μT-Kernel3.0
共通実装仕様書

Version.01.00.01

2022.7.11

目次

1. 概要

1.1 目的

本仕様書はトロンフォーラムから公開されているμT-Kernel3.0（V3.00.00）のソースコードをルネサスエレクトロニクス社のRX用に改変したソースコードの実装仕様を記載した実装仕様書である。

なお、本仕様書はハードウェアに依存しない共通仕様のみを記載する。ハードウェアに依存する実装仕様は各ハードウェア向けのμT-Kernel3.0実装仕様書を参照のこと。

以降、単にOSと称する場合はμT-Kenrel3.0を示し、本実装と称する場合は前述の改変したソースコードの実装を示す。

1.2 実装の基本方針

本実装の基本方針を以下に示す。

- アドレス空間は単一の物理アドレスのみに対応し、MMUを用いた仮想アドレスやメモリ保護などには対応しない。
- OSのプログラムは極力C言語で記述し、またC言語処理系以外の特定の開発ツールの使用を前提としない。
- OSおよびOS上で実行されるプログラムは実行モードを特権モードのみとし、静的にリンクされた一つの実行オブジェクトとする。これを前提にAPIは関数呼び出しのみとする。

1.3 バージョン情報

本実装のバージョン情報を以下に示す。この情報はtk_ref_verコールで取得される。

種別	変数	値	備考
メーカコード	maker	0x0008	個人
識別番号	prid	0x0003	
仕様書バージョン番号	spver	0x6300	μT-Kenrel3.00仕様
カーネルバージョン番号	prver	0x0000	
製品管理情報	prno[0] - [3]	0x5258 0x2020 0x4343 0x5258	RX　CCRX

1.4 ターゲット名

OSを構築するのにあたって、対象ハードウェア毎に固有のターゲット名および派生する識別名を定める。これらは機種依存部の条件コンパイルの識別子として使用される。

ターゲット名はCC-RXのコンパイルオプション（マクロ定義）で定義される。ターゲット名から派生して他の識別名が定義される。この定義は/include/sys/machine.h(inc)に記述される。

具体的なターゲット名は各ハードウェア向けのμT-Kernel3.0実装仕様書を参照のこと。

1.5 関連ドキュメント

OSの標準的な仕様は「μT-Kernel3.0仕様書」に記載される。また、ハードウェアに依存する実装仕

様は各ハードウェア向けのμT-Kernel3.0実装仕様書に記載される。

以下の関連するドキュメントを記す。

分類	名称	発行
OS仕様	μT-Kernel3.0仕様書 （Ver.3.00.00）	TRONフォーラム TEF020-S004-3.00.00
T-Monitor	T-Monitor仕様書	TRONフォーラム TEF-020-S002-01.00.01
ハードウェア 依存実装仕様	uTK3.0_AP-RX63N-0A実装仕様書	
	uTK3.0_AP-RX65N-0A実装仕様書	
	uTK3.0_AP-RX72N-0A実装仕様書	
	uTK3.0_GR-SAKURA実装仕様書	

1.6　ソースコード構成

本実装のソースコードのディレクトリ構成を以下に示す。

斜字のディレクトリはμT-Kernel3.0のソースコードに含まれない。

```
┬ config                コンフィギュレーション
├ include               インクルードファイル
│  ├ sys                        システム定義
│  │  └ sysdepend                       ハードウェア依存部
│  ├ tk                         OS関連定義
│  │  └ sysdepend                       ハードウェア依存部
│  ├ lib                        ライブラリ関連定義
│  └ tm                         T-Monitor 関連定義
├ kernel                OSソースコード
│  ├ knlinc                     OS内共通定義
│  ├ tstdlib                    OS内共通ライブラリ
│  ├ sysinit                    初期化処理
│  ├ inittask                   初期タスク
│  ├ tkernel                    OS機能
│  ├ sysdepend                  ハードウェア依存部
│  └ usermain                   ユーザメイン処理
├ lib                   ライブラリ
│  ├ libtk                      μT-Kernelライブラリ
│  └ libtm                      T-Monitorライブラリ
├ app_sample            アプリケーション（サンプル）
├ docs                  ドキュメント
└ others                その他
```

機種依存定義sysdependディレクトリは以下のように構成される。具体的には各ハードウェアの実装仕様書を参照のこと。

```
── sysdepend                          実装依存定義
    ├ <ターゲット1>                        ターゲット1 依存部
    ├ :
    ├ <ターゲットn>                        ターゲットn 依存部
    └ cpu                              CPU 依存部
        ├ <CPU1>                           CPU1 依存部
        ├ :
        └ <CPUn>                           CPUn 依存部
            └ core                         コア依存部
                ├ <core 1>                     コア1 依存部
                ├ :
                └ <core n>                     コアn 依存部
```

2.　基本実装仕様

2.1　対象マイコン

本実装は対象マイコンとして以下を想定している。

- 32ビットまたは16ビットマイコン
- 単一の物理アドレス空間（MMUは無し、または使用しない）

2.2　実行モードと保護レベル

本実装ではマイコンの実行モードは特権（スーパーバイザ）モードのみを使用し、非特権（ユーザ）モードは使用しない。よって、プログラムを実行するモードは基本的に同一である。具体的なマイコンの実行モードは実装するマイコンに依存する。各ハードウェア用の実装仕様書を参照のこと。

OSが提供する保護レベルはマイコンの実行モードが特権モードのみなので、すべて保護レベル0と同等となる。カーネルオブジェクトに対して保護レベル1～3を指定しても、メモリ保護は保護レベル0を指定されたものと同じになる。

