

TOPPERS/JSP Kernel USER'S MANUAL

本資料は、NPO法人TOPPERSプロジェクトより配布されているTOPPERS/JSPカーネル Release1.4.1のユーザズマニュアルを原文のまま掲載しています。
本資料の内容の取扱いについては、以下に示す、TOPPERSライセンスに従ってください。

TOPPERS/JSP Kernel

Toyohashi Open Platform for Embedded Real-Time Systems/
Just Standard Profile Kernel

Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN

Copyright (C) 2004 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN

上記著作権者は、以下の(1)~(4)の条件が、Free Software Foundation
によって公表されているGNU General Public LicenseのVersion 2に記
述されている条件を満たす場合に限り、本ソフトウェア(本ソフトウェア
を改変したものを含む。以下同じ)を使用・複製・改変・再配布(以下、
利用と呼ぶ)することを無償で許諾する。

- (1) 本ソフトウェアをソースコードの形で利用する場合には、上記の著作
権表示、この利用条件および下記の無保証規定が、そのままの形でソー
スコード中に含まれていること。
- (2) 本ソフトウェアを、ライブラリ形式など、他のソフトウェア開発に使用
できる形で再配布する場合には、再配布に伴うドキュメント(利用
者マニュアルなど)に、上記の著作権表示、この利用条件および下記
の無保証規定を掲載すること。
- (3) 本ソフトウェアを、機器に組み込むなど、他のソフトウェア開発に使用
できない形で再配布する場合には、次のいずれかの条件を満たすこ
と。
 - (a) 再配布に伴うドキュメント(利用者マニュアルなど)に、上記の著
作権表示、この利用条件および下記の無保証規定を掲載すること。
 - (b) 再配布の形態を、別に定める方法によって、TOPPERSプロジェクトに
報告すること。
- (4) 本ソフトウェアの利用により直接的または間接的に生じるいかなる損
害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

本ソフトウェアは、無保証で提供されているものである。上記著作権者お
よびTOPPERSプロジェクトは、本ソフトウェアに関して、その適用可能性も
含めて、いかなる保証も行わない。また、本ソフトウェアの利用により直
接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

株式会社ワイ・デー・ケーは、本資料の使用により生じた、いかなる損害に対しても、その責任を負いません。

はじめに	5
1 . TOPPERS/JSPカーネルの概要	6
1.1 ターゲットプロセッサ/ターゲットシステム	6
1.2 開発環境	7
1.3 シミュレーション環境	8
1.4 カーネルがサポートする機能	8
1.5 既知の問題点	9
1.6 注意事項	9
2 . JSPカーネルの機能	9
2.1 実装方針とモデル	9
2.2 データ型	10
2.3 オブジェクトのID番号と優先度	10
2.4 エラーチェックとエラーコード	11
2.5 割込みハンドラ	11
2.6 タイムイベントハンドラ	11
2.7 CPU例外ハンドラ	11
2.8 非タスクコンテキストからのサービスコール呼出しと割込み禁止区間	12
2.9 システム初期化手順と初期化ルーチン	12
2.9.0 静的APIとコンフィギュレータ	13
2.9.1 インクルードファイル	13
2.9.2 システム終了手順と終了処理ルーチン	14
3 . JSPカーネルのサービスコールと静的API	14
3.1 タスク管理機能	14
3.2 タスク付属同期機能	14
3.3 タスク例外処理機能	15
3.4 同期・通信機能	15
3.4.1 セマフォ	15
3.4.2 イベントフラグ	15
3.4.3 データキュー	16
3.4.4 メールボックス	16
3.5 メモリプール管理機能	16
3.5.1 固定長メモリプール	16
3.6 時間管理機能	17
3.6.1 システム時刻管理	17
3.6.2 周期ハンドラ	17
3.7 システム状態管理機能	18
3.8 割込み管理機能	18
3.9 システム構成管理機能	19
3.9.0 CPU例外発生時のシステム状態参照	19
3.9.1 性能評価用システム時刻参照機能	21
4 . システムログ機能	22
4.1 システムログ機能の位置付け	22
4.2 ログバッファへの記録と低レベル出力	22
4.3 ログ情報の種別	23
4.4 ログ情報の重要度	23
4.5 トレースログ機能	23
4.6 システムログ機能の拡張サービスコール	24
4.7 システムログ機能のためのライブラリ関数とマクロ	24
4.8 システムログ機能の設定方法	25
5 . システムサービス	26
5.1 システムインタフェースレイヤ (SIL)	26
5.1.1 割込みロック状態の制御	26
5.1.2 微少時間待ち	27
5.1.3 エンディアン	27
5.1.4 メモリ空間アクセス関数	27
5.2 システムクロックドライバ	28
5.2.1 システムクロックドライバの内部構成	28
5.3 シリアルインタフェースドライバ	29
5.3.1 シリアルインタフェースドライバのサービスコール	29

5.3.2	シリアルインタフェースドライバの内部構成	30
5.4	システムログタスク	30
6	サポートライブラリ	31
7	開発環境・インストール・ポータリング	31
7.1	ディレクトリ・ファイル構成	31
7.2	開発環境	34
7.3	コンフィギュレーションツールの構築	34
7.4	サンプルプログラムの構築	35
7.5	アプリケーションとカーネルを別々に構築する方法	35
7.6	コンフィギュレーションスクリプトの使い方	36
7.7	Makefileの修正	37
7.8	コンパイルオプション	39
7.9	コンフィギュレーションツールの使い方	39
7.10	リンクスクリプトとメモリ領域	41
7.11	他のターゲットへのポータリング	41
7.12	カーネルの内部識別子のリネーム	41
8	その他	42
8.1	ウェブサイト	42
8.2	利用条件・著作権	42
8.3	保証・サポート・適用性	42
8.4	メーリングリスト	42
8.5	TOPPERSプロジェクトへの参加	43
8.6	ITRON Club	43
9	リファレンス	43
9.1	サービスコール一覧	43
9.2	静的API一覧	45
9.3	メインエラーコード一覧 (JSPカーネルが返すもののみ)	45
9.4	バージョン履歴	46

はじめに

このユーザマニュアルは、 μ ITRON4.0 仕様書 (Ver. 4.02.00) の内容を前提に記述してあります。 μ ITRON4.0 仕様書は、以下の URL からダウンロードすることができます。

<http://www.ertl.jp/ITRON/SPEC/mitron4-j.html>

TOPPERS/JSP Kernel

Toyohashi Open Platform for Embedded Real-Time Systems/
Just Standard Profile Kernel

Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN

Copyright (C) 2004 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN

上記著作権者は、以下の (1) ~ (4) の条件が、Free Software Foundation によって公表されている GNU General Public License の Version 2 に記述されている条件を満たす場合に限り、本ソフトウェア (本ソフトウェアを改変したものを含む。以下同じ) を使用・複製・改変・再配布 (以下、利用と呼ぶ) することを無償で許諾する。

- (1) 本ソフトウェアをソースコードの形で利用する場合には、上記の著作権表示、この利用条件および下記の無保証規定が、そのままの形でソースコード中に含まれていること。
- (2) 本ソフトウェアを、ライブラリ形式など、他のソフトウェア開発に使用できる形で再配布する場合には、再配布に伴うドキュメント (利用者マニュアルなど) に、上記の著作権表示、この利用条件および下記の無保証規定を掲載すること。
- (3) 本ソフトウェアを、機器に組み込むなど、他のソフトウェア開発に使用できない形で再配布する場合には、次のいずれかの条件を満たすこと。
 - (a) 再配布に伴うドキュメント (利用者マニュアルなど) に、上記の著作権表示、この利用条件および下記の無保証規定を掲載すること。
 - (b) 再配布の形態を、別に定める方法によって、TOPPERS プロジェクトに報告すること。
- (4) 本ソフトウェアの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者および TOPPERS プロジェクトを免責すること。

本ソフトウェアは、無保証で提供されているものである。上記著作権者および TOPPERS プロジェクトは、本ソフトウェアに関して、その適用可能性も含めて、いかなる保証も行わない。また、本ソフトウェアの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

* μ ITRON4.0 仕様は、トロン協会が中心となって策定されたオープンなリアルタイムカーネル仕様です。 μ ITRON4.0 仕様の仕様書は、トロン協会のホームページ (<http://www.assoc.tron.org/>) から入手することができます。

-
- * TRON は "The Real-time Operating system Nucleus" の略称です。
 - * ITRON は "Industrial TRON" の略称です。
 - * μ ITRON は "Micro Industrial TRON" の略称です。
 - * TRON, ITRON, および μ ITRON は、特定の商品ないしは商品群を指す名称ではありません。
 - * TOPPERS は "Toyohashi Open Platform for Embedded Real-time Systems" の略称、JSP は "Just Standard Profile" の略称です。
 - * 本マニュアル中の商品名は、各社の商標または登録商標です。
-

1 . TOPPERS/JSP カーネルの概要

TOPPERS/JSP カーネル (以下, 単に JSP カーネルと書く) は, TOPPERS プロジェクトにおいて開発した μ ITRON4.0 仕様に準拠したリアルタイムカーネルである.

JSP (Just Standard Profile) の名前が示す通り, μ ITRON4.0 仕様のスタンダードプロファイル規定に従って実装されている.

1.1 ターゲットプロセッサ/ターゲットシステム

JSP カーネルは, 現時点で, 以下のターゲットプロセッサ/ターゲットシステムをサポートしている.

ディレクトリ名 プロセッサ (型番)	開発環境 システム (メーカー名)
m68k M68040 (MC68LC040)	GNU 開発環境 DVE-68K/40 (電産)
sh3 SH3 (SH7709A) SH3 (SH7729R) SH3 (SH7727) SH4 (SH7750)	GNU 開発環境 MS7709ASE01 (日立超 LSI システムズ) MS7729RSE01 (日立超 LSI システムズ) MS7727CP01 (日立超 LSI システムズ) MS7750SE01 (日立超 LSI システムズ)
sh3-ghs SH3 (SH7709A) SH3 (SH7727)	GHS 開発環境 MS7709ASE01 (日立超 LSI システムズ) MS7727CP01 (日立超 LSI システムズ)
sh1 SH1 (SH7032) SH1 (SH7032)	GNU 開発環境 KZ-SH1-01 (京都マイクロコンピュータ) RISC 評価キット SH-1 (CQ 出版) でも動作 SH1/CPUB (常盤商行)
sh2 SH2(SH7145) SH2(SH7615)	GNU 開発環境 AP_SH2F_6A (アルファプロジェクト) HSB7615IT (北斗電子)
h8 H8 (H8/3048F) H8 (H8/3052F) H8 (H8/3067F) H8 (H8/3068F) H8 (H8/3069F)	GNU 開発環境 AKI-H8/3048F (秋月電子通商) AKI-H8/3052F (秋月電子通商) AKI-H8/3067F (秋月電子通商) AKI-H8/3068F (秋月電子通商) AKI-H8/3069F (秋月電子通商)
h8s H8S (2350)	GNU 開発環境 H8S/2350 評価ボード (ミスボ)
armv4 ARM9 (ARM922T)	GNU 開発環境 KZ-ARM9EXPCI-01 (京都マイクロコンピュータ)
armv4-ghs ARM9 (ARM920T) ARM9E (ARM966E-S)	GHS 開発環境 Integrator/AP+CM920T (ARM) Integrator/AP+CM966E-S (ARM)
m32r M32R (M32102S6FP)	GNU 開発環境 M3A-2131G50 (三菱電機)
microblaze MicroBlaze	GNU 開発環境 MIREF (YDK)

MicroBlaze	MIRE_MULTI3000 (YDK)
MicroBlaze	MultiMedia Board (Xilinx)
tms320c54x	TI 社の開発環境
TMS320C54x (TSM320C5402)	TMS320VC5402 DSK (TI)
xstormy16	GNU 開発環境
Xstormy16	三洋マイコン開発ツール (三洋電機)
mips3	GNU 開発環境
MIPS3 (VR4131)	KZ-Vr4131PCI-01 (京都マイクロコンピュータ)
MIPS3 (VR5500)	RTE-VR5500-CB(64) (マイダス・ラボ)
m16c-renesas	Renesas 社の開発環境
M16C (M30620FCAFP-CPU)	OASKS16 (オックス電子)
M16C (M30262F8FG-CPU)	OAKS16-MINI (オックス電子)
s1c33	GNU 開発環境
SC33	DMT33209 (EPSON)
SC33	LUXUN2 (EPSON)
s1c33-gnu33	GNU33 開発環境
SC33	DMT33209 (EPSON)
SC33	LUXUN2 (EPSON)
powerpc32	GNU 開発環境
PowerPC32 (MPC860T)	TB6102S (タンバック)
nios2	GNU 開発環境
Nios2	NiosII Development Board (アルテラ)

JSP カーネルは、カーネルのできる限り多くの部分を C 言語で記述する、ターゲット非依存部と依存部を明確に分離するなど、他のターゲットプロセッサへのポータビリティが容易な構造になっている。ただし、ポータビリティにどの程度の手間がかかるかは、ターゲットプロセッサのアーキテクチャやシステムの構成などに依存する。

このユーザズマニュアルでは、ターゲット (ターゲットプロセッサおよびターゲットシステム) に依存しない機能についてのみ説明している。ターゲットに依存する機能については、ターゲット毎のマニュアルを参照すること。

1.2 開発環境

JSP カーネルは、GCC などの GNU 開発環境を標準のソフトウェア開発環境としているが、他の種類の開発環境も利用できるように考慮している。利用できる開発環境については、ターゲット毎または開発環境毎のマニュアルで説明する。

ターゲット非依存部は、大部分は標準的な C 言語によって記述されているが、性能と可読性を両立させるために、一部でインライン関数を用いている。インライン関数の機能を持たない開発環境の場合でも、改造なしに対応可能ではあるが、非効率 / 無駄なコードが生成されるおそれがある。

カーネル本体は、外部のライブラリ関数に依存しないように記述している。

ただし、コンパイラが標準 C ライブラリ関数を呼び出すコードを生成する場合があります、その場合には標準 C ライブラリが必要である。

また、システムサービスやサポートライブラリ、アプリケーションプログラムで標準 C ライブラリが必要になる場合も考えられる。

実際、標準配布キットに含まれる中で、システムログ機能呼び出すためのライブラリ関数内で、可変数引数を処理するための機能 (stdarg.h, va_list, va_start, va_arg) を用いている (実際には、GNU 開発環境では、可変数引数を処理するための機能は GCC 本体でサポートしているため、標準 C ライブラリは必要ない)。これらの理由により、標準 C ライブラリを用いる構成もとれるようにしている。

