスクコンテキスト

タスクの実行環境であるコンテキストを「タスクコンテキスト」と呼び、タスク以外のコンテキストを「非タスクコンテキスト」と呼びます。本 RTOS の非タスクコンテキストには、例外処理として割込みハンドラと割込みサービスルーチンが、タイムチックに同期して起動するタイムイベントハンドラがあります。また、本 RTOS のタイムイベントハンドラでは、周期ハンドラのみを実装しています。

2-2-2. 特権モードと非特権モード

本 RTOS のシステムコールと非タスクコンテキストは特権モードで実行され、タスクコンテキストは特権モードと非特権モードとをタスク生成時に選択できます。これにより、非特権モードのタスクから、RTOS のみがアクセスするメモリや指定のペリフェラルを保護することができます。また、特権モードのタスクから、特権モードでなければいけないシステムレジスタに対してアクセスできる利点もありますが、システムメモリ領域にもアクセスできるため注意が必要です。

2-2-3. オブジェクトと ID 番号

システムコールの呼出等により RTOS が操作対象とする資源を「オブジェクト」と呼びます。具体的には、タスクの他に、セマフォやイベントフラグなどがあります。このオブジェクトを識別するためにオブジェクト毎に 1 以上の番号を割付け、この番号を「ID 番号」或いは「オブジェクト ID」と呼びます。また、オブジェクトはセキュア層と非セキュア層とがあり、これらを区別する場合は、「セキュアオブジェクト」や「非セキュアオブジェクト」と呼びます。

2-2-4. タスク優先度と優先順位

RTOS がディスパッチするタスクを決めるためのパラメータは「タスク優先度」です。タスク優先度に従ってディスパッチするタスクを決める動作のことを「スケジューリング」と呼び、その機構を「スケジューラ」と呼びます。タスク優先度の値は、最高優先度を意味する 1 から昇順に、優先度が低くなります。

なお、同一優先度の場合は、実行可能状態に遷移した順でスケジューリングします。また、先にディスパッチすべきタスクの方を「優先順位」が高いと表現します。

2-2-5. 相対時間と時間監視

システムコールにより、待ち状態を解除するまでの時刻、或いは、何らかの処理を起動するまでの時刻を指定する場合は、ミリ秒単位の「相対時間」を用います。相対時間の基準時刻は、システムコールを呼出した時点のシステム時刻とします。また、システムコールにより自タスクが待ち状態に遷移する場合は、処理が完了するまで一定の時間を待ち続けることもできます。この時間待ちを「時間監視」と呼びます。

【補足説明】

μ ITRON4.0 仕様では、イベントが発生するシステムコールを呼出してからイベントの処理を行うまでの時間を保証することになっています。そのため、システムコールを呼出した時点ではなく、その後の最初のタイムチェックのシステム時刻を基準とした「相対時間」を指定します。

本 RTOS では、この相対時間に関して μ ITRON4.0 仕様を逸脱し、旧バージョンの μ ITRON3.0 仕様を採用しています。つまり、システムコールを呼出した時点のシステム時刻を基準とした「相対時間」を指定します。

2-2-6. 待ち行列

システムコールはオブジェクトに対して、処理が完了するまで待ちが発生する場合があります。その際に同一のオブジェクトに対して複数のタスクの待ちが発生した場合、優先順位に従って待つ機能を「待ち行列」と呼びます。待ち行列には、タスク優先度順と FIFO 順とがあります。FIFO 順とは、一番早い時刻から待っているタスクを先に処理することを言います。タスク優先度順とは、タスク優先度の高い方が先に処理され、同一優先度のタスクであれば、早い時刻から待っているタスクを先に処理することを言います。

2-2-7. レディキュー

実行可能状態のタスクをタスク優先度順にした行列です。この行列の最高優先順位のタスクが実行状態になります。ただし、ディスパッチ保留状態の場合には、実行状態のタスクより優先順位の高いタスクが実行可能状態になる可能性があります。

2-2-8. ディスパッチ保留状態

ディスパッチ保留状態とは、以下のいずれかに当てはまる状態です。実行中タスクの優先度よりも優先順位の高い実行可能状態のタスクがあるか否かに関わらず、ディスパッチを抑止する状態です。

- CPU ロック状態
- ディスパッチ禁止状態
- 割込み優先度マスクを変更している状態

2-2-9. キューイング

処理要求を保持する機構を「キューイング」と呼び、未処理の要求をカウントすることで実装しています。このキューイングを使用する要求として、タスクの起動要求と起床要求と強制待ち要求とがあり、これらのカウントの最大値を 255 に制限しています。また、本 RTOS では、タイムチェックによるシステム時刻の更新要求をキューイングしています。

2-2-10. エラーコード

システムコールにより本 RTOS へ処理を要求しますが、要求項目の異常、或いは本 RTOS の状態により受け入れることができない場合があります。このような場合には、負の値を持つエラーコードとして異常の種類を知らせます。ただし、本 RTOS ではエラーコードに優先順位はなく、異常と判断される要素が複数存在した場合は、どのエラーコードが返るのかは不定です。

第 3 章 RTOS の基本機能

μ ITRON4.0 仕様である本 RTOS の基本的な機能を説明します。

3-1. タスクの状態とスケジューリング

タスクが持つ状態の種類と、その状態の遷移を行うスケジューリング規則について説明します。

3-1-1. タスクの状態

本 RTOS のタスクには、次の 5 つの状態があります。

A) 実行状態 (RUNNING)

現在実行しているタスクの状態です。ただし、プリエンプトして非タスクコンテキストを実行している場合は、プリエンプト時に実行していたタスクを実行状態とします。

B) 実行可能状態 (READY)

実行できる状態ですが、より優先順位の高いタスクが実行中のため、実行を保留されている状態です。

つまり、実行可能状態のタスクの中で一番優先順位の高いタスクが実行中状態になることができます。ただし、ディスパッチ禁止状態や CPU ロック状態などの、実行状態の遷移を抑止する状態を除きます。

なお、実行可能状態に遷移する要因には、実行中タスクがプリエンプトされた場合と、システムコールにより起床、或いは待ち状態が解除されるタスクが実行状態のタスクより優先順位が低い場合等があります。

C) 待ち状態 (WAITING)

呼出したシステムコール機能の条件が成立するのを待っている状態です。具体的には、各オブジェクトの待ち条件の成立待ち、起床待ち、遅延時間待ちです。

D) 強制待ち状態 (SUSPENDED)

強制的に実行を中断された状態です。

E) 二重待ち状態 (WAITING-SUSPENDED)

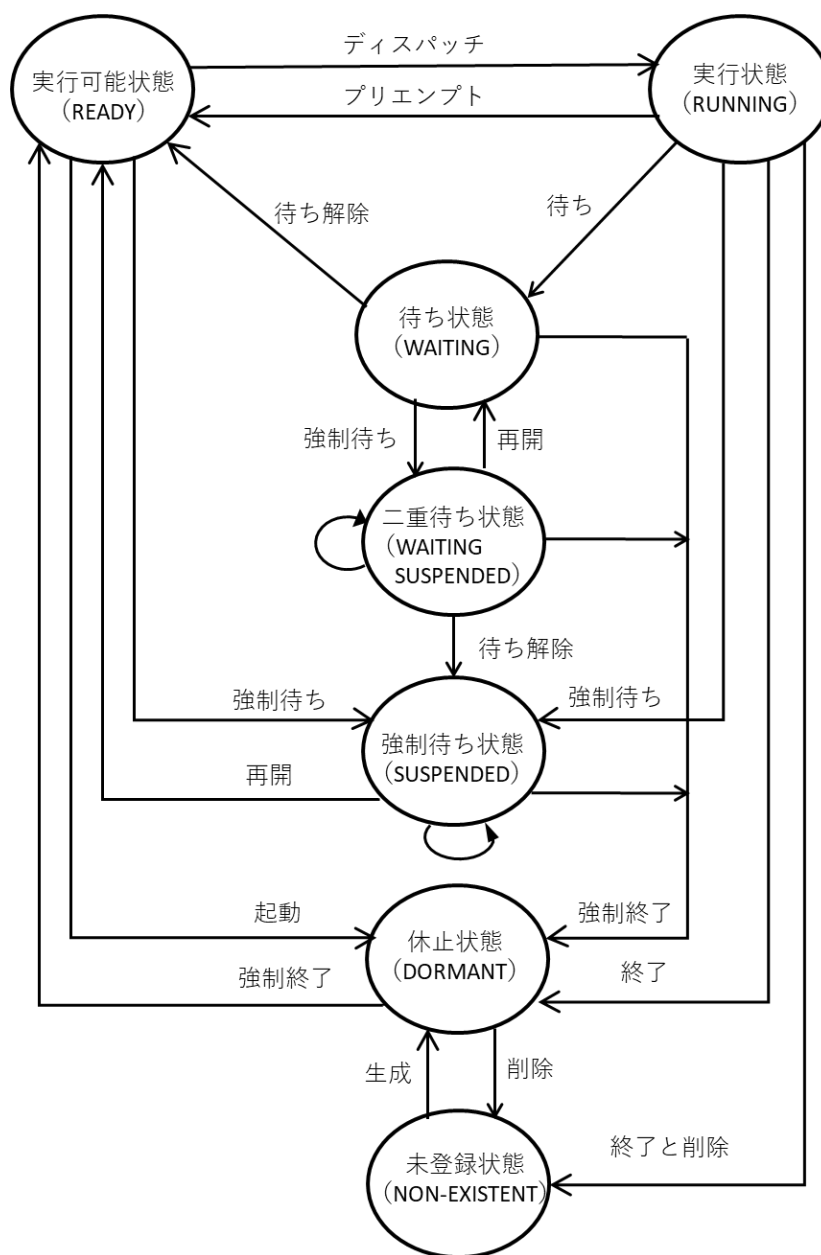
待ち状態と強制待ち状態が重なった状態です。具体的には、待ち状態のタスクに対して、強制待ち状態への移行を要求された状態です。二重待ち状態のタスクが、各オブジェクトの待ち条件の成立待ち、起床待ち、遅延時間待ちの待ちを解除されると強制待ち状態へ遷移します。二重待ち状態のタスクが、強制待ち状態が再開されると待ち状態へ遷移します。

F) 休止状態 (DORMANT)

タスクが起動する前か、実行していたタスクが終了した状態です。実行状態から休止状態に遷移した場合のコンテキストは保存されず、再度起動した場合には初期化されます。

G) 未登録状態 (NON-EXISTENT)

タスクが生成する前の状態、または削除された後の状態であり、RTOS にも登録されていないため、RTOS から見ると存在しない状態です。



タスクの状態遷移図

タスクの状態遷移は「タスクの状態遷移図」のようになります。ここで、タスクは休止状態から実行可能状態を経由して実行状態に遷移しますが、実行中タスクも含めて実行可能状態のタスクの優先順位が低い場合は、実行可能状態から即実行状態に遷移しディスパッチが発生します。

また、休止状態から実行可能状態への遷移を「タスクを起動する」と、タスクの起動から休止状態または未登録状態に遷移するまでの期間を「起動された状態」と表現します。

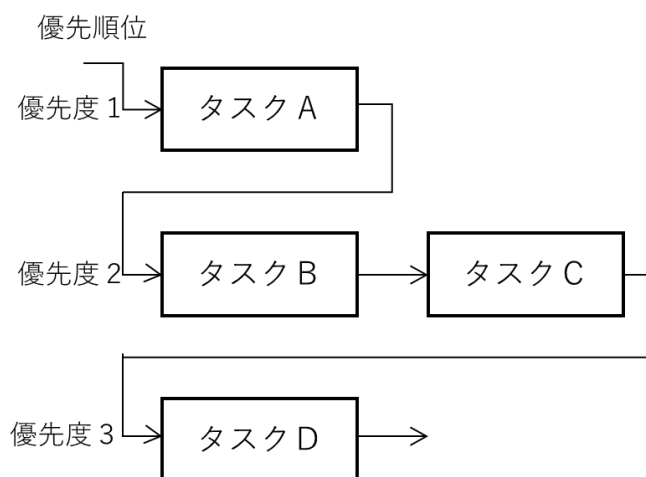
3-1-2. スケジューリング規則

スケジューリングの仕様として、タスクに紐付けられた優先度に応じたプリエンプティブな優先度ベースのスケジューリング方式を採用しています（ μ ITRON 仕様ではラウンドロビン等の方式は非対応）。同一の優先度を持つタスクでは先入れ先出し方式によるスケジューリングになります。

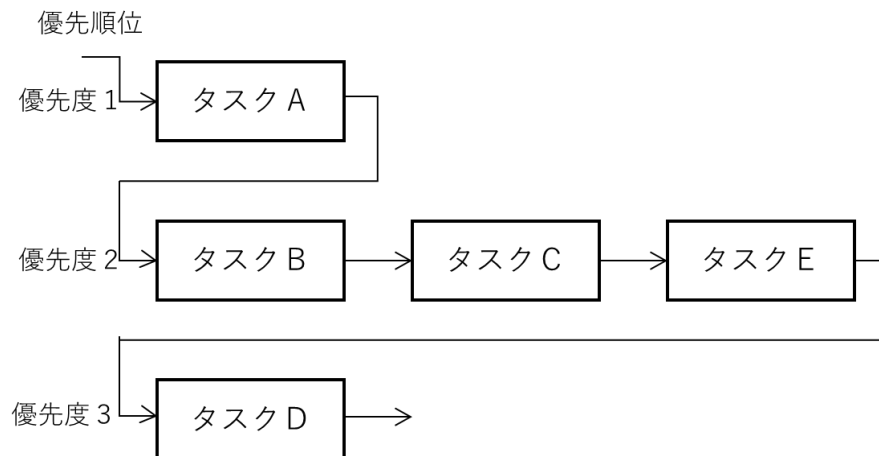
実行可能状態のタスクが複数ある場合には、一番優先度の高いタスクが、さらに同一優先度のタスクが複数ある場合には、一番先に実行可能状態になったタスクが、最も優先順位が高いタスクになります。

最も優先順位が高いタスクが入れ替わる場合にディスパッチが発生します。ただし、システムがディスパッチ保留状態にある場合には、その状態が解除されるまでディスパッチは発生しません。

例えば、次のように実行可能なタスクがあると仮定します。ここでは、最優先順位はタスク A となり、実行状態になり得るタスクとなります。

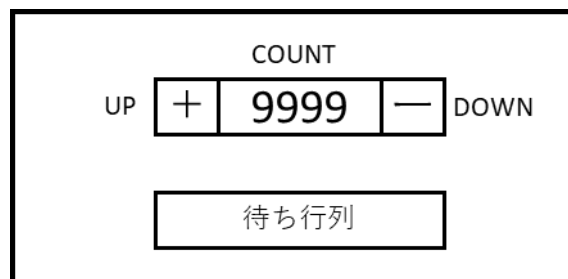


この状態で、優先度 2 のタスク E が実行可能状態になった場合、その優先順位は優先度 2 の末尾となり下記のようになります。



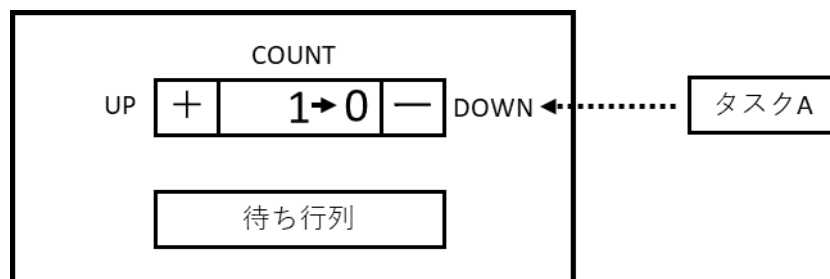
3-2. セマフォ同期・通信機能

セマフォは、仮想資源を数値として表現し管理することで、排他制御や資源の通知などの同期を行うためのオブジェクトです。このセマフォは、資源数と資源の獲得待ちタスクの待ち行列を持ち、これらにより管理します。また、獲得待ち行列に FIFO 順かタスク優先度順かを指定して生成します。

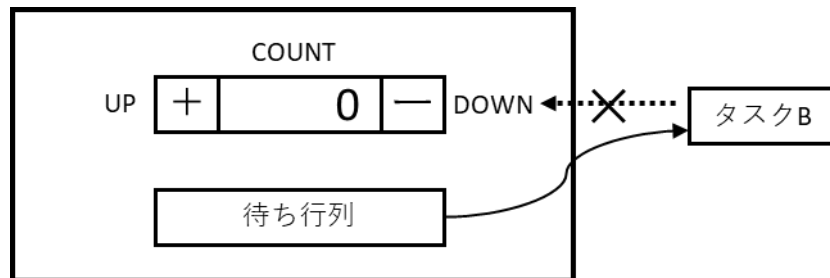


3-2-1. セマフォを排他制御に用いる場合

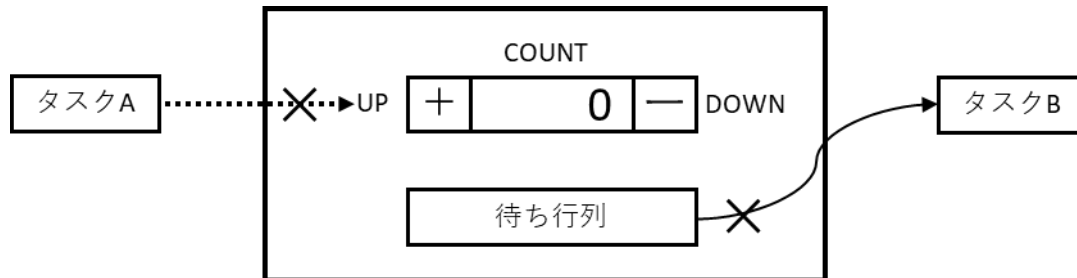
一つの資源を排他制御するため、資源数の初期値が 1 のセマフォを生成します。ここでタスク A は、資源数 count を -1 することで、資源を獲得できます。



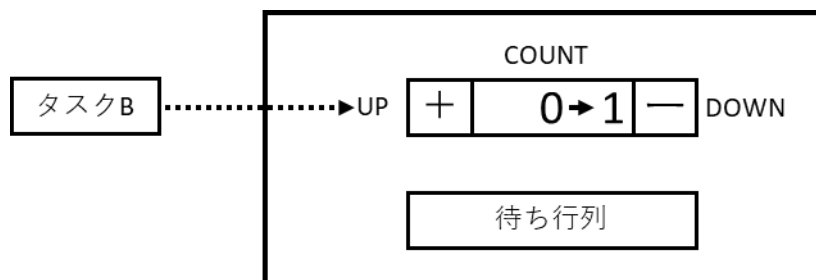
この状態でタスク B が資源の獲得を試みると、資源数が 0 のため獲得に失敗します。この時、資源を獲得できるまで、待ち行列に接続することができます。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



この状態でタスク A が資源を返却すると、獲得待ち行列にタスクが接続されているので、資源数を 1 にしません。その代わりに、タスク B の待ち状態を解除し、実行可能状態に遷移させます。これにより、タスク B が資源を獲得したことになります。



さらに、この状態でタスク B が資源を返却すると、待ち行列にタスクが接続されていないので、資源数が 1 になり、初期の状態に戻ります。

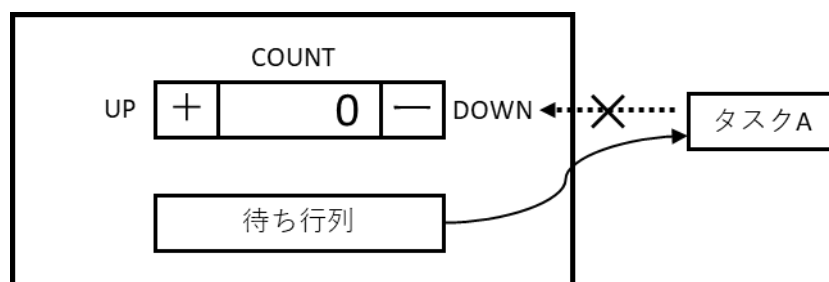


このようにして、資源を獲得するタスクを資源数に制限することで、排他制御が成立します。また、資源数が 0 と 1 に制限されているセマフォをバイナリセマフォと呼びます。

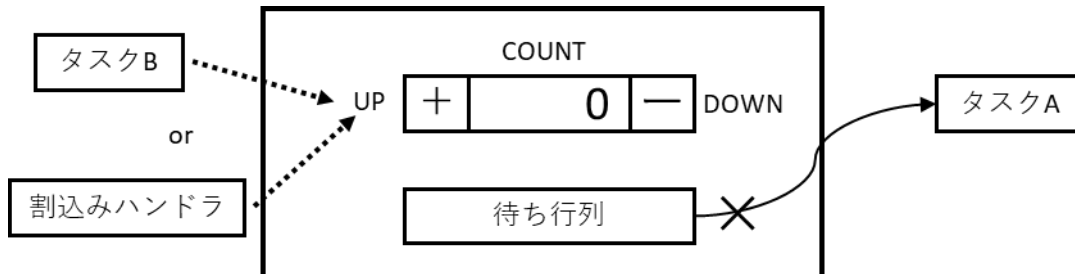
3-2-2. セマフォを資源の通知に用いる場合

タスクからタスクへ、或いは割込みハンドラからタスクへ、何らかの処理要求を通知する場合があります。このような場合、この処理要求にセマフォを使用することで、タスク間或いは割込みハンドラとタスク間で同期をとることができます。この場合は、資源数の初期値が 0 のセマフォを生成します。

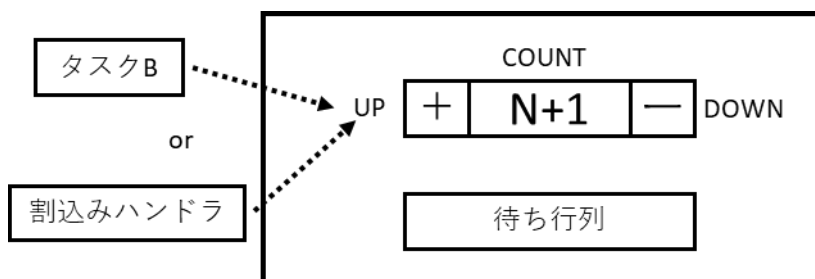
資源を受け取る側のタスク A は、資源数が 0 のため、待ち行列に接続して資源の獲得待ちになります。



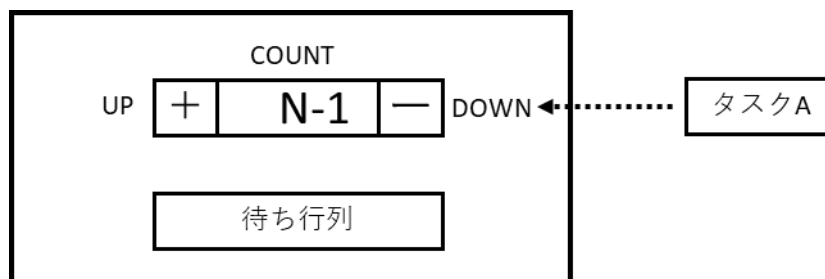
資源を返却する側のタスク B、或いは割込みハンドラから資源の返却が行われます。これにより、資源数が 0 のまま、タスク A は資源を獲得することができます。



資源を送る側のタスク B、或いは割込みハンドラから資源の返却時、待ち行列にタスクが接続されていなかった場合は、資源数に 1 を加算します。



タスク A が資源の獲得を試みて、一つ以上の資源がある場合には、資源数から 1 を減算します。

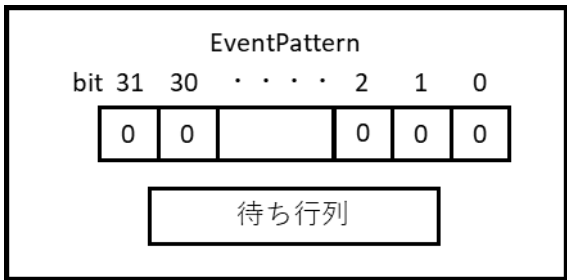


このように、資源の返却と獲得の回数が一致するため、資源を失うことなく処理することができます。ただし、最大資源数を超えない範囲の場合です。また、資源数が 2 以上になるセマフォをカウンティングセマフォと呼びます。

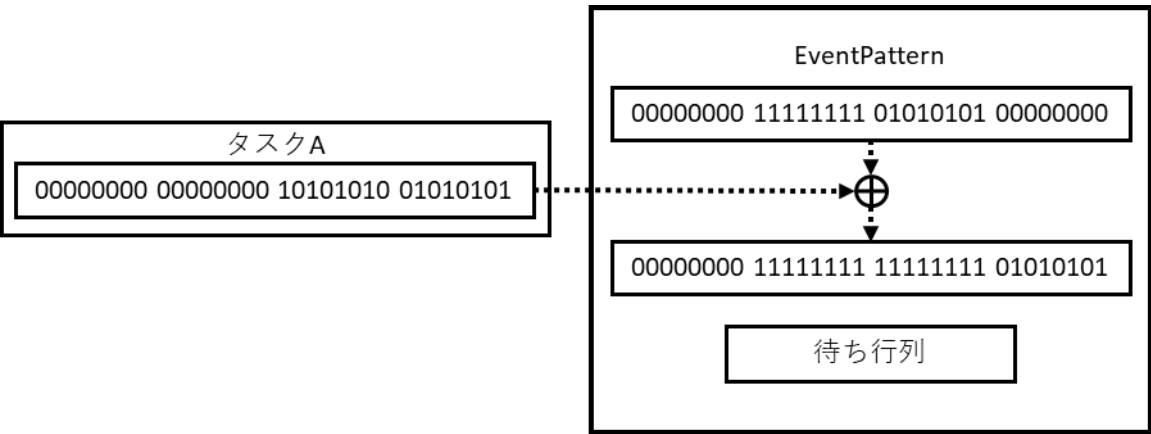
3-3. イベントフラグ同期・通信機能

イベントフラグは、複数のイベント情報を複数ビットで表現し管理することで、同期を行うためのオブジェクトです。本 RTOS では、これを 32 ビット幅で定義しています。このイベントフラグは、イベントパターンとイベントフラグのセット待ちタスクの待ち行列を持ち、これらにより管理します。

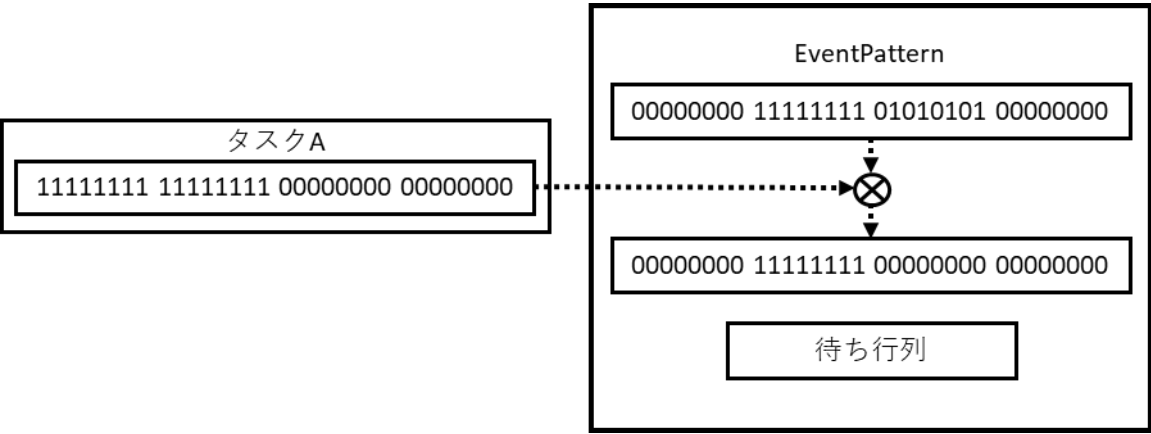
イベントフラグは、セット待ち行列 FIFO 順かタスク優先度順かを、また、複数タスクが接続できるか否かを指定して生成します。



イベントフラグのセットでは、イベントパターンをビット毎に論理和を取ります。

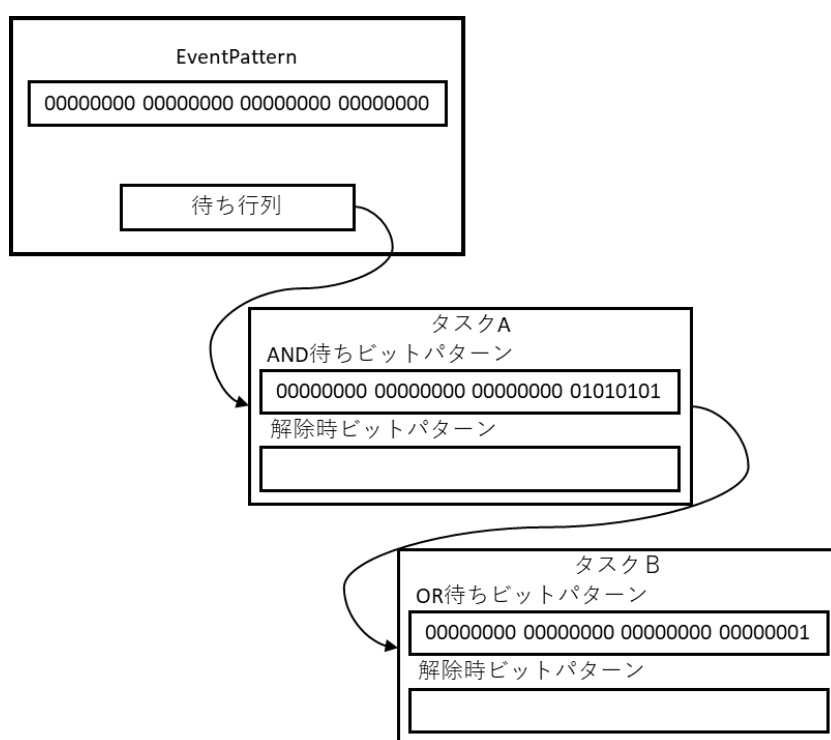


イベントフラグのクリアでは、イベントパターンとのマスクパターンを指定し、論理積をとります。



イベントフラグの条件が成り立つとは、AND 待ちでは指定ビットがすべてセットされた場合、OR 待ちでは指定ビットの一つ以上がセットされた場合です。これらは、イベントフラグ待ち毎に指定することができ、同一のイベントフラグに対して、AND 待ちのタスクと OR 待ちのタスクを混在させることができます。

タスク A がイベントフラグの AND 待ちを試みて、条件が成り立たずにセット待ちタスクの待ち行列に接続され、タスク B はイベントフラグの OR 待ちを試みて同様にセット待ちタスクの待ち行列に接続される場合もあり得ます。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



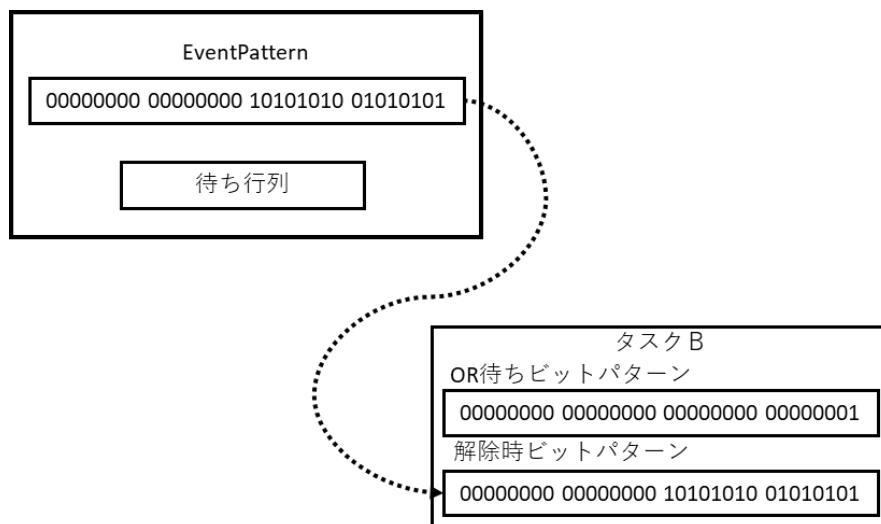
ただし、セット待ち行列に複数タスクが接続できない指定で生成された場合は、タスク B のイベントフラグのセット待ちでエラーが戻ります。

イベントフラグの生成時に、イベントフラグ待ちで条件が成り立ちセット待ちから解除される場合で、イベントパターンをクリアするように指定することができます。

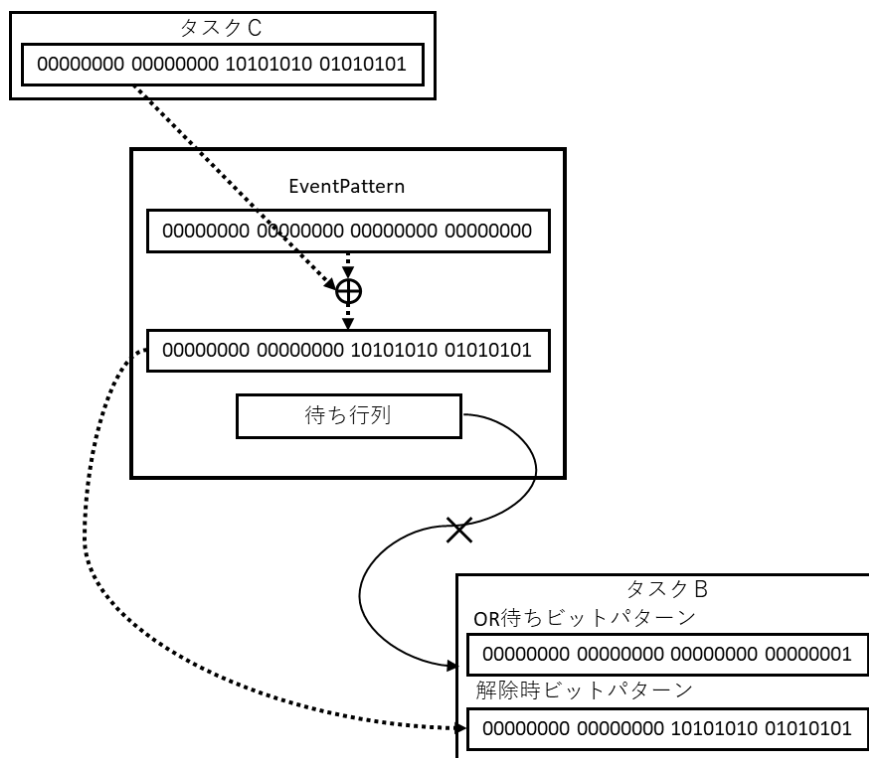
3-3-1. イベントフラグをクリアせず読み出す場合

イベントフラグ待ちを試みた際に、既に条件の成り立つ待ちビットパターンだった場合は、イベントフラグのイベントパターンを読み出して復帰します。

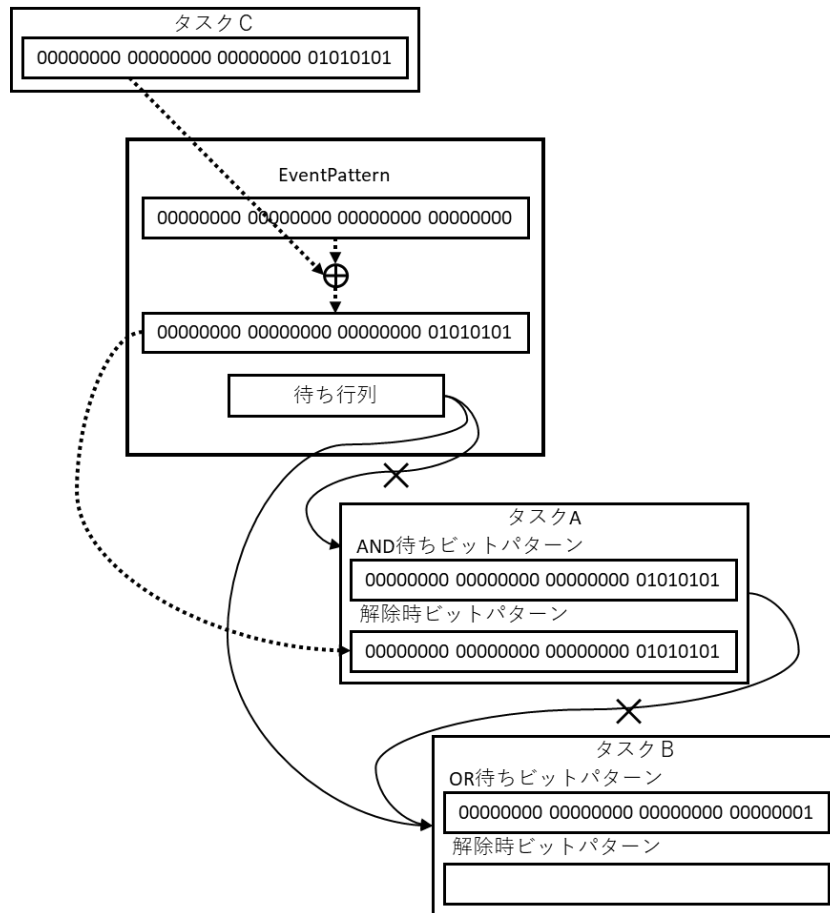
このタスク B はイベントパターンを読み出すだけで、イベントフラグに変化はありません。