プロファイル TK_MEM_RING0 ～ TK_MEM_RING3 はすべて 0 が返される。

2.3　CPU レジスタ

OSが扱うレジスタは実装するマイコンに依存する。各ハードウェア用の実装仕様書を参照のこと。

2.4　低消費電力モードと省電力機能

省電力機能は実装するマイコンに依存する。各ハードウェア向けの実装仕様書を参照のこと。

2.5　コプロセッサ対応

コプロセッサ対応は実装するマイコンに依存する。各ハードウェア向けの実装仕様書を参照のこと。

3. メモリ

3.1 メモリモデル

本実装では単一の物理アドレス空間を前提とする。アドレス空間上にはプログラムから使用可能なROMおよびRAMが存在する。

プログラムは静的にリンクされた一つの実行オブジェクトを前提とする。OSとユーザプログラム（アプリケーションなど）は静的にリンクされており、関数呼び出しが可能とする。

3.2 マイコンのアドレス・マップ

マイコンのアドレス・マップは実装するマイコンに依存する。各ハードウェア向けの実装仕様書を参照のこと。

3.3 OS のメモリマップ

本実装ではプログラムコードはROMに置くことを想定している。ただし、初期化処理においてRAM上に置いたプログラムコードをRAMに転送して使用することは可能である。

以下に本実装における一般的なROMおよびRAMの内容（用途）を示す。具体的なメモリマップは各ハードウェア向けの実装仕様書を参照のこと。

(1) ROMの用途

種別	内容
ベクタテーブル	例外や割込みのベクタテーブル 具体的な仕様はマイコンに依存する
プログラムコード	C言語のプログラムコードが配置される領域
定数データ	C言語の定数データなどが配置される領域

(2) RAMの用途

種別	内容
プログラムデータ	C言語の変数等が配置される領域
OS管理メモリ領域	OS内部の動的メモリ管理の領域

3.4 スタック

本実装では以下の種類のスタックを使用する。なお、具体的なスタックの仕様は各ハードウェア向けの実装仕様書を参照のこと。

(1) タスクスタック

タスクが使用するスタックであり、タスク毎に存在する。各タスクの生成時に指定され、OSが管理するシステムメモリ領域から動的に確保される。ただし、ユーザが確保した領域を使用することも可能である。

本実装では、すべてのプログラムは特権モードで実行しているので、システムスタックとユーザスタックの分離は行わない。一つのタスクに一つのタスクスタックが存在する。

プロファイル TK_HAS_SYSSTACK は FALSE である。

(2) 例外スタック

例外処理およびOS初期化処理の実行中に使用するスタックである。

コンフィギュレーション情報（CFN_EXC_STACK_SIZE）にてサイズが指定され、メモリマップ上に静的に領域が確保される。

3.5 OS 内の動的メモリ管理

本実装ではOS内で使用するメモリの動的な管理を行うことができる。

OSは必要に応じて、OS管理メモリ領域からメモリを確保し、また使用後はメモリを返却する。具体的には、以下のメモリ領域が動的管理の対象となる。

- タスクのスタック領域
- 固定長メモリプール領域
- 可変長メモリプール領域
- メッセージバッファ領域

OS内の動的メモリ管理は、コンフィギュレーションの USE_IMALLOC を 1 に設定することにより有効となる（初期値は有効）。コンフィギュレーションにて USE_IMALLOC を 0 の無効に設定した場合、OS内の動的メモリ管理の機能は使用できなくなる。その場合、上記のカーネルオブジェクトを生成する際にはTA_USERBUF属性（ユーザ指定のメモリ領域を使用）を指定しなくてはならない。

プログラムのコードやデータが割り当てられていないRAMの領域が、OS管理メモリ領域に割り当てられる。標準の設定では、すべてのRAMの空き領域をOS管理メモリ領域としている。

OS管理メモリ領域は、コンフィギュレーションの CFN_SYSTEMAREA_TOP と CFN_SYSTEMAREA_END により、先頭アドレスと最終アドレスを指定することで調整が可能である。また、値に 0 を指定することにより、標準の設定とすることができる。

具体的な仕様は各ハードウェア向けの実装仕様書を参照のこと。

3.6 システムメモリ管理機能

本実装ではシステムメモリ管理機能をサポートしない。

よって、μT-Kernel/SMのメモリ割り当てライブラリ関数（Kmalloc/Kcalloc/Krealloc/Kfree）は提供されない。プロファイル TK_SUPPORT_MEMLIB は FALSE である。

4.　割込みおよび例外

4.1　マイコンの割込みおよび例外

マイコンには各種の割込みおよび例外が存在する。具体的には各ハードウェア向けの実装仕様書を参照のこと。なお、OSの仕様上は割込み、例外をまとめて割込みと称している。

4.2　ベクタテーブル

対象マイコンの仕様に依存する。各ハードウェア向けの実装仕様書を参照のこと。

4.3　割込み優先度とクリティカルセクション

4.3.1　割込み優先度

対象マイコンの仕様に依存する。各ハードウェア向けの実装仕様書を参照のこと。

4.3.2　多重割込み対応

対象マイコンの仕様に依存する。各ハードウェア向けの実装仕様書を参照のこと。

4.3.3 クリティカルセクション

OS内部処理において不可分に実行しなければならない箇所をクリティカルセクションと呼ぶ。クリティカルセクションでは原則割込みは禁止である。一般にクリティカルセクション中は、割込みのマスクレベルを最高に設定することにより実現される。

具体的な実装は各ハードウェア向けの実装仕様書を参照のこと。

4.4　OS 内部で使用する割込み

OS内部の処理において、以下に示す割込みを使用する。具体的な割込みや例外への割り当て、実装などは各ハードウェア向けの実装仕様書を参照のこと。

種類	説明
割込みハンドラのハンドリング	ソフトウェア割込み
システムタイマ割込み	システムタイマによるハードウェア割込み

割込みハンドラのハンドリングにソフトウェア割込みを使用する。

システムタイマ割込みは、システムタイマにより周期的に発生するハードウェア割込みである。これによりOSの時間管理機能が実行される。

なお、本実装ではシステムコールは関数呼び出しのみであり、SVC割込みは使用しない。

4.5　μT-Kenrel/OS の割込み管理機能

各ハードウェア向けの実装仕様書を参照のこと。

4.6　μT-Kernel/SM の割込み管理機能

各ハードウェア向けの実装仕様書を参照のこと。

4.7　OS 管理外割込み

OSが管理する割込みよりも優先度の高い割込みをOS管理外割込みと呼ぶ。

OS管理外割込みはクリティカルセクションにおいても受け付けられる。つまり、管理外割込みの処理は、OS自体の動作よりも優先して実行される。このため、OS管理外割込みの中でOSのAPIなどの機能を使用することは原則としてできない。