1.3 シミュレーション環境

JSP カーネルのシミュレーション環境として、Linux 上で動作する環境と Windows 上で動作する環境を用意している。これらのシミュレーション環境は、Linux および Windows の一つのプロセスの中で複数のタスクを切り替えて動作させるもので、スレッドライブラリとして使うこともできる。

これらのシミュレーション環境についての詳細は、シミュレーション環境毎のマニュアルを参照すること。

1.4 カーネルがサポートする機能

JSP カーネルは、名前が示す通り、 μ ITRON4.0 仕様のスタンダードプロファイルに含まれる機能をすべてサポートしている。

スタンダードプロファイルでは、割込みハンドラと割込みサービスルーチンのいずれかをサポートすればよいが、JSP カーネルは、現状では割込みハンドラのみをサポートしている。

スタンダードプロファイルに含まれない機能として、ターゲット依存に以下の割込み管理機能およびサービスコールをサポートする場合がある。

これらの機能の具体的な内容については、ターゲット毎に異なる。詳しくは、ターゲット毎のマニュアルを参照すること。

dis_int	割込みの禁止
ena_int	割込みの許可
chg_ixx	割込みマスクの変更
get_ixx	割込みマスクの参照

xx はターゲット毎に定められる。

また、 μ ITRON4.0 仕様に定義されている以外に、以下の独自の拡張機能およびサービスコールをサポートしている。

(1) CPU 例外発生時のシステム状態の参照

スタンダードプロファイルでは、CPU 例外ハンドラ内で、CPU 例外が発生したコンテキストや状態を参照することが必要であるが、そのための API は定めていない。

JSP カーネルでは、CPU 例外が発生した処理で sns_yyy を呼び出した場合の結果を、CPU 例外ハンドラ内で取り出せるようにするために、以下の五つのサービスコールを独自にサポートしている。

vxsns_ctx	CPU 例外発生時のコンテキストの参照
vxsns_loc	CPU 例外発生時の CPU ロック状態の参照
vxsns_dsp	CPU 例外発生時のディスパッチ禁止状態の参照
vxsns_dpn	CPU 例外発生時のディスパッチ保留状態の参照
vxsns_tex	CPU 例外発生時のタスク例外処理禁止状態の参照

(2) 性能評価用システム時刻参照機能

性能評価用システム時刻参照機能とは、JSP カーネル上で動作するタスクや JSP カーネル自身の性能を計測するための、システム時刻を μ 秒単位で読み出す機能である。

この機能のために追加したサービスコールは次の通りである。

vxget_tim	性能評価用システム時刻の参照
-----------	----------------

性能評価用システム時刻参照機能をサポートするかどうかは、ターゲット依存部の定義ファイルで指定することができる。

また、ターゲットシステムの制限により、この機能をサポートできない場合もある。

(3) 終了処理ルーチン機能

JSP カーネルでは、システムの終了時に呼び出される終了処理ルーチンを登録するための機能をサポートしている。

この機能のために追加した静的 API は次の通りである。

VATT_TER 終了処理ルーチンの追加（静的 API）

終了処理ルーチンについては、「2.12 システム終了手順と終了処理ルーチン」を参照のこと。

（4）カーネル動作状態の参照

カーネル上で動作するタスクから呼び出される関数が、カーネルの初期化完了前や終了処理開始後にも呼び出される可能性がある場合には、その中でカーネルのサービスコールを呼び出せるかを判別することが必要となる。

JSP カーネルでは、この判別を可能にするために、次のサービスコールを追加している。

vsns_ini カーネル動作状態の参照

1.5 既知の問題点

現バージョンでは、静的 API の処理中のエラーの検出機能の中で、ターゲット依存のエラーの検出が不十分である。

例えば、割込みハンドラ番号が不正な値である場合、カーネルとコンフィギュレータのいずれもエラーを検出せず、カーネルが正しく動作しない結果となる。

kernel_cfg.c は、カーネル、システムサービス、アプリケーションのいずれのインクルードファイルもインクルードし、いずれのシンボルも参照する可能性がある。

そのため、カーネル、システムサービス、アプリケーションでシンボル等が衝突している場合や、コンパイルオプションが食い違っている場合に、kernel_cfg.c が正しくコンパイルできなくなる場合が考えられる。

カーネルのシンボルをリネームするなどの方法でかなり軽減されてはいるが、問題がなくなっているわけではない。

1.6 注意事項

CRE_DTQ のパラメータ dtqcnt は、 μ ITRON4.0 仕様の Ver. 4.01.00 では一般定数式パラメータと規定されているが、JSP カーネルでは Ver. 4.02.00 に準拠して、プリプロセッサ定数式パラメータと扱っている。

2. JSP カーネルの機能

この節では、 μ ITRON4.0 仕様で実装定義となっている事項を中心に、JSP カーネルの機能について解説する。

2.1 実装方針とモデル

μ ITRON4.0 仕様のスタンダードプロファイルは、システム全体を一つのモジュールにリンクすることを想定して規定されている。また、サービスコールの呼出しは、単なるサブルーチンコールによって行うことが想定されている。JSP カーネルは、この想定に従い、アプリケーションとカーネルを一つのモジュールにリンクし、サブルーチンコールによってサービスコールを呼び出す方法のみをサポートしている。

JSP カーネル上で動作するアプリケーションは、すべて C 言語で記述することを原則としている。そのため、タスクや割込みハンドラなどの処理単位をアセンブリ言語で記述する方法は、特別には用意していない（もちろん、インタフェースさえ C 言語の関数にあわせれば、記述にアセンブリ言語を使うことは問題ない）。

JSP カーネルでは、サービスコールの大部分を一つの割込み禁止区間として実装しているため、サービスコールの不可分性は厳密に保証される。逆に欠点としては、最大割込み禁止時間（最大割込み応答時間も同様）が、待ちキューにつながるタスクの最大数やタイムイベントの最大数に依存することになるが、スタンダードプロファイルの機能セットの範囲内では、この方法でもそれほ

ど問題にならないと思われる。

2.2 データ型

JSP カーネルでは、以下にリストアップするデータ型を、signed int 型、unsigned int 型、または size_t 型に定義している。これらの型のサイズは、JSP カーネルがポータリングされているターゲットプロセッサ/コンパイラの多くにおいて 32 ビットであるため、そうでない場合にのみターゲット毎のマニュアルに明示する。すなわち、ターゲット毎のマニュアルに明示されていない限り、以下にリストアップするデータ型のサイズは 32 ビットである。

signed int 型に定義しているデータ型

INT	符号付き整数
BOOL	真偽値
FN	機能コード
ER	エラーコード
ID	ID 番号
PRI	優先度
TMO	タイムアウト値
ER_BOOL	ER または BOOL
ER_ID	ER または ID
ER_UINT	ER または UINT

unsigned int 型に定義しているデータ型

UINT	符号無し整数
ATR	属性
STAT	状態
MODE	動作モード
RELTIM	相対時間
TEXPTN	タスク例外要因のビットパターン
FLGPTN	イベントフラグのビットパターン

size_t 型に定義しているデータ型

SIZE	サイズ
------	-----

ただし、RELTIM 型の有効ビット数は 31 ビットを越えることはない。すなわち、unsigned int 型のサイズが 32 ビットの場合には、RELTIM 型の有効ビット数は 31 ビットであり、 $(2^{31} - 1)$ を越える値を RELTIM 型のパラメータに渡した場合、E_PAR エラーとなる。unsigned int 型のサイズが 16 ビットの場合には、RELTIM 型の有効ビット数も 16 ビットである。スタンダードプロファイルでは、RELTIM 型は 16 ビット以上と規定しており、この仕様でスタンダードプロファイル規定に準拠している。

SYSTIM 型は、32 ビットの符号無し整数型に定義しており、構造体として定義する方法は用いていない。

時間をあらずデータ型 (TMO, RELTIM, SYSTIM) の時間単位は、スタンダードプロファイルの規定に従い、すべて 1 ミリ秒としている。

2.3 オブジェクトの ID 番号と優先度

オブジェクトの ID 番号には、1 から連続した正の値を用いる。オブジェクトの ID 番号に抜けがある場合 (例えば、ID=1 と ID=3 のオブジェクトが登録され、ID=2 のオブジェクトが登録されない場合) には、コンフィギュレータがエラーを報告する。負の ID 番号を用いたシステムオブジェクトとユーザオブジェクトの区別はサポートしていない。

生成できるオブジェクトの最大数は、カーネルのコード上は、ID 番号が ID 型 (signed int 型に定義している) で表現できる範囲内であるが、実際にはメモリ容量によって制限される。なお、JSP カーネルでは、オブジェクトを生成するためのサービスコールはサポートしていない。

タスクとメッセージの優先度には、1～16の正の値を用いる。

2.4 エラーチェックとエラーコード

JSP カーネルでは、以下に示すメインエラーコードを返すエラーの検出を省略している。

E_SYS	システムエラー
E_MACV	メモリアクセス違反

また、ポインタの値が不正な場合のパラメータエラー (E_PAR) の検出も省略している。メモリアクセス違反 (E_MACV) の検出も省略しているため、引数にポインタを渡すサービスコールに対して、存在しないメモリ番地を差すポインタなど、不正なアクセスを引き起こすポインタを渡した場合、プロセッサがバスエラーなどの CPU 例外を起こす場合がある (具体的な動作はターゲットプロセッサに依存)。

μITRON4.0 仕様書に定義されているメインエラーコードの中で、スタンダードプロファイルの機能では発生しないものや、JSP カーネルの実装上発生しないものがある。JSP カーネルでサービスコールが返すメインエラーコードについては、「9.3 メインエラーコード一覧」を参照のこと。

JSP カーネルでは、サブエラーコードは用いていない。サブエラーコードには常に-1が返る。

2.5 割込みハンドラ

JSP カーネルでは、割込みハンドラの機能とそれを定義する静的 API (DEF_INH) をサポートしており、割込みサービスルーチンの機能とそれを追加する静的 API (ATT_ISR) はサポートしていない。

割込みハンドラの C 言語による記述形式は次の通りとする。

```
void interrupt_handler(void)
{
    割込みハンドラ本体
}
```

JSP カーネルでは、C 言語で記述された割込みハンドラが呼ばれる時点で、CPU ロック解除状態になっている。また、割込みハンドラからリターンするには、C 言語の関数から単にリターンすればよい。

割込みハンドラをアセンブリ言語で記述する方法は、サポートしていない。

NMI (マスクできない割込み) 以外にカーネルの管理外の割込みがあるかどうかは、ターゲット依存である。具体的な仕様については、ターゲット毎のマニュアルを参照すること。

2.6 タイムイベントハンドラ

JSP カーネルでは、タイムイベントハンドラとして、周期ハンドラのみをサポートしている。周期ハンドラは、isig_tim サービスコールの中から、サブルーチンコールで呼び出される。そのため、周期ハンドラの優先順位は、isig_tim を呼び出した割込みハンドラよりも一つだけ高い (厳密に言うと、isig_tim を呼び出した割込みハンドラよりも高く、その割込みハンドラよりも高い優先順位を持つ他のいずれの処理よりも低い)。

2.7 CPU 例外ハンドラ

JSP カーネルでは、スタンダードプロファイル規定に従って、CPU 例外ハンドラの機能とそれを定義する静的 API (DEF_EXC) をサポートしている。

JSP カーネルでは、CPU 例外ハンドラは非タスクコンテキストで実行される。非タスクコンテキストから呼び出せるサービスコールは、CPU 例外ハンドラ内から呼び出すことができる。ただし、CPU 例外が CPU ロック状態で発生した場合には、CPU 例外ハンドラ中で CPU ロックを解除することはできず、非タスクコンテキストから呼び出せるサービスコールを呼び出すこともできない。

μITRON4.0仕様において、CPU例外ハンドラ内で行えるべきものとして規定されている各操作は、次のような方法で行うことができる。

- (a) CPU例外が発生したコンテキストや状態の参照は、そのために用意されたJSPカーネル独自のサービスコール(vxsns_ctx, vxsns_loc, vxsns_dsp, vxsns_dpn, vxsns_tex)を用いて行うことができる。詳しくは、「3.10 CPU例外発生時のシステム状態参照」を参照すること。
- (b) CPU例外が発生したタスクのID番号の参照は、iget_tidサービスコールを呼び出すことによって行うことができる。
- (c) タスク例外処理の要求は、iras_texサービスコールを呼び出すことによって行うことができる。

CPU例外ハンドラの優先順位は、タスクコンテキストを実行中にCPU例外が発生した場合には、ディスパッチャよりも高く、すべての割り込みハンドラおよびタイマハンドラよりも低い。非タスクコンテキストを実行中にCPU例外が発生した場合には、CPU例外が発生した処理の優先順位よりも一つだけ高い(厳密に言うと、CPU例外が発生した処理よりも高く、CPU例外が発生した処理よりも高い優先順位を持つ他のいずれの処理よりも低い)。

CPU例外ハンドラのC言語による記述形式は次の通りとする。

```
void cpu_exception_handler(VP p_excinf)
{
    CPU例外ハンドラ本体
}
```

p_excinfには、CPU例外に関する情報を記憶している領域の先頭番地が渡される。これは、CPU例外ハンドラ内で、CPU例外が発生したコンテキストや状態を参照する際に必要となる。詳しくは、「3.10 CPU例外発生時のシステム状態参照」を参照すること。CPU例外ハンドラからリターンするには、C言語の関数から単にリターンすればよい。

CPU例外ハンドラをアセンブリ言語で記述する方法は、サポートしていない。

2.8 非タスクコンテキストからのサービスコール呼出しと割り込み禁止区間

JSPカーネルでは、タスクコンテキスト専用のサービスコールと、非タスクコンテキスト専用のサービスコールを厳密に区別している。タスクコンテキスト専用のサービスコールを非タスクコンテキストから呼び出した場合や、非タスクコンテキスト専用のサービスコールをタスクコンテキストから呼び出した場合には、E_CTXエラーを返す。

また、非タスクコンテキストから呼び出されたサービスコールの遅延実行は行っていない。そのため、非タスクコンテキストから呼び出したサービスコールも、操作対象のオブジェクトの状態に依存して発生するエラーを検出することができる。

2.9 システム初期化手順と初期化ルーチン

カーネルを起動するには、ターゲットに依存して行わなければならない最低限の初期化を行った後、CPUロック状態と同等の状態では、kernel_start関数を呼び出す。JSPカーネルでは、ターゲット毎にスタートアップモジュールを用意して、この処理を行っている。詳しくは、ターゲット毎のマニュアルを参照すること。