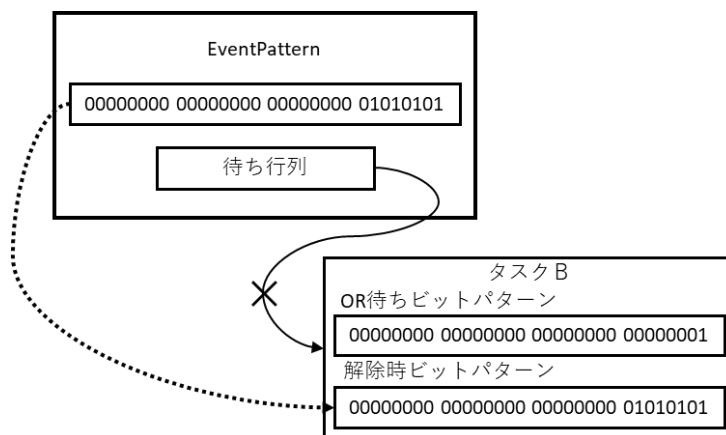
タスク B がイベントフラグ待ち行列に接続した状態で、タスク C がタスク B の待ち条件が成り立つイベントパターンをセットした場合は、イベントフラグのイベントパターンを更新した後、タスク B を待ち状態から解除し、実行可能状態に遷移させます。



イベントフラグのセット待ち行列にタスク A とタスク B が接続した状態で、タスク C が条件の成り立つイベントパターンをセットした場合は、イベントフラグのイベントパターンを更新した後、先頭のタスク A の待ち状態を解除し、実行可能状態に遷移させます。



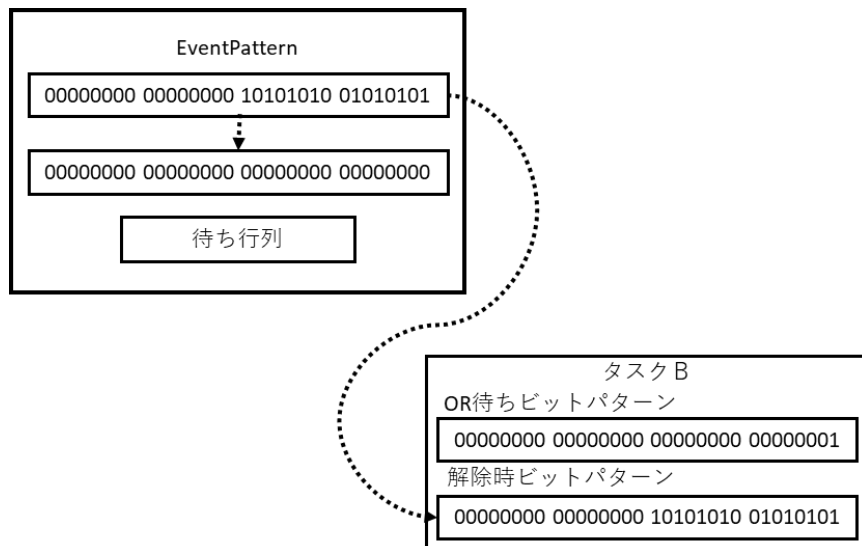
続けて、タスク B のイベントフラグのイベントパターンの条件が成り立つ場合は、これも処理します。つまり、タスク B の待ち状態を解除し、実行可能状態に遷移させます。



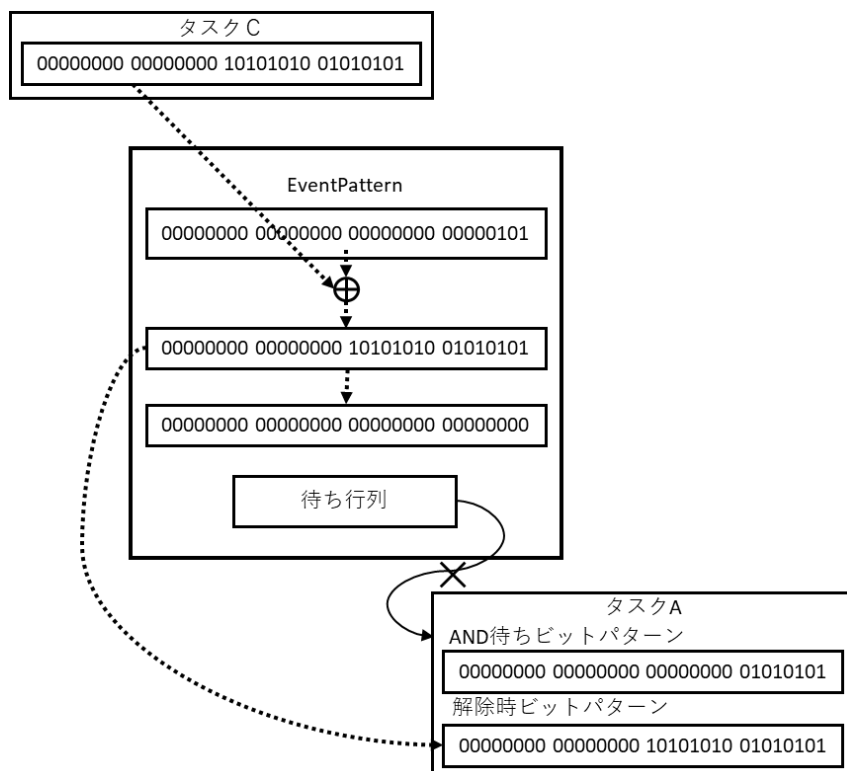
このように、イベント待ち行列に接続された全タスクを順次評価します。

3-3-2. イベントフラグを読み出し後にクリアする場合

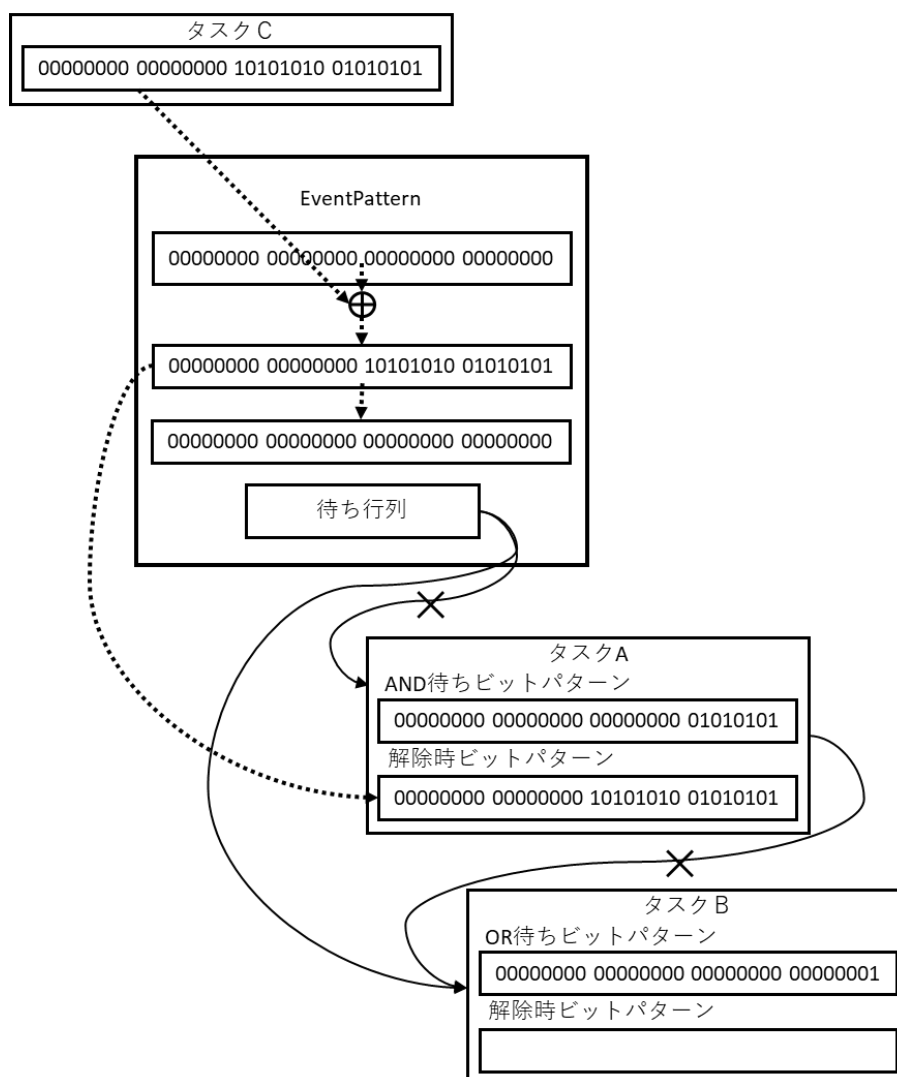
イベントフラグ待ちで、既に条件の成り立つ待ちビットパターンだった場合は、イベントフラグのイベントパターンを読み出した後、イベントパターンをクリアします。



タスク A がイベントフラグ待ち行列に接続した状態で、タスク C が条件の成り立つイベントパターンをセットした場合は、イベントフラグのイベントパターンをタスク B に渡して待ち状態を解除し、実行可能状態に遷移します。その後、イベントフラグのイベントパターンの全ビットをクリアします。



イベントフラグのセット待ち行列にタスク A とタスク B が接続した状態で、タスク C が条件の成り立つイベントパターンをセットした場合は、イベントフラグのイベントパターンを先頭のタスク A に渡して待ち状態を解除し、実行可能状態に遷移します。その後、イベントフラグのイベントパターンの全ビットをクリアします。

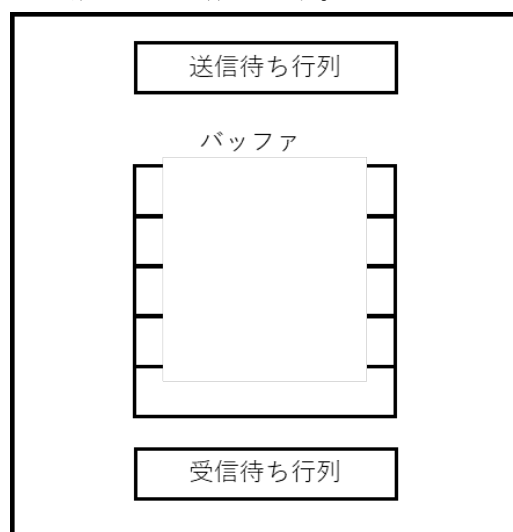


このように、条件が成り立ったタスクでイベントフラグのイベントパターンがクリアされるため、後続のタスクを評価しません。つまり、複数のタスクに条件が成り立っていても、待ち行列の先頭に近いタスクだけがセット待ちから解除されます。

3-4. データキュー同期・通信機能

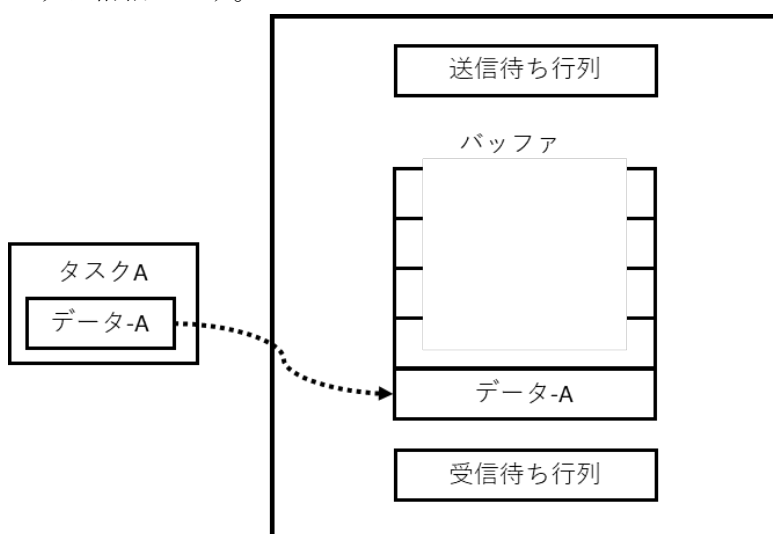
データキューは、1 ワードのデータを受け渡しすることで、同期と通信を行うためのオブジェクトです。この 1 ワードとは、整数型およびポインタ型を扱うことができ、そのビット幅はアーキテクチャに依存し、本 RTOS では 32 ビット幅で定義しています。このデータキューは、データを蓄えるバッファと、送信と受信待ちタスクの 2 つの待ち行列を持ち、これらにより管理します。

データキューは、送信待ち行列に FIFO 順かタスク優先度順かの指定と、蓄えることができるデータの個数とを指定して生成します。

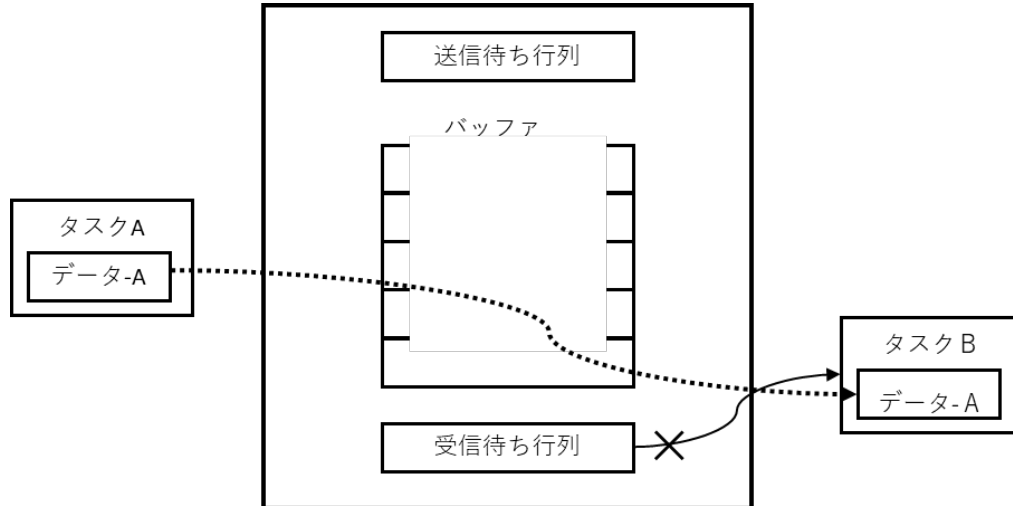


3-4-1. データキューへ送信する場合

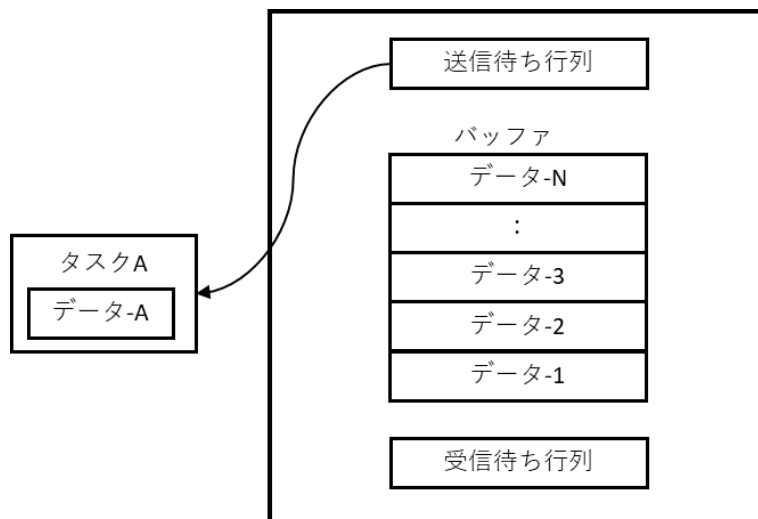
受信待ち行列にタスクが接続されておらずバッファに空きがある場合は、データをバッファに格納します。



受信待ち行列にタスクが接続されている場合は、受信待ちタスクに送信データを渡して待ち状態を解除し、実行可能状態に遷移させます。また、受信待ち行列にタスクが接続されている場合は、バッファ内にデータは存在しません。

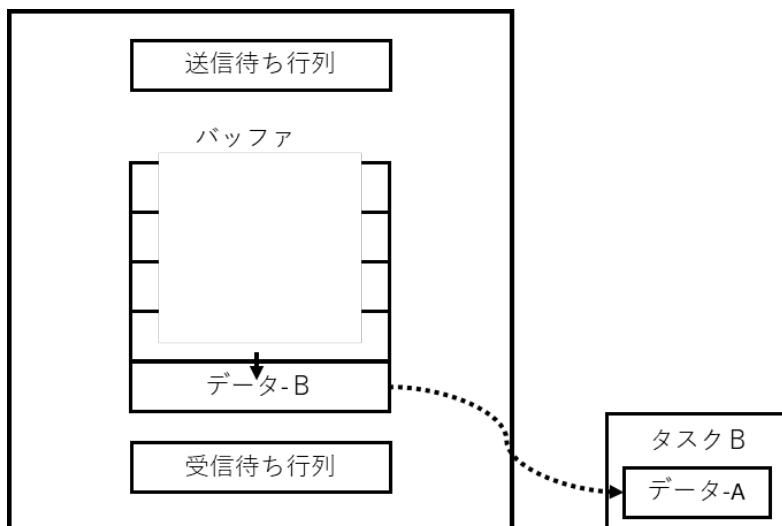


バッファが満杯で空きがない場合は、送信を試みたタスクは送信待ち行列に接続されて、データキューの送信待ち状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。

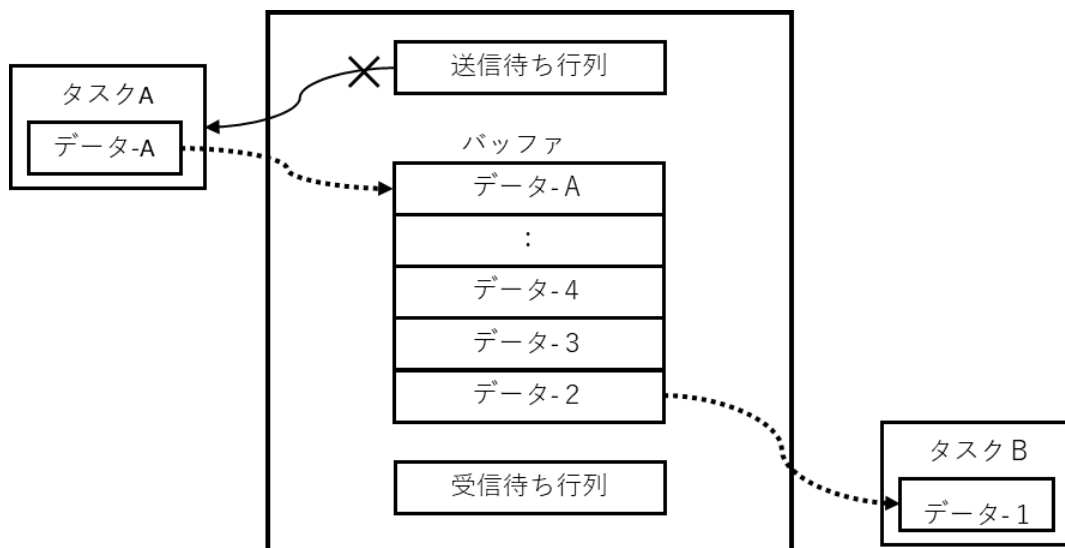


3-4-2. データキューから受信する場合

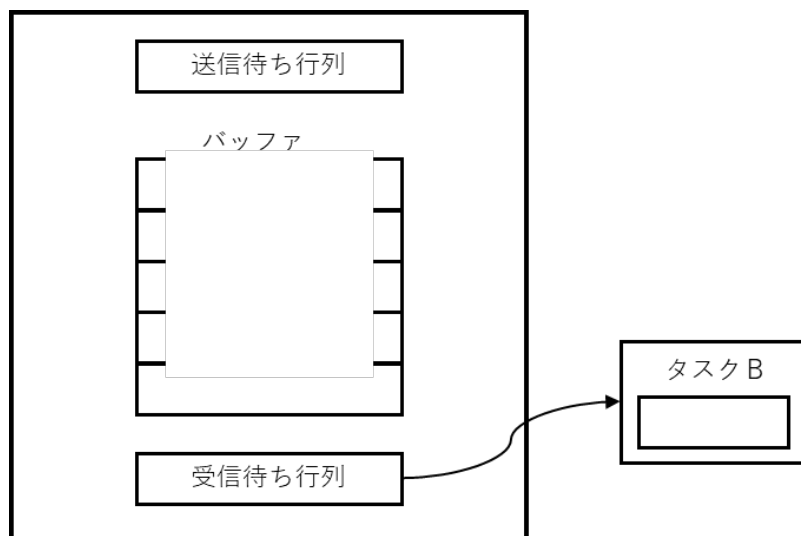
送信待ち行列にタスクが接続されておらず、バッファに一つ以上のデータがある場合は、タスクは一番古いデータをバッファから取り出します。



送信待ち行列にタスクが接続されている場合も、受信するタスクは一番古いデータをバッファから取り出します。バッファに空きができるため、送信待ちタスクが送信しようとしたデータをバッファに格納し、送信待ちを解除した後、実行可能状態に遷移させます。

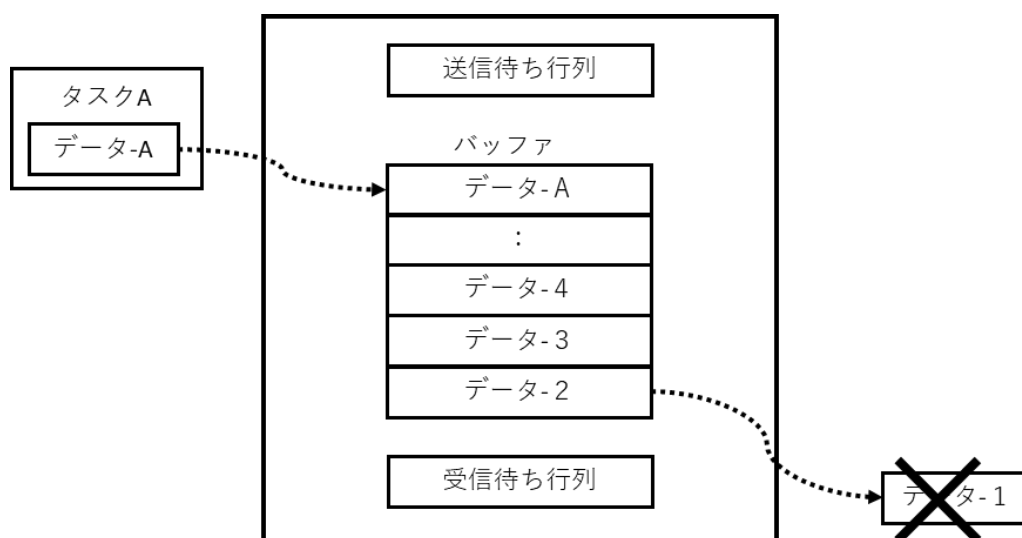


バッファに一つのデータもない場合は、受信を試みたタスクは受信待ち行列に接続されて、データキューの受信待ち状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



3-4-3. データキューへ強制送信する場合

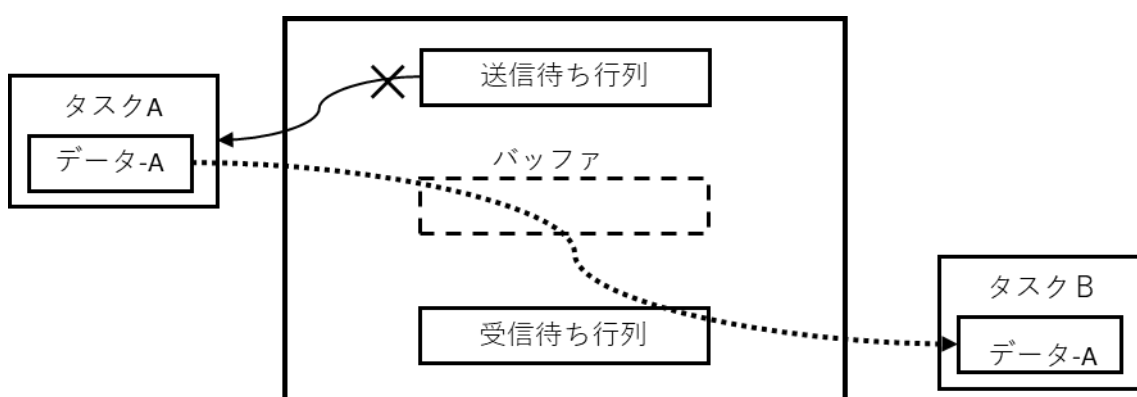
強制的にデータキューへデータを送信することができます。バッファに空きがあれば通常の送信と同じ処理を行います。しかし、バッファが満杯の場合は、一番古いデータを消去してバッファに空きを作った後に送信データを格納します。



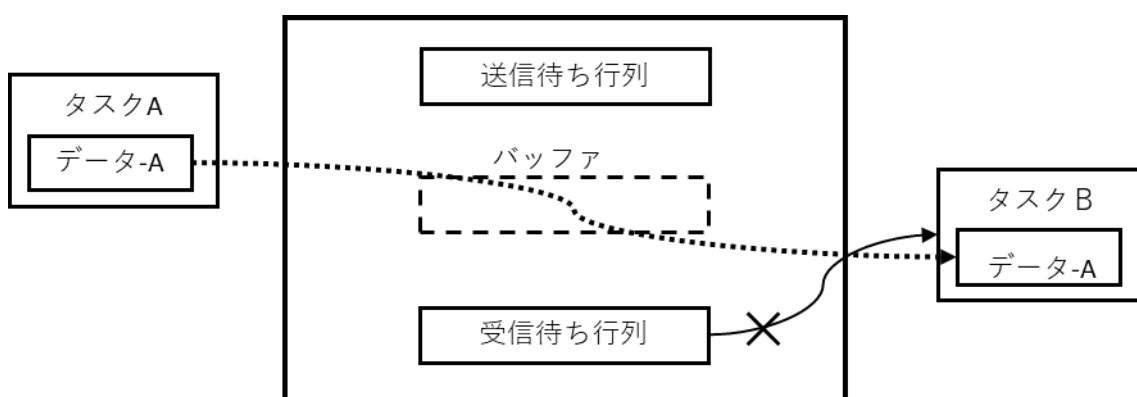
3-4-4. データキューの同期メッセージ機能の場合

データキューの送信と受信とを同じタイミングで行う同期メッセージ機能があります。この機能を使う場合は、データを蓄えることができるデータの個数を 0 として生成します。

タスク A が先に送信を試みた場合は、送信待ち行列に接続され、データキューの送信待ち状態に遷移します。その後、タスク B が受信を試みると、タスク A が送信しようとしたデータを受け取り、タスク A の待ち状態を解除し、実行可能状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



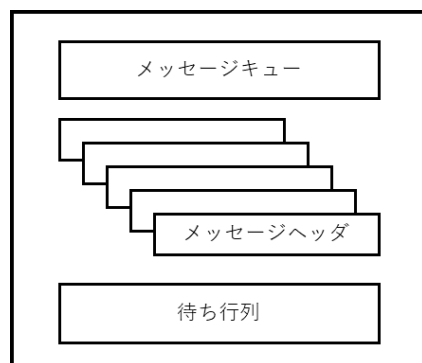
逆に、タスク B が先に受信を試みた場合は、受信待ち行列に接続され、データキューの受信待ち状態に遷移します。その後、タスク A が送信を試みると、送信データをタスク B に渡し、タスク B の待ち状態を解除し、実行可能状態に遷移します。



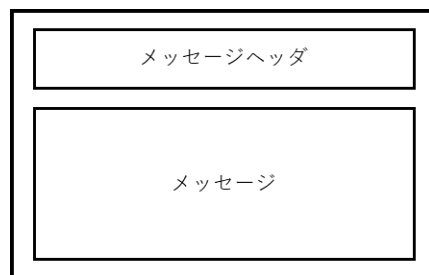
このようにして、送信と受信タスクが同時に、実行状態、或いは実行可能状態に遷移します。このとき、実行状態に遷移するのは、タスク優先順位の高い方のタスクです。

3-5. メールボックス同期・通信機能

メールボックスは、メモリ上に置かれたメッセージを受け渡すことで、同期と通信を行うためのオブジェクトです。このメールボックスは、メッセージを保管するメッセージキューと、メッセージの受信待ちタスクの待ち行列を持ち、これらにより管理します。また、受信待ち行列に FIFO 順かタスク優先度順か、メッセージキューにメッセージをいくつ接続できるか、メッセージ優先度を使用するか否かと優先度数を指定して生成します。

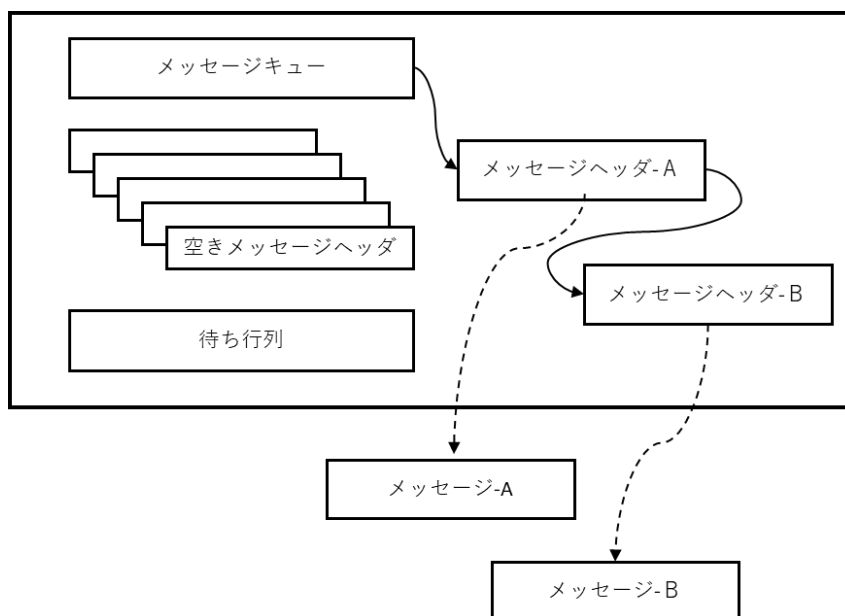


メールボックスで使用するメッセージ構造体は一般にメッセージパケットと呼び、 μ ITRON4.0 仕様ではメッセージヘッダとメッセージ本体を以下のように想定しています。

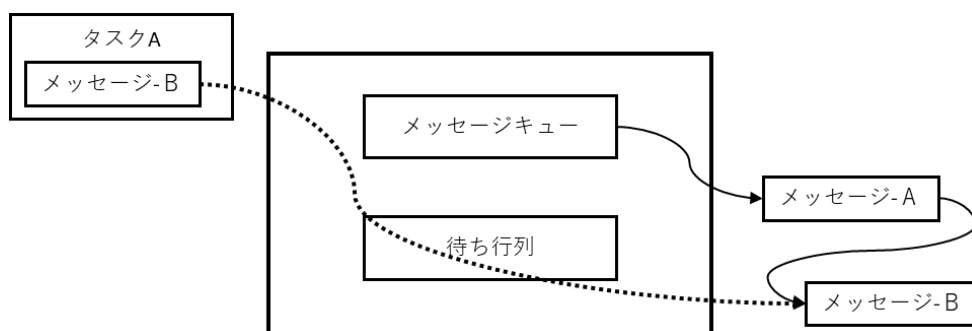


このようなメッセージパケットにメッセージヘッダを含む構造では、ユーザによるメッセージヘッダの破壊で発生する異常からシステムを保護することは難しく、本 RTOS ではメッセージヘッダとメッセージ本体とを分離して管理し、メッセージキューの破壊による異常を防いでいます。そのため、生成時にはメッセージの最大数を指定し、格納できるだけのメッセージヘッダを確保します。