OS管理外割込みは非常に応答性を要求される割込みなどに使用される。具体的な実装は各ハードウェア向けの実装仕様書を参照のこと。

5.　起動および終了処理

5.1　システム起動処理

本OSは電源投入またはリセットから以下の処理手順で起動する。

(1) リセット処理

マイコンのリセット後に実行し、ハードウェアの最小限の初期化とプログラム実行環境の初期化（C言語の変数領域の初期化など）を実行する。本処理の終了後、OSのmain関数が実行される。

本処理の具体的な内容はハードウェア依存である。各ハードウェア向けの実装仕様書を参照のこと。

(2) OS初期化処理（main）

OS初期化処理は、/kernel/sysinit/sysinit.cのmain関数にて以下の処理が実行される。ただし、コンフィギュレーションなどにより実行されない処理もある。

(2-1) T-Monitor互換ライブラリの初期化（libtm_init）

T-Monitor互換ライブラリを使用する場合に初期化を行う。

(2-2) OS内部動的管理メモリの初期化（knl_init_Imalloc）

OS内部の動的メモリ管理を使用する場合に初期化を行う。

(2-3) デバイス初期化（knl_init_device）

周辺デバイスの初期化処理を行う。

(2-4) 割込み初期化（knl_init_interrupt）

OSが使用する割込みの初期化を行う。

(2-5) カーネルオブジェクト初期化（knl_init_object）

各カーネルオブジェクトの初期化を行う。

(2-6) システムタイマ初期化（knl_timer_startup）

システムタイマの初期化および実行を開始する。

(2-7) 初期タスクの生成

初期タスクの生成およびその実行を開始する。

OS初期化処理の終了後、OSは通常の実行を開始し、最初のタスクとして初期タスクの実行が開始される。

5.2　初期タスク

5.2.1　初期タスクの設定

初期タスクはOS初期化の終了後に最初に実行されるタスクである。

初期タスクの生成情報knl_init_ctskは/include/sys/inittask.hで以下のように定義されている。なお、タスク属性の TA_DSNAME とDSオブジェクト名称はコンフィグレーションによって未指定、NULL となる場合があります。

- OS内動的メモリ管理を使用するの場合（USE_IMALLOC = 1）

項目	定義名	値
拡張情報	INITTASK_EXINF	0
タスク属性	INITTASK_TSKATR	TA_HLNG | TA_RNG0 | TA_DSNAME
タスク起動時優先度	INITTASK_ITSKPRI	1
スタックサイズ	INITTASK_STKSZ	1024
DSオブジェクト名称	INITTASK_DSNAME	"inittsk"
ユーザバッファポインタ	INITTASK_STACK	NULL

- OS内動的メモリ管理を使用しないの場合（USE_IMALLOC = 0）

項目	定義名	値
拡張情報	INITTASK_EXINF	0
タスク属性	INITTASK_TSKATR	TA_HLNG | TA_RNG0 | TA_DSNAME | TA_USERBUF
タスク起動時優先度	INITTASK_ITSKPRI	1
スタックサイズ	INITTASK_STKSZ	1024
DSオブジェクト名称	INITTASK_DSNAME	"inittsk"
ユーザバッファポインタ	INITTASK_STACK	init_task_stackへのポインタ（OS内部変数）

5.2.2　初期タスクの処理

初期タスクの実行関数は、/kernel/inittask/inittask.cのinit_task_main関数として定義される。実行関数init_task_mainの処理を以下に記す。

(1) システム起動処理（start_system）

(1-1) サブシステムの実行

サブシステムの実行を開始する。OS内で使用されるデバイス管理サブシステムの実行が開始される。なお、デバイス管理サブシステムは本実装では、サブシステムではなく、通常のOSの処理の一つとして実装されている。

(1-2) デバイスの実行（knl_start_device）

デバイスドライバの登録、実行を行う。

(2) ユーザプログラムの実行

(2-2) ユーザ定義初期化プログラムの実行（userinit）

ユーザ定義初期化プログラムuserinitが設定されていれば実行する。

(2-3) ユーザ定義メイン関数の実行（usermain）

ユーザプログラム（アプリケーション）のメイン関数である。

(3) システム終了処理（shutdown_system）

ユーザプログラムが終了するとシステムの終了処理を実行する。

処理内容は「5.3 システム終了処理」を参照のこと。

5.2.3 ユーザ定義メイン関数 usermain

ユーザ定義メイン関数usermainは以下の形式のC言語の関数である。

```
INT usermain( void );
```

usermain関数の戻り値は以下のように定義されている。ただし、システムの再起動に関して本OSは実行の枠組みのみを提供する。よって、再起動の処理コードはユーザが実装しなければならない。