ATT_INIによって追加された初期化ルーチンは、カーネル内部のデータ構造の初期化や他の静的APIの処理を終えた後に、システムコンフィギュレーションファイル中でのATT_INIの記述順と同じ順序で呼び出される。初期化ルーチン内では、サービスコールを呼び出してはならない。初期化ルーチン内でサービスコールを呼び出した場合、システムの動作は保証されない(実際には、ターゲットによって、呼び出しても差し支えないサービスコールがある)。また、初期化ルーチンを実行中にカーネルの管理外の割り込みが禁止されているかどうかは、ターゲットおよびkernel_start関数が呼び出された時の状態に依存する。具体的には、ターゲット毎のマニュアルを参照すること。

2.10 静的 API とコンフィギュレータ

JSP カーネルは、 μ ITRON4.0 仕様に規定されたシステムコンフィギュレーション手順に準拠した手順で、コンフィギュレーションを行う。

システムの構成を記述したシステムコンフィギュレーションファイルは、まず C 言語のプリプロセッサで処理され、その結果をカーネルのコンフィギュレータ (cfg プログラム) に入力する。カーネルのコンフィギュレータは、カーネル構成・初期化ファイルを kernel_cfg.c に、ID 自動割付け結果ヘッダファイルを kernel_id.h に生成する。また、静的 API のパラメータチェックに用いるファイルを kernel_chk.c に、静的 API の解析内容を含むオブジェクト定義ファイルを kernel_obj.dat に生成する。静的 API の文法エラー (および処理中のエラーの一部) が検出されれば、カーネルのコンフィギュレータがエラーを報告する。

kernel_cfg.c は、コンパイルされて、アプリケーションプログラムおよびカーネルと共にリンクされる。リンクにより生成されたロードモジュールは、カーネルのパラメータチェックプログラム (chk プログラム) によって、静的 API のパラメータチェックが行われる。パラメータの値のエラーが検出されると、パラメータチェックプログラムがエラーを報告するが、「1.5 既知の問題点」で述べた通り、現バージョンではパラメータエラーのチェックは不完全である。

以上の手順は、Makefile 内に記述されている。ソフトウェア部品のコンフィギュレータを組み込みたい場合には、Makefile を修正する必要がある。

2.11 インクルードファイル

アプリケーションが用いることができるインクルードファイルは、include ディレクトリの下に置かれている。

t_services.h は、カーネル上で動作するプログラムのソースファイルでインクルードすべき標準インクルードファイルである。この中で、kernel.h (さらにここから、t_stddef.h, itron.h, tool_defs.h, sys_defs.h, cpu_defs.h, t_syslog.h) と serial.h をインクルードしている。また、アプリケーションに有益と思われる定義をいくつか含んでいる。

s_services.h は、直接ハードウェアにアクセスするデバイスドライバのソースファイルでインクルードすべき標準インクルードファイルである。この中で、sil.h (さらにここから、t_stddef.h, itron.h, tool_defs.h, sys_defs.h, cpu_defs.h, t_syslog.h) と t_config.h (さらにここから、sys_config.h, cpu_config.h, tool_config.h) をインクルードしている。また、アプリケーションから呼ばれるデバイスドライバのインクルードファイルで、インライン関数などでシステムインタフェースレイヤを用いている場合にも、このファイルをインクルードする。

この2つのファイルからインクルードされるファイル (上に列挙したものは、直接インクルードしないのが原則であるが、次の3つのケースは例外である)。

- (1) カーネルから呼ばれるデバイスドライバのインクルードファイルで、インライン関数などでシステムインタフェースレイヤを用いている場合には、sil.h をインクルードする。
- (2) カーネル上で動作するプログラムで、ターゲット依存情報を参照したい場合には、t_config.h をインクルードする。
- (3) 他の ITRON 仕様 OS からソフトウェアをポーティングする場合などには、kernel.h を直接インクルードしてもよい。
- (4) ITRON 仕様共通規定に準拠するソフトウェア部品のインクルードファイルは、itron.h を直接インクルードしてもよい。

JSP カーネルの Release 1.3 以前のバージョンでは、t_services.h は jsp_services.h というファイル名になっていた。バージョンを問わずに動作するプログラムを作る際には、t_services.h をインクルードし、古いバージョンで jsp_services.h を t_services.h にシンボリックリンクを貼る方法を推奨する。

なお、jsp_kernel.h は、カーネルを構成するプログラムのソースファイルでインクルードするべ

- (1) slp_tsk 起床待ち
- (2) tslp_tsk 起床待ち (タイムアウトあり)
- (3) wup_tsk, iwup_tsk タスクの起床
- (4) can_wup タスク起床要求のキャンセル
- (5) rel_wai, irel_wai 待ち状態の強制解除
- (6) sus_tsk 強制待ち状態への移行
- (7) rsm_tsk 強制待ち状態からの再開
- (8) frsm_tsk 強制待ち状態からの強制再開

タスクの強制待ち要求ネスト数の最大値 (TMAX_SUSCNT) が 1 であるため, rsm_tsk と frsm_tsk の処理内容は同一である .

- (9) dly_tsk 自タスクの遅延

3.3 タスク例外処理機能

TEXPTN 型は, unsigned int 型に定義している . よって TBIT_TEXPTN は, unsigned int 型が 32 ビットの場合は 32, 16 ビットの場合は 16 になる .

- (1) DEF_TEX タスク例外処理ルーチンの定義 (静的 API)

texatr に TA_ASM が指定された場合の機能 (タスク例外処理ルーチンをアセンブリ言語で記述する) はサポートしていない .

- (2) ras_tex, iras_tex タスク例外処理の要求
- (3) dis_tex タスク例外処理の禁止
- (4) ena_tex タスク例外処理の許可
- (5) sns_tex タスク例外処理禁止状態の参照

3.4 同期・通信機能

3.4.1 セマフォ

セマフォの最大資源数は, UINT 型 (unsigned int 型に定義している) で表現できる数値の範囲内である . すなわち, unsigned int 型が 32 ビットの場合は $(2^{32} - 1)$, 16 ビットの場合は $(2^{16} - 1) = 65535$ である . TMAX_MAXSEM は定義していない .

- (1) CRE_SEM セマフォの生成 (静的 API)
- (2) sig_sem, isig_sem セマフォ資源の返却
- (3) wai_sem セマフォ資源の獲得
- (4) pol_sem セマフォ資源の獲得 (ポーリング)
- (5) twai_sem セマフォ資源の獲得 (タイムアウトあり)

3.4.2 イベントフラグ

一つのイベントフラグで複数のタスクが待ち状態になれる機能はサポートしていない .

FLGPTN 型は, unsigned int 型に定義している . よって TBIT_FLGPTN は, unsigned int 型が 32 ビットの場合は 32, 16 ビットの場合は 16 になる .

サイズが `blkksz` バイトのメモリブロックを `blkcnt` 個獲得できるのに必要な固定長メモリプール領域のサイズは、`TROUND_VP(blksz) * blkcnt` バイトである。ここで、`TROUND_VP(blksz)` は、`blkksz` をターゲットプロセッサ/コンパイラのポインタのサイズの倍数になるよう切り上げた数を表す。`TSZ_MPF` は定義していない。

- (1) `CRE_MPF` 固定長メモリプールの生成 (静的 API)

`mpf` に `NULL` 以外が指定された場合の機能 (固定長メモリプール領域の先頭番地を指定する) はサポートしていない。

- (2) `get_mpf` 固定長メモリブロックの獲得
- (3) `pget_mpf` 固定長メモリブロックの獲得 (ポーリング)
- (4) `tget_mpf` 固定長メモリブロックの獲得 (タイムアウトあり)

- (5) `rel_mpf` 固定長メモリブロックの返却

`blk` パラメータ (返却するメモリブロックの先頭番地) の値が、返却先のメモリプール領域の外や、メモリブロックの途中を指す場合には、`E_PAR` エラーを返す。未獲得のメモリブロックを返却した場合や、返却済のメモリブロックを再度返却した場合の動作は保証されない。

3.6 時間管理機能

タイムイベントハンドラに関しては、「2.6 タイムイベントハンドラ」を参照すること。

3.6.1 システム時刻管理

JSP カーネルでは、タイムティックの供給 (`isig_tim` を周期的に呼び出す処理) はシステムサービスのシステムクロックドライバによって実現している。システムクロックドライバの主要部分は、ターゲット毎にハードウェアタイマを使って実現されており、`isig_tim` を呼び出す周期はターゲット毎に定める。そのため `TIC_NUME` と `TIC_DENO` は、ターゲット依存部のアプリケーション用のインクルードファイル (`cpu_defs.h` および `sys_defs.h`) の中で定義している。ターゲットによっては、この数値を変更するだけで `isig_tim` を呼び出す周期を変更できるように実装されている場合もある。詳しくは、ターゲット毎のマニュアルを参照すること。

- (1) `set_tim` システム時刻の設定
- (2) `get_tim` システム時刻の参照
- (3) `isig_tim` タイムティックの供給

`isig_tim` は、ターゲット依存に定義された `TIC_NUME` と `TIC_DENO` で指定される時間だけシステム時刻を進め、必要なタイムイベント (タイムアウト、周期ハンドラの起動など) の処理を行う。JSP カーネルでは、システムクロックドライバがこのサービスコールを周期的に呼び出すため、アプリケーションから呼び出す必要はない。

3.6.2 周期ハンドラ

周期ハンドラの起動位相を保存する機能はサポートしていない。

- (1) `CRE_CYC` 周期ハンドラの生成 (静的 API)

`cycatr` に `TA_PHS` が指定された場合の機能 (周期ハンドラの起動位相を保存する) はサポートしていない。また、`TA_ASM` が指定された場合の機能 (周期ハンドラをアセンブリ言語で記述する) もサポートしていない。

- (2) `sta_cyc` 周期ハンドラの動作開始
- (3) `stp_cyc` 周期ハンドラの動作停止

3.7 システム状態管理機能

- | | | |
|--------|-------------------|-----------------|
| (1) | rot_rdq, irot_rdq | タスクの優先順位の回転 |
| (2) | get_tid, iget_tid | 実行状態のタスク ID の参照 |
| (3) | loc_cpu, iloc_cpu | CPU ロック状態への移行 |
| (4) | unl_cpu, iunl_cpu | CPU ロック状態の解除 |
| (5) | dis_dsp | ディスパッチの禁止 |
| (6) | ena_dsp | ディスパッチの許可 |
| (7) | sns_ctx | コンテキストの参照 |
| (8) | sns_loc | CPU ロック状態の参照 |
| (9) | sns_dsp | ディスパッチ禁止状態の参照 |
| (10) | sns_dpn | ディスパッチ保留状態の参照 |
| (11) | vsns_ini | カーネル動作状態の参照 |

【C 言語 API】

```
BOOL state = vsns_ini();
```

【パラメータ】

なし

【リターンパラメータ】

BOOL state カーネル動作状態

【機能】

カーネルの初期化完了前または終了処理開始後に呼び出された場合に TRUE, カーネルの動作中に呼び出された場合に FALSE を返す.

このサービスコールが TRUE を返す時には, 他のサービスコールを呼び出してはならない. このサービスコールが TRUE を返す時に他のサービスコールを呼び出した場合, システムの動作は保証されない.

3.8 割込み管理機能

割込みハンドラに関しては, 「2.5 割込みハンドラ」を参照すること.

- | | | |
|-------|---------|-----------------------|
| (1) | DEF_INH | 割込みハンドラの定義 (静的 API) |
|-------|---------|-----------------------|

INHNO 型の定義と inhno の意味はターゲット毎に定める. inhatr には, TA_HLNG のみを指定することができる.

- | | | |
|-------|---------|-----------|
| (2) | dis_int | 割込みの禁止 |
| (3) | ena_int | 割込みの許可 |
| (4) | chg_ixx | 割込みマスクの変更 |
| (5) | get_ixx | 割込みマスクの参照 |

これらのサービスコールがサポートされているかどうか, サポートされている場合の仕様 (xx の部分の名称, 型とパラメータの名称と意味, CPU ロック状態やディスパッチ状態との関連) については, ターゲット依存である. 具体的には, ターゲット毎のマニュアルを参照すること.

3.9 システム構成管理機能

CPU 例外ハンドラに関しては「2.7 CPU 例外ハンドラ」を，初期化ルーチンに関しては「2.9 システム初期化手順と初期化ルーチン」参照すること。

(1) DEF_EXC CPU 例外ハンドラの定義 (静的 API)

EXCNO 型の定義と excno の意味はターゲット毎に定める。excattr には，TA_HLNG のみを指定することができる。

(2) ATT_INI 初期化ルーチンの追加 (静的 API)

iniatr に TA_ASM が指定された場合の機能 (初期化ルーチンをアセンブリ言語で記述する) はサポートしていない。

(3) VATT_TER 終了処理ルーチンの追加 (静的 API)

【静的 API】

```
VATT_TER({ ATR teratr, VP_INT exinf, FP terrtn });
```

【パラメータ】

ATR	teratr	終了処理ルーチン属性
VP_INT	exinf	終了処理ルーチンの拡張情報
FP	terrtn	終了処理ルーチンの起動番地

【機能】

終了処理ルーチンを，指定される各パラメータに基づいて追加する。teratr は終了処理ルーチンの属性，exinf は終了処理ルーチンを起動する時にパラメータとして渡す拡張情報，terrtn は終了処理ルーチンの起動番地である。

VATT_TER においては，teratr はプリプロセッサ定数式パラメータである。teratr には，TA_HLNG の指定ができる。TA_HLNG (= 0x00) が指定された場合には高級言語用のインタフェースで終了処理ルーチンを起動する。

VATT_ATR によって追加された終了処理ルーチンは，システム終了処理時に実行される。詳しくは，「2.12 システム終了手順と終了処理ルーチン」を参照すること。

3.10 CPU 例外発生時のシステム状態参照

CPU 例外ハンドラ内で，CPU 例外が発生したコンテキストや状態を参照するためのサービスコールとして，JSP カーネルでは，五つのサービスコールを独自にサポートしている。サービスコール vxsns_yyy は，CPU 例外が発生した処理で sns_yyy を呼び出した場合の結果を取り出すもので，CPU 例外ハンドラに渡されるパラメータ p_excinf をパラメータとする。

(1) vxsns_ctx CPU 例外発生時のコンテキストの参照

【C 言語 API】

```
BOOL state = vxsns_ctx(VP p_excinf);
```

【パラメータ】

VP	p_excinf	CPU 例外に関する情報を記憶している領域の先頭番地
----	----------	----------------------------

【リターンパラメータ】

BOOL	state	コンテキスト
------	-------	--------

【機能】

CPU 例外が発生したコンテキストが，非タスクコンテキストの場合に TRUE，タスクコンテキスト

の場合に FALSE を返す。p_excinf には、CPU 例外ハンドラに渡される p_excinf パラメータをそのまま渡す。CPU 例外ハンドラ以外から呼び出した場合や、p_excinf を正しく渡さなかった場合の振舞いは保証されない。