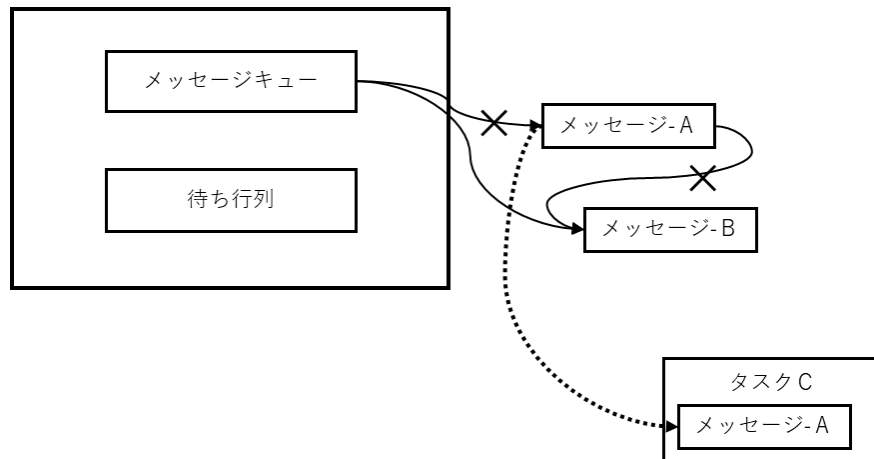
メールボックスは、メッセージヘッダを使用したリスト構造でメッセージを管理します。つまり、以下のように、メッセージヘッダによりリンクリストを構成します。本 RTOS では、メッセージパッケージのアドレスをメッセージヘッダに格納するだけで、メッセージ本体にはアクセスしません。（以降では、メッセージヘッダを省略した簡略化した図式で説明します。）



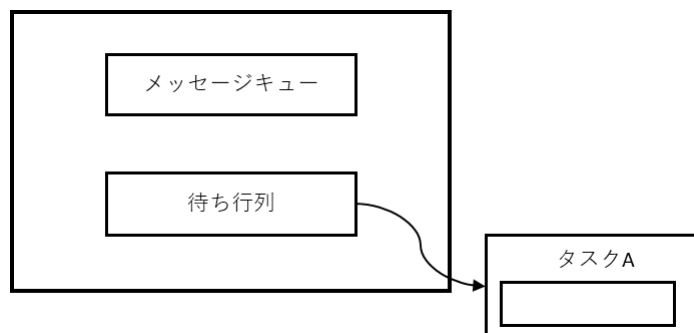
メッセージを送信する際に、メッセージの受信待ちタスクが存在しない場合は、メッセージが FIFO 順の場合はメッセージ最後尾に接続し、メッセージ優先度順の場合は待ち行列メッセージの優先度が昇順（高優先度が先頭、同優先度は後方）になるように接続します。



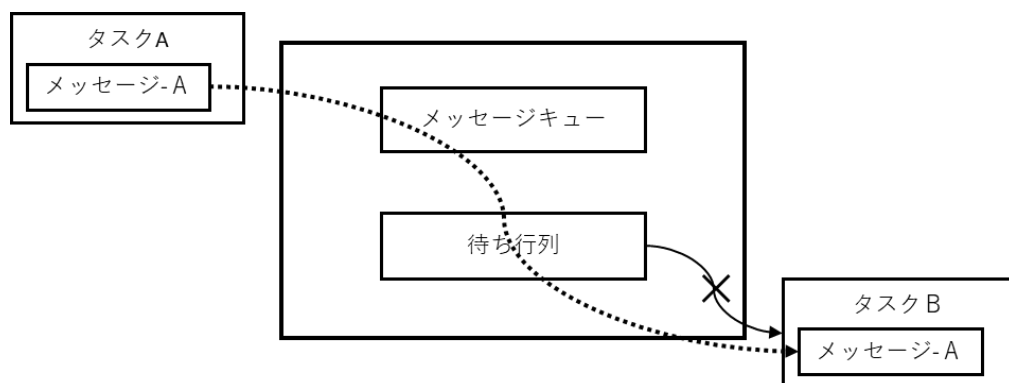
メッセージを受信する際に、メッセージが存在する場合は、常にメッセージキューの先頭メッセージを受信します。



メッセージを受信する際に、メッセージが存在しない場合は、受信待ち行列に接続し、受信待ち状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



メッセージを送信する際に、メッセージの受信待ちタスクが存在する場合は、送信メッセージを受信待ちタスクへ渡した後、受信待ちタスクの受信待ち状態を解除し、実行可能状態に遷移します。



3-6. ミューテックス拡張同期・通信機能

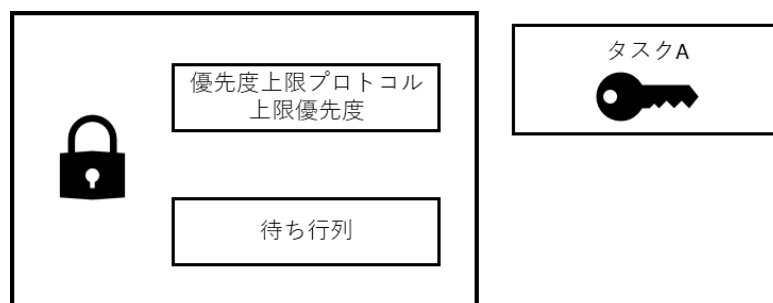
ミューテックスとは、排他制御専用のオブジェクトです。基本的には、初期値が 1 のセマフォを排他制御に用いる方法と同じですが、優先度逆転を防ぐための機構やタスクの終了とともにロックしていたミューテックスをロック解除する機構が備わっています。

優先度逆転を防ぐための機構が、ロックをする可能性のあるタスクのベース優先度で一番高い優先度を上限として設定する優先度上限プロトコルと、ロック中のタスクの優先度よりも高いタスクがロックを試みた場合にロック中のタスク優先度を上昇させる優先度継承プロトコルです。

このミューテックスは、ロックかロック解除かの状態と、ロック待ちタスクの待ち行列と、優先度上限プロトコルでの上限優先度とを持ち、これらにより管理します。また、ロック待ち行列に、FIFO 順かタスク優先度順かを、さらに優先度上限プロトコルか、優先度継承プロトコルかを指定して生成します。優先度上限プロトコルと優先度継承プロトコルの待ち行列はタスク優先度順になります。

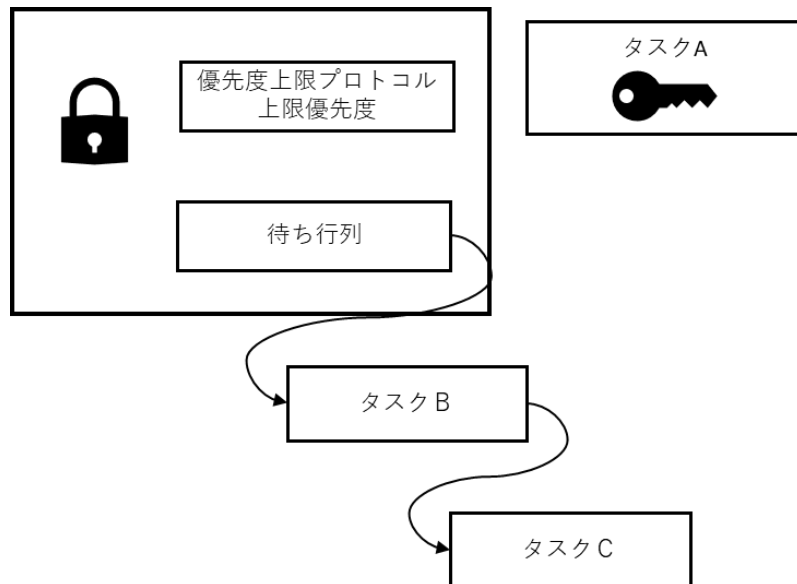


ミューテックスの初期状態は、ロック解除状態です。タスクがロックすると、ミューテックスはロック状態となり、タスクはロックした情報を内部に保持します。また、複数のミューテックスをロックすることもでき、その場合もロックした数だけの情報を保持します。



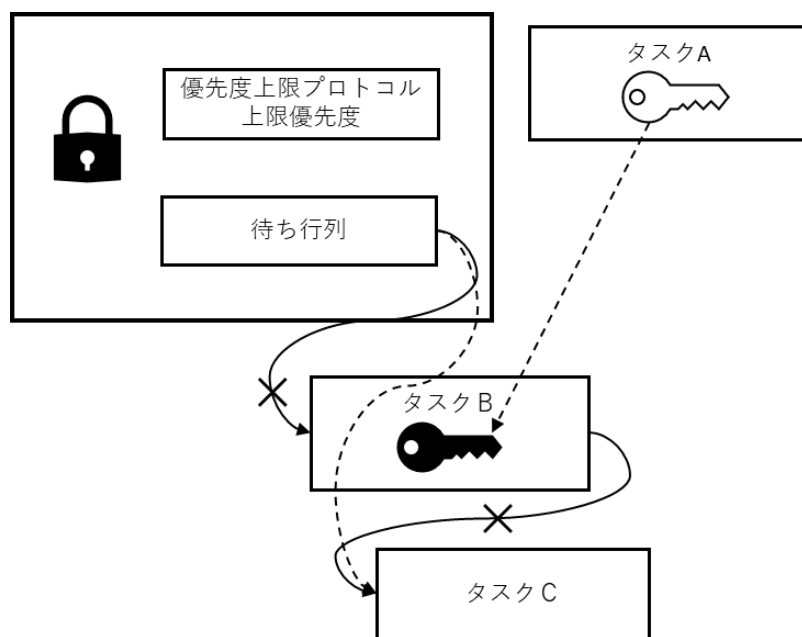
このとき、優先度上限プロトコルの場合は、設定された上限優先度まで現在優先度を上昇させます。

ロックされたミューテックスをロックしようとした場合には、ロックを試みたタスクはロック待ち行列に接続されて、ミューテックスのロック待ち状態に遷移します。

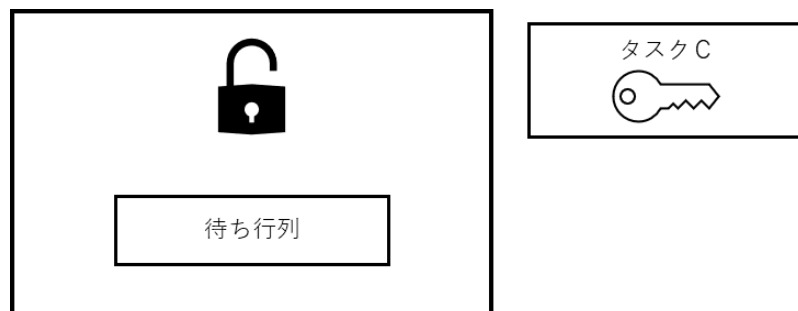


このとき、優先度継承プロトコルの場合は、どのロック待ちタスクの現在優先度よりも同じか高くなるように、ロック中のタスクの現在優先度を上昇させます。

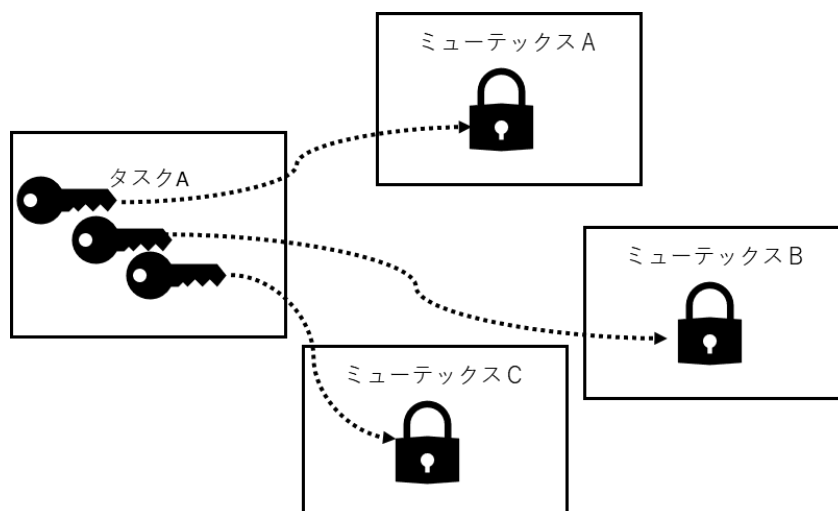
ミューテックスをロック中のタスクがロック解除した場合は、タスク内に保持していたロック情報を削除します。また、待ち行列にタスクが接続されている場合は、最優先順位のタスクにロックさせ、そのタスクがロック情報を保持します。



ミューテックスをロック解除し、さらにロック待ち行列に接続されていない場合は、ミューテックスはロック解除状態になります。



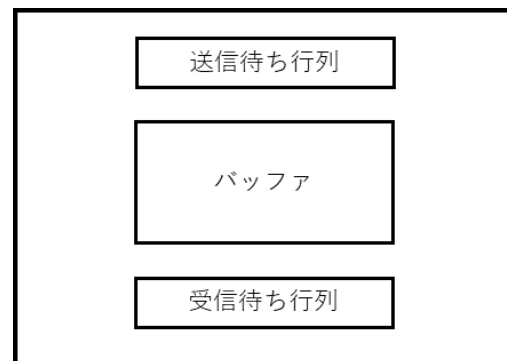
タスク終了時に一つ以上のミューテックスをロックしている可能性があり、その場合は保持していたロック情報を元にしてすべてのミューテックスを解除します。



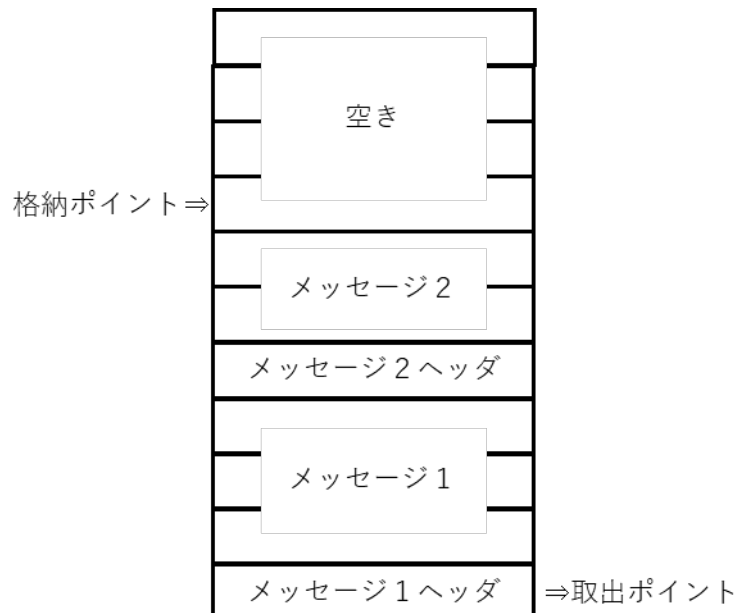
3-7. メッセージバッファ拡張同期・通信機能

メッセージバッファは、可変長サイズのメッセージを受け渡しすることで、同期と通信を行うためのオブジェクトです。このメッセージバッファは、メッセージを蓄えるバッファと、送信と受信待ちタスクの2つの待ち行列を持ち、これらにより管理します。

メッセージバッファは、送信待ち行列に FIFO 順かタスク優先度順かの指定と、メッセージを蓄えるバッファサイズを指定して生成します。

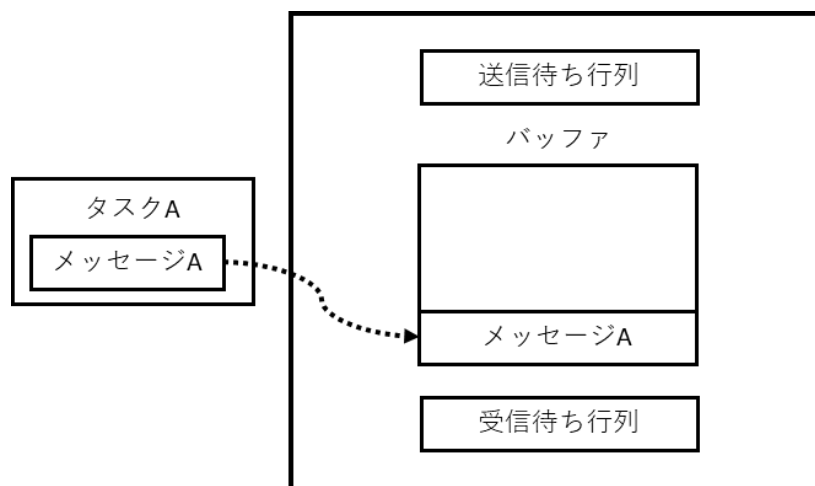


メッセージバッファのバッファ部は、メッセージの前に管理領域となる0バイト以上のメッセージヘッダを付加し、メッセージをバイトで管理しています。

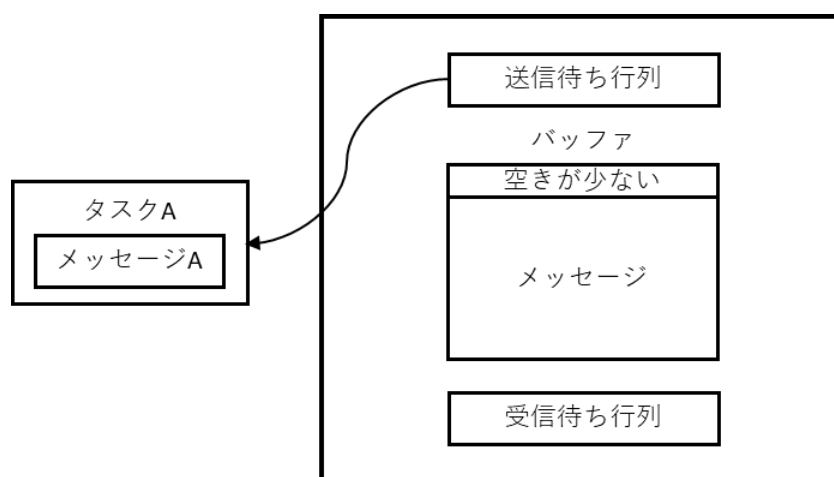


3-7-1. メッセージバッファへ送信する場合

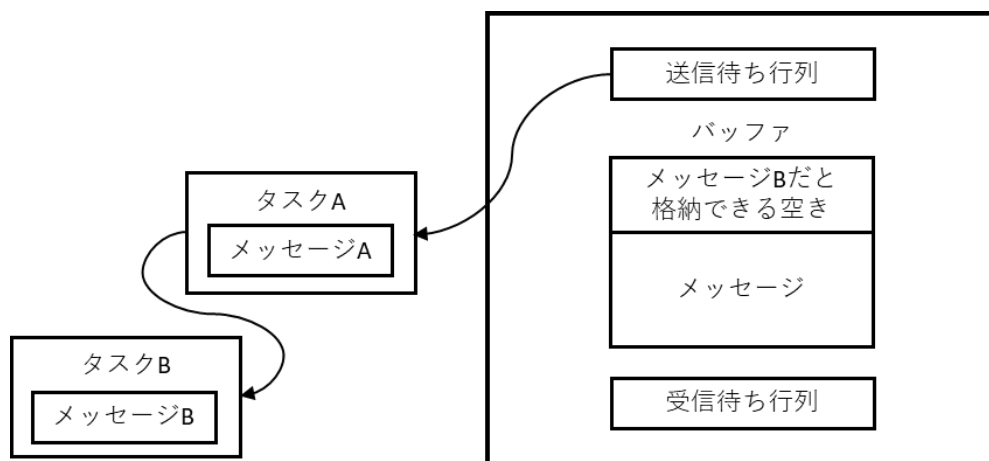
受信待ち行列にタスクが接続されておらずバッファに空きがある場合は、データをバッファに格納します。



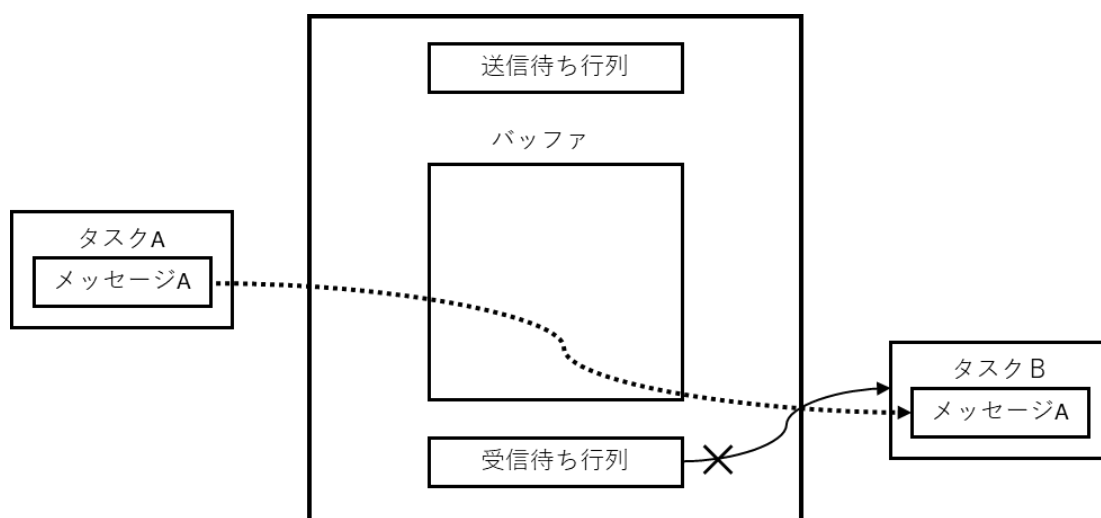
受信待ち行列にタスクが接続されておらず、バッファに空きもない場合は、送信待ち行列に接続し、メッセージバッファの送信待ち状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



送信待ち行列にタスクが接続されている場合は、メッセージをバッファに送信できるか否かに関わらず、必ず送信待ち行列に接続し、メッセージバッファの送信待ち状態に遷移します。

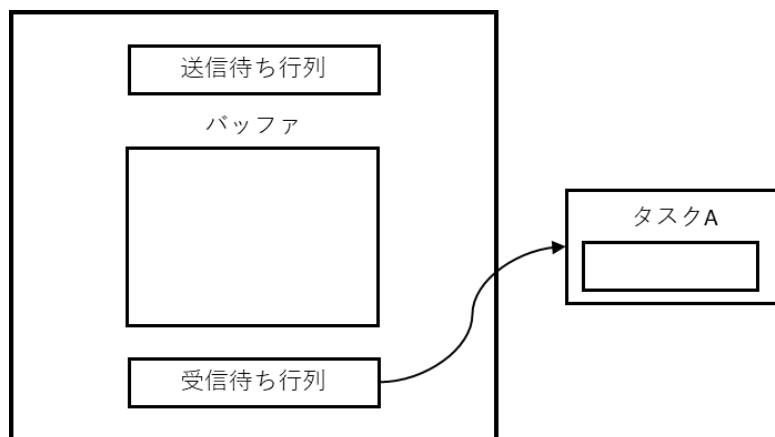


受信待ち行列にタスクが接続されている場合は、受信待ちタスクにバッファを経由せず直接メッセージを渡して待ち状態を解除し、実行可能状態に遷移させます。

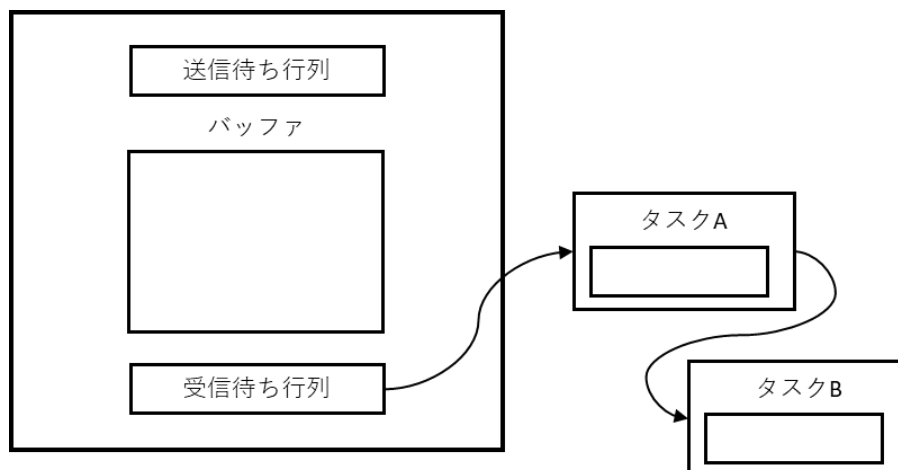


3-7-2. メッセージバッファから受信する場合

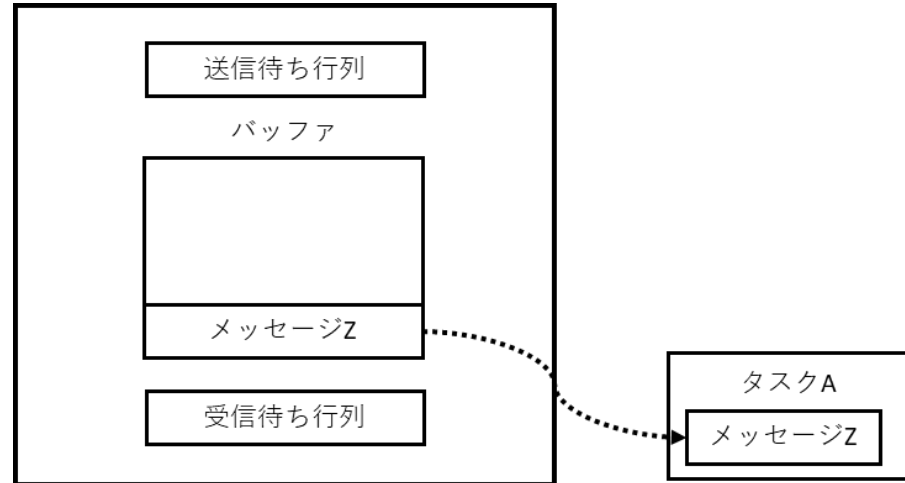
受信待ち行列にも送信待ち行列にもタスクが接続されておらず、バッファ内にメッセージがない場合は、受信待ち行列に接続し、メッセージバッファの受信待ち状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



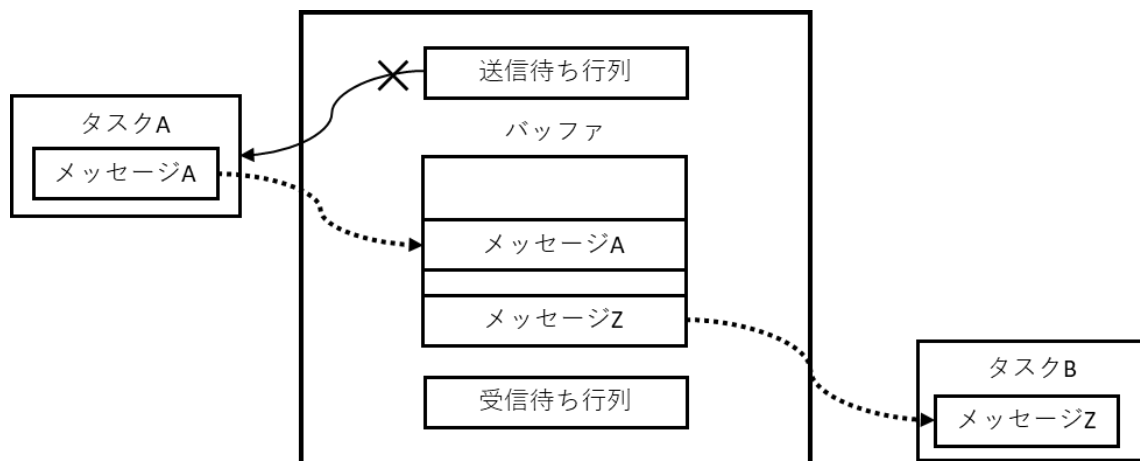
受信待ち行列にタスクが接続されている場合は、送信待ち行列の状態もバッファ内のメッセージにも関係なく、必ず、受信待ち行列に接続し、メッセージバッファの受信待ち状態に遷移します。



受信待ち行列にタスクが接続されておらず、バッファに一つ以上のメッセージがある場合は、タスクは一番古いデータをバッファから取り出します。



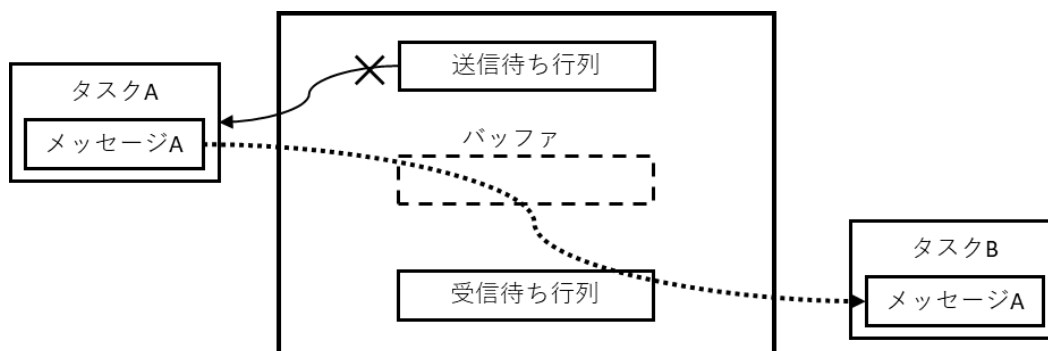
バッファを受信した後、送信待ち行列にタスクが接続されていた場合は、送信待ちタスクのメッセージがバッファに送信できるか否かをチェックします。可能な場合はメッセージをバッファに送信し、送信待ちを解除した後、実行可能状態に遷移させます。



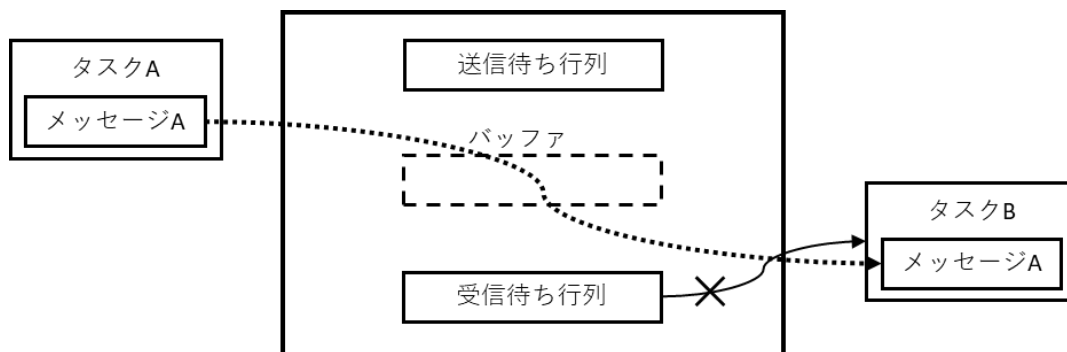
3-7-3. メッセージバッファの同期通信機能の場合

メッセージバッファの送信と受信とを同じタイミングで行う同期メッセージ機能があります。この機能を使う場合は、メッセージを蓄えるバッファのサイズを 0 として生成します。

タスク A が先に送信を試みた場合は、送信待ち行列に接続され、メッセージバッファの送信待ち状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。その後、タスク B が受信を試みると、タスク A が送信しようとしたメッセージを受け取り、タスク A の待ち状態を解除し、実行可能状態に遷移します。



逆に、タスク B が先に受信を試みた場合は、受信待ち行列に接続され、メッセージバッファの受信待ち状態に遷移します。その後、タスク A が送信を試みると、送信メッセージをタスク B に渡し、タスク B の待ち状態を解除し、実行可能状態に遷移します。

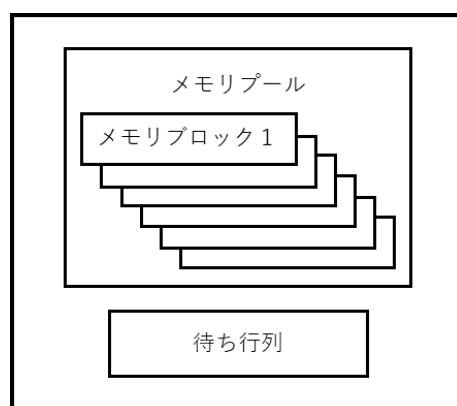


このようにして、送信と受信タスクが同時に、実行状態、或いは実行可能状態に遷移します。このとき、実行状態に遷移するのは、タスク優先順位の高い方のタスクです。

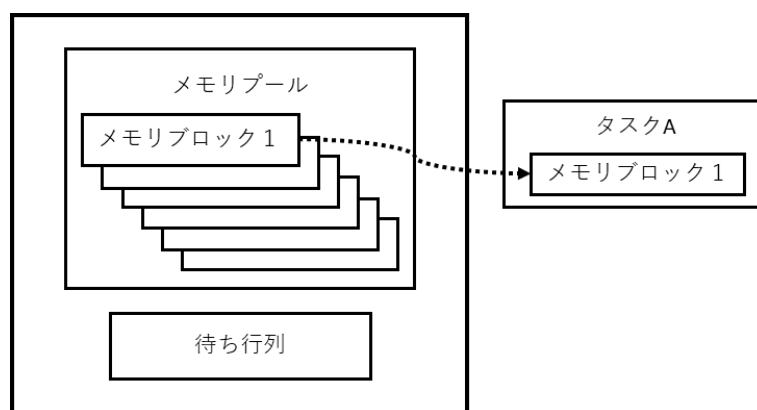
3-8. 固定長メモリプール機能

固定長メモリプールは、指定されたサイズのメモリブロックを、指定された個数を定義し、動的に獲得と返却を行うオブジェクトです。メモリブロックのサイズにアライメント境界は考慮せず、1 バイト以上の任意のサイズを指定できます。そのため、メモリブロックの番地をアライメント境界に揃えたい場合は、サイズにアライメントの倍数を指定しなければいけません。

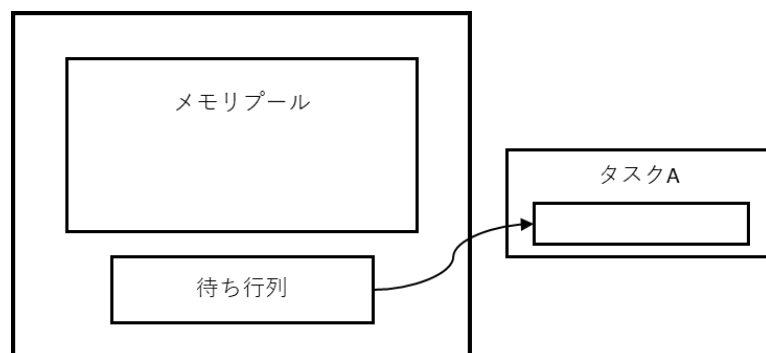
この固定長メモリプールは、メモリブロックを保管するメモリプールと、メモリブロックの獲得待ちタスクの待ち行列を持ち、これらにより管理します。また、獲得待ち行列に **FIFO** 順かタスク優先度順かを指定して生成します。