戻り値	意 味
0以上	システム終了
-1	システム再起動（ハードウェアのリセットを実行する）
-2	システム高速再起動（OSの再起動）
-3	システム再起動（ハードウェアはリセットせず、OS含むコードの初期化）

usermain関数は/kernel/usermain/usermain.cに記述されている。ユーザは作成するプログラムに応じて任意の内容を記述することができる。一般的には、作成するアプリケーションプログラムのタスクやその他のカーネルオブジェクトの生成、実行などを記述する。

usermain関数が終了すると、本OSは終了または再起動を行う。よって、アプリケーションプログラムの実行中にusermain関数は終了してはならない。

5.2.4 ユーザ定義初期化プログラム userinit

ユーザ定義初期化プログラムuserinitは、主にユーザプログラムのオブジェクトがOSを含むシステムプログラムのオブジェクトと独立（静的なリンクを行わない）場合に使用する。ただし、本OSはAPIの呼び出し方式に関数呼び出しのみ対応しているため、独立したオブジェクトからのAPI呼出しは出来ない。よって、userinitは実際には意味を持たない。

userinitはコンフィギュレーション USE_USERINIT が有効(1)の場合に使用可能となる。初期値は無効(0)である。

USE_USERINIT が有効(1)の場合、コンフィギュレーション RI_USERINIT にuserinitの実行開始アドレスを定義する。

userinitは以下の形式のC 言語の関数として定義されている。

```
typedef INT (*MAIN_FP)(INT, UB **);
```

userinit関数は、usermain関数の実行の前後で呼び出される。引数flagの値が 0 の場合はusermain関数の実行の前、-1 の場合は後である。

また、usermain関数の実行の前に呼ばれた場合は、戻り値により以下のように動作する。

戻り値	usermain関数	OSの動作
正の値	実行する	—
0	実行しない	システム終了
負の値	実行しない	システム再起動 usermainの戻り値と同じ定義

5.3 システム終了処理

5.3.1 システム終了処理の手順

初期タスクの実行関数init_task_mainにおいて、ユーザプログラム（userinitまたはusermain）の実行が終了すると、本OSはシステム終了処理shoutdown_systemを実行し、システム終了またはシステム再起動を行う。

ただし、組込みシステムではシステムの終了や再起動の処理を必要としない場合もありうる。コンフィギュレーション USE_SHUTDOWN により、システム終了処理の有無を指定できる。

USE_SHUTDOWN で 0（終了処理無し）を指定した場合は、ユーザプログラムは終了してはならない。ユーザプログラムが終了した場合の動作は保証されない。

USE_SHUTDOWN で 1（終了処理有り）を指定した場合は、ユーザプログラム（userinitまたはusermain）の戻り値に応じて、システムの終了または再起動を行う。

5.3.2 システム終了手順

以下にシステム終了処理の手順を示す。

(1) デバイス終了処理（knl_finish_device）

knl_start_deviceと対となる周辺デバイスの終了処理。

(2) カーネル終了（knl_tkernel_exit）

(2-1) システムタイマ終了（knl_timer_shoutdown）

システムタイマを停止する。knl_timer_initializeと対となる終了処理。

(2-2) デバイス停止（knl_shutdown_device）

デバイスをすべて停止し、マイコンを終了状態とする。本関数の処理でシステムは終了する。

5.3.3 システム再起動手順

以下にシステム再起動処理の手順を示す。

(1) デバイス終了処理（knl_finish_device）

knl_start_deviceと対となるハードウェアの終了処理（ユーザ定義）。

(2) デバイス再起動（knl_restart_device）

デバイスの再起動処理を行う。本関数の処理の中でシステムは再起動を行う。よって本関数から戻ることはない。ただし、再起動が出来なかった場合のみ本関数から戻る。この場合は、カーネル終了処理（knl_tkernel_exit）が実行され、システムを終了する。

5.4 デバイスの初期化および終了処理

周辺デバイスの初期化および終了の処理は、ハードウェアに依存し、また内容はユーザ定義である。

OSの初期化および終了処理から、ハードウェア依存部の各関数が呼び出されて実行される。以下に該当する関数の一覧を示す。それぞれの具体的な仕様は各ハードウェア向けの実装仕様書を参照のこと。

関数名	内容
knl_init_device	デバイスの初期化 周辺デバイスの初期化を行う。 OS初期化処理（main）から呼び出される。本関数の実行時はOSの初期化中のため、原則OSの機能は利用できない。 主にデバイスドライバ登録前に必要なハードウェアの初期化を実行する。
knl_start_device	デバイスの実行 デバイスドライバの登録、実行を行う。 初期タスク（init_task_main）から呼び出される。本関数は初期タスクのコンテキストで実行され、OSの機能を利用できる。 主にtk_def_devコールによりデバイスの登録を行う。
knl_finish_device	デバイスの終了 デバイスドライバを終了する。 システムの終了処理で呼び出される。 knl_start_deviceと対処理となり、原則として knl_start_deviceで登録、実行したデバイスを終了させる。
knl_shutdown_device	デバイスの停止 周辺デバイスをすべて終了し、マイコンを終了状態とする。 システムの終了処理でカーネル終了（knl_tkernel_exit）から呼び出される。
knl_restart_device	デバイスの再起動 周辺デバイスおよびマイコンの再起動を行う。 システムの再起動処理で呼び出される。

6. タスク

6.1 タスク属性

本実装における各タスク属性の設定の可否を以下に記す。設定不可のタスク属性を指定した場合は、TA_RSATRエラーとなる。

属性	可否	説明
TA_HLNG	○	高級言語（C言語）のみ対応
TA_ASM	×	
TA_SSTKSZ	×	独立したシステムスタックとユーザスタックを持たないため、非対応
TA_USERSTACK	×	
TA_USERBUF	○	
TA_DSNAME	△	コンフィギュレーションにて使用可否を設定する
TA_RNG0	○	実行モードは特権モードのみのため、いずれを指定してもメモリ保護はレベル0（RING0）と同等に扱われる
TA_RNG1	○	
TA_RNG2	○	
TA_RNG3	○	
TA_COPn	—	マイコンの仕様に依存する。各ハードウェアの実装仕様書を参照のこと
TA_FPU	—	

6.2 タスク優先度

設定可能なタスク優先度priは以下となる。

```
1 ≦ pri ≦ 最大優先度TK_MAX_TSKPRI
```

タスク最大優先度 TK_MAX_PRI は、コンフィギュレーション CFN_MAX_TSKPRI で設定される32以上の値である。コンフィギュレーションの初期設定は以下である。

```
#define CFN_MAX_TSKPRI 32
```