(2) vxsns_loc CPU 例外発生時の CPU ロック状態の参照

【C 言語 API】

```
BOOL state = vxsns_loc(VP p_excinf);
```

【パラメータ】

VP	p_excinf	CPU 例外に関する情報を記憶している領域の先頭番地
----	----------	----------------------------

【リターンパラメータ】

BOOL	state	CPU ロック状態
------	-------	-----------

【機能】

CPU 例外が発生した状態が CPU ロック状態の場合に TRUE、CPU ロック解除状態の場合に FALSE を返す。p_excinf には、CPU 例外ハンドラに渡される p_excinf パラメータをそのまま渡す。CPU 例外ハンドラ以外から呼び出した場合や、p_excinf を正しく渡さなかった場合の振舞いは保証されない。

(3) vxsns_dsp CPU 例外発生時のディスパッチ禁止状態の参照

【C 言語 API】

```
BOOL state = vxsns_dsp(VP p_excinf);
```

【パラメータ】

VP	p_excinf	CPU 例外に関する情報を記憶している領域の先頭番地
----	----------	----------------------------

【リターンパラメータ】

BOOL	state	ディスパッチ禁止状態
------	-------	------------

【機能】

CPU 例外が発生した状態が、ディスパッチ禁止状態の場合に TRUE、ディスパッチ許可状態の場合に FALSE を返す。p_excinf には、CPU 例外ハンドラに渡される p_excinf パラメータをそのまま渡す。CPU 例外ハンドラ以外から呼び出した場合や、p_excinf を正しく渡さなかった場合の振舞いは保証されない。

【補足説明】

CPU 例外ハンドラの起動によってディスパッチ禁止 / 許可状態は変化せず、CPU 例外ハンドラ中ではディスパッチの禁止や許可は行えないため、vxsns_dsp の返り値は sns_dsp の返り値に常に一致する。そのため、vxsns_dsp と sns_dsp の処理内容は同一となっている。

(4) vxsns_dpn CPU 例外発生時のディスパッチ保留状態の参照

【C 言語 API】

```
BOOL state = vxsns_dpn(VP p_excinf);
```

【パラメータ】

VP	p_excinf	CPU 例外に関する情報を記憶している領域の先頭番地
----	----------	----------------------------

【リターンパラメータ】

BOOL	state	ディスパッチ保留状態
------	-------	------------

【機能】

CPU 例外が発生した状態が、ディスパッチ保留状態の場合に TRUE、そうでない場合に FALSE を返す。すなわち、ディスパッチャよりも優先順位が高い処理が実行されていた時、CPU ロック状態であった時およびディスパッチ禁止状態であった時は、TRUE を返す。p_excinf には、CPU 例外ハンドラに渡される p_excinf パラメータをそのまま渡す。CPU 例外ハンドラ以外から呼び出した場合や、p_excinf を正しく渡さなかった場合の振舞いは保証されない。

(5) vxsns_tex CPU 例外発生時のタスク例外処理禁止状態の参照

【C 言語 API】

```
BOOL state = vxsns_tex(VP p_excinf);
```

【パラメータ】

VP	p_excinf	CPU 例外に関する情報を記憶している領域の先頭番地
----	----------	----------------------------

【リターンパラメータ】

BOOL	state	タスク例外処理禁止状態
------	-------	-------------

【機能】

CPU 例外が発生した時に実行状態であったタスクが、タスク例外処理禁止状態の場合に TRUE、タスク例外処理許可状態の場合に FALSE を返す。CPU 例外が非タスクコンテキストで発生し、その時に実行状態のタスクがなかった場合にも、FALSE を返す。p_excinf には、CPU 例外ハンドラに渡される p_excinf パラメータをそのまま渡す。CPU 例外ハンドラ以外から呼び出した場合や、p_excinf を正しく渡さなかった場合の振舞いは保証されない。

【補足説明】

CPU 例外ハンドラの起動によってタスク例外処理禁止 / 許可状態は変化せず、CPU 例外ハンドラ中ではタスク例外処理の禁止や許可は行えないため、vxsns_tex の戻り値は sns_tex の戻り値に常に一致する。そのため、vxsns_tex と sns_tex の処理内容は同一となっている。

3.1.1 性能評価用システム時刻参照機能

JSP カーネルでは、JSP カーネル上で動作するタスクや JSP カーネル自身の性能を計測するために、システム時刻より精度の高い性能評価用システム時刻を読み出す機能を、ターゲット依存にサポートしている。性能評価用システム時刻は、μ秒単位で表現されるが、実際の精度はターゲット依存である。具体的には、ターゲット毎のマニュアルを参照すること。

性能評価用システム時刻参照機能では、次のデータ型を用いる。

SYSUTIM	性能評価用システム時刻 (符号無し整数)
---------	----------------------

SYSUTIM 型のサイズ数はターゲット依存である。具体的には、ターゲット毎のマニュアルを参照すること。

(1) vxget_tim 性能評価用システム時刻の参照

【C 言語 API】

```
ER ercd = vxget_tim(SYSUTIM *p_sysutim);
```

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	エラーコード
SYSUTIM	sysutim	現在の性能評価用システム時刻

【エラーコード】

E_CTX コンテキストエラー

【機能】

現在の性能評価用システム時刻を読み出し，sysutim に返す．

このサービスコールは，タスクコンテキストからのみ呼び出すことができる．タスクコンテキストであれば，CPU ロック状態であっても呼び出せる．非タスクコンテキストから呼び出した場合には，E_CTX エラーとなる．

4．システムログ機能

システムログ機能は，カーネル内で発生した異常事象（アサーションの失敗，エラーコードを返せないエラー）を，システムの外部に通知するための機能である．カーネルのトレースログ，システムサービスやアプリケーション内で発生した異常事象やトレースログにも，同じ機能を利用することができる．

4.1 システムログ機能の位置付け

カーネル内で発生した異常事象をシステムの外部に通知するための方法として，シリアルインタフェースに出力する，ディスクに書き出すなどの方法が考えられる．

システムログ機能は，カーネル内部から呼び出されるという観点からは，カーネルの一部と考えるのが自然である．一方，シリアルインタフェースやディスクにアクセスするためのサービス（デバイスドライバなど）はカーネル上で動作するため，それらを用いるシステムログ機能は，カーネル上に実装されたシステムサービスと考える方が自然で，位置づけが微妙である．

そこで JSP カーネルでは，カーネルの拡張機能として，異常事象に関する情報やトレースログ情報（これを，ログ情報と総称する）を，カーネル内のバッファ（これをログバッファと呼ぶ）に記録する機能と，ログバッファからログ情報を読み出す機能を用意する．これを，システムログ機能と呼ぶ．ログ情報をログバッファから読み出し，デバイスにアクセスするサービスを用いて外部に出力する機能は，システムログタスクとしてカーネル上に実現する．

4.2 ログバッファへの記録と低レベル出力

上述したように，ログ情報をシステムの外部に出力するために，デバイスにアクセスするサービスが必要になるが，これらのサービスはカーネル上で動作しているため，カーネルの動作を継続できないような重大な異常事象が起こった場合には，これらのサービスを使うことができない．また，これらのサービス自身をデバッグする場合にも，デバイスにアクセスするサービスを使うことができない．

そこで，カーネル上で動作するサービスが使えない場合にもログ情報を出力するために，低レベル出力機能を用意する．低レベル出力機能は，ターゲット依存に用意する低レベルの文字出力関数（sys_putc）を用いてログ情報を出力する機能である．低レベルの文字出力関数は，ターゲット依存部で用意することとしているが，最終製品に組み込まれる場合などでは，文字を出力する方法がない状況も考えられる．そのような場合，低レベルの文字出力関数に送られた文字は，メモリ上に残しておくか，捨ててしまうしかない．

ログ情報を，ログバッファへ記録するか低レベル出力機能を用いて出力するかの設定は，カーネルの拡張サービスコール（vmsk_log）によって行うことができる．vmsk_log の使い方については後述する．

低レベル出力機能を用いると，ログメッセージの作成処理（printf 相当の処理）と低レベルの文字出力処理をカーネル内で行うために，カーネルの応答性が悪くなることに注意しなければならない．特に，低レベルの文字出力処理はデバイスをポーリングする形で実装するのが通常で，その場合には，カーネルの応答性は実用的と言えない程に悪くなる．

一方、カーネルの動作を継続できるような（あまり重大でない）事象については、ログ情報をカーネル内のログバッファに記録し、記録したログ情報の出力は、デバイスにアクセスするサービスを用いて動作するシステムログタスクに任せる。システムログタスクはカーネル上で動作するタスクであり、カーネルの拡張サービスコール（vrea_log）を用いて、ログバッファからログ情報を読み出す。JSP カーネルの標準配布キットには、システムログタスクの一例として、シリアルインタフェースにログ情報を文字列の形で出力するシステムログタスクを含めている。

4.3 ログ情報の種別

JSP カーネルのシステムログ機能は、ログ情報に以下の種別を設けている。

LOG_TYPE_INH	割込みハンドラ
LOG_TYPE_ISR	割込みサービスルーチン
LOG_TYPE_CYC	周期ハンドラ
LOG_TYPE_EXC	CPU 例外ハンドラ
LOG_TYPE_TEX	タスク例外処理ルーチン
LOG_TYPE_TSKSTAT	タスク状態変化
LOG_TYPE_DSP	ディスパッチャ
LOG_TYPE_SVC	サービスコール
LOG_TYPE_COMMENT	コメント
LOG_TYPE_ASSERT	アサーションの失敗

これらの種別は、ITRON デバッグインタフェース仕様を参考に定めている。ただし、ITRON デバッグインタフェース仕様におけるトレースログ形式は、RIM (RTOS Interface Module) がデバッグツールに渡す場合の形式を定めたものであり、カーネルが出力する形式と一致している必要はない (RIM が変換すればよい)。実際、上の種別の中で、LOG_TYPE_CYC と LOG_TYPE_ASSERT は、デバッグインタフェース仕様と一致していない。

ログ情報の種別の中で、LOG_TYPE_COMMENT と LOG_TYPE_ASSERT 以外はカーネルのトレースログのためのもので、どのように用いるかはターゲット依存部に任されている (4.5 節参照)。

4.4 ログ情報の重要度

JSP カーネルのシステムログ機能は、ログ情報を出力する際に指定する重要度に基づいて、実際に出力するログ情報を動的に設定することができる。これは、UNIX のシステムログ機能をまねたもので、ログの重要度の種類や指定方法も UNIX の API を参考にしている。また、低レベル出力機能を用いて出力するログ情報も、重要度に基づいて動的に設定することができる。

具体的には、ログの重要度として次の 8 段階を用意している。

LOG_EMERG	システムをシャットダウンすべきエラー
LOG_ALERT	
LOG_CRIT	
LOG_ERROR	重要性の低いシステムエラー
LOG_WARNING	警告メッセージ。システムは安全に継続動作できる
LOG_NOTICE	
LOG_INFO	
LOG_DEBUG	デバッグのためのメッセージ

なお、アサーションの失敗は、LOG_EMERG で出力する。カーネルのトレースログは、LOG_DEBUG で出力するのを標準とする (ターゲット依存)。

どの重要度のログ情報をログバッファに記録するかと、どの重要度のログ情報を低レベル出力機能を用いて出力するかは、カーネルの拡張サービスコール (vmsk_log) によって設定することができる。vmsk_log の各パラメータは、指定するログ情報の集合を表すビットマップである。また、ビットマップを作るためのマクロとして、LOG_MASK と LOG_UPTO を用意している。

4.5 トレースログ機能

JSP カーネルは、カーネルのトレースログを取得するための基本的な仕組みを持っているが、ト

レースログの実際の取得方法はターゲット依存となる。カーネルのトレースログの取得に、システムログ機能を使うのも選択肢の1つである。ただし、カーネルのトレースログをシステムログタスクを用いて取り出す方法は考えていない（システムログタスクが動作することによりトレースログが生成され、取り出すより多くのログ情報が生成される可能性があるため）。

4.6 システムログ機能の拡張サービスコール

システムログ機能の提供する拡張サービスコールは次の通りである。

(1) ER vwri_log(UINT prio, SYSLOG *p_log)

システムログ機能に、重要度 prio でログ情報を出力する（ログバッファへ記録するか低レベル出力機能を用いて出力する）。SYSLOG は、ログ情報を格納するためのデータ型（構造体）で、この拡張サービスコールには、それへのポインタを渡す。

(2) ER_UINT vrea_log(SYSLOG *p_log)

ログバッファからログ情報を1つ取り出す。ログバッファが空の時は E_OBJ, そうでない場合は、失われたログ情報の数（ログ情報が失われていない場合は 0）を返す。システムログタスクが用いることを想定している。

(3) ER vmsk_log(UINT logmask, UINT lowmask)

ログバッファに記録すべきログ情報の重要度のビットマスク（logmask）と、低レベル出力機能を用いて出力すべきログ情報の重要度のビットマスク（lowmask）を設定する。

4.7 システムログ機能のためのライブラリ関数とマクロ

システムログ機能は、上記のサービスコールに加えて、次のライブラリ関数とマクロを提供する。

(1) void _syslog_n(UINT prio, UINT type, VP_INT arg1, ..., VP_INT argn)
n は 0~6 のいずれか。

ログ種別が type, パラメータが arg1~argn のログ情報を、重要度 prio で出力するためのライブラリ関数。

(2) void syslog_n(UINT prio, const char *format, arg1, ..., argn)
n は 0~5 のいずれか。

format 文字列およびそれに続く引数から作成されるコメント（ログ種別が LOG_TYPE_COMMENT のログ情報）を、重要度 prio で出力するためのマクロ。

format はメッセージのフォーマット記述, arg1~argn はフォーマット記述中で参照される値で、printf のフォーマット記述のサブセットとなっている。arg1~argn は VP_INT 型にキャストされるため、VP_INT 型に型変換できる任意の型を渡すことができ、型チェックはされない。format および arg1~argn には、次の制限がある。

- format のフォーマット記述は、このマクロから戻った後も変化してはならない。定数文字列を渡すことを想定している。

- format 中に使えるフォーマット指定は次の通り。

%d	引数を符号付き整数とみなし、10進数で表示
%u	引数を符号無し整数とみなし、10進数で表示
%x	引数を符号無し整数とみなし、16進数（英文字は小文字）で表示
%X	引数を符号無し整数とみなし、16進数（英文字は大文字）で表示
%p	引数をポインタとみなし、16進数（英文字は小文字）で表示
%c	引数を文字コードとみなし、文字を表示
%s	引数を文字列を示すポインタとみなし、文字列を表示
%%	'%' を表示（引数は取らない）