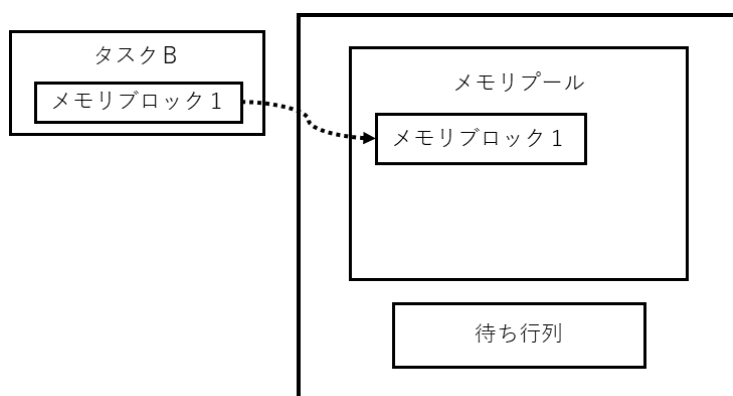
メモリブロックを獲得する際に、メモリプールにメモリブロックが存在する場合は、メモリブロックを取り出します。



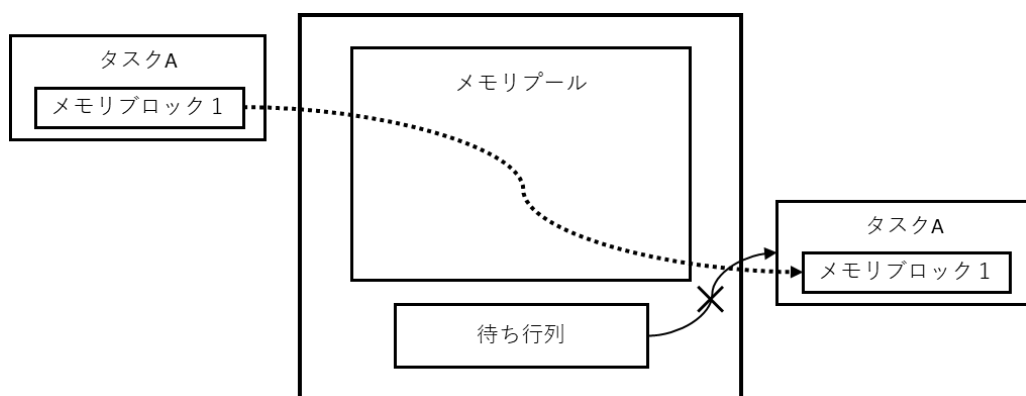
メモリブロックを獲得する際に、メモリプールにメモリブロックが存在しない場合は、獲得待ち行列に接続し、獲得待ち状態に遷移します。待ち行列への接続は、生成時の定義によって **FIFO** 順かタスク優先度順かが生成時の定義によって決まります。



メモリブロックを返却する際に、メモリブロックの獲得待ち行列にタスクが存在しない場合は、メモリプールに返却します。



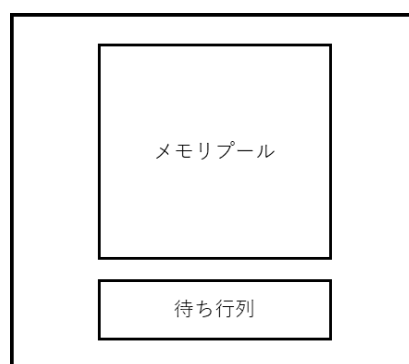
メモリブロックを返却する際に、メモリブロックの獲得待ち行列にタスクが存在する場合は、メモリブロックを獲得待ちタスクへ渡した後、獲得待ちタスクの獲得待ち状態を解除し、実行可能状態に遷移させます。



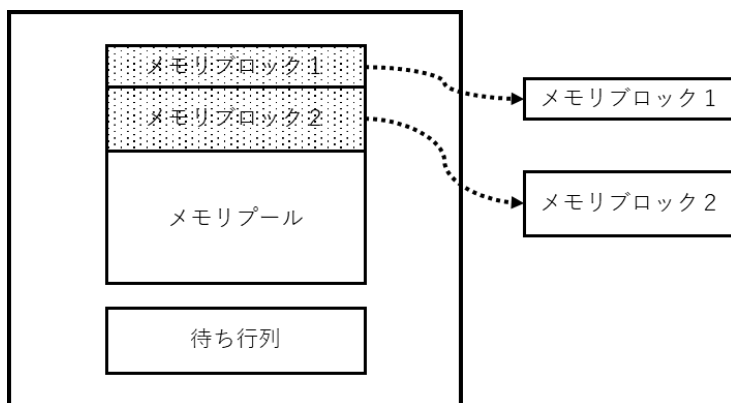
3-9. 可変長メモリプール機能

可変長メモリプールは、指定されたサイズのメモリ領域をメモリプールとして、指定されたサイズのメモリブロックを動的に獲得と返却を行うオブジェクトです。また、実際に獲得されるメモリブロックには、アライメント境界や管理領域が存在するため、要求サイズよりも多くのサイズが割当てられます。また、獲得待ち行列に **FIFO** 順かタスク優先度順かを指定して生成します。

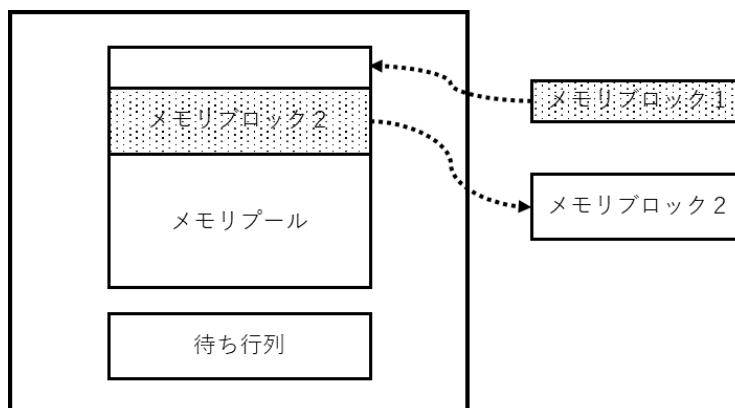
この可変長メモリプールは、割当てられるメモリブロックの元となるメモリプールと、メモリブロックの獲得待ちタスクの待ち行列を持ち、これらにより管理します。



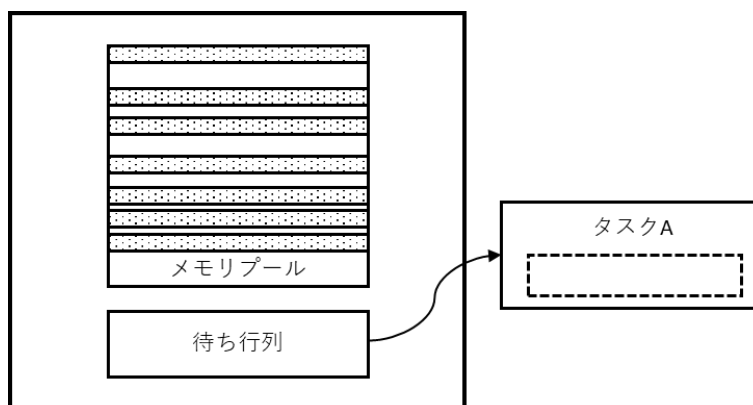
メモリブロックを獲得する際に、メモリプールにメモリブロックが存在する場合は、メモリブロックを取り出します。



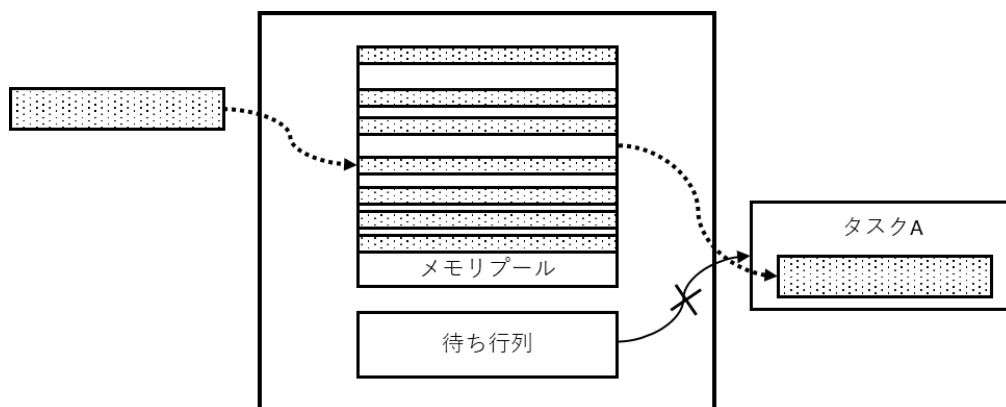
本 RTOS では、メモリブロックを獲得に、メモリプールの空き領域の先頭番地（若い番地）から順次割当てます。メモリブロックを返却した際には、空き領域に戻されます。



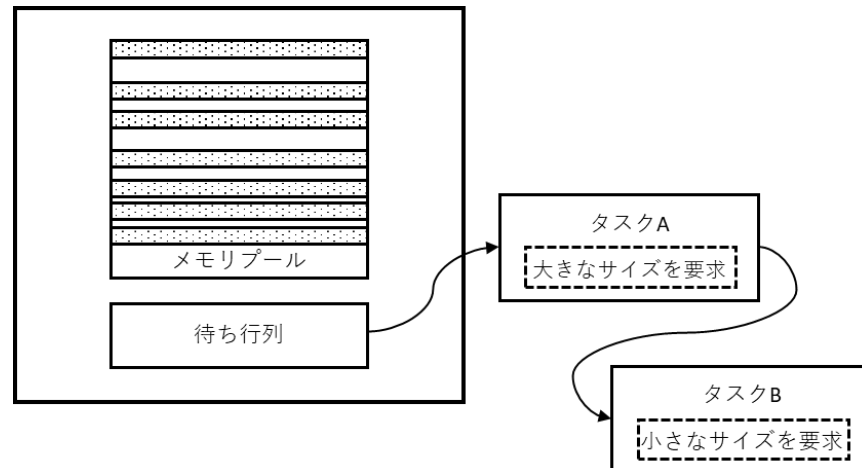
獲得と返却を繰り返すと、メモリプールの空気が無い、或いは、断片化が進み、要求するサイズのメモリブロックを割当てることができない状態になることがあります。このような場合には、要求するタスクを獲得待ち行列に接続し、獲得待ち状態に遷移します。待ち行列への接続は、生成時の定義によって FIFO 順かタスク優先度順かが生成時の定義によって決まります。



メモリブロックを返却した後、獲得待ちタスクの要求するサイズの空きメモリブロックができた場合、メモリブロックを獲得待ちタスクへ渡した後、獲得待ちタスクの獲得待ち状態を解除し、実行可能状態に遷移させます。



獲得待ちタスクが要求しているメモリブロックのサイズはいろいろあります。その結果、優先順位高いタスク A が要求しているメモリブロックのサイズは大きく、優先順位の低いタスク B が要求しているメモリブロックのサイズは小さい、このような状況が考えられます。この場合、タスク A がメモリブロックを獲得するまでは、タスク B が要求するメモリブロックの有無に関わらず待ち状態を続けます。



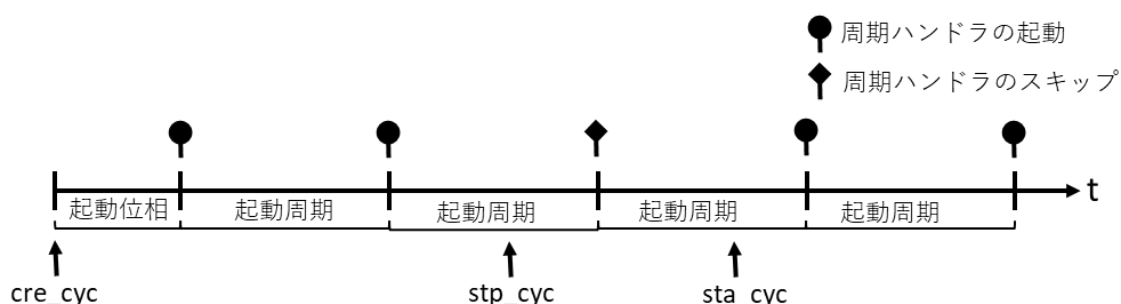
メモリプールから獲得されたメモリブロックは、そのメモリブロックの番地やサイズを使用中として管理します。本 RTOS では、ユーザによる管理領域の破壊から保護するため、RTOS のみがアクセスできるシステムメモリ領域にメモリプールとは分離して管理領域を配置します。そのため、獲得可能なメモリブロックの最大数の管理領域を生成時に予め確保します。

3-10. 周期ハンドラ機能

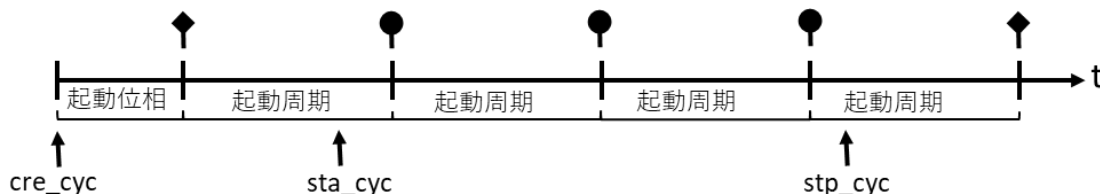
周期ハンドラは、一定の周期で起動するタイムイベントハンドラです。また、周期ハンドラはタイムチックに同期して起動します。起動タイミングは、周期ハンドラを生成時に起動すべき起動位相を保存するか否か、起動状態で生成するか否かによって異なります。

3-10-1. 起動位相を保存する場合

起動状態で生成した場合は、生成から起動位相の後に起動が始まり、動作停止から動作開始までは周期ハンドラは起動しません。ただし、起動周期が保存されているため、動作開始時刻には影響されずに、起動周期を保ちます。

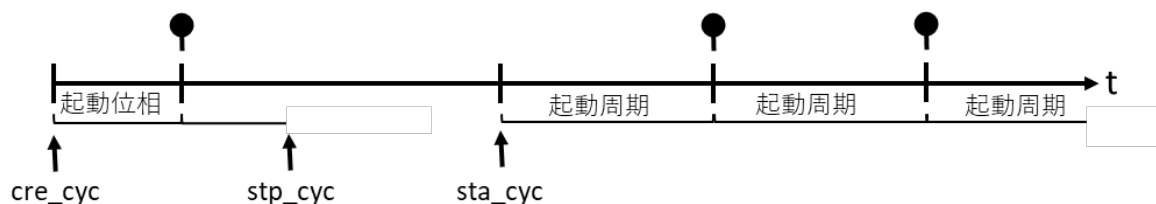


停止状態で生成した場合は、起動タイミングでも起動しないだけで、動作開始しても起動周期を保ったまま次のタイミングから起動します。

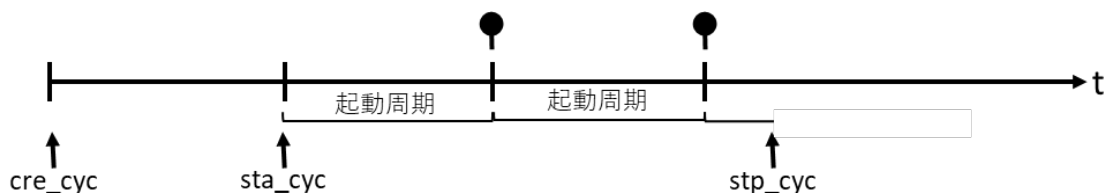


3-10-2. 起動位相を保存しない場合

起動状態で生成した場合は、生成から起動位相の後に起動が始まります。ただし、一旦、動作を停止し、再び動作開始にすると、動作開始を新たな起動周期として動作します。



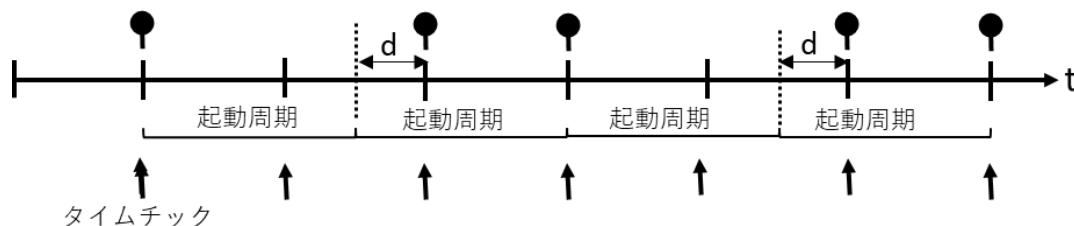
停止状態で生成した場合は、起動タイミングでも起動しませんが、起動位相も無効になり、動作開始を起動周期として起動を始めます。また、一旦、動作を停止し、再び動作開始にした場合も、動作開始を起動周期として起動を始めます。



3-10-3. 起動位相とタイムチェックの関係

周期ハンドラはタイムチェックに同期して起動するため、起動周期がタイムチェック周期の倍数でない場合は、時間通りには起動されません。

例えば、タイムチェックの周期が 10 ミリ秒で、起動周期を 15 ミリ秒とした場合です。このような場合には、起動時刻後のタイムチェックで周期ハンドラは起動されます。次のように、誤差は累積しませんが、5 ミリ秒 (=d) の誤差が発生します。



第 4 章 カーネルと RTOS の特徴と注意事項

本カーネルと本 RTOS を組み合わせた機能の特徴と、これを使用する上で重要な注意事項について説明します。

4-1. 時間の管理方法

本カーネルの場合、システム時刻を基にして各種の管理を行いますが、システム時刻の更新管理はセキュア層のみが行います。そのため、タイムチック割込みはセキュア層で受付けてタイムチックを供給し、最低優先度の割込み処理でシステム時刻を更新します。また、必要に応じてタイムイベントハンドラも処理します。

具体的には、この最低優先度の割込みは Cortex-M の PendSV 例外で実現しています。

4-1-1. システム時刻の更新

セキュア層でタイムチック割込みは受付けるが最低優先度の割込みは受け取らないよう割込みマスク優先度を変更している場合は、システム時刻の更新要求をキューイングします。そして、割込みマスク優先度を解除すると、一度に処理して本来のシステム時刻に更新されます。

システム時刻の更新はセキュア層で行うため、セキュア層の割込みを禁止している場合はシステム時刻は更新しませんが、非セキュア層の割込みを禁止していてもシステム時刻は更新されます。

4-1-2. タイムイベントハンドラの実行

セキュア層の最低優先度の割込みハンドラでシステム時刻を更新しますが、セキュア層のタイムイベントハンドラも同時に処理します。一方で、非セキュア層の最低優先度の割込みハンドラでは、非セキュア層のタイムイベントハンドラを処理します。

セキュア層でタイムチック割込みは受付けるが最低優先度の割込みは受け取らないよう割込みマスク優先度を変更している状態、或いは非セキュア層の最低優先度の割込みは受け取らない状態の場合、その状態を解除すると抑止されていたタイムイベントハンドラが一度に処理されます。つまり、複数回のタイムイベントハンドラが抑止されていた場合は、その回数分を連続して繰返します。

4-2. 割込み禁止／許可と割込みマスク優先度の変更

本 RTOS の割込みハンドラは、RTOS を介在せずに実行されます。そのため、割込み禁止にした状態や、割込みマスク優先度を変更した状態のまま、割込みハンドラを終了した場合は、RTOS は認識することができません。その結果、本来のスケジューリングに応じた

ディスパッチャが行えずに、タスク優先度の逆転など異常な状態に陥る危険性があります。このような状態にならないように、割込みハンドラで変更した状態は、必ず元の状態に戻して下さい

4-3. タスクのスタック領域

セキュアタスクも非セキュアタスクもセキュア間移行が行えるため、異なるセキュア用のタスクのスタック領域を用意します。非セキュアタスクは、タスク生成時にセキュアスタック領域が確保されます。これは、本 RTOS の仕様で、最小値 512 バイトが必須のスタック容量となり、セキュアライブラリによるスタック消費量を考慮してスタックサイズを加算することができます。

4-4. アイドル関数

非セキュア層にアイドル関数は定義できません

第 5 章 RTOS のコンフィグレーションと起動

コンフィグレーションには、RTOS に関する情報、メモリ領域に関する情報、アーキテクチャに依存する情報などがあります。これらを定義した後、RTOS の内部データを初期化する初期化関数の呼出し、そして起動関数の呼出しにより、ユーザプログラムであるタスクの起動が始まります。これらのコンフィグレーションと起動は、セキュア層と非セキュア層との RTOS 毎に行います。

5-1. コンフィグレーションのデータ作成

コンフィグレーションに使用するデータは `config.h` ファイルに記述します。データには以下の共通部コンフィグレーションと、デバイスに依存するコンフィグレーションとがあります。デバイス依存のコンフィグレーションは、各「デバイス依存部ユーザズガイド」を参照してください。

オブジェクト生成情報

MAX_TASK	タスク ID の最大値 (1～32767)
RSV_TASK	タスク ID 自動割付けの除外数 (0～32767)
MAX_SEM	セマフォ ID の最大値 (0～32767)
RSV_SEM	セマフォ ID 自動割付けの除外数 (0～32767)
MAX_EFLG	イベントフラグ ID の最大値 (0～32767)
RSV_EFLG	イベントフラグ ID 自動割付けの除外数 (0～32767)
MAX_MBX	メールボックス ID の最大値 (0～32767)
RSV_MBX	メールボックス ID 自動割付けの除外数 (0～32767)
MAX_DTQ	データキューID の最大値 (0～32767)
RSV_DTQ	データキューID 自動割付けの除外数 (0～32767)
MAX_MTX	ミューテックス ID の最大値 (0～32767)
RSV_MTX	ミューテックス ID 自動割付けの除外数 (0～32767)
MAX_MBF	メッセージバッファ ID の最大値 (0～32767)
RSV_MBF	メッセージバッファ ID 自動割付けの除外数 (0～32767)
MAX_MPF	固定長メモリプール ID の最大値 (0～32767)
RSV_MPF	固定長メモリプール ID 自動割付けの除外数 (0～32767)
MAX_MPL	可変長メモリプール ID の最大値 (0～32767)
RSV_MPL	可変長メモリプール ID 自動割付けの除外数 (0～32767)
MAX_CYC	周期ハンドラ ID の最大値 (0～32767)
RSV_CYC	周期ハンドラ ID 自動割付けの除外数 (0～32767)
MAX_ISR	割込みサービスルーチン ID の最大値 (0～32767)

RSV_ISR	割込みサービスルーチン ID 自動割付けの除外数 (0~32767)
INI_FPSCR	FPSCR レジスタの初期値 (32 ビット)
システム構成情報	
MAX_TPRI	タスク優先度の最大値 (1~128)
CLK_MSEC	タイムチックの間隔 (μ 秒単位)
DSP_PRI	最低割込み優先度
SYS_DOWN	システムダウン関数の番地 (固定値: 0)
SYS_IDLE	アイドル関数の番地 (デフォルトの場合 0)
ユーザーメモリ管理情報	
USR_MEM_ADDR	ユーザーメモリ領域の先頭番地
USR_MEM_SIZE	ユーザーメモリ領域のサイズ (バイト単位)

1. ID の最大値とは、そのオブジェクトを生成可能な個数です。
2. ID 自動割付けの除外数とは、ID 指定で生成するオブジェクトの個数です。
例えば、ID 最大値を 20 に、ID 自動割付けの除外数を 10 に指定した場合には、
`acre_xxx0`関数で ID=11~20 までは生成されます。もし、ID=1~10 に未生成があっても、ID=11~20 が生成済みであれば `E_NOID` エラーが返ります。
3. FPSCR レジスタの初期値とは、浮動小数点ユニットが有効に変化した際の初期値になります。浮動小数点ユニットを実装していないプロセッサの場合は、0 を設定します。
4. タスク優先度の最大値は 128 ですが、必要な数だけに抑えることを推奨します。
5. タイムチックの間隔を μ 秒単位で設定します。本 RTOS の設定単位がミリ秒のため、必ず 1000 の倍数になります。また、非セキュア層では無視されます。
6. 最低割込み優先度は、セキュア層では 0x7F、非セキュア層では 0xFF が定値です。
7. システムダウン関数は、本 RTOS では未使用になっています。
8. アイドル関数は、本 RTOS がアイドル状態になった場合に呼出す関数です。デフォルトでは、スリープに遷移する関数になっています。また、非セキュア層では無視されます。

5-2. RTOS が必要とするメモリ領域

RTOS が必要とする専用のメモリ領域があります。大きく分類すると、非タスクコンテキストが使用するスタック領域、各種制御ブロックのデータ領域、オブジェクト生成時に動的に使用するメモリ領域があります。

1. システムスタックメモリ領域

非タスクコンテキスト、具体的には、割込みハンドラ、割込みサービスルーチン、周期ハンドラが使用するスタック領域です。割込みハンドラは多重に発生するので、それらを賄えるだけのサイズが必要になります。その他、非セキュア層にはアイドル関数専用のスタック領域もあります。

2. システムメモリ領域

各種制御ブロックとして使用されるシステムデータ領域で、RTOS のみがアクセスし、アプリケーションプログラムではアクセスしない領域です。これには、コンフィグレーションで指定されるオブジェクトの数に依存した初期化で使用される領域と、オブジェクトの生成時に獲得される領域とがあります。生成時に獲得された領域は、オブジェクトの削除時に返却されます。また、可変長メモリプールのメモリブロック管理領域、メールボックスのメッセージ管理領域、データキューのリングバッファ領域、メッセージバッファのバッファ領域もオブジェクト生成時に獲得されます。

3. ユーザーメモリ領域

アプリケーションプログラムからアクセスされる領域です。この領域は、固定長／可変長メモリプールのメモリブロック本体の領域、タスク生成時のスタック領域で、オブジェクト生成時にメモリ領域を指定しない場合に使用されます。これらのオブジェクト生成時に獲得された領域は、オブジェクトの削除時に返却されます。ユーザーメモリ領域は、セキュア層と非セキュア層とが独立して管理されます。

5-3. 初期化から起動までの手順

本 RTOS を起動し、タスクを起動するまでは決められた手順があります。最初に初期化関数を呼び、タスクや割込みハンドラを必要に応じて生成し、最後に RTOS を起動します。

```
#include "kernel.h"

extern T_CSYS pk_csys;
ID MainTaskID = 0;
void MainTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_PRIV|TA_ACT, 0, MainTask,
                    2, 0x2000, NULL, "MainTask" };

int main(void)
{
    ER ret;

    /* Initialize kernel */
    ret = ini_krn(&pk_csys);
    if (ret == E_OK) {

        /* Create main task */
        MainTaskID = acre_tsk(&ctsk_main);
        if (MainTaskID > 0) {
            /* Start kernel */
            sta_krn();
        }
    }
    return ret;
}
```

上記が、一般的な起動方法となり、環境に応じた処理を追加します。

ini_krn()から sta_krn()まで RTOS は起動していませんが、本 RTOS の過渡状態であり、CPU ロック状態で呼出することができるシステムコールを呼出することができます。つまり、最初に起動したいタスクなどの生成や起動することができます。

5-3-1. RTOS の初期化

書式	ER ercd = ini_krn(T_CSYS *pk_csys);		
引数	T_CSYS	pk_csys	RTOS の生成情報を格納した パケットへのポインタ
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラー コード	E_PAR	パラメータエラー	
	E_NOMEM	メモリ不足	

コンフィグレーションファイル config.h より生成したデータ構造体を元にし、本 RTOS を初期化します。この構造体名 pk_csys は config.c により生成されます。

5-3-2. RTOS の起動

書式	void = sta_krn(void);
----	-----------------------

タスクスケジューラが起動を始め、生成されたタスクが起動し、定義／生成された割込みを受け付け、システムの動作が始まります。

第 6 章 各種管理機能とシステムコール

本 RTOS のシステム全体や各オブジェクトを操作または参照する各機能に応じたシステムコールがあります。

セキュア層、或いは非セキュア層内だけで使用される μ ITRON4.0 仕様で定義された以下の形式でシステムコールが定義されています。

```
xxx_yyy0
ixxx_yyy0
```

本 RTOS では、一部で機能を拡張した場合や、呼出し方法を変更した場合は、以下の形式でシステムコールを定義しています。

```
xxx_yyy20
```

【例】

タスクの起動は、セキュア層、或いは非セキュア層内でタスクを起動させるシステムコールは以下のように定義されています。

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

6-1. タスク管理機能

タスク管理機能には、以下の機能があります。

- タスクの生成と削除
- タスクの起動と終了
- タスクの起動要求のキャンセル
- タスクの優先度の変更と参照
- タスクの状態の参照

タスクは次の形式で記述します。

```
void task(VP_INT exinf)
{
    タスクで処理すべきコードを記述
    ext_tsk0;
}
```

タスク関数から戻ると自動的に `ext_tsk0` が呼出され、タスクは終了します。そのため、タスク関数の最後に記述する `ext_tsk0` を省略することができます。また、終了だけではなく、終了と削除を行う場合には、`exd_tsk0` を呼出します。

【コーディング例】

タスク管理機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_PRIV|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub = { TA_HLNG, 0, SubTask, 3, 0x2000, NULL, "SubTask" };
T_RTSK rtsk_sub;
ID SubTaskID = 0;
PRI tskpri;

void MainTask(VP_INT stacode)
{
    ER_ID erid;
    UINT cnt;
    ER ret;

    for(cnt = 0; cnt < 10; cnt++) {
        erid = acre_tsk(&ctsk_sub);
        if (erid > 0) {
            SubTaskID = erid;
            chg_pri(TSK_SELF, 4);
            act_tsk(SubTaskID);
            get_pri(SubTaskID, &tskpri);
            ref_tsk(SubTaskID, &rtsk_sub);
            if (rtsk_sub.tskstat == TTS_RDY) {
                ter_tsk(SubTaskID);
                del_tsk(SubTaskID);
            }
        }
    }
}