6.3 タスクの処理ルーチン

タスクの処理ルーチンは、以下の形式のC言語の関数である。

```
void task ( INT stacd , void * exinf )
{
        /* 処理 */
        tk_ext_tsk( ); または tk_exd_tsk( );    /* タスクの終了*/
}
```

タスクの終了には、tk_ext_tsk( ) または tk_exd_tsk( ) いずれかのAPIを使用する必要がある。これらのAPIを呼びことなく、タスクの処理関数が終了した場合の動作は保証しない。

タスクの処理ルーチンの実行開始時のマイコンの状態は、各ハードウェアの実装仕様書を参照のこと。

6.4 タスクのスタック

本OSでは、すべてのタスクは特権モードで実行されるので保護レベル0とみなし、タスクのスタックはタスク毎に一つとする。ユーザスタックとシステムスタックは独立には実装されない（プロファイル TK_HAS_SYSSTACK は FALSE となる）。スタックのサイズはタスク生成時にユーザから指定される。

7.　時間管理機能

7.1　システム時刻管理

7.1.1　システムタイマ

本実装ではシステム時刻管理のためにマイコン内蔵のタイマの一つをシステムタイマとして使用する。システムタイマのティック時間はコンフィグレーション CFN_TIMER_PERIOD で設定する。単位は1ミリ秒であり、設定範囲はハードウェア依存である。ティック時間の標準の設定値は1ミリ秒である。

本実装ではマイクロ秒の時間管理には対応しない。プロファイル TK_SUPPORT_USEC は FALSE である。システムタイマの具体的な実装は各ハードウェアの実装仕様書を参照のこと。

7.2　タイムイベントハンドラ

7.2.1　タイムイベントハンドラ属性

ハンドラはC言語記述されていることを前提とし、タイムイベントハンドラのハンドラ属性にTA_ASM属性を指定することはできない。TA_HLNG属性のみを許す。ハンドラ生成時にTA_ASM属性が指定された場合はエラーE_RSATRとする。

7.2.2　タイムイベントハンドラの実行状態

タイムイベントハンドラはOSのシステムタイマの割込み処理から実行される。よって、タイムイベントハンドラは非タスク部のコンテキストで実行される。

タイムイベントハンドラは、システムタイマの割込みレベルまで割込みがマスクされた状態で実行される。ただし、TA_STA属性の周期ハンドラで周期起動位相（cycphs）が0の場合はtk_cre_cycの処理内で最初の周期ハンドラの実行が行われる。この場合はOSのクリティカルセクション内でハンドラが実行されるため、割込みは禁止となる。また、起動時刻（almtim）が0の場合はtk_sta_almの処理内部で気アラームハンドラの実行が行われる。この場合もOSのクリティカルセクション内でハンドラが実行されるため、割込みは禁止となる。具体的な実装は、各ハードウェアの実装仕様書を参照のこと。

8.　その他の実装仕様

8.1　システムコール

本実装ではシステムコールの呼び出し形式はC言語の関数呼び出しのみとする。ソフトウェア例外（SVC例外）による呼出しには対応しない。プロファイル TK_TRAP_SVC は FALSE である。

8.2　サブシステム

トロンフォーラムから提供されているμT-Kernel3.0のソースコードではサブシステムはサポートされない。しかしながら、本実装ではサブシステムをサポートしている。ただし、サブシステムのイベント処理はサポートしていない。

結果、プロファイル TK_SUPPORT_SUBSYSTEM は TRUE（デフォルトは FALSE だが、TRUE に変更可能）、TK_SUPPORT_SSYEVENT は FALSE である。

8.3　μT-Kernel2.0 互換機能

トロンフォーラムから提供されているμT-Kernel3.0では、μT-Kernel3.0仕様からは除外されたランデブ機能をμT-Kernel2.0互換機能としてソースコードに残している。しかしながら、本実装ではランデブ機能はサポートしていない。

8.4　システム時刻の初期値

μT-Kernel3.0仕様から、システム時刻の起点は1970年1月1日0時0分0秒（UTC）となった。この変更に伴い、本実装ではシステム起動時、2000年1月1日0時0分0秒をシステム時刻に設定する。ユーザシステムでシステム時刻を利用する場合、正しいシステム時刻をtk_set_utcまたはtk_set_timシステムコールで与える必要がある。

9. デバッグサポート機能

9.1 デバッグサポート機能の有効化

本OSはコンフィギュレーション USE_DBGSPT を有効にし、OSを構築することにより、μT-Kernel/DSのデバッグサポート機能に対応し、デバッグ用APIが使用可となる。

また、DSオブジェクト名を使用するにはコンフィギュレーション USE_OBJECT_NAME を有効にする。コンフィグレーションの初期設定はすべて有効である。

9.2 対応するデバッグ機能

本OSはμT-Kernel/DSのAPIを使用可能である。ただし、マイクロ秒単位のAPIには対応していない。サービスプロファイル TK_SUPPORT_USEC は FALSE である。また、実行トレース機能には対応していない。以下に本OSにおけるμT-Kernel/DSのAPIの対応を示す。

API名	対応	説明
td_lst_tsk	○	
td_lst_sem	○	
td_lst_flg	○	
td_lst_mbx	○	
td_lst_mtx	○	
td_lst_mbf	○	
td_lst_mpf	○	
td_lst_mpl	○	
td_lst_cyc	○	
td_lst_alm	○	
td_lst_por	×	μT-Kernel3.0仕様ではない
td_lst_ssy	○	
td_rdy_que	○	
td_sem_que	○	
td_flg_que	○	
td_mbx_que	○	
td_mtx_que	○	
td_smbf_que	○	
td_rmbf_que	○	
td_mpf_que	○	
td_mpl_que	○	
td_cal_que	×	μT-Kernel3.0仕様ではない
td_acp_que	×	μT-Kernel3.0仕様ではない
td_ref_tsk	○	

td_ref_sem	○	
td_ref_flg	○	
td_ref_mbx	○	
td_ref_mtx	○	
td_ref_mbf	○	
td_ref_mpf	○	
td_ref_mpl	○	
td_ref_cyc	○	
td_ref_alm	○	
td_ref_por	×	μT-Kernel3.0仕様ではない
td_ref_sys	○	
td_ref_ssy	○	
td_get_reg	○	
td_set_reg	○	
td_get_tim	○	
td_get_utc	○	
td_get_otm	○	
td_ref_dsname	○	
td_set_dsname	○	
td_hok_svc	×	実行トレースには対応しない
td_hok_dsp	×	実行トレースには対応しない
gd_hok_int	×	実行トレースには対応しない