%d, %u, %x, %X においては, '%' の直後に表示桁数を指定する 10 進数値を記述することができる。その場合, 表示すべき文字列が指定した桁数に満たない場合には, 指定した桁数内に右詰めで表示する。10 進数値が '0' で始まる場合には, その間に '0' を埋める。

・arg1 ~ argn にポインタを渡す場合 (%s に対応する引数の場合) に, ポインタの指すデータは, このマクロから戻った後も変化してはならない。定数文字列を渡すことを想定している。

(3) void syslog(UINT prio, const char *format, ...)

format 文字列およびそれに続く引数から作成されるメッセージを, 重要度 prio でログ情報として出力するためのライブラリ関数で, 引数の数を可変にしたもの。format に続く引数は最大 5 個まで。format およびそれに続く引数には, syslog_n と同様の制限がある。

このライブラリ関数は, 可変数引数を処理するために内部で文字列をスキャンする。そのため, 実行時間が長くなる可能性があり, 割込み禁止状態で呼び出すべきではない。主にアプリケーションプログラムが用いることを想定している。そのため, このライブラリ関数のソースファイルは, ポートライブラリのディレクトリに置いている。

(4) UINT LOG_MASK(UINT prio)

重要度 prio のみセットされたビットマップを作るマクロ。vmsk_log に渡す引数を作るために用いる。

(5) UINT LOG_UPTO(UINT prio)

重要度 prio 以上の重要度がすべてセットされたビットマップを作るマクロ。vmsk_log に渡す引数を作るために用いる。

(6) void syslog_printf(const char *format, VP_INT *args, void (*putc)(char))

(7) void syslog_print(SYSLOG *p_sys, void (*putc)(char))

(8) void syslog_output(void (*putc)(char))

ログ情報をフォーマット出力するためのライブラリ関数。syslog_printf は渡されたフォーマット文字列と引数を, syslog_print は渡されたログ情報を, syslog_output はログバッファに格納されたログ情報をフォーマット出力する。

システムログタスクが用いることを想定しているため, このライブラリ関数のソースファイルはサポートライブラリのディレクトリに置いている。ただし, 低レベル出力を行うために, システムログ機能内部でも用いている。

4.8 システムログ機能の設定方法

JSP カーネルのシステムログ機能の想定されている設定方法は, 以下の通りである。

(a) 重大な異常事象を示すログ情報は低レベル出力機能を用いて出力し, そうでないログ情報の出力はシステムログタスクに任せる。

ログバッファに記録するログ情報の重要度と, 低レベル出力を用いて出力するログ情報の重要度を適切に設定する。また, ログバッファからログ情報を読み出して外部へ通知するシステムログタスクと, 低レベルの文字出力関数を用意する。

(b) すべてのログ情報を, 低レベル出力機能を用いて出力する。

必要なログ情報はすべて低レベル出力機能を用いて出力するよう設定 (vmsk_log の第 1 パラメータを 0 に設定) する。また, 低レベルの文字出力関数を用意する。システムログタスクは不要。

(c) ログ情報はメモリ上に記録するだけで, システム外部には出力しない。

必要なログ情報はすべてログバッファへ記録するように設定 (vmsk_log の第 2 パラメータを 0 に設定) する。システムログタスクは不要。

(d) ログ情報は記録も出力もしない。

いずれのログ情報も記録 / 出力しないように設定 (vmsk_log の両パラメータともに 0 に設定) する。

別の方法として、OMIT_SYSLOG を定義してコンパイルすることで、システムログ機能をカーネルから取り外し、カーネルのコードサイズを小さくすることができる。ただし、アプリケーションから syslog, syslog_printf, syslog_print, syslog_output の各関数を呼び出している場合、それらの関数のコードは外れない。また、カーネルからのログ情報は記録 / 出力しないが、アプリケーションからのログ情報は記録 / 出力したい場合には、カーネルのみ OMIT_SYSLOG を定義してコンパイルすればよい。この場合、システムログ機能の初期化関数 (_kernel_syslog_initialize) と終了処理関数 (_kernel_syslog_terminate) は、アプリケーションから呼び出す必要がある。

なお、(b) ~ (c) の設定に固定して使用する場合にも、カーネル内の一部のコードが不要になり、コードサイズを小さくできる余地があるが、簡易な方法は用意していない。

5. システムサービス

この節では、JSP カーネルがサポートしているシステムインタフェースレイヤ (SIL) と、JSP カーネルが標準的に動作させるドライバおよびシステムタスクについて説明する。

5.1 システムインタフェースレイヤ (SIL)

JSP カーネルは、ITRON デバイスドライバ設計ガイドラインの一部分として検討されているシステムインタフェースレイヤ (SIL) の中で、以下に挙げる機能をサポートしている。SIL を用いるプログラムからは、t_services.h に代えて、s_services.h をインクルードする。

ITRON デバイスドライバ設計ガイドラインでは、デバイスドライバの中で、SIL を通して直接デバイスにアクセスするモジュール (PDIC) と、カーネルの機能を用いるモジュール (GDIC) を分離することにしている。すなわち、PDIC は SIL を用いるがカーネルの機能は用いず、GDIC はカーネルの機能は用いるが SIL を用いてはならない。そのため、s_services.h には、カーネルを用いるための宣言や定義は含まれていない。

5.1.1 割込みロック状態の制御

デバイスを扱うプログラムの中では、すべての割込み (NMI を除く、以下同じ) を禁止したい場合がある。μITRON4.0 仕様の CPU ロック状態は、カーネルの管理外の割込み (NMI 以外にカーネルの管理外の割込みがあるかは、JSP カーネルではターゲット依存) を禁止するとは限らず、このような場合に用いるのは適切でない。

そこで、すべての割込みを禁止した状態を割込みロック状態と呼び、SIL では割込みロック状態を制御するための以下の機能を用意している。

(1) SIL_PRE_LOC

割込みロック状態の制御に必要な変数を宣言し、それを初期化するマクロ。このマクロを記述した時点で、割込みの禁止状態を記録する。SIL_LOC_INT, SIL_UNL_INT を用いる関数 (ブロック) の先頭の変数宣言部に記述しなければならない。

(2) SIL_LOC_INT()

すべての割込みを禁止し、割込みロック状態に移行する。

(3) SIL_UNL_INT()

SIL_PRE_LOC を記述した時点の状態に戻す .

割込みロック状態の制御機能の使用例は次の通り .

```
{  
    SIL_PRE_LOC;  
  
    SIL_LOC_INT();  
    この間はすべての割込みが禁止される  
    この間にサービスコールを呼び出してはならない  
    SIL_UNL_INT();  
}
```

なお、JSP カーネル自身は割込みロック状態は管理していないため、割込ロック状態ではサービスコールを呼び出してはならない（呼び出した場合の動作は保証されない）。

5.1.2 微少時間待ち

デバイスをアクセスする際に、微少な時間待ちを入れなければならない場合がある。そのような場合に、nop をいくつか入れるなどの方法で対応すると、ポータビリティが悪くなる。そこで SIL では、微少な時間待ちを行うための機能を用意している。

(1) void sil_dly_nse(UINT dlytim)

dlytim で指定された以上の時間（単位はナノ秒）、ループなどによって待つ。指定した値によっては、指定した時間よりもかなり長く待つ場合があるので注意すること。

5.1.3 エンディアン

プロセッサのエンディアンを知るためのマクロとして、以下のマクロを定義している。

(1) SIL_ENDIAN

リトルエンディアンプロセッサでは SIL_ENDIAN_LITTLE (=0)、ビッグエンディアンプロセッサでは SIL_ENDIAN_BIG (=1) にマクロ定義される。

5.1.4 メモリ空間アクセス関数

メモリ空間にマッピングされたデバイスレジスタや、デバイスとの共有メモリをアクセスするために、以下の関数を用意している。

(1) VB sil_reb_mem(VP mem)

mem で指定されるアドレスから、8 ビット単位で読んだ値を返す。

(2) void sil_wrb_mem(VP mem, VB data)

mem で指定されるアドレスに、data で指定される値を 8 ビット単位で書き込む。

(3) VH sil_reh_mem(VP mem)

mem で指定されるアドレスから、16 ビット単位で読んだ値を返す。

(4) void sil_wrh_mem(VP mem, VH data)

mem で指定されるアドレスに、data で指定される値を 16 ビット単位で書き込む。

(5) VH sil_reh_lem(VP mem)

mem で指定されるアドレスから , 16 ビット単位でリトルエンディアンで読んだ値を返す . リトルエンディアンプロセッサでは , sil_reh_mem と一致する .

(6) void sil_wrh_lem(VP mem, VH data)

mem で指定されるアドレスに , data で指定される値を 16 ビット単位でリトルエンディアンで書き込む . リトルエンディアンプロセッサでは , sil_wrh_mem と一致する .

(7) VH sil_reh_bem(VP mem)

mem で指定されるアドレスから , 16 ビット単位でビッグエンディアンで読んだ値を返す . ビッグエンディアンプロセッサでは , sil_reh_mem と一致する .

(8) void sil_wrh_bem(VP mem, VH data)

mem で指定されるアドレスに , data で指定される値を 16 ビット単位でビッグエンディアンで書き込む . ビッグエンディアンプロセッサでは , sil_wrh_mem と一致する .

(9) VW sil_rew_mem(VP mem)

mem で指定されるアドレスから , 32 ビット単位で読んだ値を返す .

(10) void sil_rwr_mem(VP mem, VW data)

mem で指定されるアドレスに , data で指定される値を 32 ビット単位で書き込む .

(11) VW sil_rew_lem(VP mem)

mem で指定されるアドレスから , 32 ビット単位でリトルエンディアンで読んだ値を返す . リトルエンディアンプロセッサでは , sil_rew_mem と一致する .

(12) void sil_rwr_lem(VP mem, VW data)

mem で指定されるアドレスに , data で指定される値を 32 ビット単位でリトルエンディアンで書き込む . リトルエンディアンプロセッサでは , sil_rwr_mem と一致する .

(13) VW sil_rew_bem(VP mem)

mem で指定されるアドレスから , 32 ビット単位でビッグエンディアンで読んだ値を返す . ビッグエンディアンプロセッサでは , sil_rew_mem と一致する .

(14) void sil_rwr_bem(VP mem, VW data)

mem で指定されるアドレスに , data で指定される値を 32 ビット単位でビッグエンディアンで書き込む . ビッグエンディアンプロセッサでは , sil_rwr_mem と一致する .

なお , JSP カーネルのターゲット非依存部では , I/O 空間にアクセスするための関数を用意していないが , ターゲット依存部でサポートすることは可能である . 詳しくは , ターゲット毎のマニュアルを参照すること .

5.2 システムクロックドライバ

システムクロックドライバは , ハードウェアタイマを用いて周期的に割り込みを発生させ , isig_tim を呼び出してカーネルにタイムティックを供給する . システムクロックドライバは , システムコンフィギュレーションファイルに timer.cfg をインクルードすることで , システムに組み込むことができる .

5.2.1 システムクロックドライバの内部構成

システムクロックドライバは、タイマの起動処理、タイマ割込みハンドラ、タイマの停止処理で構成される。

(1) void timer_initialize(VP_INT exinf)

タイマの起動処理。タイマを初期化し、周期的なタイマ割込み要求を発生させる。カーネルに初期化ルーチンとして登録する。exinfは無視する。

(2) void timer_handler()

タイマ割込みハンドラ。タイマ割込み要求をクリアした後、isig_timを呼び出してタイムティックを供給する。カーネルに割込みハンドラとして登録する。

(3) void timer_terminate(VP_INT exinf)

タイマの停止処理。周期的なタイマ割込み要求を停止させる。カーネルに終了処理ルーチンとして登録する。exinfは無視する。

5.3 シリアルインタフェースドライバ

シリアルインタフェースドライバは、シリアルポートを扱うためのドライバである。シリアルインタフェースドライバは、システムコンフィギュレーションファイルにserial.cfgをインクルードすることで、システムに組み込むことができる。

NEWLIBやGLIBCなどの標準Cライブラリを使用する場合には、標準Cライブラリの低レベル入出力ルーチンをシリアルインタフェースドライバを呼び出すものにする。タスクの標準入出力をシリアルインタフェースドライバ経由に切り替えることができる。具体的な方法は、用いる標準Cライブラリに依存する。

5.3.1 シリアルインタフェースドライバのサービスコール

シリアルインタフェースドライバを呼び出すサービスコールの仕様は下記の通りである。この中で、シリアルポートのID番号(portid)の解釈はターゲット依存となる。

これらのサービスコールは、非タスクコンテキストから呼び出すことはできない。また、serial_rea_datとserial_wri_datは、ディスパッチ保留状態で呼び出すことはできない。いずれも、呼び出した場合にはE_CTXエラーとなる。

(1) ER serial_opn_por(ID portid)

portidで示されるシリアルポートをオープンし、受信/送信が可能な状態にする。

(2) ER serial_cls_por(ID portid)

portidで示されるシリアルポートをクローズする。

(3) ER_UINT serial_rea_dat(ID portid, char *buf, UINT len)

portidで示されるシリアルポートから、lenバイトの文字列を受信し、bufからの領域に入れる。受信した文字数またはエラーコードを返す。

(4) ER_UINT serial_wri_dat(ID portid, char *buf, UINT len)

portidで示されるシリアルポートに、bufからのlenバイトの文字列を送信する。送信した文字数またはエラーコードを返す。

(5) ER serial_ctl_por(ID portid, UINT ioctl)

portidで示されるシリアルポートの制御情報を、ioctlで示される値に設定する。

ioctl には、以下の制御情報を表す定数を、ビット毎に論理和をとったものを指定する。

IOCTL_ECHO (エコーバックモード)
このビットを設定すると、シリアルインタフェースドライバがエコーバックを行う。具体的には、バッファから文字を取り出す度に、その文字を書き出す。

IOCTL_CRLF (改行モード)
LF (line feed) を書き出すと、CR (carriage return) + LF に変換して書き出す。

IOCTL_FCSND (出力フロー制御)
文字を送信する処理に対して、XON/XOFF によるフロー制御を行う。すなわち、STOP (コントロール-S) を受信すると送信を停止し、START (コントロール-Q) を受信すると送信を再開する。

IOCTL_FCANY (送信フロー制御で任意の文字で送信再開)
IOCTL_FCOUT を指定している時に、送信停止中に受信した任意の文字で送信を再開する。

IOCTL_FCRCV (受信フロー制御)
文字を受信する処理に対して、XON/XOFF によるフロー制御を行う。すなわち、受信バッファの残り領域が少なくなると STOP (コントロール-S) を送出し、残り領域が増えれば START (コントロール-Q) を送出する。