void SubTask(VP_INT stacode)
{
    chg_pri(MainTaskID, TPRI_INI);
    exd_tsk();
}
```

6-1-1. タスクの生成

書式	ER ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk); ER_ID tskid = acre_tsk(T_CTSK *pk_ctsk);		
引数	ID	tskid	生成対象のタスク ID
	T_CTSK	*pk_ctsk	タスク生成情報を格納したパケットへのポインタ
	pk_ctsk の内容 (T_CTSK 型)		
	ATR	tskatr	タスク属性
	VP_INT	exinf	タスクの拡張情報
	FP	task	タスクの起動番地
	PRI	itskpri	タスクの起動時優先度
	SIZE	stksz	タスクのスタック領域のサイズ (バイト数)
	VP	stk	タスクのスタック領域の先頭番地
	SIZE	sstksz	タスクのセキュアスタック領域のサイズ (バイト数)
	VB const	*name	タスクの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_tsk の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_tsk の場合		
	ER_ID	tskid	生成したタスクの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOID	ID 番号不足 (登録可能なタスク ID がない、acre_tsk のみ)	
	E_NOMEM	メモリ不足	
	E_RSATR	予約属性 (tskatr が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_PAR	パラメータエラー (pk_ctsk、task、itskpri、stksz、stk が不正)	
	E_OBJ	オブジェクト状態エラー (対象タスクが登録済み、cre_tsk のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクを、pk_ctsk で指定されるタスク生成情報を基にして生成します。具体的には、タスク ID で指定される未登録状態のタスクを休止状態または実行可能状態に遷移させます。

tskatrはタスク属性、**exinf**はタスクが起動した際に引数として渡されるパラメータ、**itskpri**はタスク起動時のタスク優先度、**stksz**はスタック領域のサイズ (バイト数)、**stk**はタスクのスタック領域の先頭番地、**task**はタスクの起動番地です。また、**sstksz**は非セキュアタスクのセキュア層用のスタック領域の加算値で、セキュアタスクの場合は無視されます。**acre_tsk**は、生成可能なタスク ID を検索し割付け、そのタスク ID を返します。**tskatr**には、(**TA_HLNG** | [**TA_PRIV**] | [**TA_ACT**]) を指定でき、**TA_ACT**を指定した場合は実行可能状態のタスクを、そうでない場合には休止状態のタスクとして生成します。**TA_PRIV**を指定した場合は特権モードのタスクを、そうでない場合には非特権モードのタスクとして生成します。**stk**指定された番地から、**stksz**で指定されたバイト数の領域をタスクのスタック領域として使います。また、**stk**に **NULL (=0)** を指定した場合には、**RTOS** が管理するユーザーメモリ領域から切り出して割付けます。非セキュアタスクでは、**sstksz**で指定されたバイト数の領域をタスクのセキュア層用のスタック領域の加算値として、セキュア層のユーザーメモリ領域から切り出して割付けます。これは、非セキュアタスクがセキュアライブラリを呼び出す場合に、必要なスタックサイズを確保するためです。この値に **0** を指定すると、セキュア層用のスタック領域は **512** バイトが確保され例外発生時のレジスタ待避領域などに使用されます。**name**で指定するタスクの名称は、ヌル ('¥0') を終端とする **15** 文字までの文字列とし、**16** 文字以上を指定した場合は、**16** 文字目以降は無視されます。**name**に **NULL**を指定することでタスクの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-1-2. タスクの削除

書式	ER ercd = del_tsk(ID tskid);		
引数	ID	tskid	削除対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象タスクが休止状態でない)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskidで指定される ID 番号のタスクを削除します。具体的には、指定される ID 番号のタスクを休止状態から未登録状態に遷移させます。

6-1-3. タスクの起動

書式	ER ercd = act_tsk(ID tskid); ER ercd = iact_tsk(ID tskid);		
引数	ID	tskid	起動対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_QOVR	キューイングオーバフロー (起動要求キューイング数がオーバフロー)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクを起動します。具体的には、指定される ID 番号のタスクが休止状態の場合は、タスクの起動番地から実行できるよう実行可能状態に遷移させます。そのタスクに渡される引数には、タスクの生成時に指定した拡張情報が使われます。休止状態でも未登録状態でもない場合には、対象タスクの起動要求キューイング数を+1します。その際に、キューイング数が最大値を超える場合には E_QOVR エラーを返します。tskid に TSK_SELF (=0) を指定すると起動対象タスクを自タスクとします。

6-1-4. タスク起動要求のキャンセル

書式	ER_UINT actcnt = can_act(ID tskid);		
引数	ID	tskid	起動要求キャンセル対象のタスク ID
戻り値	ER_UINT	actcnt	キューイングされていた起動要求の回数 (0 以上) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクにキューイングされている起動要求をキャンセルし、キューイングされていた起動要求の回数を返します。具体的には、タスクの起動要求キューイング数に0を設定し、0を設定する前の値を返します。can_actの場合、tskidにTSK_SELF (=0) を指定すると起動要求のキャンセルを自タスクとします。

6-1-5. タスクの起動（起動コード指定）

書式	ER ercd = sta_tsk(ID tskid, VP_INT stacd);		
引数	ID	tskid	起動対象のタスク ID
	VP_INT	stacd	タスクの起動コード
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（タスク ID が不正）	
	E_NOEXS	オブジェクト未生成（対象タスクが未登録状態）	
	E_OBJ	オブジェクト状態エラー（対象タスクが休止状態でない）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

tskid で指定される ID 番号のタスクを起動します。具体的には、指定される ID 番号のタスクが休止状態の場合のみに、stacd で指定される起動コードを引数とし、タスクの起動番地から実行できるよう実行可能状態に遷移させます。

6-1-6. 自タスクの終了

書式	void ext_tsk(void);		
戻り値	なし		
エラーコード	なし	システムコールから戻らないため	

自タスクを終了させます。具体的には、自タスクを実行状態から休止状態に遷移させます。ただし、起動要求がキューイングされていた場合には、自タスクの起動要求キューイング数を-1 し、実行可能状態に遷移させます。この際のタスクに渡される引数には、タスクの拡張情報が適用されます。タスクのメイン関数からの戻りで、この関数が自動的に呼出されます。

自タスクの概念がない非タスクコンテキストから呼出された場合には、戻り値、つまりエラーコードなしでシステムコールから戻ります。また、非セキュア移行タスクから呼出された場合も、エラーコードなしで戻ります。

6-1-7. 自タスクの終了と削除

書式	void exd_tsk(void);		
戻り値	なし		
エラーコード	なし	システムコールから戻らないため	

自タスクを終了させた後、削除します。具体的には、自タスクを実行状態から未登録状態に遷移させます。未登録状態にすることで、起動要求キューイング数を破棄され、再び実行可能状態にはなりません。

自タスクの概念がない非タスクコンテキストから呼出された場合には、戻り値、つまりエラーコードなしでシステムコールから戻ります。また、非セキュア移行タスクから呼出された場合も、エラーコードなしで戻ります。

6-1-8. タスクの強制終了

書式	ER ercd = ter_tsk(ID tskid);		
引数	ID	tskid	起動対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_ILUSE	システムコール不正利用 (対象タスクが自タスク、非セキュア移行タスクの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象タスクが休止状態)	

tskid で指定される ID 番号のタスクを強制的に終了させます。具体的には、指定される ID 番号のタスクが実行可能状態または待ち状態の場合のみ、休止状態に遷移させます。起動要求がキューイングされていた場合には、そのタスクの起動要求キューイング数を-1 し、実行可能状態に遷移させます。この際のタスクに渡される引数には、タスクの拡張情報が使われます。

6-1-9. タスク優先度の変更

書式	ER ercd = chg_pri(ID tskid, PRI tskpri);		
引数	ID	tskid	変更対象のタスク ID
	PRI	tskpri	変更後のタスク優先度
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_PAR	パラメータエラー (tskpri が不正)	
	E_OBJ	オブジェクト状態エラー (対象タスクが休止状態)	
	E_ILUSE	システムコール不正利用 (セキュア移行タスク自身を対象とした呼出し、 非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクの優先度を変更します。具体的には、指定される ID 番号のタスクが実行可能状態または待ち状態の場合のみ、tskpri で指定されるタスク優先度に変更します。

tskid に TSK_SELF(=0)を指定すると変更対象タスクを自タスクとし、tskpri に TPRI_INI(=0)を指定すると起動時タスク優先度を変更後のタスク優先度とします。

このシステムコールは、変更対象のタスクが実行状態または実行可能状態の場合は、レディキューからの切り離しと接続を行うため、タスク優先順位が変更される可能性があります。つまり、変更対象タスクは、同一タスク優先度のなかで最低優先順位になります。

6-1-10. タスク優先度の参照

書式	ER ercd = get_pri(ID tskid, PRI *p_tskpri);		
引数	ID	tskid	参照対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	PRI	tskpri	参照対象タスクの現在優先度
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_PAR	パラメータエラー (p_tskpri が不正)	
	E_OBJ	オブジェクト状態エラー (対象タスクが休止状態)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクの現在優先度を参照します。具体的には、指定される ID 番号のタスクが未登録状態でも休止状態でもない場合のみ、tskpri に現在優先度を返します。

tskid に TSK_SELF (=0) を指定すると自タスクの優先度を返します。

6-1-11. タスクの状態参照

書式	ER ercd = ref_tsk(ID tskid, T_RTSK *p_rtsk);		
引数	ID	tskid	参照対象のタスク ID
	T_RTSK	*pk_rtsk	タスク状態を返すパケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
pk_rtsk の内容 (T_RTSK 型)			
	STAT	tskstat	タスク状態
	PRI	tskpri	タスクの現在優先度
	PRI	tskbpri	タスクのベース優先度
	STAT	tskwait	待ち要因
	ID	wobjid	待ち対象オブジェクトの ID 番号
	TMO	lefttmo	タイムアウトするまでの時間
	UINT	actcnt	起動要求キューイング数
	UINT	wupcnt	起床要求キューイング数
	UINT	suscnt	強制待ち要求キューイング数 (常に 0)
	VB	name[16]	タスクの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_PAR	パラメータエラー (pk_rtsk が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクの各種状態を参照し、未登録状態でない場合に限り、pk_rtsk で指定されるパケットに返します。tskid に TSK_SELF (=0) を指定すると、自タスクの情報を返します。

tskstat には、タスクの状態により次の値が返ります。

TTS_RUN	0x01	実行状態
TTS_RDY	0x02	実行可能状態
TTS_WAI	0x04	待ち状態
TTS_DMT	0x10	休止状態

tskpri には現在優先度が、tskbpri にはベース優先度 (本 RTOS では現在優先度と同じ値) が返ります。タスクが休止状態の場合は、共に起動時優先度が返ります。

tskwait には、タスクが待ち状態の場合には次の値が返ります。待ち状態以外の場合には 0

が返ります。

TTW_SLP	0x0001	起床待ち状態
TTW_DLY	0x0002	時間経過待ち状態
TTW_SEM	0x0004	セマフォ資源獲得待ち状態
TTW_FLG	0x0008	イベント待ち状態
TTW_SDTQ	0x0010	データキューへの送信待ち状態
TTW_RDTQ	0x0020	データキューからの受信待ち状態
TTW_MBX	0x0040	メールボックスからの受信待ち状態
TTW_MPF	0x2000	固定長メモリブロックの獲得待ち状態
TTW_MPL	0x4000	可変長メモリブロックの獲得待ち状態

lefttmo には、タスクが時間待ち状態の場合は、タイムアウトまでの時間を返します。タスクが時間待ち以外の待ち状態の場合は、TMO_FEVR (-1) を返し、タスクが待ち状態以外の場合は 0 を返します。

actcnt には起動要求数を、actent には起床要求数を、suscnt には強制待ち要求数を返します。

name には、タスク生成時に指定されたタスクの名称を、指定されなかった場合は自動で生成された名称を返します。

6-1-12. タスクの状態参照（簡易版）

書式	ER ercd = ref_tst(ID tskid, T_RTST *p_rtst);		
引数	ID	tskid	参照対象のタスク ID
	T_RTST	*pk_rtst	タスク状態を返すパケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
pk_rtst の内容 (T_RTST 型)			
	STAT	tskstat	タスク状態
	STAT	tskwait	待ち要因
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_PAR	パラメータエラー (pk_rtst が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクの各種状態を参照し、未登録状態でない場合に限り、pk_rtst で指定されるパケットに返します。tskid に TSK_SELF (=0) を指定すると、自タスクの情報を返します。

このシステムコールは、`ref_tsk` の簡易版で、`tskstat` と `tskwait` で返す値は `ref_tsk0` と同じです。

6-1-13. セキュアタスクの非セキュアスタック定義

書式	ER ercd = vset_psp(ID tskid, SIZE stksz);		
引数	ID	tskid	定義対象のタスク ID
	SIZE	stksz	非セキュアスタック領域のサイズ (バイト数)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOMEM	メモリ不足	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_CTX	コンテキストエラー (非タスクコンテキスト)	
	E_OBJ	オブジェクト状態エラー	
	E_SYS	(対象タスクが休止状態でない、自タスクではない、 非セキュアスタックポインタが初期状態でない) システムエラー (非セキュアが起動していない)	

`tskid` で指定される ID 番号のタスクの非セキュアスタックを定義します。`stksz` で指定されたバイト数の領域を非セキュアスタック領域として、非セキュアのユーザーメモリ領域から切り出して割付けます。指定できるタスクは休止状態か、`TSK_SELF` (=0) を指定した非セキュアスタックポインタが初期値の自タスクのみです。

また、既に非セキュアスタックを定義した状態でも、非セキュアスタックポインタが初期値の場合、つまり、スタック領域にデータが存在しない場合は、指定されたサイズの非セキュアスタックに変更されます。

6-2. タスク付属同期機能

タスク付属同期機能は、他のオブジェクトの関与なしにタスクのみで可能な同期機能で、以下の機能があります。

- タスクの起床待ちと起床
- タスクの起床要求のキャンセル
- タスクの待ち状態の強制解除
- タスクの強制待ち状態への移行と解除
- 自タスクの遅延

【コーディング例】

タスク付属同期機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub = { TA_HLNG|TA_ACT, 0, SubTask, 3, 0x2000, NULL, "SubTask" };

void MainTask(VP_INT stacode)
{
    ER_UINT wupcnt;
    ER ret;

    for(;;) {
        wupcnt = can_wup(TSK_SELF);
        if (wupcnt >= 0) {
            ret = slp_tsk();
            if (ret == E_OK) {
                dly_tsk(50);
                wup_tsk(SubTaskID);
            } else if (ret == E_RLWAI) {
                dly_tsk(50);
                rel_wai(SubTaskID);
                break;
            } else {
                ;
            }
        }
    }
}
```

```
}  
  
void SubTask(VP_INT stacode)  
{  
    ER ret;  
  
    for(;;) {  
        ret = tslp_tsk(100);  
        if (ret == E_OK) {  
            rel_wai(MainTaskID);  
        } else if (ret == E_TMOUT) {  
            wup_tsk(MainTaskID);  
        } else {  
            ;  
        }  
    }  
}
```

6-2-1. 起床待ち

書式	ER ercd = slp_tsk(void); ER ercd = tslp_tsk(TMO tmout);		
引数	TMO	tmout	タイムアウト指定 (tslp_tsk のみ)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (tmout が不正、tslp_tsk のみ)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付)	
	E_TMOUT	ポーリング失敗またはタイムアウト (tslp_tsk のみ)	
	E_CTX	コンテキストエラー (非タスクコンテキスト、またはディスパッチ保留状態)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

自タスクを起床待ち状態に遷移させます。ただし、起床要求がキューイングされている場合には、起床要求キューイング数を-1し、実行状態のまま待ち状態には遷移しません。tslp_tsk()は、slp_tsk()にタイムアウト機能を追加したシステムコールです。tmout には、最大値以下の正のタイムアウト指定 (ミリ秒単位) の他に、TMO_POL (=0: ポーリング) と TMO_FEVR (=1: 永久待ち) を指定することができます。

6-2-2. タスクの起床

書式	ER ercd = wup_tsk(ID tskid); ER ercd = iwup_tsk(ID tskid);		
引数	ID	tskid	起床対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_OBJ	オブジェクト状態エラー (対象タスクが休止状態)	
	E_QOVR	キューイングオーバフロー (起床キューイング数がオーバフロー)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクを、起床待ち状態を解除します。待ち解除されたタスクは、起床待ちシステムコールから E_OK が返ります。そのタスクが起床待ち状態ではない場合には、タスクの起床キューイング数が最大値以下では+1し、最大値では E_QOVR エラーを返します。(i)wup_tsk の場合、tskid に TSK_SELF (=0) を指定すると起床対象タスクを自タスクとします。

6-2-3. タスク起床要求のキャンセル

書式	ER_UINT wupcnt = can_wup(ID tskid);		
引数	ID	tskid	起床要求のキャンセル対象のタスク ID
戻り値	ER_UINT	wupcnt	キューイングされた起床要求の回数 (0 以上) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_OBJ	オブジェクト状態エラー (対象タスクが休止状態)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクのキューイングされた起床要求をクリアし、クリアする前のキューイング数を返します。can_wup の場合、tskid に TSK_SELF (=0) を指定すると、起床要求のキャンセル対象タスクを自タスクとします。

6-2-4. 待ち状態の強制解除

書式	ER ercd = rel_wai(ID tskid); ER ercd = irel_wai(ID tskid);		
引数	ID	tskid	待ち状態の強制解除対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_OBJ	オブジェクト状態エラー (対象タスクが待ち状態でない)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクが待ち状態だった場合、強制的に待ち状態を解除します。待ちを解除されたタスクは、待ちに遷移するシステムコールからは E_RLWAI エラーが返ります。そのタスクが待ち状態ではない場合は、E_OBJ エラーを返します。

6-2-5. 強制待ち状態への移行

書式	ER ercd = sus_tsk(ID tskid);		
引数	ID	tskid	移行対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_CTX	コンテキストエラー (ディスパッチ保留状態で、自タスクが対象)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_OBJ	オブジェクト状態エラー (対象タスクが休止状態)	
	E_QOVR	キューイングオーバフロー (強制待ち要求ネスト数がオーバフロー)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクを強制待ち状態にします。強制待ち状態にされたタスクは、実行状態または実行可能状態のタスクは強制待ち状態に、待ち状態のタスクは二重待ち状態に遷移させます。また、強制待ち要求ネスト数が、最大値より小さい場合は+1 し、最大値の場合は E_QOVR エラーが返ります。sus_tsk の場合、tskid に TSK_SELF (=0) を指定すると移行対象タスクを自タスクとします。

6-2-6. 強制待ち状態からの再開

書式	ER ercd = rsm_tsk(ID tskid);		
引数	ID	tskid	再開対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_OBJ	オブジェクト状態エラー (対象タスクが強制待ち状態でない)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクの強制待ち状態を解除し再開します。強制待ち要求ネスト数を-1 して 0 になると、対象タスクが強制待ち状態だった場合は実行可能状態に、二重待ち状態だった場合は待ち状態に遷移させます。対象タスクが強制待ち状態ではない場合は、E_OBJ エラーを返します。

6-2-7. 強制待ち状態からの強制再開

書式	ER ercd = frsm_tsk(ID tskid); ER ercd = vn_frsm_tsk(ID tskid);		
引数	ID	tskid	再開対象のタスク ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (タスク ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象タスクが未登録状態)	
	E_OBJ	オブジェクト状態エラー (対象タスクが強制待ち状態でない)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskid で指定される ID 番号のタスクの強制待ち状態を解除し再開します。強制待ち要求ネスト数を 0 にし、対象タスクが強制待ち状態だった場合は実行可能状態に、二重待ち状態だった場合は待ち状態に遷移させます。対象タスクが強制待ち状態ではない場合は、E_OBJ エラーを返します。

6-2-8. 自タスクの遅延

書式	ER ercd = dly_tsk(RELTIM dlytim);		
引数	RELTIM	dlytim	自タスクの遅延時間
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (dlytim が不正)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付)	
	E_CTX	コンテキストエラー (非タスクコンテキスト、またはディスパッチ保留状態)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

自タスクを時間経過待ち状態に遷移させ、dlytim で指定された時間 (ミリ秒単位) だけ、実行を一時的に停止します。具体的には、システムコールを呼出した時刻から dlytim で指定された時刻に待ちを解除させるように設定し、自タスクを時間経過待ちに遷移させます。遅延時間経過後に待ち解除した場合には E_OK を返します。それ以前に rel_wai で待ち解除された場合には、E_RLWAI エラーが返ります。

6-3. セマフォ同期・通信機能

セマフォとは、何らかの資源を対象に、使用しているか否か、或いは資源の数などを数値で表現し管理するオブジェクトです。セマフォ同期・通信機能には、以下の機能があります。

- セマフォの生成と削除
- セマフォの資源の獲得と返却
- セマフォの状態の参照

【コーディング例】

セマフォ同期・通信機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub  = { TA_HLNG|TA_ACT, 0, SubTask,  3, 0x2000, NULL, "SubTask" };
T_CSEM csem1     = { TA_TFIFO, 0, 1, "Semaphore1" };
ID SemID1 = 0;
T_RSEM rsem;

void MainTask(VP_INT stacode)
{
    ER_ID erid;
    ER ret;

    erid = acre_sem(&csem1);
    if (erid > 0) {
        SemID1 = (ID)erid;
        for (;;) {
            ret = sig_sem(SemID1);
            if (ret != E_OK) {
                break;
            }
            dly_tsk(10);
        }
        ret = ref_sem(SemID1, &rsem);
        if (ret == E_OK) {
            del_sem(SemID1);
        }
    }
}
```

```
    }  
}
```

```
void SubTask(VP_INT stacode)  
{  
    UINT cnt;  
    ER ret;  
  
    if (SemID1 > 0) {  
        for (cnt = 0; cnt < 100; cnt++) {  
            ret = wai_sem(SemID1);  
            if (ret != E_OK) {  
                break;  
            }  
        }  
    }  
}
```

6-3-1. セマフォの生成

書式	ER ercd = cre_sem(ID semid, T_CSEM *pk_csem); ER_ID semid = acre_sem(T_CSEM *pk_csem);		
引数	ID	semid	生成対象のセマフォ ID
	T_CSEM	*pk_csem	セマフォ生成情報を格納したパケットへのポインタ
	pk_csem の内容 (T_CSEM 型)		
	ATR	sematr	セマフォ属性
	UINT	isemcnt	セマフォの資源数の初期値
	UINT	maxsem	セマフォの最大資源数
	VB	*name	セマフォの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_sem の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_sem の場合		
	ER_ID	semid	生成したセマフォの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (セマフォ ID が不正、cre_sem のみ)	
	E_NOID	ID 番号不足 (未登録状態のセマフォ ID がない、acre_sem のみ)	
	E_RSATR	予約属性 (sematr が不正)	
	E_PAR	パラメータエラー (pk_csem、isemcnt、maxsem が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象セマフォが登録済み、cre_sem のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

semid で指定される ID 番号のセマフォを、pk_csem で指定されるセマフォ生成情報を基にして生成します。具体的には、セマフォ ID で指定される未登録状態のセマフォを、isemcnt で指定される値を初期値の資源数、maxsem で指定される値を最大資源数とするセマフォを登録済み状態に遷移させます。sematr には、(TA_FIFO | TA_TPRI) を指定でき、TA_FIFO は FIFO 順待ち行列を、TA_TPRI はタスク優先度順待ち行列を構成します。acre_sem は、生成可能なセマフォ ID を検索し割付け、そのセマフォ ID を返します。name で指定するセマフォの名称は、ヌル ('¥0') を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。name に NULL を指定することでセマフォの名称を省略することができますが、その場合は自動的に任意の名称が割

付けられます。

6-3-2. セマフォの削除

書式	ER ercd = del_sem(ID semid);		
引数	ID	semid	削除対象のセマフォ ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (セマフォ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象セマフォが未登録)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

semid で指定される ID 番号のセマフォを削除します。具体的には、指定される ID 番号のセマフォを登録状態から未登録状態に遷移させます。

セマフォ獲得待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを獲得待ち状態から実行可能状態に遷移させ、E_DLT エラーを返します。

6-3-3. セマフォ資源の返却

書式	ER ercd = sig_sem(ID semid); ER ercd = isig_sem(ID semid);		
引数	ID	semid	資源返却対象のセマフォ ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (セマフォ ID が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_NOEXS	オブジェクト未生成 (対象セマフォが未登録)	
	E_QOVR	キューイングオーバフロー (最大資源数を超える返却)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

semid で指定される ID 番号のセマフォに資源を 1 つ返却します。具体的には、指定される ID 番号のセマフォに資源の獲得待ちタスクが存在する場合には、待ち行列の最優先順位タスクの待ちを解除し、そのタスクには戻り値として E_OK を返します。また、資源の獲得待ちタスクが存在しない場合には、セマフォの資源数が最大値以下では+1 し、最大値では E_QOVR エラーを返します。

本 RTOS では、isig_sem と sig_sem とは同じ実装をしています。

6-3-4. セマフォ資源の獲得

書式	ER ercd = wai_sem(ID semid); ER ercd = pol_sem(ID semid); ER ercd = twai_sem(ID semid, TMO tmout);		
引数	ID	semid	資源獲得対象のセマフォ ID
	TMO	tmout	タイムアウト指定 (twai_sem のみ)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (セマフォ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象セマフォが未登録)	
	E_PAR	パラメータエラー (tmout が不正、twai_sem のみ)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、pol_sem 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (wai_sem 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 pol_sem 以外のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に対象セマフォが削除、pol_sem 以外)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

semid で指定される ID 番号のセマフォから資源を 1 つ獲得します。具体的には、指定される ID 番号のセマフォの資源数が一つ以上ある場合には、資源数を-1 して処理を終えます。セマフォの資源数が 0 の場合には、自タスクを資源獲得待ち行列に接続し、実行状態から資源獲得待ち状態に遷移させます。

pol_sem は待ち状態には遷移しないポーリング動作を、wai_sem は条件が成り立つまで永久に待ち状態になる動作を、twai_sem は条件が成り立つまでかタイムアウト時刻になるまで待ち状態になる動作をします。twai_sem のタイムアウト指定 (ミリ秒単位) に、TMO_POL が指定された場合には pol_sem として、TMO_FEVR が指定された場合には wai_sem と同じ動作をします。

6-3-5. セマフォの状態参照

書式	ER ercd = ref_sem(ID semid, T_RSEM *p_rsem);		
引数	ID	semid	参照対象のセマフォ ID
	T_RSEM	*pk_rsem	セマフォ状態を返すパケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
pk_rsem の内容 (T_RSEM 型)			
	ID	wtskid	セマフォの資源獲得待ち行列の先頭のタスク ID
	UINT	sement	セマフォの現在の資源数
	VB	name[16]	セマフォの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号 (セマフォ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象セマフォが未登録状態)	
	E_PAR	パラメータエラー (pk_rsem が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

semid で指定される ID 番号のセマフォの各種状態を参照し、未登録状態でない場合に限り、pk_rsem で指定されるパケットに返します。

対象セマフォの資源獲得待ち行列にタスクが存在する場合は、wtskid には獲得待ち行列の先頭のタスク ID を、sement には 0 を返します。また、資源獲得待ちタスクが存在しない場合は、wtskid には TSK_NONE (=0) を、sement には対象セマフォの現在の資源数を返します。

name には、セマフォ生成時に指定されたセマフォの名称を、指定されなかった場合は自動で生成された名称を返します。

6-4. イベントフラグ同期・通信機能

イベントフラグとは、複数ビット（本 RTOS では 32 ビット）で構成するビットパターンを持ち、ビット毎にイベントの有無を情報として与え、同期を行うためのオブジェクトです。

イベントフラグのビットパターンをセットやクリアすることができます。一方、ビットパターンで指定した全ビット或いは 1 つ以上のビットがセットされることを待つことができ、これにより同期と通信を行います。

イベントフラグ同期・通信機能には、以下の機能があります。

- イベントフラグの生成と削除
- イベントフラグのセットとクリア
- イベントフラグのイベントを待つ機能
- イベントフラグの状態の参照

【コーディング例】

イベントフラグ同期・通信機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub = { TA_HLNG|TA_ACT, 0, SubTask, 3, 0x2000, NULL, "SubTask" };
T_CFLG cflg1 = { TA_TFIFO|TA_WMUL|TA_CLR, 0x00000000, "EventFlag1" };
T_CFLG cflg2 = { TA_TFIFO|TA_WMUL, 0x00000000, "EventFlag2" };
ID FlgID1 = 0;
ID FlgID2 = 0;
T_RFLG rflg;

void MainTask(VP_INT stacode)
{
    FLGPTN flgptns;
    FLGPTN flgptn;
    ER_ID erid;
    ER ret;

    erid = acre_flg(&cflg1);
    if (erid > 0) {
        FlgID1 = erid;
        erid = acre_flg(&cflg2);
    }
}
```

```

    if (erid > 0) {
        FlgID2 = erid;
    }
}
if ((FlgID1 > 0) && (FlgID2 > 0)) {
    for (flgptns = 0x00000001; flgptns != 0; flgptns <= 1) {
        set_flg(FlgID2, flgptns);
        ret = twai_flg(FlgID1, 0xFFFFFFFF, TWF_ORW, &flgptn, 100);
        if (ret != E_OK) {
            break;
        }
    }
}
ret = ref_flg(FlgID1, &rflg);
if (ret == E_OK) {
    del_flg(FlgID1);
}
ret = ref_flg(FlgID2, &rflg);
if (ret == E_OK) {
    del_flg(FlgID2);
}
}

void SubTask(VP_INT stacode)
{
    FLGPTN flgptn;
    ER ret;

    if ((FlgID1 > 0) && (FlgID2 > 0)) {
        for(;;) {
            ret = wai_flg(FlgID2, 0xFFFFFFFF, TWF_ORW, &flgptn);
            if (ret != E_OK) {
                break;
            }
            clr_flg(FlgID2, ~flgptn);
            set_flg(FlgID1, flgptn);
        }
    }
}

```


6-4-1. イベントフラグの生成

書式	ER ercd = cre_flg(ID flgid, T_CFLG *pk_cflg); ER_ID flgid = acre_flg(T_CFLG *pk_cflg);		
引数	ID	flgid	生成対象のイベントフラグ ID
	T_CFLG	*pk_cflg	イベントフラグ生成情報を格納した パケットへのポインタ
	pk_cflg の内容（T_CFLG 型）		
	ATR	flgatr	イベントフラグ属性
	FLGPNTN	iflgptn	イベントフラグのビットパターンの初期値
	VB	*name	イベントフラグの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_flg の場合		
	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	acre_flg の場合		
	ER_ID	flgid	生成したイベントフラグの ID 番号（正の値） またはエラーコード
	E_ID	不正 ID 番号（イベントフラグ ID が不正、cre_flg のみ）	
	E_NOID	ID 番号不足 (未登録状態のイベントフラグ ID がない、acre_flg のみ)	
	E_RSATR	予約属性（flgatr が不正）	
	E_PAR	パラメータエラー（pk_cflg、iflgptn が不正）	
	E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）	
	E_OBJ	オブジェクト状態エラー (対象イベントフラグが登録済み、cre_flg のみ)	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

flgid で指定される ID 番号のイベントフラグを、**pk_cflg** で指定されるイベントフラグ生成情報を基にして生成します。具体的には、イベントフラグ ID で指定される未登録状態のイベントフラグを、**iflgptn** で指定される値をイベントフラグのビットパターンの初期値として登録済み状態に遷移させます。**flgatr** には、 $((TA_FIFO \mid TA_TPRI) \mid (TA_WSGL \mid TA_WMUL) \mid [TA_CLR])$ を指定できます。**TA_FIFO** は FIFO 順待ち行列を、**TA_TPRI** はタスク優先度順待ち行列を構成します。**TA_WMUL** を指定した場合は複数のタスクが同一のイベントフラグ待ちにできますが、**TA_WSGL** を指定した場合は 1 つのタスクだけしかイベント待ちにできません。**TA_CLR** を指定した場合は、イベント待ちシステムコールを呼出したタスクが、条件が成り立ちイベントフラグを読み出した後にビットパターンの

全ビットをクリアします。

acre_flg は、生成可能なイベントフラグ ID を検索し割付け、そのイベントフラグ ID を返します。

name で指定するイベントフラグの名称は、ヌル（'¥0'）を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。**name** に NULL を指定することでイベントフラグの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-4-2. イベントフラグの削除

書式	ER ercd = del_flg(ID flgid);		
引数	ID	flgid	削除対象のイベントフラグ ID
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（イベントフラグ ID が不正）	
	E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）	
	E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

flgid で指定される ID 番号のイベントフラグを削除します。具体的には、指定される ID 番号のイベントフラグを登録状態から未登録状態に遷移させます。

イベントフラグ待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを獲得待ち状態から実行可能状態に遷移させ、E_DLT エラーを返します。

6-4-3. イベントフラグのセット

書式	ER ercd = set_flg(ID flgid, FLGPTN setptn); ER ercd = iset_flg(ID flgid, FLGPTN setptn);		
引数	ID	flgid	セット対象のイベントフラグ ID
	FLGPTN	setptn	セットするビットパターン
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（イベントフラグ ID が不正）	
	E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）	
	E_OBJ	オブジェクト状態エラー (対象イベントフラグが生成済み、cre_sem のみ)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

flgid で指定される ID 番号のイベントフラグのビットパターンに **setptn** で指定されるビッ

トをセットします。具体的には、指定される ID 番号のイベントフラグのビットパターンに **setptn** で指定されるビットの論理和 (OR) で更新します。更新された結果、イベントフラグ待ち行列にタスクが存在する場合は、優先順位の高いタスクから条件が成り立つか順次検索し、成り立つとそのタスクの待ちを解除します。待ちを解除されたタスクは **E_OK** を返し、その時のビットパターンも返します。また、**TA_CLR** が指定されたイベントフラグの場合にはイベントフラグのビットパターン全ビットをクリアし、検索は終了します。本 RTOS では、**iset_flg** と **set_flg** とは同じ実装をしています。

6-4-4. イベントフラグのクリア

書式	ER ercd = clr_flg(ID flgid, FLGPtn clrptn);		
引数	ID	flgid	クリア対象のイベントフラグ ID
	FLGPtn	clrptn	クリアするビットパターン
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (イベントフラグ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象イベントフラグが未登録)	
	E_PAR	パラメータエラー (clrptn が不正)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

flgid で指定される ID 番号のイベントフラグの **clrptn** で指定されたビットをクリアします。ビットの指定は、0 のビットをクリアし、1 のビットは変更しません。具体的には、指定される ID 番号のイベントフラグのビットパターンと **clrptn** との論理積 (AND) で更新します。

6-4-5. イベントフラグ待ち

書式	ER ercd = wai_flg(ID flgid, MODE wfmode, FLGPNTN *p_flgptn); ER ercd = pol_flg(ID flgid, MODE wfmode, FLGPNTN *p_flgptn); ER ercd = twai_flg(ID flgid, MODE wfmode, FLGPNTN *p_flgptn, TMO tmout);		
引数	ID	flgid	待ち獲得対象のイベントフラグ ID
	FLGPNTN	waipntn	待ちビットパターン
	MODE	wfmode	待ちモード
	FLGPNTN	*p_flgptn	ビットパターンを格納する領域の番地
	TMO	tmout	タイムアウト指定 (twai_flg のみ)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	FLGPNTN	flgptn	待ち解除時のビットパターン
エラーコード	E_ID	不正 ID 番号 (イベントフラグ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象イベントフラグが未登録)	
	E_PAR	パラメータエラー (waipntn, wfmode, p_flgptn, tmout が不正)	
	E_ILUSE	システムコール不正使用 (TA_WSGL 属性が指定された イベントフラグに待ちタスクがある)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、pol_flg 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (wai_flg 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 pol_flg 時のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に対象イベントフラグが削除、pol_flg 以外)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

semid で指定される ID 番号のイベントフラグのビットパターンと waipntn で指定されるビットパターンと wfmode で指定される待ちモードとで評価し、条件が成り立つのを待ちます。具体的には、条件が成り立つ場合にはビットパターンを読み出し、E_OK を返すとともに処理を終了します。ビットパターンを読み出した場合は、TA_CLR が指定されたイベントフラグではビットパターンを全ビットクリアします。一方で、条件が成り立たなかった場合には自タスクをイベントフラグ待ち行列に接続し、実行状態からイベントフラグ待ち状態に遷移させます。

wfmode には、(TWF_ANDW || TWF_ORW) が指定できます。条件の評価とは、TWF_ANDW が指定されている場合には、waiptn で指定されるビットがイベントフラグのビットパターンですべてセットされていると条件が成り立ちます。一方で、TWF_ORW が指定されている場合には、waiptn で指定されるビットがイベントフラグのビットパターンで一つ以上のビットがセットされていると条件が成り立ちます。

TA_WSGL が指定されたイベントフラグに、イベントフラグ待ち行列にタスクが接続されている場合は、条件が成り立っているか否かに関わらず、E_ILUSE エラーを返します。

pol_flg は待ち状態には遷移しないポーリング動作を、wai_flg は条件が成り立つまで永久に待ち状態になる動作を、twai_flg は条件が成り立つまでかタイムアウト時刻になるまで待ち状態になる動作をします。twai_flg のタイムアウト指定（ミリ秒単位）に、TMO_POL が指定された場合には pol_flg として、TMO_FEVR が指定された場合には wai_flg と同じ動作をします。

6-4-6. イベントフラグの状態参照

書式	ER ercd = ref_flg(ID flgid, T_RFLG *p_rflg);		
引数	ID	flgid	参照対象のイベントフラグ ID
	T_RFLG	*pk_rflg	イベントフラグ状態を返すパッケージへの ポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
	pk_rsem の内容 (T_RSEM 型)		
	ID	wtskid	イベントフラグの待ち行列の先頭のタスク ID
	FLGPTN	flgpntn	イベントフラグの現在のビットパターン
	VB	name[16]	イベントフラグの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号 (イベントフラグ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象イベントフラグが未登録状態)	
	E_PAR	パラメータエラー (pk_rflg が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

flgid で指定される ID 番号のイベントフラグの各種状態を参照し、未登録状態でない場合に限り、pk_rflg で指定されるパッケージに返します。

対象イベントフラグの待ち行列にタスクが存在する場合は wtskid に待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。flgpntn にイベントフラグの現在のビットパターンを返します。

name には、イベントフラグ生成時に指定されたイベントフラグの名称を、指定されなかつ

た場合は自動で生成された名称を返します。

6-5. データキュー同期・通信機能

データキューとは、1 単位 of データを受け渡すことにより同期と通信を行うためのオブジェクトです。1 単位とは、アーキテクチャ毎に異なり、INT 型或いはポインタ型の両方を扱えるよう大きい方のビットサイズで、本 RTOS では 32 ビットです。

データキューは、定義されたデータの数を格納できるバッファを持ち、データをバッファに格納する送信と、バッファからデータを取り出す受信で通信を行います。また、バッファが満杯な場合でも、一番古いデータ、つまり次に受信するはずだったデータを破棄し、強制的に送信する機能があります。

データキューはデータの数を 0 個とするバッファを定義することができ、この場合はバッファを介さずに、送信と受信を同時に行い同期させることができます。

データキュー同期・通信機能には、以下の機能があります。

- データキューの生成と削除
- データキューへのデータの送信／強制送信とデータの受信
- データキューの状態の参照

データキューのバッファサイズ（バイト数）を算出するマクロとして以下があります。

dtqcnt で指定される個数データを格納できるバッファのサイズ dtqsz が返ります。

```
SIZE dtqsz = TSZ_DTQ(UINT dtqcnt)
```

【コーディング例】

データキュー同期・通信機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub = { TA_HLNG|TA_ACT, 0, SubTask, 3, 0x2000, NULL, "SubTask" };
T_CDTQ cdtq1 = { TA_TFIFO, 10, 0, "Dataqueue1" };
ID DtqID1 = 0;
T_RDTQ rdtq;

void MainTask(VP_INT stacode)
{
    VP_INT data;
    ER_ID erid;
    UINT cnt;
    ER ret;
```

