μT-Kernel3.0
共通実装仕様書

Version.01.00.02

2020.05.29

変更履歴

版数(日付)	内　容
1.00.02 (2020.05.29)	● 1.3 バージョン情報 バージョン番号等を更新 ● 1.5 関連ドキュメント バージョン番号等を更新 ● 1.6 ソースコード構成 Make 用ビルドディレクトリの追加 ● 11.1 基本コンフィギュレーション (4) バージョン情報バージョン番号等を更新 ● 4.4 OS 内部で使用される割込み 実装依存に関する説明の見直し
1.00.01 (2020.03.13)	● 1.4 ターゲット名 ターゲット名はソースコード中には記述せず、開発ツールにて指定するように変更。 ● 1.5 関連ドキュメント　更新 ● 5.1 システム起動処理 説明分の見直し、修正 ● 5.4 ハードウェアの初期化および終了処理 5.5 デバイスドライバの実行および終了 ハードウェアの周辺デバイスと、OS が管理するデバイスドライバが、デバイスという同一の名称で一緒に説明されていたので、節を分けるように修正。また、後者をデバイスと呼び前者はハードウェアと呼ぶこととした。 ● 11. コンフィギュレーション 「(9) コプロセッサ対応」をの項目を追加
1.00.00 (2019.12.11)	● 初版

目次

1. 概要

1.1 目的

本仕様書は、μT-Kernel3.0の実装仕様を記載した実装仕様書である。
対象は、TRONフォーラムから公開されているμT-Kernel 3.0 (V3.00.01)のソースコードの実装とする。
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	0x0000	TRONフォーラム
識別番号	prid	0x0000	TRONフォーラム
仕様書バージョン番号	spver	0x6300	μT-Kenrel 3.00仕様
カーネルバージョン番号	prver	0x0001	
製品管理情報	prno[4]	-	全て0

1.4 ターゲット名

OSを構築するのあたって、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。

ターゲット名はソースコード中には記載されない。開発ツールにおいて指定する必要がある。たとえば、コンパイラのオプションや、統合開発環境の設定においてターゲット名を指定する。
ソースコードでは、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 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-Kernel 3.0 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-Kernel 3.0 仕様書 (Ver.3.00.00)	TRON フォーラム TEF020-S004-3.00.00
ハードウェア 依存実装仕様	TX03 M367 IoT-Engine 向け 実装仕様書 (Ver.01.00.02)	TRON フォーラム TEF033-W003-200529
	RX231 IoT-Engine 向け 実装仕様書 (Ver.01.00.01)	TRON フォーラム TEF033-W005-200529
T-Monitor	T-Monitor 仕様書	TRON フォーラム TEF-020-S002-01.00.01