なお、オープン直後のデフォルトの設定値は (IOCTL_ECHO | IOCTL_CRLF | IOCTL_FCOUT | IOCTL_FCIN) である。

(6) ER serial_ref_por(ID portid, T_SERIAL_RPOR *pk_rpor)

portid で示されるシリアルポートの状態を参照し、pk_rpor で指定されるパケットに返す。パケット中の reacnt には受信バッファ中の文字数を、wricnt には送信バッファ中の文字数を返す。

5.3.2 シリアルインタフェースドライバの内部構成

シリアルインタフェースドライバは、前記のサービスコールに加えて、初期化処理と割り込みハンドラで構成される。初期化処理は、カーネルに初期化ルーチンとして登録する。割り込みハンドラは、カーネルに割り込みハンドラとして登録する。これらの登録処理は serial.cfg に含まれる。

(1) void serial_initialize(VP_INT exinf)

シリアルインタフェースドライバを初期化する。カーネルに初期化ルーチンとして登録する。exinf は無視する。

(2) 割り込みハンドラ

シリアル I/O デバイスの種類によって、割り込みハンドラの種類や数は異なる。具体的には、送信割り込みと受信割り込みが別れているものと別れていないものや、ポートを複数持つシリアル I/O デバイスでポート毎に割り込みハンドラが別れているものと別れていないものがある。シリアルインタフェースドライバの割り込みハンドラは、カーネルに割り込みハンドラとして登録する。

5.4 システムログタスク

システムログタスクは、カーネル内のログバッファからログ情報を取り出し、デバイスにアクセスするサービスを用いて外部に出力するタスクである。

JSP カーネルの標準配布キットに含まれるシステムログタスクは、シリアルインタフェースにログ情報を文字列の形で出力するもので、システムログタスクの一例という位置付けで提供している。このシステムログタスクは、システムコンフィギュレーションファイルに logtask.cfg をインクルードすることで、システムに組み込むことができる。

6 . サポートライブラリ

サポートライブラリは、アプリケーションやシステムサービスを作成するために利用できるライブラリ関数群である。現バージョンでは、システムサービスやサンプルプログラムで使う最低限の関数しか用意していない。

(1) `const char *itron_strerror(ER ercd)`

`ercd` で示されるメインエラーコードに対応するエラーコードの文字列を返す。返された文字列を書き換えてはならない。

(2) `void t_perror(const char *file, int line, const char *expr, ER ercd)`

エラーメッセージをシステムログサービスに出力する `.assert` マクロなどで利用することを想定している。

7 . 開発環境・インストール・ポータリング

7.1 ディレクトリ・ファイル構成

ソースファイルのディレクトリ構成は次の通り。

<code>include/</code>	共通ヘッダファイル
<code>kernel/</code>	カーネルソースファイル
<code>systask/</code>	システムサービスソースファイル
<code>library/</code>	サポートライブラリソースファイル
<code>config/</code>	ターゲット依存部
<code> m68k/</code>	<code>M68040</code> プロセッサ依存ファイル
<code> dve68k/</code>	<code>DVE-68K/40</code> システム依存ファイル
<code> sh3/</code>	<code>SH3</code> プロセッサ依存ファイル
<code> ms7727cp01/</code>	<code>MS7727CP01</code> システム依存ファイル
<code> solution_engine/</code>	<code>Solution Engine</code> システム依存ファイル
<code> sh3-ghs/</code>	<code>SH3</code> プロセッサ依存ファイル (GHS 開発環境)
<code> ms7727cp01/</code>	<code>MS7727CP01</code> システム依存ファイル
<code> solution_engine/</code>	<code>Solution Engine</code> システム依存ファイル
<code> sh1/</code>	<code>SH1</code> プロセッサ依存ファイル
<code> kz_sh1/</code>	<code>KZ-SH1-01</code> システム依存ファイル
<code> tokiwa_sh1/</code>	<code>SH1/CPUB</code> システム依存ファイル
<code> h8/</code>	<code>H8</code> プロセッサ依存ファイル
<code> akih8_3048f/</code>	<code>AKI-H8/3048F</code> システム依存ファイル
<code> akih8_3052f/</code>	<code>AKI-H8/3052F</code> システム依存ファイル
<code> akih8_3067f/</code>	<code>AKI-H8/3067F</code> システム依存ファイル
<code> akih8_3068f/</code>	<code>AKI-H8/3068F</code> システム依存ファイル
<code> akih8_3069f/</code>	<code>AKI-H8/3069F</code> システム依存ファイル
<code> armv4/</code>	<code>ARMV4</code> プロセッサ依存ファイル
<code> excalibur/</code>	<code>Excalibur-ARM</code> システム依存ファイル
<code> armv4-ghs/</code>	<code>ARMV4</code> プロセッサ依存ファイル (GHS 開発環境)
<code> integrator/</code>	<code>Integrator</code> システム依存ファイル
<code> m32r/</code>	<code>M32R</code> プロセッサ依存ファイル
<code> m3a2131g50/</code>	<code>M3A-2131G50</code> システム依存ファイル
<code> microblaze/</code>	<code>MicroBlaze</code> プロセッサ依存ファイル
<code> miref/</code>	<code>MIREF</code> システム依存ファイル
<code> mire_multi/</code>	<code>MIRE_MULTI3000</code> システム依存ファイル
<code> mutlimedia/</code>	<code>MultiMedia Board</code> システム依存ファイル
<code> tms320c54x/</code>	<code>TMS320C54x</code> プロセッサ依存ファイル
<code> c5402dsk/</code>	<code>TMS320VC5402 DSK</code> システム依存ファイル
<code> xstormy16/</code>	<code>Xstormy16</code> プロセッサ依存ファイル
<code> simulator/</code>	三洋マイコン開発ツール環境 依存ファイル

mips3/	MIPS3 プロセッサ依存ファイル
vr4131/	VR4131 システム依存ファイル
vr5500/	VR5500 システム依存ファイル
linux/	Linux 上のシミュレーション環境依存ファイル
windows/	Windows 上のシミュレーション環境依存ファイル
tools/	開発環境依存ディレクトリ
WINDOWS/	Windows 上のサンプルプログラムとプロジェクトファイル
GHS/	GHS (Green Hills Software) 開発環境用のファイル
C5402DSK/	TMS320VC5402 DSK 用のプロジェクトファイル
pdic/	PDIC (デバイスドライバの OS 非依存部分)
simple_sio/	簡易 SIO ドライバ (シリアルドライバが使用するもの)
cfg/	カーネルコンフィギュレータ
utils/	ユーティリティ
sample/	サンプルプログラムと Makefile
doc/	ドキュメント
windev/	Windows デバイスマネージャ

ターゲット非依存部 (カーネルコンフィギュレータは除く) の各ファイルの概要は次の通り。

README	TOPPERS/JSP カーネルの簡単な紹介
configure	コンフィギュレーションスクリプト
include/	
itron.h	ITRON 仕様共通規定に関連する定義
kernel.h	μITRON4.0 仕様に関連する定義
kernel_debug.h	μITRON4.0 仕様 デバッグ用インクルードファイル
sil.h	システムインタフェースレイヤ (SIL)
t_stddef.h	カーネル・アプリケーション 共通インクルードファイル
t_config.h	ターゲット依存情報の定義
t_syslog.h	システムログサービス関連の定義
t_services.h	アプリケーション用 標準インクルードファイル
s_services.h	デバイスドライバ用 標準インクルードファイル
kernel_cfg.h	kernel_cfg.c 用のインクルードファイル
timer.h	システムクロックドライバ関連の定義
serial.h	シリアルインタフェースドライバ関連の定義
logtask.h	システムログタスク関連の定義
linux_sigio.h	Linux 用 ノンブロッキング I/O サポート
kernel/	
Makefile.kernel	カーネルのファイル構成の定義
jsp_kernel.h	JSP カーネル用 標準インクルードファイル
jsp_rename.def	カーネルの内部識別名のリネーム定義
jsp_rename.h	カーネルの内部識別名のリネーム
jsp_unrename.h	カーネルの内部識別名のリネーム解除
check.h	エラーチェック用マクロ
queue.h	ダブルリンクキューの構造と操作
startup.c	カーネルの初期化処理
banner.c	カーネルの起動メッセージの出力
task.h	タスク操作ルーチン関連の定義
task.c	タスク操作ルーチン
wait.h	待ち状態操作ルーチン関連の定義
wait.c	待ち状態操作ルーチン
time_event.h	タイムイベント管理関連の定義
time_event.c	タイムイベント管理
syslog.h	システムログ機能関連の定義
syslog.c	システムログ機能
task_manage.c	タスク管理機能
task_sync.c	タスク付属同期機能
task_except.c	タスク例外処理機能

semaphore.h	セマフォ機能関連の定義
semaphore.c	セマフォ機能
eventflag.h	イベントフラグ機能関連の定義
eventflag.c	イベントフラグ機能
dataqueue.h	データキュー機能関連の定義
dataqueue.c	データキュー機能
mailbox.h	メールボックス機能関連の定義
mailbox.c	メールボックス機能
mempfix.h	固定長メモリプール関連の定義
mempfix.c	固定長メモリプール
time_manage.c	システム時刻管理機能
cyclic.h	周期ハンドラ機能関連の定義
cyclic.c	周期ハンドラ機能
sys_manage.c	システム管理機能
interrupt.h	割込み管理機能関連の定義
interrupt.c	割込み管理機能
exception.h	CPU 例外管理機能関連の定義
exception.c	CPU 例外管理機能
systask/	
timer.c	システムクロックドライバ
timer.cfg	システムクロックドライバの設定記述
serial.c	シリアルインタフェースドライバ
serial.cfg	シリアルインタフェースドライバの設定記述
logtask.c	システムログタスク
logtask.cfg	システムログタスクの設定記述
linux_sigio.c	Linux 用 ノンブロッキング I/O サポート
linux_sigio.cfg	Linux 用 ノンブロッキング I/O サポートの設定記述
linux_serial.c	Linux 用 疑似シリアルドライバ
linux_serial.cfg	Linux 用 疑似シリアルドライバの設定記述
cxxrt.c	C++対応ランタイム本体
cxxrt.cfg	C++対応ランタイム用オブジェクト設定
newlibrt.c	NEWLIB 対応ランタイム
library/	
log_output.c	システムログ機能用ライブラリ関数 (syslog_output など)
strerror.c	itron_strerror 関数
t_perror.c	t_perror 関数
vasyslog.c	syslog 関数
utils/	
makedep	依存関係定義の生成
genoffset	offset.h 生成プログラム
gencheck	パラメータチェック用ファイルの生成
genrename	内部シンボルリネーム定義の生成
rename	内部シンボルのリネーム処理
sample/	
Makefile	サンプルの Makefile
Makefile.linux	サンプルの Makefile (Linux 用)
sample1.cfg	サンプルプログラム(1)の設定記述
sample1.h	サンプルプログラム(1)に関する定義
sample1.c	サンプルプログラム(1)の本体
cxx_sample1.cfg	C++用サンプルプログラム(1)の設定記述
cxx_sample1.h	C++用サンプルプログラム(1)に関する定義
cxx_sample1.c	C++用サンプルプログラム(1)の本体
cxx_sample2.cfg	C++用サンプルプログラム(1)の設定記述
cxx_sample2.h	C++用サンプルプログラム(1)に関する定義
cxx_sample2.c	C++用サンプルプログラム(1)の本体

```

doc/
  user.txt          TOPPERS/JSP カーネル ユーザズマニュアル
  gnu_install.txt  GNU 開発環境構築マニュアル
  m68k.txt         M68040 ターゲット依存部マニュアル
  sh3.txt          SH3 ターゲット依存部マニュアル
  sh1.txt          SH1 ターゲット依存部マニュアル
  h8.txt           H8 ターゲット依存部マニュアル
  armv4.txt        ARMV4 ターゲット依存部マニュアル
  m32r.txt         M32R ターゲット依存部マニュアル
  microblaze.txt   MicroBlaze ターゲット依存部マニュアル
  tsm320c54x.txt  TMS320C54x ターゲット依存部マニュアル
  xstormy16.txt    Xstormy16 ターゲット依存部マニュアル
  mips3.txt        MIPS3 ターゲット依存部マニュアル
  linux.txt        Linux シミュレーション環境依存部マニュアル
  windows.txt      Windows シミュレーション環境依存部マニュアル
  config.txt       JSP カーネル ターゲット依存部 ポーティングガイド
  configurator.txt JSP カーネル コンフィギュレータ仕様
  design.txt       JSP カーネル 設計メモ

```

7.2 開発環境

JSP カーネルを用いたシステム構築には、以下のツールが必要である。

ホスト環境用のツール

標準規格に準拠した C コンパイラ, C ライブラリ
 C++ コンパイラ, C++ ライブラリ, STL
 動作確認: GNU C++ 2.95.3, 3.2, 3.3 (Linux 環境)
 GNU C++ 3.2 (Cygwin 環境)
 Visual C++ 6.0, .NET
 perl (動作確認は 5.6.1)
 GNU Make (動作確認は 3.79.1)

クロス環境用のツール

GNU 開発環境
 BINUTILS (アセンブラ, リンカなど)
 GCC または GCC-CORE (C コンパイラ)
 GDB (デバッガ)
 NEWLIB (標準 C ライブラリ)

GNU 開発環境をインストール方法については、「GNU 開発環境構築マニュアル」を用意しているので、それを参照するとよい。また、動作確認バージョンについては、ターゲット毎のマニュアルを参照すること。

ホスト環境用の C コンパイラと C ライブラリは、クロス環境用のツールのインストールに必要な。また、C++ コンパイラ, C++ ライブラリと STL (Standard Template Library) は、カーネルのコンフィギュレーションツールのコンパイルに必要な。クロス環境用のツールとコンフィギュレーションツールをバイナリで入手した場合には、これらのツールは必要ない。

クロス環境用の標準 C ライブラリは、アプリケーションが標準 C ライブラリを使用しない場合には、必要ない。ただし、コンパイラが標準 C ライブラリ関数 (memcpy, memset など) を呼び出すコードを生成する場合があります、その場合には標準 C ライブラリが必要である。ないしは、生成したコードが呼び出す関数のみを自分で用意してもよい。

以下では、これらのツールが用意できていることを前提に、UNIX マシン (動作確認は Linux) 上で構築手順を説明する。また以下の説明では、make コマンドが GNU Make であるものとする (JSP カーネルの Makefile は、GNU Make の拡張機能を用いている)。

7.3 コンフィギュレーションツールの構築

カーネルを構築する前に、まず、コンフィギュレーションツールをコンパイルする必要がある(コ

ンフィギュレーションツールをバイナリで入手した場合には、このステップは必要ない)。

JSP カーネルのコンフィギュレーションツールは、コンフィギュレータ (cfg プログラム) とパラメータチェックプログラム (chk プログラム) から構成される。コンフィギュレーションツールの使い方については、「7.9 コンフィギュレーションツールの使い方」を参照すること。