```

erid = acre_dtq(&cdtq1);
if (erid > 0) {
    DtqID1 = erid;
    for(cnt = 1; cnt < 100; cnt++) {
        data = (VP_INT)cnt;
        ret = tsnd_dtq(DtqID1, data, 10);
        if (ret != E_OK) {
            break;
        }
    }
}
ret = ref_dtq(DtqID1, &rdtq);
if (ret == E_OK) {
    del_dtq(DtqID1);
}
}

void SubTask(VP_INT stacode)
{
    VP_INT data;
    UINT cnt;
    ER ret;

    if (DtqID1 > 0) {
        for(;;) {
            ret = trcv_dtq(DtqID1, &data, 100);
            if (ret != E_OK) {
                break;
            }
            cnt = (UINT)data;
            if (cnt >= 20) {
                break;
            }
        }
    }
}

```


6-5-1. データキューの生成

書式	ER ercd = cre_dtq(ID dtqid, T_CDTQ *pk_cdtq); ER_ID dtqid = acre_dtq(T_CDTQ *pk_cdtq);		
引数	ID	dtqid	生成対象のデータキューID
	T_CDTQ	*pk_cdtq	データキュー生成情報を格納した パケットへのポインタ
	pk_cdtq の内容 (T_CDTQ 型)		
	ATR	dtqatr	データキュー属性
	UINT	dtqcnt	データキュー領域の容量 (データの個数)
	VP	dtq	データキュー領域の先頭番地
	VB	*name	データキューの名称 (終端をヌル'¥0'とする文字列)
	戻り値	cre_dtq の場合 ER ercd 正常終了 (E_OK) またはエラーコード acre_dtq の場合 ER_ID dtqid 生成したデータキューの ID 番号 (正の値) またはエラーコード	
エラーコード	E_ID	不正 ID 番号 (データキューID が不正、cre_dtq のみ)	
	E_NOID	ID 番号不足 (未登録状態のデータキューID がない、acre_dtq のみ)	
	E_NOMEM	メモリ不足 (データキューのバッファ、管理領域が確保できない)	
	E_RSATR	予約属性 (dtqatr が不正)	
	E_PAR	パラメータエラー (pk_cdtq、dtq、dtqcnt が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象データキューが登録済み、cre_dtq のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

dtqid で指定される ID 番号のデータキューを対象とし、pk_cdtq で指定されるデータキュー生成情報を基にして生成します。具体的には、未登録状態の対象データキューを、dtqcnt で指定される個数のデータを持つ、dtq で指定されるアドレスのバッファとして登録済み状態に遷移させます。また、dtq に NULL (=0) を指定した場合には、RTOS 管理のシステムメモリ領域から切り出してバッファとして割付けます。dtqatr には、(TA_FIFO | TA TPRI) を指定でき、TA FIFO は FIFO 順待ち行列を、TA TPRI はタスク優先度順待ち

ち行列を構成します。

`acre_dtq` は、生成可能なデータキューID を検索し割付け、そのデータキューID を返します。

`name` で指定するデータキューの名称は、ヌル（'¥0'）を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。`name` に NULL を指定することでデータキューの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-5-2. データキューの削除

書式	ER ercd = del_dtq(ID dtqid);		
引数	ID	dtqid	削除対象のデータキューID
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（データキューID が不正）	
	E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）	
	E_NOEXS	オブジェクト未生成（対象データキューが未登録）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

`dtqid` で指定される ID 番号のデータキューを削除します。具体的には、指定される ID 番号のデータキューを登録状態から未登録状態に遷移させます。

データキューの受信待ち行列または送信待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを獲得待ち状態から実行可能状態に遷移し、E_DLT エラーを返します。

6-5-3. データキューへの送信

書式	ER ercd = snd_dtq(ID dtqid, VP_INT data); ER ercd = psnd_dtq(ID dtqid, VP_INT data); ER ercd = ipsnd_dtq(ID dtqid, VP_INT data); ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);		
引数	ID	dtqid	送信対象のデータキューID
	VP_INT	data	データキューへ送信するデータ
	TMO	tmout	タイムアウト指定（tsnd_dtq のみ）
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（データキューID が不正）	
	E_NOEXS	オブジェクト未生成（対象データキューが未登録）	
	E_PAR	パラメータエラー（tmout が不正、tsnd_dtq のみ）	
	E_RLWAI	待ち状態の強制解除	

	(待ち状態の間に rel_wai を受付、snd_dtq、tsnd_dtq のみ)
E_TMOUT	ポーリング失敗またはタイムアウト (snd_dtq 以外)
E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 snd_dtq、tsnd_dtq 時のディスパッチ保留状態での呼出し)
E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象データキューが削除、snd_dtq、tsnd_dtq のみ)
E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)

dtqid で指定される ID 番号のデータキューを送信対象とし、data で指定されるデータを送信します。具体的には、送信対象データキューに受信待ちタスクが存在する場合には、受信待ち行列の最優先順位タスクの待ちを解除し、そのタスクには data で指定されるデータと戻り値として E_OK を返します。また、受信待ちタスクが存在しない場合には、バッファに空きがあれば data で指定されるデータをバッファに格納し、空きがなければ E_TMOUT エラー返します。

psnd_dtq は待ち状態には遷移しないポーリング動作を、snd_dtq は送信できるまで永久に待ち状態になる動作を、tsnd_dtq は送信できるかタイムアウト時刻になるまで待ち状態になる動作をします。tsnd_dtq のタイムアウト指定 (ミリ秒単位) に、TMO_POL が指定された場合には psnd_dtq として、TMO_FEVR が指定された場合には snd_dtq と同じ動作をします。

また、本 RTOS では、ipsnd_dtq と psnd_dtq とは同じ実装をしています。

6-5-4. データキューへの強制送信

書式	ER ercd = fsnd_dtq(ID dtqid, VP_INT data); ER ercd = ifsnd_dtq(ID dtqid, VP_INT data);		
引数	ID	dtqid	送信対象のデータキューID
	VP_INT	data	データキューへ送信するデータ
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (データキューID が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_NOEXS	オブジェクト未生成 (対象データキューが未登録)	
	E_ILUSE	システムコール不正使用 (対象データキューの容量が 0、 FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

dtqid で指定される ID 番号のデータキューに data で指定されるデータを強制的に送信しま

す。具体的には、対象のデータキューに受信待ちタスクが存在する場合には、受信待ち行列の最優先順位タスクの待ちを解除し、そのタスクには **data** で指定されるデータと戻り値として **E_OK** を返します。また、受信待ちタスクが存在しない場合には、バッファに空きがあれば **data** で指定されるデータをバッファに格納し、空きがなければバッファの一番古いデータを破棄してバッファに格納します。

対象データキューの容量が 0 の場合には、**E_ILUSE** エラーを返します。

6-5-5. データキューからの受信

書式	ER ercd = rcv_dtq(ID dtqid, VP_INT *p_data); ER ercd = prcv_dtq(ID dtqid, VP_INT *p_data); ER ercd = trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);		
引数	ID	dtqid	受信対象のデータキューID
	VP_INT	*p_data	受信したデータを格納する領域の番地
	TMO	tmout	タイムアウト指定 (trcv_dtq のみ)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	VP_INT	data	データキューから受信したデータ
エラーコード	E_ID	不正 ID 番号 (データキューID が不正)	
	E_NOEXS	オブジェクト未生成 (対象データキューが未登録)	
	E_PAR	パラメータエラー (p_data 、 tmout が不正)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、 prcv_dtq 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (rcv_dtq 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 prcv_dtq 以外のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象データキューが削除、 prcv_dtq 以外)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

dtqid で指定される ID 番号のデータキューを受信対象とし、受信した **data** を返します。具体的には、受信対象データキューのバッファにデータが存在する場合には、バッファから一番古いデータを取り出した **data** とともに **E_OK** を返します。また、送信待ち行列にタスクが存在する場合は、最優先順位のタスクから送信データを取り出しバッファに格納し、タスクに **E_OK** を返すと同時に送信待ちから実行可能状態へと遷移させます。一方で、受信対象データキューのバッファにデータが存在しない場合には、送信待ち行列にタスクが存在しなければ、自タスクを受信待ち行列に接続し実行状態から受信待ち状態へ遷移させ

ます。送信待ち行列にタスクが存在していれば、最優先順位のタスクから送信データを取り出した **data** とともに **E_OK** を返し、送信待ちだったタスクは **E_OK** を返すとともに送信待ちから実行可能状態へと遷移させます。

prcv_dtq は待ち状態には遷移しないポーリング動作を、**rev_dtq** は送信できるまで永久に待ち状態になる動作を、**trcv_dtq** は送信できるかタイムアウト時刻になるまで待ち状態になる動作をします。**trcv_dtq** のタイムアウト指定（ミリ秒単位）に、**TMO_POL** が指定された場合には **prcv_dtq** として、**TMO_FEVR** が指定された場合には **rev_dtq** と同じ動作をします。

6-5-6. データキューの状態参照

書式	ER ercd = ref_dtq(ID dtqid, T_RDTQ *p_rdtq); ER ercd = vn_ref_dtq(ID dtqid, T_RDTQ *p_rdtq);		
引数	ID	dtqid	参照対象のデータキューID
	T_RDTQ	*pk_rdtq	データキュー状態を返すパッケージへのポインタ
戻り値	ID	ercd	正常終了（E_OK）またはエラーコード
pk_rdtq の内容（T_RDTQ 型）			
	ID	stskid	データキューの送信待ち行列の先頭のタスク ID
	ID	rtskid	データキューの受信待ち行列の先頭のタスク ID
	UINT	sdtqcnt	データキューのバッファに格納されたデータ数
	VB	name[16]	データキューの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号（データキューID が不正）	
	E_NOEXS	オブジェクト未生成（対象データキューが未登録状態）	
	E_PAR	パラメータエラー（pk_rdtq が不正）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

dtqid で指定される ID 番号のデータキューを対象とし、その各種状態を参照し、未登録状態でない場合に限り、**pk_rdtq** で指定されるパッケージに返します。

対象データキューの送信待ち行列にタスクが存在する場合は **stskid** に待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。受信待ち行列にタスクが存在する場合は **rtskid** に待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。**sdtqcnt** にデータキューに格納されているデータ数を返します。

name には、データキュー生成時に指定されたデータキューの名称を、指定されなかった場合は自動で生成された名称を返します。

6-6. メールボックス同期・通信機能

メールボックスとは、メモリ上に配置されたメッセージを受け渡すことで、同期と通信を行うためのオブジェクトです。

メールボックスでは、メッセージパケットの先頭部分に配置したメッセージヘッダの番地を送信または受信に使用します。

メールボックス同期・通信機能には、以下の機能があります。

- メールボックスの生成と削除
- メールボックスのメッセージの送信とメッセージの受信
- メールボックスの状態の参照

メッセージヘッダの型は、本 RTOS では次のように定義しています。メッセージパケットの先頭部分に、このメッセージヘッダを配置します。

```
typedef struct t_msg {
    struct t_msg    *msgque;    /* メッセージヘッダ */
} T_MSG;
```

【コーディング例】

メールボックス同期・通信機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub  = { TA_HLNG|TA_ACT, 0, SubTask,  3, 0x2000, NULL, "SubTask" };
T_CMBX cmbx1    = { TA_TFIFO, 0, 0, "Mailbox1" };
ID MbxID1 = 0;
T_RMBX rmbx;
#define MSGMAX 10
typedef struct t_testmsg {
    T_MSG head;
    UINT data;
} T_TESTMSG;
T_TESTMSG msg_data[MSGMAX];

void MainTask(VP_INT stacode)
{
    ER_ID erid;
```

```

UINT cnt;
ER ret;

erid = acre_mbx(&cmbx1);
if (erid > 0) {
    MbxID1 = erid;
    for(cnt = 1; cnt <= MSGMAX; cnt++) {
        msg_data[cnt-1].data = cnt;
        ret = snd_mbx(MbxID1, &msg_data[cnt-1].head);
        if (ret != E_OK) {
            break;
        }
    }
}
}

```

```

void SubTask(VP_INT stacode)
{
    T_TESTMSG *msg;
    ER ret;

    for(;;) {
        ret = trcv_mbx(MbxID1, (T_MSG **)&msg, 0);
        if (ret != E_OK) {
            break;
        }
        if (msg->data == MSGMAX) {
            break;
        }
    }
    ret = ref_mbx(MbxID1, &rmbx);
    if (ret == E_OK) {
        del_mbx(MbxID1);
    }
}

```

6-6-1. メールボックスの生成

書式	ER ercd = cre_mbx(ID mbxid, T_CMBX *pk_cmbx); ER_ID mbxid = acre_mbx(T_CMBX *pk_cmbx);		
引数	ID	mbxid	生成対象のメールボックス ID
	T_CMBX	*pk_cmbx	メールボックス生成情報を格納した パケットへのポインタ
	pk_cmbx の内容 (T_CMBX 型)		
	ATR	mbxatr	メールボックス属性
	PRI	maxmpri	送信するメッセージ優先度の最大値
	VP	mprihd	優先度別メッセージキューの ヘッダ領域の先頭番地 (NULL)
	UINT	msgmax	メールボックスに格納可能な最大メッセージ数
	VB	*name	メールボックスの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_mbx の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_mbx の場合		
	ER_ID	mbxid	生成したメールボックスの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (メールボックス ID が不正、cre_mbx のみ)	
	E_NOID	ID 番号不足 (未登録状態のメールボックス ID がない、acre_mbx のみ)	
	E_RSATR	予約属性 (mbxatr が不正)	
	E_PAR	パラメータエラー (pk_cmbx が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象メールボックスが登録済み、cre_mbx のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mbxid で指定される ID 番号のメールボックスを対象とし、pk_cdtq で指定されるメールボックス生成情報を基にして生成します。具体的には、対象メールボックスを登録済み状態に遷移させます。mbxatr には、((TA_FIFO | TA_TPRI) | (TA_MFIFO | TA_MPRI)) を指定でき、TA_FIFO は FIFO 順待ち行列を、TA_TPRI はタスク優先度順待ち行列を構成します。また、TA_MFIFO は FIFO 順で、TA_MPRI はメッセージ優先度順でメッセージキューを構成します。

acre_mbx は、生成可能なメールボックス ID を検索し割付け、そのメールボックス ID を返します。maxpri はメッセージ優先度順メッセージキューの場合のみ有効になります。本 RTOS では、msgmax にメールボックス内に格納可能な最大メッセージ数を指定します。mprihd にはシステムメモリを使用するため NULL とし、指定された値は無効になります。name で指定するメールボックスの名称は、ヌル（'¥0'）を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。name に NULL を指定することでメールボックスの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-6-2. メールボックスの削除

書式	ER ercd = del_mbx(ID mbxid);		
引数	ID	mbxid	削除対象のメールボックス ID
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（メールボックス ID が不正）	
	E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）	
	E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

mbxid で指定される ID 番号のメールボックスを削除します。具体的には、指定される ID 番号のメールボックスを登録状態から未登録状態に遷移させます。メールボックスの受信待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを受信待ち状態から実行可能状態に遷移し、E_DLT エラーを返します。また、未読メッセージが存在する場合は、そのすべてのメッセージは破棄されます。そのため、メモリプールなどと連携している場合など、使用中メモリブロックとして存在し続け悪影響を及ぼします。このような問題を防ぐために、削除する前にメッセージを空読みするなどの処理が必要とされます。

6-6-3. メールボックスへの送信

書式	ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg); ER ercd = snd_mbx2(ID mbxid, PRI msgpri, VP pk_msg);		
引数	ID	mbxid	送信対象のメールボックス ID
	PRI	msgpri	メッセージ優先度
	T_MSG	*pk_msg	メールボックスへ送信する メッセージパケットの先頭番地
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (メールボックス ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象メールボックスが未登録)	
	E_PAR	パラメータエラー (pk_msg が不正)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mbxid で指定される ID 番号のメールボックスを対象とし、pk_msg で指定されるメッセージを送信します。具体的には、対象メールボックスの受信待ち行列にタスクが存在する場合は、そのタスクにメッセージを渡すとともに E_OK を返します。対象メールボックスの受信待ち行列にタスクが存在しない場合は、メッセージキューの最後尾にメッセージを接続し E_OK を返します。

snd_mbx は μ ITRON4.0 仕様で定義されたメッセージヘッダを持ったメッセージパケット用です。一方、snd_mbx2 は本 RTOS の独自仕様で、msgpri にメッセージ優先度を、msg にメッセージのアドレスを指定します。また、FIFO 順メッセージキューの場合の msgpri には NULL を指定します。

snd_mbx は既存のアプリケーションプログラムを利用するためですが、新規では snd_mbx2 を使用することを推奨します。

6-6-4. メールボックスからの受信

書式	ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg); ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg); ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout); ER ercd = rcv_mbx2(ID mbxid, VP *ppk_msg); ER ercd = prcv_mbx2(ID mbxid, VP *ppk_msg); ER ercd = trcv_mbx2(ID mbxid, VP *ppk_msg, TMO tmout);		
引数	ID	mbxid	受信対象のメールボックス ID
	T_MSG	**ppk_msg	メッセージパケットの番地を 格納する領域の番地
	TMO	tmout	タイムアウト指定 (trcv_mbx のみ)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	T_MSG	*pk_msg	メールボックスから受信した メッセージパケットの先頭番地
エラーコード	E_ID	不正 ID 番号 (メールボックス ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象メールボックスが未登録)	
	E_PAR	パラメータエラー (ppk_msg, tmout が不正)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、prcv_mbx 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (rcv_mbx 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 prcv_mbx 以外のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象メールボックスが削除、prcv_mbx 以外)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mbxid で指定される ID 番号のメールボックスを対象とし、メッセージを受信します。具体的には、対象メールボックスのメッセージキューにメッセージが存在する場合には、先頭のメッセージを取りだし、pk_msg に格納するとともに E_OK を返します。対象メールボックスのメッセージキューにメッセージが存在しない場合には、受信待ち行列に接続し、実行状態からメールボックスからの待ち状態に遷移させます。

prcv_mbx、rcv_mbx、trcv_mbx は μ ITRON4.0 仕様で定義されたメッセージヘッダを持ったメッセージパケット用です。一方、prcv_mbx2、rcv_mbx2、trcv_mbx2 は本 RTOS の独自仕様で、msg にメッセージの格納アドレスを指定します。

prcv_mbx、prcv_mbx2 は待ち状態には遷移しないポーリング動作を、rcv_mbx と rcv_mbx2 は条件が成り立つまで永久に待ち状態になる動作を、trcv_mbx と trcv_mbx2 は条件が成り立つまでかタイムアウト時刻になるまで待ち状態になる動作をします。trcv_mbx と trcv_mbx2 のタイムアウト指定（ミリ秒単位）に、TMO_POL が指定された場合には prcv_mbx、prcv_mbx2 として、TMO_FEVR が指定された場合には rcv_mbx、rcv_mbx2 と同じ動作をします。

prcv_mbx、rcv_mbx、trcv_mbx は既存のアプリケーションプログラムを利用するためですが、新規では prcv_mbx2、rcv_mbx2、trcv_mbx2 を使用することを推奨します。

6-6-5. メールボックスの状態参照

書式	ER ercd = ref_mbx(ID mbxid, T_RMBX *p_rmbx); ER ercd = vn_ref_mbx(ID mbxid, T_RMBX *p_rmbx);		
引数	ID	mbxid	参照対象のメールボックス ID
	T_RMBX	*pk_rmbx	メールボックス状態を返すパケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
	pk_rmbx の内容 (T_RMBX 型)		
	ID	wtskid	メールボックスの待ち行列の先頭のタスク ID
	T_MSG	*pk_msg	メールボックスのメッセージキューの 先頭のメッセージパケットの番地
	VB	name[16]	メールボックスの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号 (メールボックス ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象メールボックスが未登録状態)	
	E_PAR	パラメータエラー (pk_rmbx が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mbxid で指定される ID 番号のメールボックスを対象とし、その各種状態を参照し、未登録状態でない場合に限り、pk_rmbx で指定されるパケットに返します。

対象メールボックスの待ち行列にタスクが存在する場合は、wtskid には待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。メッセージキューにメッセージが存在する場合は先頭メッセージパケットの番地を、存在しない場合は 0 を、pk_msg に返します。

name には、メールボックス生成時に指定されたメールボックスの名称を、指定されなかった場合は自動で生成された名称を返します。

6-7. ミューテックス拡張同期・通信機能

ミューテックスとは、排他制御専用のオブジェクトで、優先度逆転を防ぐための機構やタスクの終了とともにロックしていたミューテックスをロック解除する機構などが備わっています。

ミューテックス拡張同期・通信機能には、以下の機能があります。

- ミューテックスの生成と削除
- ミューテックスのロックとロック解除
- ミューテックスの状態の参照

【コーディング例】

ミューテックス同期・通信機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
void MainTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CDTQ cmtx      = { TA_TFIFO, 0, "Mutex1" };
ID MtxID = 0;
T_RMTX rmtx;

void MainTask(VP_INT stacode)
{
    ER_ID erid;
    UINT cnt;
    ER ret;

    erid = acre_mtx(&cmtx1);
    if (erid > 0) {
        MtxID = erid;
        ret = tloc_mtx(MtxID, 10);
        if (ret == E_OK) {
            uln_mtx(MtxID);
        }
        ret = ref_mtx(MtxID, &rmtx);
        if (ret == E_OK) {
            del_mtx(MtxID);
        }
    }
}
```

6-7-1. ミューテックスの生成

書式	ER ercd = cre_mtx(ID mtxid, T_CMTX *pk_cmtx); ER_ID mtxid = acre_mtx(T_CMTX *pk_cmtx);		
引数	ID	mtxid	生成対象のミューテックス ID
	T_CMTX	*pk_cmtx	ミューテックス生成情報を格納した パケットへのポインタ
	pk_cmtx の内容 (T_CMTX 型)		
	ATR	mtxatr	ミューテックス属性
	PRI	ceilpri	ミューテックスの上限優先度
	VB	*name	ミューテックスの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_mtx の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_mtx の場合		
	ER_ID	mtxid	生成したミューテックスの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (ミューテックス ID が不正、cre_mtx のみ)	
	E_NOID	ID 番号不足 (未登録状態のミューテックス ID がない、 acre_mtx のみ)	
	E_RSATR	予約属性 (mtxatr が不正)	
	E_PAR	パラメータエラー (pk_cmtx、ceilpri が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象ミューテックスが登録済み、cre_mtx のみ)	
	E_ILUSE	サービスコール不正使用 (移行タスクの呼出し)	

mtxid で指定される ID 番号のセマフォを、pk_cmtx で指定されるミューテックス生成情報を基にして生成します。具体的には、ミューテックス ID で指定される未登録状態のミューテックスを登録済み状態に遷移させます。mtxatr には、(TA_FIFO | TA_TPRI | TA_INHERIT | TA_CEILING) を指定でき、TA_FIFO は FIFO 順待ち行列を、それ以外はタスク優先度順待ち行列を構成します。また、TA_INHERIT が指定されると優先度継承プロトコル、TA_CEILING が指定されると優先度上限プロトコルとして制御されます。acre_mtx は、生成可能なミューテックス ID を検索し割付け、そのミューテックス ID を返します。

name で指定するセマフォの名称は、ヌル ('¥0') を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。name に NULL を指定する

ことでセマフォの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-7-2. ミューテックスの削除

書式	ER ercd = del_mtx(ID mtxid);		
引数	ID	mtxid	削除対象のミューテックス ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (ミューテックス ID が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_NOEXS	オブジェクト未生成 (対象ミューテックスが未登録)	
	E_ILUSE	サービスコール不正使用 (移行タスクの呼出し)	

mtxid で指定される ID 番号のセマフォを削除します。具体的には、指定される ID 番号のセマフォを登録状態から未登録状態に遷移させます。

ミューテックス待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを待ち状態から実行可能状態に遷移させ、E_DLT エラーを返します。

6-7-3. ミューテックスのロック

書式	ER ercd = loc_mtx(ID mtxid); ER ercd = ploc_mtx(ID mtxid); ER ercd = tloc_mtx(ID mtxid, TMO tmout);		
引数	ID	mtxid	ロック対象のミューテックス ID
	TMO	tmout	タイムアウト指定 (tloc_mtx のみ)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (ミューテックス ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象ミューテックスが未登録)	
	E_PAR	パラメータエラー (tmout が不正、tloc_mtx のみ)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、ploc_mtx 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (loc_mtx 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 ディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象ミューテックスが削除、ploc_mtx 以外)	
	E_ILUSE	サービスコール不正使用 (移行タスクの呼出し)	

mtxid で指定される ID 番号のミューテックスをロックします。ミューテックスが既にロッ

クされていた場合は、自タスクをロック待ち行列に接続し、実行状態からロック待ち状態に遷移させます。

ロックしたミューテックスが優先度上限プロトコルの場合は、自タスクの現在優先度をミューテックスの上限優先度に設定します。

ロック待ちしたミューテックスが優先度継承プロトコルの場合は、ロックしているタスクの現在優先度が、自タスクの現在優先度よりも低ければ、ロックしているタスクの現在優先度を自タスクの現在優先度と同じ優先度に設定します。

`ploc_mtx` は待ち状態には遷移しないポーリング動作を、`loc_mtx` は条件が成り立つまで永久に待ち状態になる動作を、`tloc_mtx` は条件が成り立つまでかタイムアウト時刻になるまで待ち状態になる動作をします。`tloc_mtx` のタイムアウト指定(ミリ秒単位)に、`TMO_POL` が指定された場合には `ploc_mtx` として、`TMO_FEVR` が指定された場合には `loc_mtx` と同じ動作をします。

6-7-4. ミューテックスのロック解除

書式	ER ercd = unl_mtx(ID mtxid);		
引数	ID	mtxid	ロック解除対象のミューテックス ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (ミューテックス ID が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_NOEXS	オブジェクト未生成 (対象ミューテックスが未登録)	
	E_ILUSE	サービスコール不正使用 (移行タスクの呼出し、 対象ミューテックスをロックしていない)	

`mtxid` で指定される ID 番号のミューテックスをロック解除します。指定される ID 番号のミューテックスにロック待ちタスクが存在する場合には、待ち行列の最優先順位タスクの待ちを解除し、そのタスクには戻り値として `E_OK` を返します。また、ロック解除により自タスクがロックしているミューテックスがなくなった場合は、自タスクの現在優先度をベース優先度に設定します。

6-7-5. ミューテックスの状態参照

書式	ER ercd = ref_mtx(ID mtxid, T_RMTX *p_rmtx);		
引数	ID	mtxid	参照対象のミューテックス ID
	T_RMTX	*pk_rmtx	ミューテックス状態を返すパケットへの ポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
	pk_rmtx の内容 (T_RMTX 型)		
	ID	htskid	ミューテックスをロックしているタスク ID
	ID	wtskid	ミューテックスのロック待ち行列の 先頭のタスク ID
	VB	name[16]	ミューテックスの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号 (ミューテックス ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象ミューテックスが未登録状態)	
	E_PAR	パラメータエラー (pk_rmtx が不正)	
	E_ILUSE	サービスコール不正使用 (移行タスクの呼出し)	

mtxid で指定される ID 番号のミューテックスの各種状態を参照し、未登録状態でない場合に限り、pk_rmtx で指定されるパケットに返します。

対象ミューテックスがロックされている場合は、htskid にロックしているタスク ID を返します。ロック待ち行列にタスクが存在する場合は、wtskid には獲得待ち行列の先頭のタスク ID を返します。

name には、セマフォ生成時に指定されたセマフォの名称を、指定されなかった場合は自動で生成された名称を返します。

6-8. メッセージバッファ拡張同期・通信機能

メッセージバッファとは、可変長サイズのメッセージを送受信するオブジェクトです。

メッセージバッファ管理機能には、以下の機能があります。

- メッセージバッファの生成と削除
- メッセージバッファの送信と受信
- メッセージバッファの状態の参照

【コーディング例】

メッセージバッファ同期・通信機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub = { TA_HLNG|TA_ACT, 0, SubTask, 3, 0x2000, NULL, "SubTask" };
T_CMBF cmbf1 = { TA_TFIFO, sizeof(UINT), 128, 0, "MsgBuffer1" };

ID MbfID1 = 0;
T_RMBF rmbf;

void MainTask(VP_INT stacode)
{
    UNT data;
    ER_ID erid;
    UINT cnt;
    ER ret;

    erid = acre_mbf(&cmbf1);
    if (erid > 0) {
        MbfID1 = erid;
        for(;;) {
            ret = trcv_mbf(MbfID1, &data, 10);
            if (ret == E_OK) {
                if (data == 100) {
                    break;
                }
            } else {
                break;
            }
        }
    }
}
```

```
    }  
  }  
}  
  
void SubTask(VP_INT stacode)  
{  
    UINT cnt;  
    ER ret;  
  
    if (MbfID1 > 0) {  
        for (cnt = 1; cnt <= 100; cnt++) {  
            ret = tsnd_mbf(MbfID1, cnt, 100);  
            if (ret != E_OK) {  
                break;  
            }  
        }  
        ret = ref_mbf(MbfID1, &rmbf);  
        if (ret == E_OK) {  
            del_mbf(MbfID1);  
        }  
    }  
}
```

6-8-1. メッセージバッファの生成

書式	ER ercd = cre_mbf(ID mbfid, T_CMBF *pk_cmbf); ER_ID mbfid = acre_mbf(T_CMBF *pk_cmbf);		
引数	ID	mbfid	生成対象のメッセージバッファ ID
	T_CMBF	*pk_cmbf	メッセージバッファ生成情報を格納した パケットへのポインタ
	pk_cmbf の内容 (T_CMBF 型)		
	ATR	dtqatr	メッセージバッファ属性
	UINT	maxmsg	メッセージの最大サイズ (バイト数)
	SIZE	mbfsz	メッセージバッファ領域のサイズ (バイト数)
	VP	mbf	メッセージバッファ領域の先頭番地
	VB	*name	メッセージバッファの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_mbf の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_mbf の場合		
	ER_ID	mbfid	生成したメッセージバッファの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (メッセージバッファ ID が不正、 cre_mbf のみ)	
	E_NOID	ID 番号不足 (未登録状態のメッセージバッファ ID がない、 acre_mbf のみ)	
	E_NOMEM	メモリ不足 (メッセージバッファのバッファ、 管理領域が確保できない)	
	E_RSATR	予約属性 (mbfatr が不正)	
	E_PAR	パラメータエラー (pk_cmbf、mbf、maxmsz が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象メッセージバッファが登録済み、cre_mbf のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mbfid で指定される ID 番号のメッセージバッファを対象とし、pk_cmbf で指定されるメッセージバッファ生成情報を基にして生成します。具体的には、未登録状態の対象メッセージバッファを、受信可能な最大メッセージサイズが maxmsg で、mbf で指定されるアドレスでサイズが mbfsz のバッファとして登録済み状態に遷移させます。また、mbf に NULL

(=0) を指定した場合には、RTOS 管理のシステムメモリ領域から切り出してバッファとして割付けます。mbfatr には、(TA_FIFO | TA_TPRI) を指定でき、TA_FIFO は FIFO 順待ち行列を、TA_TPRI はタスク優先度順待ち行列を構成します。

acre_mbf は、生成可能なメッセージバッファ ID を検索し割付け、そのメッセージバッファ ID を返します。

name で指定するメッセージバッファの名称は、ヌル ('¥0') を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。name に NULL を指定することでデータキューの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-8-2. メッセージバッファの削除

書式	ER ercd = del_mbf(ID mbfid);		
引数	ID	mbfid	削除対象のメッセージバッファ ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (メッセージバッファ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象メッセージバッファが未登録)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mbfid で指定される ID 番号のメッセージバッファを削除します。具体的には、指定される ID 番号のメッセージバッファを登録状態から未登録状態に遷移させます。

メッセージバッファの受信待ち行列または送信待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを獲得待ち状態から実行可能状態に遷移し、E_DLT エラーを返します。

6-8-3. メッセージバッファへの送信

書式	ER ercd = snd_mbf(ID mbfid, VP msg, UINT msgsz); ER ercd = psnd_mbf(ID mbfid, VP msg, UINT msgsz); ER ercd = tsnd_mbf(ID mbfid, VP msg, UINT msgsz, TMO tmout);		
引数	ID	mbfid	送信対象のメッセージバッファ ID
	VP	msg	送信するメッセージの先頭番地
	UINT	msgsz	送信するメッセージのサイズ (バイト数)
	TMO	tmout	タイムアウト指定 (tsnd_mbf のみ)
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (メッセージバッファ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象メッセージバッファが未登録)	
	E_PAR	パラメータエラー (tmout が不正、tsnd_mbf のみ)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、 snd_mbf、tsnd_mbf のみ)	
	E_TMOUT	ポーリング失敗またはタイムアウト (snd_mbf 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 snd_mbf、tsnd_mbf 時のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象メッセージバッファが削除、snd_mbf、tsnd_mbf のみ)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mbfid で指定される ID 番号のメッセージバッファを送信対象とし、msgsz で指定されるサイズ (バイト数) のメッセージ msg を送信します。具体的には、送信対象メッセージバッファに受信待ちタスクが存在する場合には、受信待ち行列の最優先順位タスクの待ちを解除し、そのタスクには msg で指定されるデータと戻り値としてそのサイズを返します。また、受信待ちタスクが存在しない場合には、バッファに空きがあれば msgsz と msg で指定されるメッセージをバッファに格納し、空きがなければ自タスクを送信待ち行列に接続し実行状態から送信待ち状態へ遷移させます。

psnd_mbf は待ち状態には遷移しないポーリング動作を、snd_mbf は送信できるまで永久に待ち状態になる動作を、tsnd_mbf は送信できるかタイムアウト時刻になるまで待ち状態になる動作をします。tsnd_mbf のタイムアウト指定 (ミリ秒単位) に、TMO_POL が指定された場合には psnd_mbf として、TMO_FEVR が指定された場合には snd_mbf と同じ動作をします。