10. T-Monitor 互換ライブラリ

10.1 T-Monitor 概要

T-MonitorはT-Kernel1.0の標準開発環境であったT-Engineのモニタプログラムである。μT-Kernel2.0以降はT-Engineは使用していない。また、T-Monitorはμ T-Kernelの仕様には含まれていない。

本実装ではT-Monitorの一部の機能を主にデバッグ用途として、T-Monitor互換ライブラリの形で提供する。

10.2 ライブラリの有効化

T-Monitor互換ライブラリは、コンフィグレーション USE_TMONITOR を有効(1)に設定すると使用可能となる。OSの起動処理の中でライブラリの初期化が行われる。

また、コンフィグレーション USE_SYSTEM_MESSAGE を(1)に設定すると、OSの起動メッセージなどが、T-Monitorのコンソールに出力される。

コンフィグレーション USE_EXCEPTION_DBG_MSG を有効(1)に設定すると、暫定的な例外ハンドラのデバッグ出力が、T-Monitorのコンソールに出力される。

コンフィグレーションの初期設定はすべて有効(1)である。

10.3 対応 API

以下のT-MonitorのAPIを提供する。なお、APIの仕様は「T-Monitor仕様書」を参照のこと。また、具体的な実現方法は、各ハードウェアの実装仕様書を参照のこと。

API名	機能
tm_getchar	コンソールから1文字入力
tm_putchar	コンソールへの1文字出力
tm_getline	コンソールから1行入力
tm_putstring	コンソールへの文字列出力
tm_printf	コンソールへの書式付文字列出力（※）
tm_sprintf	文字列変数への書式付文字列出力（※）

（※）本APIはT-Monitor使用には存在せず、本ライブラリにて追加したものである。

11. コンフィギュレーション

11.1 基本コンフィグレーション

OSの変更可能な設定値は、コンフィギュレーションファイル（/config/config.h(inc)）にて設定される。設定値を変更しOSを再構築することにより、OSの設定を変更することができる。

以下にコンフィグレーションの一覧を示す。値は初期値であり、変更可能である。

(1) カーネル基本設定

名称	値	説明
CFN_MAX_TSKPRI	32	タスク優先度の最大値
CFN_TIMER_PERIOD	1	システムタイマの割込み周期(単位ミリ秒)

(2) カーネルオブジェクト設定

名称	値	説明
CFN_MAX_TSKID	8	最大タスク数
CFN_MAX_SEMID	4	最大セマフォ数
CFN_MAX_FLGID	4	最大イベントフラグ数
CFN_MAX_MBXID	4	最大メールボックス数
CFN_MAX_MTXID	4	最大ミューテックス数
CFN_MAX_MBFID	4	最大メッセージバッファ数
CFN_MAX_MPLID	4	最大可変長メモリプール数
CFN_MAX_MPFID	4	最大固定長メモリプール数
CFN_MAX_CYCID	4	最大周期ハンドラ数
CFN_MAX_ALMID	4	最大アラームハンドラ数
CFN_MAX_SSYID	4	最大サブシステム数
CFN_MAX_SSYPRI	1	サブシステム優先度の最大値（※）

（※）サービスプロファイル TK_SUPPORT_SSYEVENT が FALSE なので意味を持たない

(3) デバイス情報

名称	値	説明
CFN_MAX_REGDEV	2	デバイスの最大登録数
CFN_MAX_OPNDEV	4	デバイスの最大同時オープン数
CFN_MAX_REQDEV	4	デバイスの最大要求数
CFN_DEVT_MBFSZ0	-1	デバイスイベント用メッセージバッファサイズ（-1:使用しない）
CFN_DEVT_MBFSZ1	-1	デバイスイベント用メッセージバッファサイズ（-1:使用しない）

(4) バージョン情報

本情報はカーネルのバージョン情報として、tk_ref_verコールで取得できる。

名称	値	説明
CFN_VER_MAKER	8	カーネルのメーカコード（8:個人または個人事業主）
CFN_VER_PRID	3	カーネルの識別番号（3:RX用μT-Kernel3.0、Ver1.00）
CFN_VER_PRNO1	0x5258	製品管理情報 （ASCIIコード：RX CCRX）
CFN_VER_PRNO2	0x2020	
CFN_VER_PRNO3	0x4343	
CFN_VER_PRNO4	0x5258	

(5) OS内部設定

名称	値	説明
CFN_EXC_STACK_SIZE	1024	初期化スタックのサイズ
USE_IMALLOC	1	OS内部の動的メモリ管理の指定　0:使用しない 1:使用する
USE_SHUTDOWN	1	OS終了処理の指定　0:使用しない 1:使用する

(6) APIのパラメータチェック

名称	値	説明
CHK_NOSPT	1	APIパラメータの指定（E_NOSPT）
CHK_RSATR	1	APIパラメータの指定（E_RSATR）
CHK_PAR	1	APIパラメータの指定（E_PAR）
CHK_ID	1	APIパラメータの指定（E_ID）
CHK_OACV	1	APIパラメータの指定（E_OACV）
CHK_CTX	1	APIパラメータの指定（E_CTX）
CHK_CTX1	1	APIパラメータの指定（E_CTX）　ディスパッチ禁止中
CHK_CTX2	1	APIパラメータの指定（E_CTX）　タスク独立部実行中
CHK_SELF	1	APIパラメータの指定（E_OBJ）　自タスクを指定
CHK_TKERNEL_CONST	1	CONST指定のチェック