コンフィギュレーションツール (cfg プログラムと chk プログラム) は、cfg ディレクトリに移動し、make depend で依存関係ファイル (Makefile.depend) を生成した後、make コマンドにより生成される。

```
% cd cfg
% make depend
% make
```

7.4 サンプルプログラムの構築

次に、サンプルプログラムを構築する方法を説明する。

まず、サンプルプログラムのオブジェクトファイルを置くディレクトリを作成し、コンフィギュレーションスクリプトを実行する。例えば、オブジェクトファイルを置くディレクトリを、JSP カーネルのソースファイルを展開したディレクトリの下に OBJ という名称のディレクトリにする場合には、次のコマンドを実行する (ディレクトリの場所は名称は任意に決めてよい)。

```
% mkdir OBJ
% cd OBJ
% perl ../configure -C m68k -S dve68k
```

ここで、m68k はターゲットプロセッサ名、dve68k はターゲットシステム名である。これらのコンフィギュレーションスクリプトのオプションについては、次の節で説明する。

コンフィギュレーションスクリプトの実行により、カレントディレクトリには、サンプルプログラムを構築するための Makefile、サンプルプログラム用のコンフィギュレーションファイル (sample1.cfg)、サンプルプログラム本体 (sample1.h および sample1.c) が生成される。

コンフィギュレーションスクリプトの実行後、必要であれば Makefile を修正する。Makefile の修正方法については、「7.7 Makefile の修正」を参照すること。

その後、make depend で依存関係ファイル (Makefile.depend) を生成した後、make コマンドによりサンプルプログラムのロードモジュール (jsp または jsp.exe) が生成できる。依存関係ファイルの生成には若干時間がかかる。

```
% make depend
% make
```

ここで構築したサンプルプログラム (sample1.h, sample1.c, sample1.cfg) は、JSP カーネルの基本的な動作を確認するためのものである。このプログラムの概要説明は、sample1.c の先頭のコメントにある。

7.5 アプリケーションとカーネルを別々に構築する方法

前節で説明した方法では、アプリケーションとカーネルを同時に生成するため、オブジェクトファイルを置くディレクトリに非常に多くのファイルが作成されて、扱いにくくなる。そこで、カーネルを修正する頻度が低い場合には、カーネルは事前に構築しておき、後でアプリケーションだけを構築する方法を用意している。以下では、サンプルプログラムを構築を例に、その手順について説明する。

まず、カーネルを構築するディレクトリを作成し、コンフィギュレーションスクリプトを実行する。例えば、カーネルを構築するディレクトリを、JSP カーネルのソースファイルを展開したディレクトリの下に KERNEL という名称のディレクトリにする場合には、次のコマンドを実行する (ディレクトリの場所は名称は任意に決めてよい)。

```
% mkdir KERNEL
% cd KERNEL
% perl ../configure -C m68k -S dve68k
```

これにより、カーネルを構築するディレクトリに、Makefile、sample1.cfg、sample1.h、sample1.c が生成されるが、Makefile 以外は使用しない。

make depend で依存関係ファイル (Makefile.depend) を生成した後、make ibkernel.a によりカーネルライブラリ (libkernel.a) が生成できる。

```
% make depend
% make libkernel.a
```

次に、アプリケーションを構築するディレクトリを作成し、コンフィギュレーションスクリプトを実行する。例えば、アプリケーションを構築するディレクトリを、JSP カーネルのソースファイルを展開したディレクトリの下に APL という名称のディレクトリにする場合には、次のコマンドを実行する (ディレクトリの場所は名称は任意に決めてよい)。

```
% cd ..
% mkdir APL
% cd APL
% perl ../configure -C m68k -S dve68k -L ../KERNEL
```

ここで -L オプションには、カーネルを構築したディレクトリのパスを指定する。

最後に、make depend で依存関係ファイル (Makefile.depend) を生成した後、make コマンドによりサンプルプログラムのロードモジュール (jsp または jsp.exe) が生成できる。

```
% make depend
% make
```

この手順では、アプリケーション構築時にはカーネルの再構築が必要かチェックしないため、カーネルのソースコードを修正した場合には、カーネルを構築したディレクトリで make libkernel.a を再実行する必要がある。また、アプリケーション構築時にカーネルライブラリが更新されたかチェックしないため、アプリケーションを構築したディレクトリで、ロードモジュールを削除した後 make を再実行する必要がある。

以上では、カーネルとアプリケーションを別々のディレクトリで構築したが、-L オプションにレントディレクトリ (".") を指定することで、同じディレクトリで (別々に) 構築することもできる。具体的には、次の手順となる。

```
% mkdir OBJ
% cd OBJ
% perl ../configure -C m68k -S dve68k -L .
% make depend
% make libkernel.a
% make cleankernel
% make
```

ここで、make cleankernel は、カーネルライブラリを生成するための中間ファイルを削除するものである。この手順では、make depend によりカーネルライブラリに関する依存関係を生成しないため、カーネルのソースコードを修正した場合には、必ず make cleankernel (または、make clean) してから、make ibkernel.a する必要があるので注意すること。さらに、ロードモジュールを削除した後 make を再実行する必要があるのは、前の場合と同様である。

7.6 コンフィギュレーションスクリプトの使い方

コンフィギュレーションスクリプトは、JSP カーネルおよびアプリケーションプログラムを構築するために必要な基本的なコンフィギュレーションを行うためのプログラムである。JSP カーネル

を用いてアプリケーションを作成する場合には、まずオブジェクトファイルを置くディレクトリを作成し、そのディレクトリでコンフィギュレーションスクリプトを実行する。オブジェクトファイルを置くディレクトリの場所や名称は、任意に決めてよい。

コンフィギュレーションスクリプトに対するオプションは次の通り。

- C <プロセッサ名>
ターゲットプロセッサ名またはシミュレーション環境名を、config ディレクトリの中のディレクトリ名称で指定する（必須）。
- S <システム名>
ターゲットシステム名を、config の下のプロセッサのディレクトリの中のディレクトリ名称で指定する。シミュレーション環境の場合には、指定する必要がない。
- T <開発環境名>
開発環境名を、config の下のディレクトリ名称の後半の名称で指定する。GNU 開発環境を用いる場合には、指定する必要がない。
- A <アプリケーションプログラム名>
アプリケーションプログラムの名称を指定する。省略した場合には、標準のサンプルプログラム（sample1）となる。
- U <オブジェクトファイル名>
アプリケーションプログラムのメインのオブジェクトファイル（-A で指定したアプリケーションプログラム名に".o"を付加したもの）以外に、リンクすべきオブジェクトファイルの名称を、".o"を付加した形で指定する。""で囲むことによって、複数のファイルを指定することも可能である（-U オプションを複数使ってはならない）。
- L <カーネルライブラリのディレクトリ名>
事前に構築したカーネルを用いて、アプリケーションのみを構築する場合には、このオプションにカーネルライブラリ(libkernel.a)の置かれたディレクトリ名を指定する。このオプションの使用方法については、「7.5 アプリケーションとカーネルを別々に構築する方法」を参照すること。
- D <JSP カーネルソースディレクトリ名>
JSP カーネルのソースコードを置いたディレクトリ名を指定する。省略した場合には、configure の置かれているディレクトリとなる。

コンフィギュレーションスクリプトが行う処理は次の通りである。

(1) Makefile の生成

sample ディレクトリから適切な Makefile を選択し、必要な箇所を書き換えて、Makefile を生成する。

(2) サンプルプログラムの生成

指定したアプリケーションプログラムが sample ディレクトリにある場合、適切なサンプルプログラムのソースファイルを選択し、必要な箇所を書き換えて、サンプルプログラムのソースファイル（例えば、sample1.h, sample1.c, sample1.cfg）を生成する。

7.7 Makefile の修正

JSP カーネルの実行環境によっては、コンフィギュレーションスクリプトが生成した Makefile を修正することが必要になる。ここでは、Makefile の中で、修正が必要となる可能性の高い箇所について説明する。

なお、Makefile を修正した後にコンフィギュレーションスクリプトを再実行すると、修正した Makefile が上書きされてしまうので注意すること（古いものが Makefile.bak に保存される）。

(A) ターゲット名の定義

CPU はターゲットプロセッサ名, SYS はターゲットシステム名, TOOL は開発環境名に定義する。これらの定義は, コンフィギュレーションスクリプトが行う。

(B) オブジェクトファイルの拡張子の設定

Cygwin 環境でコンパイルする時には, OBJEXT を "exe" に定義する必要がある。これは, Cygwin 環境では, オブジェクトプログラムに拡張子 "exe" が付加されるのに対応するためのものである。Cygwin 環境であることを判定できれば, コンフィギュレーションスクリプトがこの定義を行う。

(C) 実行環境の定義 (ターゲット依存)

ターゲットによっては, 実行環境に対応してターゲット依存部のコードを差し換える場合がある。これを可能にするために, 実行環境の名称を DBGENV に定義している。標準では, GDB スタブを用いることを想定して, これを GNU_STUB に定義しているが, ターゲット依存の定義を入れた Makefile.config で上書きされる場合もある。どのターゲットがどの実行環境に対応しているかは, ターゲット毎のマニュアルを参照すること。

(D) カーネルライブラリのディレクトリ名の定義

KERNEL_LIB には, カーネルライブラリの置かれたディレクトリ名を定義する。この定義は, 通常はコンフィギュレーションスクリプトが行うが, 事後に KERNEL_LIB の定義を変更してもかまわない。

(E) 共通コンパイルオプションの定義

全体に共通するコンパイルオプションの追加が必要な場合には, 下の変数の定義を変更する。そのコンパイルオプションが, 特定のターゲットで常に必要な場合には, ターゲット依存の定義を入れた Makefile.config を修正すべきである。追加の可能性のあるコンパイルオプションについては, 「7.8 コンパイルオプション」を参照のこと。

CDEFS	-D オプションを記述する。
INCLUDES	-I オプションを記述する。
COPTS	コンパイラに対するその他のオプションを記述する。
LDFLAGS	リンカに対するオプションを記述する。
LIBS	ライブラリリンクのためのオプションを記述する。

(F) アプリケーションプログラムに関する定義

アプリケーションプログラムが一つの C ソースファイル (*.c) のみで構成されている場合には, UNAME にそのファイル名を定義すればよい。アプリケーションプログラムが複数のソースファイルで構成される場合には, UNAME にそのアプリケーション名を定義し, オブジェクトファイル名を UTASK_ASMOBS および UTASK_COBS に列挙する。いずれの場合にも, コンフィギュレーションファイルは, UNAME に定義した名前に拡張子 "cfg" を付加した名前とする。

ソースファイルをコンパイルするのとは別のディレクトリに置く場合には, UTASK_DIRS にそのディレクトリを追加する。また, アプリケーションのコンパイルに必要なコンパイルオプションや, アプリケーションがライブラリを必要とする場合には, UTASK_CFLAGS および UTASK_LIBS に定義する。

(G) オブジェクトファイル名の定義

オブジェクトファイル名を OBJNAME に定義する。デフォルトは jsp である。

(H) ターゲットファイルの定義

ロードモジュールの形式を指定する。具体的には, ELF 形式の時は \$(OBJFILE) または

\$(OBJNAME).out (PARTNER-J 環境の時) , バイナリ形式の時は\$(OBJNAME).bin , モトローラ S 形式の時は\$(OBJNAME).srec を指定する .
\$(OBJFILE) は , Cygwin 環境で OBJEXT を "exe" に定義した時には\$(OBJNAME).exe , そうでない場合には\$(OBJNAME)となる .

(1) カーネルのコンフィギュレーションファイルの生成

ソフトウェア部品のコンフィギュレータを追加する場合には , この規則を修正することが必要である .

7.8 コンパイルオプション

JSP カーネルのコード中には , assert マクロが使われている . assert マクロは , NDEBUG を定義することで , オブジェクトコード中から消すことができる . カーネルのデバッグが終了すれば , CDEFS に -DNDEBUG を指定してコンパイルした方が効率がよくなる .

また , システムログ機能を取り外すための OMIT_SYSLOG を用意している . 詳しくは , 「4.8 システムログ機能の設定方法」を参照すること .

7.9 コンフィギュレーションツールの使い方

以下では , cfg プログラムと chk プログラムのオプションについて説明する . これらのプログラムによるコンフィギュレーション手順については , 「2.10 静的 API とコンフィギュレータ」を参照すること .

cfg プログラムと chk プログラムに共通のオプションは次の通り .

- cpu <プロセッサ名>
ターゲットプロセッサ名を指定する .
- system <システム名>
ターゲットシステム名を指定する .
- h, --help
ヘルプメッセージを表示する .
- v
処理の途中結果を表示する .
- le, --english
メッセージを英語で表示する (デフォルト) .
- lj, --japanese
メッセージを日本語で表示する .

cfg プログラムに対するオプションは次の通り .

- s, --source <ファイル名>
<ファイル名>で指定されたシステムコンフィギュレーションファイル (C 言語のプリプロセッサで処理したもの) を読み込む . <ファイル名> を省略した場合は , システムコンフィギュレーションファイルを標準入力から読み込む (オプション自身を省略してはならない) .
- c, --check
静的 API のパラメータチェックに用いるファイルを kernel_chk.c に生成する (デフォルトでは生成しない) .
- obj, --dump-object <ファイル名>
静的 API の解析内容を含むオブジェクト定義ファイルを , <ファイル名> で指定されたファイルに生成する . <ファイル名> を省略した場合は ,

kernel_obj.dat に生成する .

-z, --nonzero
__EMPTY_LABEL マクロの使用を抑止する .

-ao=xxx
ID の割付順序を指定する . xxx に指定できる内容は次の通り .

alphabetic	名称の昇順 (A に近いものほど小さな値)
fcfs	定義順の昇順 (先に宣言したものほど小さな値)
alphabetic,reverse	名称の降順
fcfs,reverse	定義順の降順

-id=<ファイル名>
自動 ID 割付け結果ヘッダファイル (kernel_id.h) の名称を変更する .

-cfg=<ファイル名>
カーネル構成ファイル(kernel_cfg.c)の名称を変更する .

-oproto
ハンドラやタスク本体などの関数のプロトタイプ宣言をカーネル構成
ファイル (kernel_cfg.c) に出力する .

-iI
カーネル関連のヘッダをインクルードする際に , "ファイル名" ではなく
<ファイル名> を使用する .

-1.3
TOPPERS/JSP カーネル Release 1.3 互換の形式で生成する .
(注意: -1.3 で生成したファイルは 1.4 以降では使用できない)

-iapi
カーネルコンフィギュレータが処理できない静的 API を無視する .