6-8-4. メッセージバッファからの受信

書式	ER_UINT ercd = rcv_mbf(ID mbfid, VP msg); ER_UINT ercd = prcv_mbf(ID mbfid, VP msg); ER_UINT ercd = trcv_mbf(ID mbfid, VP msg, TMO tmout);		
引数	ID	mbfid	受信対象のメッセージバッファ ID
	VP	msg	受信したメッセージを格納する領域の番地
	TMO	tmout	タイムアウト指定 (trcv_mbf のみ)
戻り値	ER_UINT	ercd	受信メッセージのサイズ (バイト数、正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (メッセージバッファ ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象メッセージバッファが未登録)	
	E_PAR	パラメータエラー (p_data、tmout が不正)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、prcv_mbf 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (rcv_mbf 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 prcv_mbf 以外のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象データキューが削除、prcv_mbf 以外)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mbfid で指定される ID 番号のメッセージバッファを受信対象とし、受信したメッセージを msg に格納し、受信したメッセージサイズを返します。具体的には、受信対象メッセージバッファのバッファにメッセージが存在する場合には、バッファから一番古いメッセージを取り出し msg に格納し、受信したメッセージサイズを返します。また、送信待ち行列にタスクが存在する場合は、最優先順位のタスクから送信メッセージを取り出しバッファに格納し、タスクに E_OK を返すとともに送信待ちから実行可能状態へと遷移させます。一方で、受信対象データキューのバッファにデータが存在しない場合には、送信待ち行列にタスクが存在しなければ、自タスクを受信待ち行列に接続し実行状態から受信待ち状態へ遷移させます。送信待ち行列にタスクが存在していれば、最優先順位のタスクから送信データを取り出したメッセージを msg に格納し、そのメッセージのサイズを返し、送信待ちだったタスクは E_OK を返すとともに送信待ちから実行可能状態へと遷移させます。

prcv_mbf は待ち状態には遷移しないポーリング動作を、rcv_mbf は送信できるまで永久に待ち状態になる動作を、trcv_mbf は送信できるかタイムアウト時刻になるまで待ち状態に

なる動作をします。trev_mbf のタイムアウト指定（ミリ秒単位）に、TMO_POL が指定された場合には prcv_mbf として、TMO_FEVR が指定された場合には rev_mbf と同じ動作をします。

6-8-5. メッセージバッファの状態参照

書式	ER ercd = ref_mbf(ID mbfid, T_RDTQ *p_rmbf);		
引数	ID	mbfid	参照対象のデータキューID
	T_RMBF	*pk_rmbf	メッセージバッファ状態を返すパケットへの ポインタ
戻り値	ID	ercd	正常終了（E_OK）またはエラーコード
pk_rmbf の内容（T_RDTQ 型）			
	ID	stskid	メッセージバッファの送信待ち行列の 先頭のタスク ID
	ID	rtskid	メッセージバッファの受信待ち行列の 先頭のタスク ID
	UINT	smsgcnt	メッセージバッファに格納されたメッセージ数
	VB	name[16]	メッセージバッファの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号（メッセージバッファ ID が不正）	
	E_NOEXS	オブジェクト未生成（対象メッセージバッファが未登録状態）	
	E_PAR	パラメータエラー（pk_rmbf が不正）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

mbfid で指定される ID 番号のメッセージバッファを対象とし、その各種状態を参照し、未登録状態でない場合に限り、pk_rmbf で指定されるパケットに返します。

対象メッセージバッファの送信待ち行列にタスクが存在する場合は stskid に待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。受信待ち行列にタスクが存在する場合は rtskid に待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。smbfcnt にメッセージバッファに格納されているメッセージ数を返します。

name には、メッセージバッファ生成時に指定されたメッセージバッファの名称を、指定されなかった場合は自動で生成された名称を返します。

6-9. 固定長メモリプール管理機能

固定長メモリプールとは、固定長サイズのメモリブロックを管理するオブジェクトです。

メモリブロックは固定長メモリプール毎に、サイズと個数を定義できます。

固定長メモリプール管理機能には、以下の機能があります。

- 固定長メモリプールの生成と削除
- 固定長メモリブロックの獲得と返却
- 固定長メモリプールの状態の参照

本 RTOS では 1 バイト以上の任意のサイズのメモリブロックができるため、何らかのアドレス境界が必要な場合は、予め考慮したメモリブロックのサイズにします。

固定長メモリプールのプールサイズ (バイト数) を算出するマクロとして以下があります。
blksz で指定されるメモリブロックのサイズ (バイト数) を blkcnt で指定される個数のプールサイズ mpfsz が返ります。本 RTOS では、サイズと個数を掛けた値が返ります。

```
SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blksz)
```

【コーディング例】

固定長メモリプール管理機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub = { TA_HLNG|TA_ACT, 0, SubTask, 3, 0x2000, NULL, "SubTask" };
#define MPFCNT 10
T_CMPF cmpf1 = { TA_TFIFO, MPFCNT, 256, NULL, "MemoryPool1" };
VP_p_blk[MPFCNT];
ID MpfID1 = 0;
T_RMPF rmpf;

void MainTask(VP_INT stacode)
{
    ER_ID erid;
    UINT cnt;
    ER ret;

    erid = acre_mpf(&cmpf1);
    if (erid > 0) {
```

```

    MpfID1 = erid;
    for(cnt = 0; cnt < MPFCNT; cnt++) {
        ret = tget_mpf(MpfID1, &p_blk[cnt], 100);
        if (ret != E_OK) {
            break;
        }
    }
}
}
}

```

```

void SubTask(VP_INT stacode)
{
    UINT cnt;
    ER ret;

    if (MpfID1 > 0) {
        for(cnt = 0; cnt < MPFCNT; cnt++) {
            ret = rel_mpf(MpfID1, p_blk[cnt]);
            if (ret != E_OK) {
                break;
            }
        }
        ret = ref_mpf(MpfID1, &rmpf);
        if (ret == E_OK) {
            del_mpf(MpfID1);
        }
    }
}
}

```

6-9-1. 固定長メモリアンプールの生成

書式	ER ercd = cre_mpf(ID mpfid, T_CMPF *pk_cmpf); ER_ID mpfid = acre_mpf(T_CMPF *pk_cmpf);		
引数	ID	mpfid	生成対象の固定長メモリアンプール ID
	T_CMPF	*pk_cmpf	固定長メモリアンプール生成情報を格納した パケットへのポインタ
	pk_cmpf の内容 (T_CMPF 型)		
	ATR	blkatr	固定長メモリアンプール属性
	UINT	blkcnt	固定長メモリアンプールのメモリアンプールの個数
	UINT	blksiz	固定長メモリアンプールのメモリアンプールのサイズ
	VP	mpf	固定長メモリアンプール領域の先頭番地
	VB	*name	固定長メモリアンプールの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_mpf の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_mpf の場合		
	ER_ID	mpfid	生成した固定長メモリアンプールの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (固定長メモリアンプール ID が不正、cre_mpf のみ)	
	E_NOID	ID 番号不足 (未登録状態の固定長メモリアンプール ID がない、 acre_mpf のみ)	
	E_NOMEM	メモリアンプール不足 (固定長メモリアンプールのプールの領域が確保できない)	
	E_RSATR	予約属性 (mpfatr が不正)	
	E_PAR	パラメータエラー (pk_cmpf、blkcnt、blksiz、mpf が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象固定長メモリアンプールが登録済み、cre_mpf のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mpfid で指定される ID 番号の固定長メモリアンプールを対象とし、pk_cmpf で指定される固定長メモリアンプール生成情報を基にして生成します。具体的には、未登録状態の対象固定長メモリアンプールを、blksiz で指定されるメモリアンプールのブロックを blkcnt で指定される個数だけ持つ mpf で指定されるアドレスのメモリアンプールとして登録済み状態に遷移させます。また、mpf に

NULL (=0) を指定した場合には、RTOS のユーザーメモリ領域から獲得してメモリプールに割付けます。mpfatr には、(TA_FIFO | TA_TPRI) を指定でき、TA_FIFO は FIFO 順待ち行列を、TA_TPRI はタスク優先度順待ち行列を構成します。

acre_mpf は、生成可能な固定長メモリプール ID を検索し割付け、その固定長メモリプール ID を返します。

name で指定する固定長メモリプールの名称は、ヌル ('¥0') を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。name に NULL を指定することで固定長メモリプールの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-9-2. 固定長メモリプールの削除

書式	ER ercd = del_mpf(ID mpfid);		
引数	ID	mpfid	削除対象の固定長メモリプール ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (固定長メモリプール ID が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_NOEXS	オブジェクト未生成 (対象固定長メモリプールが未登録)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mpfid で指定される ID 番号の固定長メモリプールを削除します。具体的には、指定される ID 番号の固定長メモリプールを登録状態から未登録状態に遷移させます。

固定長メモリプールの獲得待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを獲得待ち状態から実行可能状態に遷移し、E_DLT エラーを返します。

また、返却されていないメモリブロックが存在する場合は、そのすべてのメモリブロックは返却されたと見なします。そのため、返却されていないメモリブロック領域を重複して使うなど悪影響を及ぼします。このような問題を防ぐために、削除する前にすべてのメモリブロックが返却されたか、使用が終了したことを確認するなどの処理が必要とされます。

6-9-3. 固定長メモリブロックの獲得

書式	ER ercd = get_mpf(ID mpfid, VP *p_blk); ER ercd = pget_mpf(ID mpfid, VP *p_blk); ER ercd = tget_mpf(ID mpfid, VP *p_blk, TMO tmout);		
引数	ID	mpfid	獲得対象の固定長メモリプール ID
	TMO	tmout	タイムアウト指定 (tget_mpf のみ)
	VP	p_blk	メモリブロックの番地を格納する領域の番地
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	VP	blk	獲得したメモリブロックの先頭番地
エラーコード	E_ID	不正 ID 番号 (固定長メモリプール ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象固定長メモリプールが未登録)	
	E_PAR	パラメータエラー (p_blk, tmout が不正)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、pget_mpf 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (get_mpf 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 pget_mpf 以外のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象固定長メモリプールが削除、pget_mpf 以外)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mpfid で指定される ID 番号の固定長メモリプールを対象とし、メモリブロックを獲得します。具体的には、対象固定長メモリプールにメモリブロックが存在する場合には、メモリブロックを 1 つ取りだし、その先頭番地 blk とともに E_OK を返します。対象固定長メモリプールにメモリブロックが存在しない場合には、獲得待ち行列に接続し、実行状態から固定長メモリブロック待ち状態に遷移させます。

pget_mpf は待ち状態には遷移しないポーリング動作を、get_mpf は獲得できるまで永久に待ち状態になる動作を、tget_mpf は獲得できるかタイムアウト時刻になるまで待ち状態になる動作をします。tget_mpf のタイムアウト指定 (ミリ秒単位) に、TMO_POL が指定された場合には pget_mpf として、TMO_FEVR が指定された場合には get_mpf と同じ動作をします。

6-9-4. 固定長メモリブロックの返却

書式	ER ercd = rel_mpf(ID mpfid, VP blk);		
引数	ID	mpfid	返却対象の固定長メモリプール ID
	VP	blk	返却する固定長メモリブロックの先頭番地
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (固定長メモリプール ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象固定長メモリプールが未登録)	
	E_PAR	パラメータエラー (blk が不正、 返却対象の固定長メモリプールのメモリブロック以外、 既に返却済みのメモリブロック)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mpfid で指定される ID 番号の固定長メモリプールを対象とし、blk で指定されるメモリブロックを返却します。具体的には、対象固定長メモリプールの獲得待ち行列にタスクが存在する場合は、そのタスクにメモリブロックを渡すとともに E_OK を返します。対象固定長メモリプールの獲得待ち行列にタスクが存在しない場合は、メモリブロックをメモリプールに返却し E_OK を返します。

6-9-5. 固定長メモリアンプールの状態参照

書式	ER ercd = ref_mpf(ID mpfid, T_RMPF *p_rmpf);		
引数	ID	mpfid	参照対象の固定長メモリアンプール ID
	T_RMPF	*p_rmpf	固定長メモリアンプール状態を返す パケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
pk_rmpf の内容 (T_RMPF 型)			
	ID	wtskid	固定長メモリアンプールの待ち行列の先頭のタスク ID
	UINT	fblkent	固定長メモリアンプールの空きメモリアンプールブロック数
	VB	name[16]	固定長メモリアンプールの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号 (固定長メモリアンプール ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象固定長メモリアンプールが未登録状態)	
	E_PAR	パラメータエラー (pk_rmpf が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mpfid で指定される ID 番号の固定長メモリアンプールを対象とし、その各種状態を参照し、未登録状態でない場合に限り、pk_rmpf で指定されるパケットに返します。

対象固定長メモリアンプールの待ち行列にタスクが存在する場合は、wtskid に待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。現在の固定長メモリアンプール内の空きメモリアンプールブロックの個数を fblkent に返します。

name には、固定長メモリアンプール生成時に指定された固定長メモリアンプールの名称を、指定されなかった場合は自動で生成された名称を返します。

6-10. 可変長メモリプール管理機能

可変長メモリプールとは、任意サイズのメモリブロックを管理するオブジェクトです。可変長メモリプール毎に、プールサイズを定義できます。

可変長メモリプール管理機能には、以下の機能があります。

- 可変長メモリプールの生成と削除
- 可変長メモリブロックの獲得と返却
- 可変長メモリプールの状態の参照

本 RTOS では 1 つのメモリブロックは 16 バイト境界の揃えられます。そのため、1 バイトのメモリブロックを獲得すると、メモリプールから 16 バイトが消費されます。

可変長メモリプールのプールサイズ (バイト数) を算出するマクロとして以下があります。
blksz で指定されるメモリブロックのサイズ (バイト数) を **blkcnt** で指定される個数を獲得できるメモリプール領域のサイズを返します。ただし、一般的に獲得するメモリブロックのサイズは一定では無いため、この値は目安でしかありません。

SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)

【コーディング例】

可変長メモリプール管理機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
#define SubTaskID 2
void MainTask(VP_INT stacode);
void SubTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CTSK ctsk_sub = { TA_HLNG|TA_ACT, 0, SubTask, 3, 0x2000, NULL, "SubTask" };
#define MPLSZ 0x10000
T_CMPL cmpl1 = { TA_TFIFO, MPLSZ, NULL, "MemoryPool2" };
ID MpfID1 = 0;

#define MPLCNT 10
VP_p_blk[MPLCNT];
T_RMPL rmpl;

void MainTask(VP_INT stacode)
{
    ER_ID erid;
    UINT cnt;
    ER ret;
```



```

erid = acre_mpl(&cmpl1);
if (erid > 0) {
    MplID1 = erid;
    for(cnt = 0; cnt < MPFCNT; cnt++) {
        ret = tget_mpl(MplID1, &p_blk[cnt], 0x100);
        if (ret != E_OK) {
            break;
        }
    }
}
}

```

```

void SubTask(VP_INT stacode)
{
    UINT cnt;
    ER ret;

    if (MplID1 > 0) {
        for(cnt = 0; cnt < MPLCNT; cnt++) {
            ret = rel_mpl(MplID1, p_blk[cnt]);
            if (ret != E_OK) {
                break;
            }
        }
        ret = ref_mpl(MplID1, &rmpl);
        if (ret == E_OK) {
            del_mpl(MplID1);
        }
    }
}

```

6-10-1. 可変長メモリプールの生成

書式	ER ercd = cre_mpl(ID mplid, T_CMPL *pk_cmpl); ER_ID mplid = acre_mpl(T_CMPL *pk_cmpl);		
引数	ID	mplid	生成対象の可変長メモリプール ID
	T_CMPL	*pk_cmpl	可変長メモリプール生成情報を格納した パケットへのポインタ
	pk_cmpl の内容 (T_CMPL 型)		
	ATR	mplatr	可変長メモリプール属性
	UINT	mplsz	可変長メモリプール領域のサイズ
	VP	mpl	可変長メモリプール領域の先頭番地
	UINT	blkmax	可変長メモリプールの最大メモリブロック数
	VB	*name	可変長メモリプールの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_mpf の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_mpl の場合		
	ER_ID	mplid	生成した可変長メモリプールの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (可変長メモリプール ID が不正、cre_mpl のみ)	
	E_NOID	ID 番号不足 (未登録状態の固定長メモリプール ID がない、 acre_mpl のみ)	
	E_NOMEM	メモリ不足 (可変長メモリプールのプール領域が確保できない)	
	E_RSATR	予約属性 (mplatr が不正)	
	E_PAR	パラメータエラー (pk_cmpl、mplsz、mpl が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象可変長メモリプールが登録済み、cre_mpl のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mplid で指定される ID 番号の可変長メモリプールを対象とし、pk_cmpl で指定される可変長メモリプール生成情報を基にして生成します。具体的には、未登録状態の対象可変長メモリプールを、blkmax で指定されるメモリブロックを持つ mpl で指定されるアドレスのメモリプールとして登録済み状態に遷移させます。また、mpl に NULL (=0) を指定した場合

には、RTOS のユーザーメモリ領域から獲得してメモリプールに割付けます。mplatr には、(TA_FIFO | TA_TPRI) を指定でき、TA_FIFO は FIFO 順待ち行列を、TA_TPRI はタスク優先度順待ち行列を構成します。

本 RTOS では、可変長メモリプールから取得できるメモリブロック数を指定します。

acre_mpl は、生成可能な可変長メモリプール ID を検索し割付け、その可変長メモリプール ID を返します。

name で指定する可変長メモリプールの名称は、ヌル ('¥0') を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。name に NULL を指定することで固定長メモリプールの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-10-2. 可変長メモリプールの削除

書式	ER ercd = del_mpl(ID mplid);		
引数	ID	mplid	削除対象の可変長メモリプール ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (可変長メモリプール ID が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_NOEXS	オブジェクト未生成 (対象可変長メモリプールが未登録)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mplid で指定される ID 番号の可変長メモリプールを削除します。具体的には、指定される ID 番号の可変長メモリプールを登録状態から未登録状態に遷移させます。

可変長メモリプールの獲得待ち行列にタスクが存在する場合は、待ち行列に接続されているすべてのタスクを獲得待ち状態から実行可能状態に遷移し、E_DLT エラーを返します。また、返却されていないメモリブロックが存在する場合は、そのすべてのメモリブロックは返却されたと見なします。そのため、返却されていないメモリブロック領域を重複して使うなど悪影響を及ぼします。このような問題を防ぐために、削除する前にすべてのメモリブロックが返却されたか、使用が終了したことを確認するなどの処理が必要とされます。

6-10-3. 可変長メモリブロックの獲得

書式	ER ercd = get_mpl(ID mplid, UINT blksz, VP *p_blk); ER ercd = pget_mpl(ID mplid, UINT blksz, VP *p_blk); ER ercd = tget_mpl(ID mplid, UINT blksz, VP *p_blk, TMO tmout);		
引数	ID	Mplid	獲得対象の固定長メモリプール ID
	UINT	blksz	獲得するメモリブロックのサイズ
	TMO	tmout	タイムアウト指定 (tget_mpl のみ)
	VP	p_blk	メモリブロックの番地を格納する領域の番地
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	VP	blk	獲得したメモリブロックの先頭番地
エラーコード	E_ID	不正 ID 番号 (可変長メモリプール ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象可変長メモリプールが未登録)	
	E_PAR	パラメータエラー (p_blk, tmout が不正)	
	E_RLWAI	待ち状態の強制解除 (待ち状態の間に rel_wai を受付、pget_mpl 以外)	
	E_TMOUT	ポーリング失敗またはタイムアウト (get_mpl 以外)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し、 pget_mpl 以外のディスパッチ保留状態での呼出し)	
	E_DLT	待ちオブジェクトの削除 (待ち状態の間に 対象可変長メモリプールが削除、pget_mpl 以外)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mplid で指定される ID 番号の可変長メモリプールを対象とし、メモリブロックを獲得します。具体的には、対象可変長メモリプールに十分なメモリ領域が存在する場合には、メモリブロックを取りだし、その先頭番地 blk とともに E_OK を返します。対象可変長メモリプールに獲得するだけの空きメモリ領域が存在しない場合、或いは可変長メモリプールから獲得できる最大メモリブロック数に達している場合には、獲得待ち行列に接続し、実行状態から可変長メモリブロック待ち状態に遷移させます。

pget_mpl は待ち状態には遷移しないポーリング動作を、get_mpl は獲得できるまで永久に待ち状態になる動作を、tget_mpl は獲得できるかタイムアウト時刻になるまで待ち状態になる動作をします。tget_mpl のタイムアウト指定 (ミリ秒単位) に、TMO_POL が指定された場合には pget_mpl として、TMO_FEVR が指定された場合には get_mpl と同じ動作をします。

6-10-4. 可変長メモリブロックの返却

書式	ER ercd = rel_mpl(ID mplid, VP blk);		
引数	ID	mplid	返却対象の可変長メモリプール ID
	VP	blk	返却する可変長メモリブロックの先頭番地
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (可変長メモリプール ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象可変長メモリプールが未登録)	
	E_PAR	パラメータエラー (blk が不正、 返却対象の可変長メモリプールのメモリブロック以外、 既に返却済みのメモリブロック)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_ILUSE	サービスコール不正使用 (FIFO 順待ち行列以外へのセキュア移行タスクの呼出し、 非セキュア移行タスクの呼出し)	

mplid で指定される ID 番号の可変長メモリプールを対象とし、blk で指定されるメモリブロックを返却します。具体的には、メモリブロックを対象可変長メモリプールに返却した後、可変長メモリプールの獲得待ち行列にタスクが存在し、獲得できた場合のみ、そのタスクに獲得したメモリブロックを渡すとともに E_OK を返します。対象可変長メモリプールの獲得待ち行列にタスクが存在しない場合と、メモリブロックを獲得できなかった場合は、メモリブロックをメモリプールに返却し E_OK を返します。

6-10-5. 可変長メモリプールの状態参照

書式	ER ercd = ref_mpl(ID mplid, T_RMPL *p_rmpl);		
引数	ID	Mp l id	参照対象の可変長メモリプール ID
	T_RMPL	*pk_rmpl	可変長メモリプール状態を返す パケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
pk_rmpl の内容 (T_RMPL 型)			
	ID	wtskid	可変長メモリプールの待ち行列の先頭のタスク ID
	SIZE	fmplsz	可変長メモリプールの空きメモリの合計サイズ
	UINT	fblksz	獲得可能な最大メモリブロックサイズ
	VB	name[16]	可変長メモリプールの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号 (可変長メモリプール ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象可変長メモリプールが未登録状態)	
	E_PAR	パラメータエラー (pk_rmpf が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

mplid で指定される ID 番号の可変長メモリプールを対象とし、その各種状態を参照し、未登録状態でない場合に限り、pk_rmpl で指定されるパケットに返します。

対象可変長メモリプールの待ち行列にタスクが存在する場合は、wtskid には待ち行列の先頭のタスク ID を、存在しない場合は 0 を返します。可変長メモリプール内の空き領域の合計を fmplsz に、すぐに獲得可能な最大のメモリブロックサイズを fblksz に返します。

name には、可変長メモリプール生成時に指定された可変長メモリプールの名称を、指定されなかった場合は自動で生成された名称を返します。

6-11. システム時刻管理

システム時刻管理機能には、以下の機能があります。

- システム時刻の設定と参照
- タイムチックの供給

本 RTOS では、システム時刻を次の構造体として定義しています。これらはすべてミリ秒単位で設定または参照されます。

```
typedef struct t_sysim {  
    UW utime;           /* 上位 32 ビット */  
    UW ltime;           /* 下位 32 ビット */  
} SYSTIM;
```

6-11-1. システム時刻の設定

書式	ER ercd = set_tim(SYSTIM *p_systim);		
引数	SYSTIM	*p_systim	設定するシステム時刻の格納領域の番地
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (p_systim が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

p_systim で指定される時刻をシステム時刻に設定します。この設定により、相対時間でタイムアウトを指定している場合でも、その相対時間が変化することはありません。つまり、システム時刻を変更した場合は、タイムアウト時刻が変化することになります。

6-11-2. システム時刻の参照

書式	ER ercd = get_tim(SYSTIM *p_systim);		
引数	SYSTIM	*p_systim	システム時刻を格納する領域の番地
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	SYSTIM	systim	現在のシステム時刻
エラーコード	E_PAR	パラメータエラー (p_systim が不正)	

現在のシステム時刻 systim を返します。

6-12. 周期ハンドラ機能

周期ハンドラとは、一定周期でタイムイベントハンドラが起動するオブジェクトです。タイムイベントハンドラが周期ハンドラの生成時から指定された起動位相の時間経過後に最初の起動を行い、以降起動周期の時間経過毎に起動を続けます。

周期ハンドラには、以下の機能があります。

- 周期ハンドラの生成と削除
- 周期ハンドラの動作の開始と停止
- 周期ハンドラの状態参照

また、周期ハンドラの周期を変更せずに、開始時刻に起動させないことも、新たな周期で開始させることもできます。これらの設定はミリ秒単位です。

【コーディング例】

周期ハンドラ機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
void MainTask(VP_INT stacode);
void CycHandler(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_CCYC ccyc1      = { TA_HLNG|TA_PHS, 0, CycHandler, 100, 50, "CycHandler1" };
ID CycID1 = 0;
T_RCYC rcyc;

void MainTask(VP_INT stacode)
{
    ER_ID erid;
    UINT cnt;
    ER ret;

    erid = acre_cyc(&ccyc1);
    if (erid > 0) {
        CycID1 = erid;
        for(cnt = 1; cnt < 5; cnt++) {
            sta_cyc(CycID1);
            dly_tsk(800);
            stp_cyc(CycID1);
            dly_tsk(200);
        }
    }
    ret = ref_cyc(CycID1, &rcyc);
    if (ret == E_OK) {
        del_cyc(CycID1);
    }
}

void CycHandler(VP_INT stacode)
{
    ;
}
```

6-12-1. 周期ハンドラの生成

書式	ER ercd = cre_cyc(ID cycid, T_CCYC *pk_ccyc); ER_ID cycid = acre_cyc(T_CCYC *pk_ccyc);		
引数	ID	cycid	生成対象の周期ハンドラ ID
	T_CCYC	*pk_ccyc	周期ハンドラ生成情報を格納した パケットへのポインタ
	pk_ccyc の内容 (T_CCYC 型)		
	ATR	cycatr	周期ハンドラ属性
	VP_INT	exinf	周期ハンドラの拡張情報
	FP	cychdr	周期ハンドラの起動番地
	RELTIM	cycetim	周期ハンドラの起動周期
	RELTIM	cycphs	周期ハンドラの起動位相
	VB	*name	周期ハンドラの名称 (終端をヌル'¥0'とする文字列)
	戻り値	cre_cyc の場合	
ER	ercd	正常終了 (E_OK) またはエラーコード	
	acre_cyc の場合		
	ER_ID	cycid	生成した周期ハンドラの ID 番号 (正の値) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (周期ハンドラ ID が不正、cre_cyc のみ)	
	E_NOID	ID 番号不足 (未登録状態の周期ハンドラ ID がない、acre_cyc のみ)	
	E_RSATR	予約属性 (cycatr が不正)	
	E_PAR	パラメータエラー (pk_ccyc が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象周期ハンドラが登録済み、cre_cyc のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

cycid で指定される ID 番号の周期ハンドラを対象とし、**pk_ccyc** で指定される周期ハンドラ生成情報を基にして生成し、対象周期ハンドラを登録済み状態に遷移させます。**cycatr** には、(TA_HLNG | [TA_STA] | [TA_PHS]) を指定できます。TA_STA を指定した場合は、周期ハンドラを起動した状態で生成します。TA_PHS を指定した場合は、起動位相を保存し、周期ハンドラを停止しても起動位相は変化しません。

acre_cyc は、生成可能な周期ハンドラ ID を検索し割付け、その周期ハンドラ ID を返します。

name で指定する周期ハンドラの名称は、ヌル（'¥0'）を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。name に NULL を指定することで周期ハンドラの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-12-2. 周期ハンドラの削除

書式	ER ercd = del_cyc(ID cycid);		
引数	ID	cycid	削除対象の周期ハンドラ ID
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（周期ハンドラ ID が不正）	
	E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）	
	E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

cycid で指定される ID 番号の周期ハンドラを削除します。具体的には、対象周期ハンドラを登録状態から未登録状態に遷移させます。また、動作が開始中の周期ハンドラだった場合には動作を停止させます。

6-12-3. 周期ハンドラの動作開始

書式	ER ercd = sta_cyc(ID cycid);		
引数	ID	cycid	動作開始対象の周期ハンドラ ID
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（周期ハンドラ ID が不正）	
	E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

cycid で指定される ID 番号の周期ハンドラを対象とし、動作を開始状態にします。また、周期ハンドラの属性で TA_PHS を指定していない場合は、このシステムコールを呼出した時刻を基準に起動周期間隔で周期ハンドラを起動させるため起動時刻を設定します。

6-12-4. 周期ハンドラの動作停止

書式	ER ercd = stp_cyc(ID cycid);		
引数	ID	cycid	動作停止対象の周期ハンドラ ID
戻り値	ER	ercd	正常終了（E_OK）またはエラーコード
エラーコード	E_ID	不正 ID 番号（周期ハンドラ ID が不正）	

E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）
E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）

cycid で指定される ID 番号の周期ハンドラを対象とし、動作を停止状態にします。

6-12-5. 周期ハンドラの状態参照

書式	ER ercd = ref_cyc(ID cycid, T_RCYC *p_rcyc);		
引数	ID	cycid	参照対象の周期ハンドラ ID
	T_RCYC	*pk_rcyc	周期ハンドラ状態を返すパケットへのポインタ
戻り値	ID	ercd	正常終了（E_OK）またはエラーコード
pk_rcyc の内容（T_RCYC 型）			
	STAT	cycstat	周期ハンドラ状態
	RELTIM	lefttim	次に周期ハンドラが起動するまでの時間
	RELTIM	cycstim	周期ハンドラの起動周期
	VP_INT	exinf	周期ハンドラの拡張情報
	VB	name[16]	周期ハンドラの名称 (終端をヌル'¥0'とする文字列)
エラーコード	E_ID	不正 ID 番号（周期ハンドラ ID が不正）	
	E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録状態）	
	E_PAR	パラメータエラー（pk_rcyc が不正）	
	E_ILUSE	サービスコール不正使用（非セキュア移行タスクの呼出し）	

cycid で指定される ID 番号の周期ハンドラを対象に各種状態を参照し、未登録状態でない場合に限り、pk_rcyc で指定されるパケットに返します。

cycstat には、周期ハンドラの状態により次の値が返ります。

TCYC_STP	0x00	周期ハンドラは開始状態
TCYC_STA	0x01	周期ハンドラは停止状態

周期ハンドラが開始状態の場合の lefttim には、現在時刻から次の起動時刻までの相対時間を返します。停止状態の場合では-1 を返します。cycstim には、生成時に指定された起動周期が返ります。

6-13. システム状態管理機能

システム状態管理機能には、システムの状態を変更や参照するための機能として、以下の機能があります。

- タスクの優先順位を回転
- 実行状態のタスクを参照
- CPU ロック状態への移行と解除
- タスクディスパッチの禁止と解除
- コンテキストの参照
- システム状態の参照

【コーディング例】

システム状態管理機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
void MainTask(VP_INT stacode);
T_CTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
T_RSYS rsys;
BOOL state;

void MainTask(VP_INT stacode)
{
    ID tskid;

    get_tid(&tskid);
    rot_rdq(tskid);

    loc_cpu();
    state = sns_loc();
    state = sns_dpn();
    unl_cpu();

    dis_dsp();
    state = sns_dsp();
    state = sns_dpn();
    ena_dsp();

    ref_sys(&rsys);
}
```

6-13-1. タスクの優先順位の回転

書式	ER ercd = rot_rdq(PRI tskpri); ER ercd = irot_rdq(PRI tskpri);		
引数	PRI	tskpri	優先順位を回転する対象の優先度
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (tskpri が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

tskpri で指定される優先度のタスクの優先順位を回転します。具体的には、優先度のタスクの最高優先順位のタスクを最低優先順にします。本 RTOS では、レディキュー、つまり実行状態と実行可能状態のタスクにのみ影響を与えます。

tskpri に TPRI_SELF (=0) を指定すると、自タスクのベース優先度 (本 RTOS では現在優先度と一致) が対象優先度になります。ただし、非タスクコンテキストから呼出された場合は、自タスクの概念がないため E_PAR エラーが返ります。

本 RTOS では、irot_rdq と rot_rdq とは同じ実装をしています。

6-13-2. 実行状態のタスク ID の参照

書式	ER ercd = get_tid(ID *p_tskid); ER ercd = iget_tid(ID *p_tskid);		
引数	ID	*p_tskid	実行状態のタスク ID を 格納する領域へのポインタ
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	ID	tskid	実行状態のタスク ID
エラーコード	E_PAR	パラメータエラー (p_tskid が不正)	

実行状態のタスクの ID 番号を参照して tskid に返します。タスクからこのシステムコールを呼出した場合には、必ず、自タスクのタスク ID が返ります。ただし、セキュア移行タスクの場合はタスクの ID 番号に 0x8000 を追加した値が、非セキュア移行タスクの場合は TSK_NONE (=0) が返ります。

各種ハンドラなどの非タスクコンテキストから呼出した場合は、そのハンドラが起動する前に実行していたタスクが、ハンドラ内でそのタスクの状態を変更しない限り、プリエンプトされたタスクのタスク ID が返ります。なお、実行中タスクが存在しない場合は、TSK_NONE (=0) が返ります。また、注意事項として、セキュアの非タスクコンテキストから呼出し、セキュア移行タスクか非セキュアタスクが実行状態だった場合は、そのタスクの ID 番号に 0x8000 を追加した値が返ります。

本 RTOS では、iget_tid と get_tid とは同じ実装をしています。

6-13-3. CPU ロック状態への移行

書式	ER ercd = loc_cpu(void); ER ercd = iloc_cpu(void);		
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

CPU ロック状態に移行します。既に CPU ロック状態だった場合は、何もしません。

CPU ロック状態とは、割込みの禁止やディスパッチの禁止を含み、具体的には **primask** レジスタをセット (=1) した状態を言います。

タスク内で CPU ロック状態に移行した場合は、同時に特権モードに移行します。

本 RTOS では、iloc_cpu と loc_cpu とは同じ実装をしています。

6-13-4. CPU ロック状態の解除

書式	ER ercd = unl_cpu(void); ER ercd = iunl_cpu(void);		
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

CPU ロック状態を解除します。既に CPU ロック解除状態だった場合は、何もしません。

CPU ロック状態の解除とは、具体的には **primask** レジスタをクリア (=0) した状態を言います。

タスク内で CPU ロック状態を解除した場合は、タスク生成時の特権／非特権モードに戻ります。つまり、特権モードで生成したタスクではモードに変化はありませんが、非特権モードで生成したタスクは非特権モードに移行します。

本 RTOS では、iunl_cpu と unl_cpu とは同じ実装をしています。

6-13-5. ディスパッチの禁止

書式	ER ercd = dis_dsp(void);		
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_CTX	コンテキストエラー	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

ディスパッチ禁止状態に移行します。既にディスパッチ禁止状態だった場合は、何もしません。各種ハンドラなどの非タスクコンテキストから呼出した場合は、**E_CTX** エラーを返します。

6-13-6. ディスパッチの許可

書式	ER ercd = ena_dsp(void);		
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_CTX	コンテキストエラー	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

ディスパッチ許可状態に移行します。既にディスパッチ許可状態だった場合は、何もしません。各種ハンドラなどの非タスクコンテキストから呼出した場合は、E_CTX エラーを返します。