全項目 0:チェック無 1:チェック有

(7) ユーザ定義初期化プログラム

名称	値	説明
USE_USERINIT	0	ユーザ定義初期化プログラムの有無　0:無 1:有
RI_USERINIT	0	ユーザ定義初期化プログラムの開始アドレス

(8) デバッグサポート機能

名称	値	説明
USE_DBGSPT	0	デバッグサポート機能の指定　0:無 1:有
USE_OBJECT_NAME	1	オブジェクト名機能の指定　0:無 1:有

OBJECT_NAME_LENGTH	8	オブジェクト名の最大長
USE_TMONITOR	1	T-Monitor互換ライブラリの使用　0:使用しない 1:使用する
USE_SYSTEM_MESSAGE	1	OSからのメッセージ（T-Monitor出力）　0:無 1:有
USE_EXCEPTION_DBG_MSG	1	例外ハンドラメッセージ（T-Monitor出力）0:無 1:有

11.2　機能コンフィギュレーション

OSの取り外しが可能な機能は、機能コンフィグレーションとして、有効・無効の指定ができる。無効とした機能は、プログラムコードも生成されない。本機能は、使用しない機能を取り外すことにより、OSのプログラムコードのサイズを小さくすることが主な目的である。

機能コンフィギュレーションは、機能単位とAPI 単位で有効・無効の指定ができる。機能単位で無効化した場合、関連するAPIはすべて使用できなくなる。API単位の指定は、その機能単位が有効な場合に指定可能である。

機能コンフィグレーションは、すべての機能に対して指定できるわけではない。OSとして必須の機能や、ある機能単位で必須のAPIなどが指定できない。

機能コンフィグレーションは、機能コンフィグレーションファイル（/config/config_func.h）に記述する。

(1) 機能単位

機能単位はそれぞれに有効(1)、無効(0)を設定する。以下にコンフィグレーションの一覧を示す。

名称	機能単位	初期値
USE_SEMAPHORE	セマフォ	有効(1)
USE_EVENTFLAG	イベントフラグ	有効(1)
USE_MAILBOX	メールボックス	有効(1)
USE_MUTEX	ミューテックス	有効(1)
USE_MESSAGEBUFFER	ミューテックス	有効(1)
USE_FIX_MEMORYPOOL	固定長メモリプール	有効(1)
USE_MEMORYPOOL	可変長メモリプール	有効(1)
USE_TIMEMANAGEMENT	時間管理	有効(1)
USE_ALARMHANDLER	周期ハンドラ	有効(1)
USE_CYCLICHANDLER	アラームハンドラ	有効(1)
USE_SUBSYSTEM	サブシステム	有効(1)
USE_DEVICE	デバイスドライバ	有効(1)
USE_FAST_LOCK	高速ロック	有効(1)
USE_MULTI_LOCK	高速マルチロック	有効(1)

注）USE_DEVICEを有効(1)にする際はUSE_MULTI_LOCKも有効(1)にする必要があります。

(2) API単位

API単位の指定は、以下の形式で記述される。

```
#define USE_FUNC_XX_YYY_ZZZ
```

XX_YYY_ZZZは対応するAPI名を大文字としたものである。例えば、tk_del_tskは以下のように定義される。

```
#define USE_FUNC_TK_DEL_TSK
```

定義が存在する場合、対応するAPIは有効となる。定義がない場合は対応するAPIは無効である。
以下に全定義を示す。ここにないAPIは無効化ができない。

種別	定義
タスク管理	USE_FUNC_TK_DEL_TSK USE_FUNC_TK_EXT_TSK USE_FUNC_TK_EXD_TSK USE_FUNC_TK_TER_TSK USE_FUNC_TK_CHG_PRI USE_FUNC_TK_REL_WAI USE_FUNC_TK_GET_REG USE_FUNC_TK_SET_REG USE_FUNC_TK_GET_CPR USE_FUNC_TK_SET_CPR USE_FUNC_TK_REF_TSK USE_FUNC_TK_SUS_TSK USE_FUNC_TK_RSM_TSK USE_FUNC_TK_FRSM_TSK USE_FUNC_TK_SLP_TSK USE_FUNC_TK_WUP_TSK USE_FUNC_TK_CAN_WUP USE_FUNC_TK_DLY_TSK USE_FUNC_TD_LST_TSK USE_FUNC_TD_REF_TSK USE_FUNC_TD_INF_TSK USE_FUNC_TD_GET_REG USE_FUNC_TD_SET_REG
セマフォ管理	USE_FUNC_TK_DEL_SEM USE_FUNC_TK_REF_SEM USE_FUNC_TD_LST_SEM USE_FUNC_TD_REF_SEM USE_FUNC_TD_SEM_QUE
イベントフラグ管理	USE_FUNC_TK_DEL_FLG USE_FUNC_TK_REF_FLG USE_FUNC_TD_LST_FLG USE_FUNC_TD_REF_FLG USE_FUNC_TD_FLG_QUE
メールボックス管理	USE_FUNC_TK_DEL_MBX USE_FUNC_TK_REF_MBX USE_FUNC_TD_LST_MBX USE_FUNC_TD_REF_MBX