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）
├ build_make              Make 用ビルドディレクトリ
├ docs                    ドキュメント
└ etc                     その他
```

機種依存定義 sysdeped 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
            ├ <core1>                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。
特別なレジスタとして、taskmode レジスタを定義する。taskmode レジスタは、メモリへのアクセス権限（保護レベル）を保持する仮想レジスタである。マイコンの物理的なレジスタを割り当てるのではなく、OS 内の仮想的なレジスタとして実装する場合もある。
なお、本実装ではプログラムは特権モードでのみ実行されるので、常に値は 0 となる。

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(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) テンポラリスタック
OS の内部処理（タスクディスパッチ処理など）で使用されるスタックである。
コンフィギュレーション情報(TMP_STACK_SIZE)にてサイズの指定が指定され、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(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。よって、上記のカーネルオブジェクトの生成の際には、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 CNF_SYSTEMAREA_TOP と CNF_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により、調整が可能である。また、値に 0 を指定することにより、デフォルト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。

OS 管理メモリ領域のアドレスを調整することにより、OS 管理外のメモリ領域をつくることができる。OS 管理外のメモリ領域は、OS からは使用されないので、ユーザプログラムで自由に使用することができる。

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 K_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使用する割込みの種類を以下に示す。

種類	説明
SVC 割込み （スーパーバイザコール）	システムコールまたは拡張 SVC の呼び出しに使用する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
ディスパッチ要求	ディスパッチを要求する割込み
強制ディスパッチ要求	強制ディスパッチを要求する割込み

SVC 割込みは、システムコールや拡張 SVC を発行する際に使用されるソフトウェア割込みである。ただし、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 には対応しない。拡張 SVC の機能は無い。よって SVC 割込みは使用しない。

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。

ディスパッチ要求は、タスクのディスパッチを発生させるために使用される割込みである。OS の API 処理にて、タスクのディスパッチが必要となった場合（実行タスクが変更になった場合）に発行される。

強制ディスパッチ要求は、OS の処理にて強制ディスパッチを発生させるために使用される割込みである。強制ディスパッチは、ディスパッチ元のタスクが存在しない場合に行われる特別なディスパッチである。具体的には OS の起動時とタスクの終了時のみに行われる。

上記の各種割込みには、各ハードウェアへの実装に応じて、具体的な割込みや例外が割り当てられる。実装によっては使用されない割込みもある。詳細は各ハードウェア向けの実装仕様書を参照のこと。

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も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。main 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。

(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）

デバイスドライバの登録に先立ち、必要なハードウェアの初期化を行う。。

(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）

タスクなどの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。

- 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
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_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。ただし、本実装ではサブシステムには対応していないため、この処理は行われない。

例外として 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　ハードウェアの初期化および終了処理

OS の起動および終了に際して、マイコンおよび周辺デバイスなどのハードウェアの初期化、終了処理を行う。これらの処理は、ハードウェアに依存し、また内容はユーザ定義である。

OS の標準のソースコード内では、OS が使用する周辺デバイスのみの必要最低限の初期化および終了処理が実装されている。ユーザは必要に応じて各関数の内容を変更してよい。ただし、これらの関数は OS の共通部からも呼ばれるため、関数の呼び出し形式を変更してはならない。

本実装では、ハードウェア依存部に以下の関数が実装されている。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_startup_hw	**ハードウェアの初期化（リセット）** リセット時の必要最小限のハードウェアの初期化を行う。 主に以下の処理が行われる。 ・CPU クロックの設定 ・I/O 端子の機能設定 ・使用する周辺デバイスの有効化
knl_shutdown_hw	**ハードウェアの停止** 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。
knl_restart_hw	**ハードウェアの再起動** 周辺デバイスおよびマイコンの再起動を行う。 主にソフトウェアリセットなど実行する。

5.5　デバイスドライバの実行および終了

OS の起動および終了に際して、デバイスドライバの登録、実行、終了を行う。

OS の標準のソースコード内では、以下の関数を実装している。ユーザは使用するデバイスに応じて処理を記述する必要がある。

関数名	内容
knl_init_device	**デバイスの初期化** デバイスドライバの登録に先立ち、必要なハードウェアの初期化を行う。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。

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。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。
knl_finish_device	**デバイスの終了** デバイスドライバを終了する。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。

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為、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 CNF_MAX_TSKPRI で設定される 16 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_PRI             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 10 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の実行中は、原則として、タスクと同様にすべての割込みを受け付ける。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、API(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。