6-13-7. コンテキストの参照

書式	BOOL state = sns_ctx(void);		
戻り値	BOOL	state	コンテキストの種類

非タスクコンテキストから呼出された場合は TRUE を、タスクコンテキストから呼出された場合は FALSE を、state に返します。また、セキュア移行タスクから呼出された場合はタスクコンテキストの FALSE を、非セキュア移行タスクから呼出された場合は非タスクコンテキストの TRUE を返します。

6-13-8. CPU ロック状態の参照

書式	BOOL state = sns_loc(void);		
戻り値	BOOL	state	CPU ロック状態

CPU ロック状態で呼出された場合は TRUE を、CPU ロック解除状態で呼出された場合は FALSE を、state に返します。

6-13-9. ディスパッチ禁止状態の参照

書式	BOOL state = sns_dsp(void);		
戻り値	BOOL	state	ディスパッチ禁止状態

ディスパッチ禁止状態で呼出された場合は TRUE を、ディスパッチ許可状態で呼出された場合は FALSE を、state に返します。

6-13-10. ディスパッチ保留状態の参照

書式	BOOL state = sns_dpn(void);		
戻り値	BOOL	state	ディスパッチ保留状態

ディスパッチ保留状態で呼出された場合は TRUE を、ディスパッチ保留状態以外で呼出された場合は FALSE を、state に返します。

6-13-11. システムの状態参照

書式	ER ercd = ref_sys(T_RSYS *pk_rsys);		
引数	T_RSYS	*pk_rsys	システム状態を返すパケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード

pk_rsys の内容 (T_RSYS 型)

SIZE	free_sysmem	空きシステムメモリのサイズ (バイト数)
SIZE	free_usrmem	空きユーザーメモリのサイズ (バイト数)
UINT	used_task	生成済みタスク数
UINT	used_sem	生成済みセマフォ数
UINT	used_eflag	生成済みイベントフラグ数
UINT	used_mbox	生成済みメールボックス数
UINT	used_dtq	生成済みデータキュー数
UINT	used_mpf	生成済み固定長メモリプール数
UINT	used_mpl	生成済み可変長メモリプール数
UINT	used_isr	生成済み割込みサービスルーチン数
UINT	used_cyc	生成済み周期ハンドラ数

エラーコード	E_PAR	パラメータエラー (pk_rsys が不正)
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)

システムの状態を参照し、pk_rsys で指定されるパケットに返します。

システムメモリの空き領域サイズ、ユーザーメモリの空き領域サイズ、タスク、セマフォ、イベントフラグ、メールボックス、データキュー、固定長メモリプール、可変長メモリプール、割込みサービスルーチン、周期ハンドラのそれぞれの生成済みのオブジェクト数を、pk_rsys に返します。

6-14. 割込み管理機能

割込み管理機能とは、割込みハンドラ及び割込みサービスルーチンを管理する機能として、以下の機能があります。

- 割込みハンドラの定義
- 割込みサービスルーチンの生成と削除
- 割込みサービスルーチンの状態参照
- 割込みの禁止と許可
- 割込みマスクの変更と参照

【コーディング例】

割込み管理機能のシステムコールを用いたコーディング例です。システムコールの詳細な使い方は、それぞれの説明を参照してください。

```
#include "kernel.h"

#define MainTaskID 1
void MainTask(VP_INT stacode);
void IntHandler(void);
void IsrHandler(VP_INT stacode);
T_GTSK ctsk_main = { TA_HLNG|TA_ACT, 0, MainTask, 2, 0x2000, NULL, "MainTask" };
#define ISR_PRI 1
#define ISR_NO 16
T_CISR cisr1 = { TA_HLNG, 0, ISR_NO, IsrHandler, ISR_PRI, "IsrHandler1" };
#define INH_PRI 2
#define INH_NO 32
T_DINH dinh1 = { TA_HLNG, (FP)IntHandler, INH_PRI };
ID IsrID1 = 0;
T_RISR risr;

void MainTask(VP_INT stacode)
{
    ER_ID erid;
    ER ret;

    erid = acre_isr(&cisr1);
    if (erid > 0) {
        IsrID1 = erid;
        ena_int(ISR_NO);
        dly_tsk(1000);
        dis_int(ISR_NO);
    }
    ret = ref_isr(IsrID1, &risr);
    if (ret == E_OK) {
```

```
        del_isr(IsrID1);
    }

    def_inh(INH_NO, &dinh1);
    ena_int(INH_NO);
    dly_tsk(1000);
    def_inh(INH_NO, NULL);
    dis_int(INH_NO);
}

void IntHandler(void)
{
    ;
}

void IsrHandler(VP_INT stacode)
{
    ;
}
```

6-14-1. 割込みハンドラの定義

書式	ER ercd = def_inh(INHNO inhno, T_DINH *pk_dinh);		
引数	INHNO	inhno	定義対象の割込みハンドラ番号
	T_DINH	*pk_dinh	割込みハンドラ定義情報を格納した パケットへのポインタ
	pk_dinh の内容 (T_DINH 型)		
	ATR	inhatr	割込みハンドラ属性
	FP	inthdr	割込みハンドラの起動番地
	IMASK	imask	割込みハンドラの割込み優先度
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (inth、pk_dinh が不正)	
	E_OBJ	オブジェクト状態エラー (対象割込みの定義が不正)	

inhno で指定される割込み番号の割込みを対象とし、**pk_dinh** で指定される割込みハンドラ定義情報を基にして定義します。具体的には、**inthdr** で指定される起動番地の割込みハンドラを、**imask** で指定される割込み優先度で定義します。同一の割込み番号に複数の割込みハンドラを定義することはできません。また、**pk_dinh** に **NULL** を指定することにより、既に定義している割込みハンドラの定義を解除することができます。

割込みハンドラ属性には、本 RTOS では使用しないため、0 を指定します。割込み番号と割込み優先度は、プロセッサに依存するため、詳細は「デバイス依存部ユーザズガイド」を参照してください。

6-14-2. 割込みサービスルーチンの生成

書式	<pre>ER ercd = cre_isr(ID isrid, T_CISR *pk_cisr); ER_ID isrid = acre_isr(T_CISR *pk_cisr);</pre>		
引数	ID	isrid	生成対象の割込みサービスルーチン ID
	T_CISR	*pk_cisr	割込みサービスルーチン生成情報を格納した パケットへのポインタ
	pk_cisr の内容 (T_CISR 型)		
	ATR	isratr	割込みサービスルーチン属性
	VP_INT	exinf	割込みサービスルーチンの拡張情報
	INTNO	intno	割込みサービスルーチンの割込み番号
	FP	isr	割込みサービスルーチンの起動番地
	IMASK	imask	割込みサービスルーチンの割込み優先度
	VB	*name	割込みサービスルーチンの名称 (終端をヌル'¥0'とする文字列)
戻り値	cre_isr の場合		
	ER	ercd	正常終了 (E_OK) またはエラーコード
	acre_isr の場合		
	ER_ID	isrid	生成した割込みサービスルーチンの ID 番号 (正の値)、またはエラーコード
エラーコード	E_ID	不正 ID 番号 (割込みサービスルーチン ID が不正、cre_isr のみ)	
	E_NOID	ID 番号不足 (未登録状態の 割込みサービスルーチン ID がない、acre_isr のみ)	
	E_RSATR	予約属性 (isratr が不正)	
	E_PAR	パラメータエラー (pk_cisr、intno、isr、imask が不正)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_OBJ	オブジェクト状態エラー (対象割込みサービスルーチンが登録済み、cre_isr のみ)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

isrid で指定される ID 番号の割込みサービスルーチンを対象とし、pk_cisr で指定される割込みサービスルーチン生成情報を基にして生成し、対象割込みサービスルーチンを登録済み状態に遷移させます。isratr に割込みサービスルーチンは属性、exinf は割込みサービスルーチンを起動する際に引数、intno は割込みサービスルーチンを起動する割込み番号、imask で指定される割込み優先度、isr は割込みサービスルーチンの起動番地です。また、

同一の割込み番号に複数の割込みサービスルーチンを生成することができます。

割り込みサービスルーチン属性と割り込み番号と割り込み優先度は、アーキテクチャ及びプロセッサに依存するため、詳細は「デバイス依存部ユーザズガイド」を参照してください。

`acre_cyc` は、生成可能な周期ハンドラ ID を検索し割付け、その周期ハンドラ ID を返します。

name で指定する割込みサービスルーチンの名称は、ヌル（'¥0'）を終端とする 15 文字までの文字列とし、16 文字以上を指定した場合は、16 文字目以降は無視されます。**name** に NULL を指定することで割込みサービスルーチンの名称を省略することができますが、その場合は自動的に任意の名称が割付けられます。

6-14-3. 割込みサービスルーチンの削除

書式	ER ercd = del_isr(ID isrid);		
引数	ID	isrid	削除対象の割込みサービスルーチン ID
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_ID	不正 ID 番号 (割込みサービスルーチン ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象割込みサービスルーチンが未登録)	
	E_CTX	コンテキストエラー (非タスクコンテキストからの呼出し)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

isrid で指定される ID 番号の割込みサービスルーチンを削除し、対象割込みサービスルーチンを登録状態から未登録状態に遷移させます。

6-14-4. 割込みサービスルーチンの状態参照

書式	ER ercd = ref_isr(ID isrid, T_RISR *p_risr);		
引数	ID	isrid	参照対象の割込みサービスルーチン ID
	T_RISR	*pk_risr	割込みサービスルーチン状態を返す パケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
	pk_risr の内容 (T_RISR 型)		
	INTNO	intno	割込みサービスルーチンの割込み番号
	FP	isr	割込みサービスルーチンの起動番地
	VB	name[16]	割込みサービスルーチンの名称
	(終端をヌル'¥0'とする文字列)		
エラーコード	E_ID	不正 ID 番号 (割込みサービスルーチン ID が不正)	
	E_NOEXS	オブジェクト未生成 (対象割込みサービスルーチンが未登録状態)	
	E_PAR	パラメータエラー (pk_risr が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

isrid で指定される ID 番号の割込みサービスルーチンを対象に各種状態を参照し、未登録状態でない場合に限り、pk_risr で指定されるパケットに返します。

割込みサービスルーチン生成時に指定された値として、**intno** には割込み番号、**isr** には割込みサービスルーチンの起動番地、**name** には割込みサービスルーチンの名称が返ります。**name** には、割込みサービスルーチン生成時に指定された割込みサービスルーチンの名称を、指定されなかった場合は自動で生成された名称を返します。

6-14-5. 割込みの禁止

書式	ER ercd = dis_int(INTNO intno);		
引数	INTNO	intno	割込みを禁止する対象の割込み番号
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (intno が不正)	

`intno` で指定される割込み番号の割込みを禁止します。割込み番号は、プロセッサに依存するため、詳細は「デバイス依存部ユーザズガイド」を参照してください。

6-14-6. 割込みの許可

書式	ER ercd = ena_int(INTNO intno);		
引数	INTNO	intno	割込みを許可する対象の割込み番号
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (intno が不正)	

intno で指定される割込み番号の割込みを許可します。割込み番号は、プロセッサに依存するため、詳細は「デバイス依存部ユーザズガイド」を参照してください。

6-14-7. 割込みマスクの変更

書式	ER ercd = chg_ims(IMASK imask);		
引数	IMASK	imask	変更後の割込みマスク優先度
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
エラーコード	E_PAR	パラメータエラー (imask が不正)	
	E_ILUSE	サービスコール不正使用 (非セキュア移行タスクの呼出し)	

imask で指定される割込み優先度で割込みをマスクします。割込み優先度のマスクとは、具体的にはマスクしたい優先度を basepri レジスタに設定することを言います。また、優先度マスクの解除とは、basepri レジスタをクリア (=0) することを言います。

6-14-8. 割込みマスクの参照

書式	ER ercd = chg_ims(IMASK *p_imask);		
引数	IMASK	*p_imask	割込みマスク優先度を格納する領域へのポインタ
戻り値	ER	ercd	正常終了 (E_OK) またはエラーコード
	IMASK	imask	割込みマスク優先度
エラーコード	E_PAR	パラメータエラー (p_imask が不正)	

現在設定されている割込みマスク優先度を imask に返します。

6-15. システム構成管理機能

システム管理機能として、以下の機能があります。

- コンフィグレーション情報の参照
- バージョン情報の参照

6-15-1. コンフィグレーション情報の参照

書式	ER ercd = ref_cfg(T_RCFG *pk_rcfg);		
引数	T_RCFG	*pk_rcfg	コンフィグレーション情報を返す パケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
pk_rcfg の内容 (T_RCFG 型)			
SIZE	sysmem_size	初期システムメモリのサイズ (バイト数)	
SIZE	usrmem_size	初期ユーザーメモリのサイズ (バイト数)	
UINT	max_task	生成可能なタスク数	
UINT	max_sem	生成可能なセマフォ数	
UINT	max_eflag	生成可能なイベントフラグ数	
UINT	max_mbox	生成可能なメールボックス数	
UINT	max_dtq	生成可能なデータキュー数	
UINT	max_mpf	生成可能な固定長メモリプール数	
UINT	max_mpl	生成可能な可変長メモリプール数	
UINT	max_isr	生成可能な割込みサービスルーチン数	
UINT	max_cyc	生成可能な周期ハンドラ数	
エラーコード	E_PAR	パラメータエラー (pk_rcfg が不正)	

システムのコンフィグレーション情報を参照し、pk_rcfg で指定されるパケットに返します。システムメモリの初期の領域サイズ、ユーザーメモリの初期の領域サイズと、タスク、セマフォ、イベントフラグ、メールボックス、データキュー、固定長メモリプール、可変長メモリプール、割込みサービスルーチン、周期ハンドラのそれぞれの生成可能なオブジェクト数 (ID 番号の最大値) を、pk_rcfg に返します。

6-15-2. バージョン情報の参照

書式	ER ercd = ref_ver(T_RVER *pk_rver);		
引数	T_RVER	*pk_rver	バージョン情報を返すパケットへのポインタ
戻り値	ID	ercd	正常終了 (E_OK) またはエラーコード
pk_rver の内容 (T_RVER 型)			
	UH	maker	カーネルのメーカコード
	UH	prid	カーネルの識別コード
	UH	spver	ITRON 仕様のバージョン番号
	UH	prver	カーネルのバージョン番号
	UH	prno[4]	カーネル製品の管理番号
エラーコード	E_PAR	パラメータエラー (pk_rver が不正)	

カーネルのバージョン番号を参照し、pk_rver で指定されるパケットに返します。
メーカコードを maker に返しますが、本 RTOS ではメーカコードを取得していないため 0x0000 が返ります。ITRON 仕様のバージョン番号 spver に 0x5403 (本書の執筆時のバージョン番号) が返ります。
カーネルの識別番号を prid に、カーネルのバージョン番号を prver に、カーネルの製品の管理情報を prno に、それぞれ返します。これらは製品毎、或いはバージョンアップなどにより変わります。

第 7 章 付録

7-1. データの型

D	符号付き 64 ビット整数
W	符号付き 32 ビット整数
H	符号付き 16 ビット整数
B	符号付き 8 ビット整数
UD	符号なし 64 ビット整数
UW	符号なし 32 ビット整数
UH	符号なし 16 ビット整数
UB	符号なし 8 ビット整数
VD	データ型が定まらない 64 ビットの値
VW	データ型が定まらない 32 ビットの値
VH	データ型が定まらない 16 ビットの値
VB	データ型が定まらない 8 ビットの値
VP	データ型が定まらないものへのポインタ
FP	プログラムの起動番地 (ポインタ)
INT	符号付き 32 ビット整数
UINT	符号なし 32 ビット整数
BOOL	真偽値 (TRUE または FALSE)
FN	機能コード (符号付き整数)
ER	エラーコード (符号付き整数)
ID	オブジェクトの ID 番号 (符号付き整数)
ATR	オブジェクト属性 (符号なし整数)
STAT	オブジェクトの状態 (符号なし整数)
MODE	システムコールの動作モード (符号なし整数)
PRI	優先度 (符号付き整数)
SIZE	メモリ領域のサイズ (符号なし整数)
TMO	タイムアウト指定 (符号付き整数)
RELTIM	相対時間 (符号なし整数)
SYSTEMIM	システム時刻 (符号なし整数)
VP_INT	符号なし 32 ビット整数
ER_BOOL	エラーコードまたは真偽値
ER_ID	エラーコードまたは ID 番号
ER_UINT	エラーコード、または符号なし整数 (符号なし整数の有効ビット数は UINT より 1 ビット短い)

T_MSG	メールボックスのメッセージヘッダ
INHNO	割込みハンドラ番号
INTNO	割込み番号
IMASK	割込みマスク

7-2. パケット形式

A) タスク管理機能

タスク生成情報のパケット形式

```
typedef struct t_ctsk {
    ATR      tskatr;          /* タスク属性 */
    VP_INT   exinf;          /* タスクの拡張情報 */
    FP       task;           /* タスクの起動番地 */
    PRI      itskpri;        /* タスクの起動時優先度 */
    SIZE     stksz;          /* タスクのスタックサイズ (バイト数) */
    VP       stk;            /* タスクのスタック領域の先頭番地 */
    SIZE     sstksz;         /* タスクのセキュアスタック加算サイズ
                                (バイト数) */
    VB const  *name;         /* タスクの名称 */
} T_CTSK;
```

タスク生状態のパケット形式

```
typedef struct t_rtsk {
    STAT      tskstat;       /* タスク状態 */
    PRI      tskpri;         /* タスクの現在優先度 */
    PRI      tsbpri;         /* タスクのベース優先度 */
    STAT      tskwait;       /* 待ち要因 */
    ID        wobjid;        /* 待ち対象のオブジェクトの ID 番号 */
    TMO       lefttmo;       /* タイムアウトするまでの時間 */
    UINT      actcnt         /* 起動要求キューイング数 */
    UINT      wupcnt;        /* 起床要求キューイング数 */
    UINT      suscnt;        /* 強制待ち要求ネスト数 */
    VB        name[16];      /* タスクの名称 */
} T_RTSK;
```

タスク生状態（簡易版）のパケット形式

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

B) セマフォ機能

セマフォ生成情報のパケット形式

```
typedef struct t_csem {
    ATR      sematr;          /* セマフォ属性 */
    UINT      isemcnt;        /* セマフォの資源数の初期値 */
    UINT      maxsem;         /* セマフォの最大資源数 */
    VB const  *name;          /* セマフォの名称 */
} T_CSEM;
```

セマフォ状態のパケット形式

```
typedef struct t_rsem {
    ID        wtskid;         /* セマフォ待ち行列の先頭タスク ID */
    UINT      semcnt;         /* セマフォの現在の資源数 */
    VB        name[16];       /* セマフォの名称 */
} T_RSEM;
```

C) イベントフラグ機能

イベントフラグ生成情報のパケット形式

```
typedef struct t_cflg {
    ATR      flgatr;          /* イベントフラグ属性 */
    FLGPTN   iflgptn;         /* イベントフラグの初期ビットパターン */
    VB const  *name;          /* イベントフラグの名称 */
} T_CFLG;
```

イベントフラグ状態のパケット形式

```
typedef struct t_rflg {
    ID        wtskid;         /* イベントフラグ待ち行列の
                                先頭タスク ID */
    FLGPTN   flgptn;         /* イベントフラグの現在のビットパターン */
    VB        name[16];       /* イベントフラグの名称 */
} T_RFLG;
```

D) データキュー機能

データキュー生成情報のパケット形式

```
typedef struct t_cdtq {
    ATR      dtqatr;          /* データキュー属性 */
    UINT      dtqcnt;         /* データキュー領域の容量（データ個数） */
    VP        dtq;            /* データキュー領域の先頭番地 */
    VB const  *name;          /* データキューの名称 */
} T_CDTQ;
```

データキュー状態のパケット形式

```
typedef struct t_rdtq {
    ID          stskid;          /* データキュー送信待ち行列の
                                   先頭タスク ID */
    ID          rtskid;          /* データキュー受信待ち行列の
                                   先頭タスク ID */
    UINT         sdtqcnt;        /* データキューに格納されている
                                   データ数 */
    VB          name[16];        /* データキューの名称 */
} T_RDTQ;
```

E) メールボックス機能

メールボックス生成情報のパケット形式

```
typedef struct t_cmbx {
    ATR          mbxatr;         /* メールボックス属性 */
    PRI          mbxmpri;        /* 送信できるメッセージの
                                   最大優先度（不使用） */
    VP          mprihd;         /* 優先度別のメッセージキューヘッダ領域の
                                   先頭番地（不使用） */
    UINT         msgmax;         /* 格納できる最大メッセージ数 */
    VB const     *name;          /* メールボックスの名称 */
} T_CMBX;
```

メールボックス状態のパケット形式

```
typedef struct t_rmbx {
    ID          wtskid;          /* メッセージ待ち行列の先頭タスク ID */
    T_MSG       *pk_msg;         /* メッセージキュー先頭の
                                   メッセージパケットの先頭番地 */
    UINT         msgcnt;         /* メールボックス内のメッセージ数 */
    VB          name[16];        /* メールボックスの名称 */
} T_RMBX;
```


F) ミューテックス管理機能

ミューテックス生成情報のパケット形式

```
typedef struct t_cmtx {
    ATR      mtxatr;          /* ミューテックス属性 */
    PRI      ceilpri;         /* ミューテックスの上限優先度 */
    VB const  *name;          /* ミューテックスの名称 */
} T_CMTX;
```

ミューテックス状態のパケット形式

```
typedef struct t_rmtx {
    ID      htsskid;          /* ミューテックスをロックしている
                               タスク ID */
    ID      wtsskid;          /* ミューテックスの待ち行列の先頭
                               タスク ID */
    VB      name[16];         /* ミューテックスの名称 */
} T_RMTX;
```

G) メッセージバッファ管理機能

メッセージバッファ生成情報のパケット形式

```
typedef struct t_cmbf {
    ATR      mpfatr;          /* メッセージバッファ属性 */
    UINT      maxmsz;         /* メッセージの最大サイズ (バイト数) */
    SIZE      mbfsz;          /* メッセージバッファ領域のサイズ
                               (バイト数) */
    VP      mbf;              /* メッセージバッファ領域の先頭番地 */
    VB const  *name;          /* メッセージバッファの名称 */
} T_CMBF;
```

メッセージバッファ状態のパケット形式

```
typedef struct t_rmbf {
    ID      stsskid;          /* メッセージバッファの送信待ち行列の
                               先頭タスク ID */
    ID      rtsskid;          /* メッセージバッファの受信待ち行列の
                               先頭タスク ID */
    UINT      msgcnt;         /* メッセージバッファに入っている
                               メッセージの数 */
    SIZE      fmbfsz;         /* メッセージバッファの空きサイズ
                               (バイト数) */
}
```

```

        VB          name[16];          /* メッセージバッファの名称 */
    } T_RMBF;

```

H) 固定長メモリプール管理機能

固定長メモリプール生成情報のパケット形式

```

typedef struct t_cmpf {
    ATR          mpfatr;          /* 固定長メモリプール属性 */
    UINT         blkcnt;          /* 生成するメモリブロック個数 */
    UINT         blkksz;          /* メモリブロックのサイズ（バイト数） */
    VP           mpf;             /* 固定長メモリプール領域の先頭番地 */
    VB const     *name;           /* 固定長メモリプールの名称 */
} T_CMPF;

```

固定長メモリプール状態のパケット形式

```

typedef struct t_rmpf {
    ID           wtskid;          /* 固定長メモリブロック待ち行列の
                                   先頭タスク ID */
    UINT         fblkcnt;         /* 固定長メモリプールの
                                   空きメモリブロック数 */
    VB           name[16];        /* 固定長メモリプールの名称 */
} T_RMPF;

```

I) 可変長メモリプール管理機能

可変長メモリプール生成情報のパケット形式

```

typedef struct t_cmpl {
    ATR          mplatr;          /* 可変長メモリプール属性 */
    UINT         mplksz;          /* 可変長メモリプール領域のサイズ
                                   （バイト数） */
    VP           mpl;             /* 可変長メモリプール領域の先頭番地 */
    UINT         blkmax;          /* 可変長メモリプールの
                                   獲得可能ブロック数 */
    VB const     *name;           /* 可変長メモリプールの名称 */
} T_CMPL;

```

可変長メモリプール状態のパケット形式

```

typedef struct t_rmpf {
    ID          wtskid;          /* 可変長メモリブロック待ち行列の
                                先頭タスク ID */
    SIZE        fmplsz;          /* 可変長メモリプールの空き領域の
                                合計サイズ (バイト数) */
    UINT        fblksiz;        /* すぐに獲得可能な最大メモリブロック
                                サイズ (バイト数) */
    VB          name[16];        /* 可変長メモリプールの名称 */
} T_RMPF;

```

J) 周期ハンドラ機能

周期ハンドラ生成情報のパケット形式

```

typedef struct t_ccyc {
    ATR         cycatr;          /* 周期ハンドラ属性 */
    VP_INT      exinf;           /* 周期ハンドラの拡張情報 */
    FP          cychdr;          /* 周期ハンドラの起動番地 */
    RELTIM      cyctim;          /* 周期ハンドラの起動周期 */
    RELTIM      cycphs;          /* 周期ハンドラの起動位相 */
    VB const    *name;           /* 周期ハンドラの名称 */
} T_CCYC;

```

周期ハンドラ状態のパケット形式

```

typedef struct t_rcyc {
    STAT        cycstat;         /* 周期ハンドラの動作状態 */
    RELTIM      lefttim;         /* 次に周期ハンドラが
                                起動するまでの時間 */
    RELTIM      cyctim;          /* 周期ハンドラの起動周期 */
    VP_INT      exinf;           /* 周期ハンドラの拡張情報 */
    VB          name[16];        /* 周期ハンドラの名称 */
} T_RCYC;

```

K) システム状態管理機能

システム状態の packets 形式

```
typedef struct t_rsys {
    SIZE    free_sysmem;    /* 空きシステムメモリサイズ (バイト数) */
    SIZE    free_usrmem;    /* 空きユーザーメモリサイズ (バイト数) */
    UINT    used_task;      /* 生成済みのタスク数 */
    UINT    used_sem;       /* 生成済みのセマフォ数 */
    UINT    used_eflg;      /* 生成済みのイベントフラグ数 */
    UINT    used_mbox;      /* 生成済みのメールボックス数 */
    UINT    used_dtq;       /* 生成済みのデータキュー数 */
    UINT    used_mtx;       /* 生成済みのミューテックス数 */
    UINT    used_mbf;       /* 生成済みのメッセージバッファ数 */
    UINT    used_mpf;       /* 生成済みの固定長メモリプール数 */
    UINT    used_mpl;       /* 生成済みの可変長メモリプール数 */
    UINT    used_isr;       /* 生成済みの割り込みサービスルーチン数 */
    UINT    used_cyc;       /* 生成済みの周期ハンドラ数 */
} T_RSYS;
```

L) 割り込み管理機能

割り込みハンドラ定義情報の packets 形式

```
typedef struct t_dinh {
    ATR      inhatr;        /* 割り込みハンドラ属性 */
    FP       inthdr;        /* 割り込みハンドラの起動番地 */
    IMASK     imask;        /* 割り込みハンドラの割り込み優先度 */
} T_DINH;
```

割り込みサービスルーチン生成情報の packets 形式

```
typedef struct t_cisr {
    ATR      isratr;        /* 割り込みサービスルーチン属性 */
    VP_INT   exinf;        /* 割り込みサービスルーチンの拡張情報 */
    INTNO    intno;        /* 割り込みサービスルーチンの割り込み番号 */
    FP       isr;          /* 割り込みサービスルーチンの起動番地 */
    IMASK     imask;        /* 割り込みサービスルーチンの
                                割り込み優先度 */
    VB const *name;        /* 割り込みサービスルーチンの名称 */
} T_CISR;
```

割込みサービスルーチン状態のパケット形式

```
typedef struct t_risr {
    INTNO    intno;          /* 割込みサービスルーチンの割込み番号 */
    FP       isr;           /* 割込みサービスルーチンの起動番地 */
    VB       name[16];      /* 割込みサービスルーチンの名称 */
} T_RISR;
```

M) システム構成管理機能

コンフィグレーション情報のパケット形式

```
typedef struct t_rcfg {
    SIZE      sysmem_size;   /* システムメモリ領域サイズ (バイト数) */
    SIZE      usrmem_size;  /* ユーザーメモリ領域サイズ (バイト数) */
    UINT      max_task;     /* タスク ID の最大値 */
    UINT      max_sem;      /* セマフォ ID の最大値 */
    UINT      max_eflg;     /* イベントフラグ ID の最大値 */
    UINT      max_mbox;     /* メールボックス ID の最大値 */
    UINT      max_dtq;      /* データキューID の最大値 */
    UINT      max_mtx;      /* ミューテックス ID の最大値 */
    UINT      max_mbf;      /* メッセージバッファ ID の最大値 */
    UINT      max_mpf;      /* 固定長メモリプール ID の最大値 */
    UINT      max_mpl;      /* 可変長メモリプール ID の最大値 */
    UINT      max_isr;      /* 割込みサービスルーチン ID の最大値 */
    UINT      max_cyc;      /* 周期ハンドラ ID の最大値 */
} T_RCFG;
```

バージョン情報のパケット形式

```
typedef struct t_rver {
    UH        maker;        /* カーネルのメーカーコード */
    UH        prid;         /* カーネルの識別番号 */
    UH        spver;        /* ITRON 仕様のバージョン番号 */
    UH        prver;        /* カーネルのバージョン番号 */
    UH        prno[4];      /* カーネル製品の管理番号 */
} T_RVER;
```

7-3. 定数とマクロ

A) 一般

NULL	0	無効ポインタ
TRUE	1	真
FALSE	0	偽
E_OK	0	正常終了

B) オブジェクト属性

TA_NULL	0	オブジェクト属性を指定しない
TA_HLNG	0x00	高級言語用のインタフェースで処理を起動
TA_ASM	0x01	アセンブラ言語用のインタフェースで処理を起動
TA_TFIFO	0x00	タスクの待ち行列は FIFO 順
TA_TPRI	0x01	タスクの待ち行列はタスク優先度順
TA_MFIFO	0x00	メッセージのキューは FIFO 順
TA_MPRI	0x02	メッセージのキューはメッセージの優先度順
TA_ACT	0x02	タスクを起動された状態で生成
TA_WSGL	0x00	イベントフラグを複数のタスクが待つことは不許可
TA_WMUL	0x02	イベントフラグを複数のタスクが待つことを許可
TA_CLR	0x04	待ち解除時にイベントフラグをクリア
TA_STA	0x02	周期ハンドラを動作している状態で生成
TA_PHS	0x04	周期ハンドラの位相を保存

C) タイムアウト指定

TMO_POL	0	ポーリング
TMO_FEVR	-1	永久待ち

D) システムコールの動作モード

TWF_ANDW	0x00	イベントフラグの AND 待ち
TWF_ORW	0x01	イベントフラグの OR 待ち

E) オブジェクトの状態

TTS_RUN	0x01	実行状態
TTS_RDY	0x02	実行可能状態
TTS_WAI	0x04	待ち状態
TTS_DMT	0x08	休止状態
TTW_SLP	0x0001	起床待ち状態
TTW_DLY	0x0002	時間経過待ち状態
TTW_SEM	0x0004	セマフォ資源の獲得待ち状態

TTW_FLG	0x0008	イベントフラグ待ち状態
TTW_SDTQ	0x0010	データキューへの送信待ち状態
TTW_RDTQ	0x0020	データキューからの受信待ち状態
TTW_MBX	0x0040	メールボックスからの受信待ち状態
TTW_MPF	0x2000	固定長メモリプールの獲得待ち状態
TTW_MPL	0x4000	可変長メモリプールの獲得待ち状態
TCYC_STP	0x00	周期ハンドラは動作中
TCYC_STA	0x01	周期ハンドラは停止中

F) その他の定数

TSK_SELF	0	自タスクの指定
TSK_NONE	0	該当タスクなし
TPRI_SELF	0	自タスクのベース優先度を指定
TPRI_INI	0	タスクの起動時優先度を指定

7-4. 構成定数とマクロ

A) 優先度の範囲

TMIN_TPRI タスク優先度の最小値（固定値：1）
TMAX_TPRI タスク優先度の最大値（固定値：128）

B) バージョン情報

TKERNEL_MAKER カーネルのメーカコード
TKERNEL_PRID カーネルの識別番号
TKERNEL_SPVER ITRON 仕様のバージョン番号
TKERNEL_PPVER カーネルのバージョン番号

C) キューイング／ネスト回数の最大値

TMAX_ACTCNT タスク起動要求キューイング数の最大値（固定値：255）
TMAX_WUPCNT タスク起床要求キューイング数の最大値（固定値：255）
TMAX_SUSCNT タスク強制待ち要求ネスト数の最大値（固定値：255）

D) ビットパターンビット数

TBIT_FLGPTN イベントフラグのビット数（固定値：32）

E) 必要なメモリのサイズ

SIZE dtqsz = TSZ_DTQ(UINT dtqcnt)
 dtqcnt 個のデータを格納するのに必要なデータキュー領域の
 サイズ（バイト数）

SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blkksz)
 blkksz バイトのメモリブロックを blkcnt 個生成できる固定長メ
 モリプール領域のサイズ（バイト数）

SIZE mpls = TSZ_MPL(UINT blkcnt, UINT blkksz)
 blkksz バイトのメモリブロックを blkcnt 個生成できる可変長メ
 モリプール領域のサイズ（目安のバイト数）

F) その他

MAX_DTQ_COUNT データキューのデータ個数の最大値（固定値：32767）
MAX_MPL_COUNT 固定長メモリアンプールのメモリブロック個数の最大値
 （固定値：32767）
TMAX_MAXSEM セマフォの最大資源数の最大値（固定値：65535）
TMAX_TMO TMO 型タイムアウト値の最大値（固定値：2000000）
TMAX_RELTIM RELTIM 型相対時間の最大値（固定値：4000000）

7-5. エラーコード

E_NG	-1	規定されていないエラーコード
E_SYS	-5	システムエラー
E_NOSPT	-9	未サポート機能
E_RSFN	-10	予約機能コード
E_RSATR	-11	予約属性
E_PAR	-17	パラメータエラー
E_ID	-18	不正 ID エラー
E_CTX	-25	コンテキストエラー
E_ILUSE	-28	サービスコール不正使用
E_NOMEM	-33	メモリ不足
E_NOID	-34	ID 番号不足
E_OBJ	-41	オブジェクト状態エラー
E_NOEXS	-42	オブジェクト未生成
E_QOVR	-43	キューイングオーバーフロー
E_RLWAI	-49	待ち状態の強制解除
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_DLT	-51	待ちオブジェクトの削除

(余白)

Cortex-M 対応 MULCOS-MK

μ ITRON4.0 仕様版共通部簡易ユーザーズガイド

2025 年 1 月 第 1 版発行

株式会社スパイラルテック

Mail tech-info@spiral-tech.co.jp