	USE_FUNC_TD_MBX_QUE
ミューテックス管理	USE_FUNC_TK_DEL_MTX USE_FUNC_TK_REF_MTX USE_FUNC_TD_LST_MTX USE_FUNC_TD_REF_MTX USE_FUNC_TD_MTX_QUE
メッセージバッファ管理	USE_FUNC_TK_DEL_MBF USE_FUNC_TK_REF_MBF USE_FUNC_TD_LST_MBF USE_FUNC_TD_REF_MBF USE_FUNC_TD_SMBF_QUE USE_FUNC_TD_RMBF_QUE
固定長メモリプール管理	USE_FUNC_TK_DEL_MPF USE_FUNC_TK_REF_MPF USE_FUNC_TD_LST_MPF USE_FUNC_TD_REF_MPF USE_FUNC_TD_MPF_QUE
可変長メモリプール管理	USE_FUNC_TK_DEL_MPL USE_FUNC_TK_REF_MPL USE_FUNC_TD_LST_MPL USE_FUNC_TD_REF_MPL USE_FUNC_TD_MPL_QUE
時間管理	USE_FUNC_TK_SET_UTC USE_FUNC_TK_GET_UTC USE_FUNC_TK_SET_TIM USE_FUNC_TK_GET_TIM USE_FUNC_TK_GET_OTM USE_FUNC_TD_GET_TIM USE_FUNC_TD_GET_OTM
周期ハンドラ管理	USE_FUNC_TK_DEL_CYC USE_FUNC_TK_STA_CYC USE_FUNC_TK_STP_CYC USE_FUNC_TK_REF_CYC USE_FUNC_TD_LST_CYC USE_FUNC_TD_REF_CYC
アラームハンドラ管理	USE_FUNC_TK_DEL_ALM USE_FUNC_TK_STP_ALM USE_FUNC_TK_REF_ALM USE_FUNC_TD_LST_ALM USE_FUNC_TD_REF_ALM
サブシステム管理	USE_FUNC_TK_REF_SSY USE_FUNC_TD_LST_SSY USE_FUNC_TD_REF_SSY
システム情報管理	USE_FUNC_TK_ROT_RDQ USE_FUNC_TK_GET_TID USE_FUNC_TK_DIS_DSP USE_FUNC_TK_ENA_DSP USE_FUNC_TK_REF_SYS USE_FUNC_TK_REF_VER USE_FUNC_TD_REF_SYS USE_FUNC_TD_RDY_QUE

12. サービスプロファイル

本実装におけるサービスプロファイルの一覧を以下に示す。一部のサービスプロファイルに関しては、の切り替えが可能である。

```
#define TK_SUPPORT_TASKEVENT          FALSE
#define TK_SUPPORT_DISWAI             FALSE
#define TK_SUPPORT_IOPORT             TRUE
#define TK_SUPPORT_MICROWAIT          FALSE

#define TK_SUPPORT_LOWPOWER           FALSE          // TRUE に変更可能

#define TK_SUPPORT_USERBUF            TRUE
#define TK_SUPPORT_AUTOBUF            FALSE
#define TK_SUPPORT_MEMLIB             FALSE

#define TK_SUPPORT_TASKEXCEPTION      FALSE

#define TK_SUPPORT_SUBSYSTEM          TRUE           // FALSE に変更可能
#define TK_SUPPORT_SSYEVENT           FALSE

#define TK_SUPPORT_SYSCONF            FALSE

#define TK_HAS_DOUBLEWORD             TRUE
#define TK_SUPPORT_USEC               FALSE
#define TK_SUPPORT_LARGEDEV           FALSE
#define TK_SUPPORT_SERCD              FALSE

#define TK_SUPPORT_INTCTRL            TRUE           // FALSE に変更可能
#define TK_HAS_ENAINTLEVEL            TRUE           // FALSE に変更可能
#define TK_SUPPORT_CPUINTLEVEL        TRUE           // FALSE に変更可能
#define TK_SUPPORT_CTRLINTLEVEL       FALSE
#define TK_SUPPORT_INTMODE            FALSE

#define TK_SUPPORT_CACHECTRL          FALSE
#define TK_SUPPORT_SETCACHEMODE       FALSE
#define TK_SUPPORT_WBCACHE            FALSE
#define TK_SUPPORT_WTCACHE            FALSE

#define TK_SUPPORT_FPU                FALSE          // RXv3 コアは TRUE
#define TK_SUPPORT_COP0               FALSE          // RXv3 コアは TRUE
#define TK_SUPPORT_COP1               FALSE
#define TK_SUPPORT_COP2               FALSE
#define TK_SUPPORT_COP3               FALSE

#define TK_SUPPORT_ASM                FALSE
#define TK_SUPPORT_REGOPS             TRUE
#define TK_ALLOW_MISALIGN             FALSE
#define TK_BIGENDIAN                  FALSE          // CC-RX のオプションで変更可能
#define TK_TRAP_SVC                   FALSE
#define TK_HAS_SYSSTACK               FALSE
#define TK_SUPPORT_PTIMER             FALSE
#define TK_SUPPORT_UTC                TRUE           // FALSE に変更可能
#define TK_SUPPORT_TRONTIME           TRUE           // FALSE に変更可能
```

```
#define TK_SUPPORT_DSNAME               TRUE            // FALSE に変更可能
#define TK_SUPPORT_DBGSPT               FALSE           // TRUE に変更可能

#define TK_SPECVER_MAGIC                6
#define TK_SPECVER_MAJOR                3
#define TK_SPECVER_MINOR                0
#define TK_SPECVER                      (( TK_SPECVER_MAJOR << 8) | TK_SPECVER_MINOR )

#define TK_MAX_TSKPRI                   32              // コンフィグレーションで変更可能
#define TK_WAKEUP_MAXCNT                2147483647
#define TK_SEMAPHORE_MAXCNT             2147483647
#define TK_SUSPEND_MAXCNT               2147483647
#define TK_MEM_RNG0                     0
#define TK_MEM_RNG1                     0
#define TK_MEM_RNG2                     0
#define TK_MEM_RNG3                     0
#define TK_MAX_PTIMER                   0
```

13.　問い合わせ先

本実装に関する問い合わせや他のRXファミリへのポーティングに関する相談は以下のメールアドレス宛にお願い致します。

yuji_katori@yahoo.co.jp　　トロンフォーラム学術・教育WGメンバ
鹿取　祐二（かとり　ゆうじ）

なお、上記のメールアドレスは余儀なく変更される場合がありますが、その際はご了承ください。

以上