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

本実装では、サブシステムはサポートされない。
プロファイル TK_SUPPORT_SUBSYSTEM および TK_SUPPORT_SSYEVENT は FALSE である。

8.3 μT-Kernel2.0 互換機能

本実装では、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す。ただし、今後実装から削除される可能性があるので、使用は推奨しない。
μT-Kernel2.0 互換機能は、コンフィギュレーションの USE_LEGACY_API を 1 に設定すると有効となる。コンフィグレーションの初期設定は 1(有効)である。

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。

（＊）μT-Kernel2.0 互換仕様である。コンフィグレーション USE_LEGACY_API を有効にした場合のみ使用可能。

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μT-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 の変更可能な種設定値は、コンフィギュレーションファイル（/config/comfig.h）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。ただし、オブジェクトの数などはその値に応じてメモリなどの資源が確保されるので、ユーザのシステムに応じた使用可能な資源から適切な値とする必要がある。

(1) カーネル基本設定

名称	値	説明
CFN_SYSTEMAREA_TOP	0	システムメモリ領域の開始アドレス 0: システムのデフォルト値を使用
CFN_SYSTEMAREA_END	0	システムメモリ領域の終了アドレス 0: システムのデフォルト値を使用
CFN_MAX_TSKPRI	32	タスク優先度の最大値
CFN_TIMER_PERIOD	10	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	32	最大タスク数
CFN_MAX_SEMID	16	最大セマフォ数
CFN_MAX_FLGID	16	最大イベントフラグ数
CFN_MAX_MBXID	8	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	8	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	8	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	8	最大アラームハンドラ数

(3) デバイス情報

名称	値	説明
CFN_MAX_REGDEV	8	デバイスの最大登録数
CFN_MAX_OPNDEV	16	デバイスの最大同時オープン数
CFN_MAX_REQDEV	16	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。ただし、本実装では tk_ref_ver コールは未対応である。

名称	値	説明
CNF_VER_MAKER	0	カーネルのメーカコード 0: TRON フォーラム
CFN_VER_PRID	0	カーネルの識別番号 0: TRON フォーラム
CFN_VER_PRVER	1	カーネルのバージョン番号
CFN_VER_PRNO1	0	製品管理情報
CFN_VER_PRNO2	0	
CFN_VER_PRNO3	0	
CFN_VER_PRNO4	0	

(5) OS 内部設定

名称	値	説明
USE_LEGACY_API	0	T-Kernel2.0 互換 API 0: 使用しない 1: 使用する
CFN_MAX_PORID	0	最大ランデブ数
CNF_EXC_STACK_SIZE	2048	初期化スタックのサイズ
CNF_TMP_STACK_SIZE	256	テンポラリスタックのサイズ
USE_NOINIT	0	初期値をもたない静的変数領域（BSS）の初期化の指定 0: ゼロクリアする 1: 初期化しない

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
USE_STATIC_IVT	0	静的割込みベクタテーブルの指定 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) 0: チェック無　1: チェック有
CHK_RSATR	1	API パラメータの指定 (E_RSATR) 0: チェック無　1: チェック有
CHK_PAR	1	API パラメータの指定 (E_PAR) 0: チェック無　1: チェック有
CHK_ID	1	API パラメータの指定 (E_ID) 0: チェック無　1: チェック有
CHK_OACV	1	API パラメータの指定 (E_OACV) 0: チェック無　1: チェック有
CHK_CTX	1	API パラメータの指定 (E_CTX) 0: チェック無　1: チェック有
CHK_CTX1	1	API パラメータの指定 (E_CTX) ディスパッチ禁止中 0: チェック無　1: チェック有
CHK_CTX2	1	API パラメータの指定 (E_CTX) タスク独立部実行中 0: チェック無　1: チェック有
CHK_SELF	1	API パラメータの指定 (E_OBJ) 自タスクを指定 0: チェック無　1: チェック有
CHK_TKERNEL_CONST	1	CONST 指定のチェック 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	1	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

(9) コプロセッサ制御

名称	値	説明
USE_FPU	0	FPU 対応（※） 0: 無　1: 有
USE_DSP	0	DSP 対応（※） 0: 無　1: 有

（※）対象のハードウェアが FPU や DSP を有するか、および OS の機種依存の実装により指定可能か否かが決まる。各ハードウェアｍｐ実装仕様書を参照のこち。

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/comfig_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。初期値ではすべての機能は有効(1)である。

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
USE_SEMAPHORE	セマフォ
USE_MUTEX	ミューテックス
USE_EVENTFLAG	イベントフラグ
USE_MAILBOX	メールボックス
USE_MESSAGEBUFFER	ミューテックス
USE_RENDEZVOUS	ランデブ
USE_MEMORYPOOL	可変長メモリプール
USE_FIX_MEMORYPOOL	固定長メモリプール
USE_TIMEMANAGEMENT	時間管理
USE_CYCLICHANDLER	アラームハンドラ
USE_ALARMHANDLER	周期ハンドラ
USE_DEVICE	アラームハンドラ
USE_FAST_LOCK	高速ロック
USE_MULTI_LOCK	高速マルチロック

(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。

たとえば、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_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
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
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
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
ランデブ管理	USE_FUNC_TK_DEL_POR
	USE_FUNC_TK_FWD_POR
	USE_FUNC_TK_REF_POR
	USE_FUNC_TD_LST_POR
	USE_FUNC_TD_REF_POR
	USE_FUNC_TD_CAL_QUE
	USE_FUNC_TD_ACP_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
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
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
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

以上