-t
カーネルコンフィギュレータが処理できない静的 API を標準出力に出力
する .

-ext
標準外の拡張機能の使用を許可する .

chk プログラムに対するオプションは次の通り .

-m, --module <モジュール記述>
チェックするロードモジュールを指定する . 標準の chk プログラムは ,
ロードモジュールのシンボルフファイル (GNU BINUTILS の nm が出力する
形式) とモトローラ S レコードファイルを読み込んでパラメータチェッ
クを行なう . この場合 , <モジュール記述>には , シンボルフファイルと
S レコードファイルの 2 つを " , " で区切って指定する .

-cs, --script <ファイル名>
<ファイル名>で指定されたチェックファイルを用いてチェックする .
チェックファイルとは , cfg プログラムが生成した kernel_chk.c を ,
コンパイラおよび utils/gencheck により加工したファイルのこと
である . このオプションを省略した場合 , いくつかのチェックが行われ
なくなる . オプションを指定した場合には , <ファイル名>は省略でき
ない .

-obj, --load-object <ファイル名>
静的 API の解析内容を含むオブジェクト定義ファイルを , <ファイル名>

で指定されたファイルから読み込む。<ファイル名>を省略した場合は、kernel_obj.dat から読み込む。

-cl <エラーレベル>

エラー検出レベルを変更する。レベルは LAZY (重大なエラーのみ検出)、STANDARD (ITRON 仕様の範囲のみで検出)、TOPPERS (JSP カーネルの制限違反まで検出)、RESTRICTED (すべてのエラーを検出) の 4 種類のうちから選択する (デフォルトは RESTRICTED)。

7.10 リンカスクリプトとメモリ領域

JSP カーネルのリンク方法は、ターゲット依存のリンカスクリプト (*.ld) に記述されている。サンプルプログラムの Makefile では、ターゲット依存の定義を入れた Makefile.config の中で LDSCRIPT を定義すると、定義した名前のファイルをリンカスクリプトに用いる。

JSP カーネル動作時には、以下のメモリ領域が必要になる。

(a) コード領域

カーネルおよびアプリケーションのプログラムおよび定数データが置かれる領域。ROM 上に置くことも可能である。先頭アドレスを、カーネルをリンクする際の -Ttext オプションで指定する。サンプルプログラムの Makefile では、ターゲット依存の定義を入れた Makefile.config の中で TEXT_START_ADDRESS を定義すると、リンク時に -Ttext オプションが付加される。

(b) データ領域

カーネルおよびアプリケーションの使用するデータ領域。固定的なデータ領域と、sbrk 関数によって取られるヒープ領域からなる。カーネルはヒープ領域を使用しない。先頭アドレスを、カーネルをリンクする際の -Tdata オプションで指定する。サンプルプログラムの Makefile では、ターゲット依存の定義を入れた Makefile.config の中で DATA_START_ADDRESS を定義すると、リンク時に -Tdata オプションが付加される。

(c) 非タスクコンテキスト用のスタック領域

割込みハンドラなどの非タスクコンテキストが使用するスタック領域。領域の設定方法はターゲット依存であるが、通常は、ターゲットシステム依存のインクルードファイル (sys_config.h) でスタックの初期値を定義し、ターゲットプロセッサ依存のスタートアップモジュール (start.S) 中で初期化される。

7.11 他のターゲットへのポーティング

JSP カーネルを他のターゲットへポーティングするために必要な作業は、カーネル自身のポーティング、システムサービスのポーティング、開発環境の構築と標準の開発環境との差異の吸収などからなる。詳しくは、「JSP カーネル ターゲット依存部 ポーティングガイド」(config.txt) を参照すること。

7.12 カーネルの内部識別子のリネーム

μITRON4.0 仕様は、カーネルの内部識別子を _kernel_ または _KERNEL_ で始めることを要求している。ところが、カーネルのソースコード中で直接このような識別子を用いると、識別子の長さが長くなり、可読性を損なう。そこで JSP カーネルでは、xxxxx というカーネルの内部識別子を _kernel_xxxxx にリネームする仕組みを入れている。

ところが、この仕組みにより、デバッグ作業が非効率になるケースが考えられる。具体的には、ソースコード中の識別子がオブジェクトコード中の識別子と一致しないために、ソースコード中の変数を指定してその値を読んだり、関数を指定してそこにブレークポイントを置くといったことができない。

この問題を解決するために、JSP カーネルでは、ソースコード中の必要な識別子をリネームする

ためのユーティリティ (utils/rename) を用意している。rename ユーティリティに、リネーム定義ファイル (xxx_rename.def) のプリフィックス (xxx の部分) と、リネームしたいファイルリストを与えると、リネーム処理を行なう。例えば、kernel ディレクトリのすべてのファイルに対して、カーネルの内部識別名をリネームするには、次のコマンドを用いればよい。

```
% cd kernel
% ../utils/rename jsp *
```

8. その他

8.1 ウェブサイト

TOPPERS プロジェクトおよび JSP カーネルのためのウェブサイトを、以下の URL に用意している。

<http://www.toppers.jp/>

配付キットの最新版は、このウェブサイトからダウンロードすることができる。
また、後述のメーリングリストのアーカイブなども、このウェブサイトで閲覧することができる。

8.2 利用条件・著作権

JSP カーネルの利用条件は、各ファイルの先頭に明示されている（このドキュメントの先頭にもついている）。著作権は、各ファイルの先頭に表示されている著作権者が保有している。

利用条件の (3) の (b) において、利用の形態を TOPPERS プロジェクトに報告する方法としては、JSP カーネルを利用した製品の種別と名称、利用した JSP カーネルの構成（どのターゲット依存部を用いたか）を、次のアドレスへメールで連絡する方法を標準とする。JSP カーネルを製品以外に利用した場合には、それに準じる情報を報告するものとする。

report@toppers.jp

またその際に、JSP カーネルを使用しているコメントやご意見もいただけると幸いです。この方法での報告が難しい場合には、別途相談されたい。

8.3 保証・サポート・適用性

JSP カーネルは無保証で提供されているものである。開発者は、その適用可能性も含めて、いかなる保証も行わない。また、サポートの約束もしていない。質問がある場合は、後述のメーリングリストを利用していただけると幸いです。

8.4 メーリングリスト

JSP カーネルのユーザに対する情報提供およびユーザ相互間の情報交換を容易にするために、TOPPERS ユーザメーリングリストを用意している。このメーリングリストには、誰でも自由にメールを送付することができる。また、送付されたメールは、誰でも自由にウェブサイトで読むことができる。JSP カーネルにバグや問題点を発見した場合には、このメーリングリストに報告して欲しい。

メーリングリストへのメールの送付先は次の通り。

users@toppers.jp

メーリングリストにバグや問題点などを報告する場合には、必要に応じて、次の情報を知らせて欲しい。

- ターゲットに関する情報
- ・ターゲットプロセッサの種類
 - ・ターゲットボードの種類

ホストに関する情報

- ・OS のバージョン (サービスパックの適用状況も)
- ・コンパイラなどの開発環境のバージョン (Cygwin のバージョンも)

このメーリングリストへの登録を希望する場合は、まず、
users-ctl@toppers.jp 宛てに、本文に

```
subscribe あなたの名前  
例: subscribe Hiroaki Takada
```

と書いたメールを送付する (上記のコマンド中には半角英文字のみを使うこと) 。
折り返し、登録確認のためのメールが送られてくるので、その指示に従って登録する。

8.5 TOPPERS プロジェクトへの参加

TOPPERS プロジェクトでは、何からの形でプロジェクトに貢献されたい方、プロジェクトで開発したソフトウェアをお使いの方、プロジェクトに興味をお持ちの方の参加を求めている。TOPPERS プロジェクトへの参加方法については、TOPPERS プロジェクトのウェブサイトを参照すること。

8.6 ITRON Club

仕事・勉強・趣味を問わず ITRON 仕様および ITRON プロジェクトに興味を持つ人のための情報交換・議論の場として、ITRON Club メーリングリストを用意している。このメーリングリストには、ITRON に興味を持っている人なら誰でも参加できる。また、このメーリングリストに送付されたメールは、誰でも自由にウェブサイトで読むことができる。

このメーリングリストへの登録を希望する場合は、まず、
itron-club-ctl@ertl.jp 宛てに、本文に

```
subscribe あなたの名前  
例: subscribe Hiroaki Takada
```

と書いたメールを送付する (上記のコマンド中には半角英文字のみを使うこと) 。
折り返し、登録確認のためのメールが送られてくるので、その指示に従って登録する。

9. リファレンス

9.1 サービスコール一覧

(1) タスク管理機能

```
ER ercd = act_tsk(ID tskid);  
ER ercd = iact_tsk(ID tskid);  
ER_UINT actcnt = can_act(ID tskid);  
void ext_tsk();  
ER ercd = ter_tsk(ID tskid);  
ER ercd = chg_pri(ID tskid, PRI tskpri);  
ER ercd = get_pri(ID tskid, PRI *p_tskpri);
```

(2) タスク付属同期機能

```
ER ercd = slp_tsk();  
ER ercd = tslp_tsk(TMO tmout);  
ER ercd = wup_tsk(ID tskid);  
ER ercd = iwup_tsk(ID tskid);  
ER_UINT wupcnt = can_wup(ID tskid);  
ER ercd = rel_wai(ID tskid);
```

```
ER ercd = irel_wai(ID tskid);
ER ercd = sus_tsk(ID tskid);
ER ercd = rsm_tsk(ID tskid);
ER ercd = frsm_tsk(ID tskid);
ER ercd = dly_tsk(RELTIM dlytim);
```

(3) タスク例外処理機能

```
ER ercd = ras_tex(ID tskid, TEXPTN rasptn);
ER ercd = iras_tex(ID tskid, TEXPTN rasptn);
ER ercd = dis_tex();
ER ercd = ena_tex();
BOOL state = sns_tex();
```

(4) 同期・通信機能

```
ER ercd = sig_sem(ID semid);
ER ercd = isig_sem(ID semid);
ER ercd = wai_sem(ID semid);
ER ercd = pol_sem(ID semid);
ER ercd = twai_sem(ID semid, TMO tmout);

ER ercd = set_flg(ID flgid, FLGPTN setptn);
ER ercd = iset_flg(ID flgid, FLGPTN setptn);
ER ercd = clr_flg(ID flgid, FLGPTN clrptn);
ER ercd = wai_flg(ID flgid, FLGPTN waiptn,
MODE wfmode, FLGPTN *p_flgptn);
ER ercd = pol_flg(ID flgid, FLGPTN waiptn,
MODE wfmode, FLGPTN *p_flgptn);
ER ercd = twai_flg(ID flgid, FLGPTN waiptn,
MODE wfmode, FLGPTN *p_flgptn, TMO tmout);

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

ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg);
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);
```

(5) メモリプール管理機能

```
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);
ER ercd = rel_mpf(ID mpfid, VP blk);
```

(6) 時間管理機能

```
ER ercd = set_tim(const SYSTIM *p_sysstim);
ER ercd = get_tim(SYSTIM *p_sysstim);
ER ercd = isig_tim();
```

```
ER ercd = sta_cyc(ID cycid);
ER ercd = stp_cyc(ID cycid);
```

(7) システム状態管理機能

```
ER ercd = rot_rdq(PRI tskpri);
ER ercd = irot_rdq(PRI tskpri);
ER ercd = get_tid(ID *p_tskid);
ER ercd = iget_tid(ID *p_tskid);
ER ercd = loc_cpu();
ER ercd = iloc_cpu();
ER ercd = unl_cpu();
ER ercd = iunl_cpu();
ER ercd = dis_dsp();
ER ercd = ena_dsp();
BOOL state = sns_ctx();
BOOL state = sns_loc();
BOOL state = sns_dsp();
BOOL state = sns_dpn();
BOOL state = vsns_ini();
```

(8) 割り込み管理機能

```
ER ercd = dis_int(INTNO intno);
ER ercd = ena_int(INTNO intno);
ER ercd = chg_ixx(IXXXX ixxxx);
ER ercd = get_ixx(IXXXX *p_ixxxx);
xx, xxx, XXXX はターゲット毎に定められる。
```

(9) CPU 例外発生時のシステム状態参照

```
BOOL state = vxsns_ctx(VP p_excinf);
BOOL state = vxsns_loc(VP p_excinf);
BOOL state = vxsns_dsp(VP p_excinf);
BOOL state = vxsns_dpn(VP p_excinf);
BOOL state = vxsns_tex(VP p_excinf);
```

(10) 性能評価用システム時刻参照機能

```
ER ercd = vxget_tim(SYSUTIM *p_sysutim);
```

9.2 静的 API 一覧

```
CRE_TSK(tskid, { ATR tskatr, VP_INT exinf, FP task,
                PRI itskpri, SIZE stksz, VP stk });
DEF_TEX(ID tskid, { ATR texatr, FP texrtn });
CRE_SEM(ID semid, { ATR sematr, UINT isemcnt, UINT maxsem });
CRE_FLG(ID flgid, { ATR flgatr, FLGPTN iflgptn });
CRE_DTQ(ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtq });
CRE_MBX(ID mbxid, { ATR mbxatr, PRI maxmpri, VP mprihd });
CRE_MPF (ID mpfid, { ATR mpfatr, UINT blkcnt, UINT blkksz, VP mpf } );
CRE_CYC (ID cycid, { ATR cycatr, VP_INT exinf, FP cychdr,
                  RELTIM cyctim, RELTIM cycphs } );
DEF_INH(INHNO inhno, { ATR inhatr, FP inthdr });
DEF_EXC(EXCNO excno, { ATR excatr, FP exchdr });
ATT_INI({ ATR iniatr, VP_INT exinf, FP inirtn });
VATT_TER({ ATR teratr, VP_INT exinf, FP terrtn });
```

9.3 メインエラーコード一覧 (JSP カーネルが返すもののみ)

E_PAR	-17	パラメータエラー
E_ID	-18	不正 ID 番号
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_OBJ	-41	オブジェクト状態エラー
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

9.4 バージョン履歴

2000年11月15日	Release 1.0	最初のリリース
2000年11月24日	Release 1.0 (PL=1)	問題点の修正
2001年2月24日	Release 1.1	V850 の追加など
2001年5月9日	Release 1.1 (PL=1)	SH1 の追加など
2001年11月15日	Release 1.2	SH4, H8, ARM7TDMI の追加など
2002年4月15日	Release 1.3	M32R, MicroBlaze, TMS320C54x, i386, H8S の追加など
2003年12月25日	Release 1.4	多数の改良
2004年10月14日	Release 1.4 (PL=1)	SH2, M16C, SC33, PowerPC32, Nios2 の追加など