

TOPPERS初級実装セミナー

(JSP-1.4|OAKS16 mini|カップラーメンタイマー版)

2日目

TOPPERSプロジェクト
教育ワーキング・グループ

本ドキュメントに関して

■ 本ドキュメントの利用に関して以下の点にご留意ください

1. 著作権に関しての表記

<TOPPERS 初級実装セミナー (JSP1.4/OAKS16-MINI/タイマー版)2日目>

Copyright (C) 2004 by 竹内良輔 (株)リコー プラットフォーム開発センター

Copyright (C) 2004 by 山本雅基 (株)デンソークリエイト

上記著作権者は、以下の (1)～(3) の条件を満たす場合に限り、本資料 (本資料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) 本資料を利用する場合には、上記の著作権表示およびこの利用条件が、そのままの形で資料中に含まれていること。
 - (2) 本資料を改変する場合には、資料を改変した旨の記述を、資料中に含めること。ただし、TOPPERSプロジェクトの活動の一環として本資料を改変する場合には、資料を改変した旨の記述を含める必要はない。
 - (3) 本資料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。
-

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。

3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

本ドキュメントでは、Microsoft社のClip Art Gallery コンテンツを使用しています。

TRONは"The Realtime Operating system Nucleus"の略称です。ITRONは"Industrial TRON"の略称です。
μITRONは"Micro Industrial TRON"の略称です。TOPPERS/JSPはToyohashi Open Platform for Embedded Real-Time System/Just Standard Profile Kernelの略称です。」

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

スケジュール

■ 1日目

1. 開発環境の確認 0. 5時間
2. 組込みとは、SESSAMEの復習
1. 5時間
3. SESSAME版教材の動作確認
1時間
4. リアルタイムOSとは何か
1. 5時間
RTOSの状態遷移、HW I/F
教材を用いた確認
5. カップラーメンタイマーの設計
設計を行う 1. 5時間

■ 2日目

1. カップラーメンタイマーの実装
実装を行う 1. 5時間
2. レビュー 0. 5時間
設計、実装の評価
3. 同期と通信 1時間
タスク間通信について
同期、排他制御について
4. イベントフラグを用いた改造
1. 5時間
5. まとめ、宿題 1時間

タイマーの実装

1. カップラーメンタイマーの実装
2. レビュー
3. 同期と通信
4. イベントフラグを用いた改造
5. まとめ、宿題

カップラーメンタイマーのTOPPERS/JSP版を作成してみましょう

この開発環境では、デバッガが使用できませんので、タスクモニターやシステムログのダンプ機能を使ってデバックしましょう

TOPPERS/JSP実装の為の基礎知識 1

システムの初期化手順と初期化プログラム

- カーネルを起動するには、ターゲットに依存した最低限の初期化を行った後、`kernel_start`を呼び出す
- これらの処理はCPUロック状態で行う
- 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、この処理を行う
- 詳細はdocの下のマニュアルを参照

`config`ディレクトリの下に、ターゲットに依存したソースがあります。

この下のディレクトリにはネーミング規則があります。

<プロセッサ名>-<開発環境名>

<開発環境名>が省略された場合は、GNU-GCCでの開発環境となります。

その下のディレクトリが<システム名>を示します。システム名はターゲット・ボード名を使用することが多いようです。M16Cの環境は以下のディレクトリ構成となっています。

`config/m16c-renesas/oaks16`

`/oaks16_mini`

M16C-CPUはGNUの環境はなく、ルネサステクノロジのツールとなります。ボードはOAKS16とOAKS16 MINIの対応を行っています。OBJの下の開発環境に両方のTMファイルを用意しています。

OBJ/Jsp14Timer2

OAKS16用

OBJ/Jsp14Timer2m

OAKS16 MINI用

通常、パワーオンの初期化を行うプログラムは<CPU名>-<開発環境名>の下に`start.S` (M16Cの場合は`start.a30`)というアセンブラ言語のプログラムにより行われます。このプログラムの最後に`_kernel_start`の呼び出しが設定されています。

`kernel_start()`は`kernel`ディレクトリ中の`startup.c`というC言語のプログラムに記述されています。`startup.c`はTOPPERS/JSPカーネルの初期化と終了処理を行うプログラムです。

TOPPERS/JSP実装の為の基礎知識 2

コンフィギュレータの確認

- RTOSでは、カーネルのオブジェクトで使用するメモリ・リソースを動的に確保するものと静的に確保するものがある
- TOPPERS/JSPでは、コンパイル時にメモリ・リソースを決定する静的に確保を行う
- コンフィギュレータはカーネルのオブジェクトの静的な生成を自動的に行うプログラムです

RTOSには、カーネルのオブジェクトのメモリ確保を電源がオンされた後で、動的に確保するタイプのものと、コンパイルとリンク時に決定するものがある。実は、ほとんどのRTOSは動的な確保を行う。TOPPERSのフルセットカーネルも動的な確保を行うカーネルである。静的に確保を行う方が無駄な管理用の領域を必要としない為に、トータルのメモリサイズが小さい。TOPPERS/JSPは静的な確保をコンフィギュレータという自動割当プログラムを用いて行う

コンフィギュレータはコンフィグレーションファイル(.cfg)からメモリ領域を定義したソースファイルとインクルードファイルを生成する。(カッパーメンタイマーの内容は「イベントフラグを用いた改造のイベントフラグによるタスク間通信2に記載されています」)

コンフィギュレータはcfgディレクトリの下でcfgコマンドにより実行されます。このプログラムはCPUによる依存性はありませんので、どこCPUでも共通に使用されます。ビルドの最初でcfgコマンドの呼び出しを行っていますので、ビルドのログにて確認ができます。

TOPPERS/JSPを用いたタイマーの実装1

メインがなくても、250ms実行可能

- ノンタスクの実装では、main()が強制的に250msごとに、2つのプロセスを実行していた
- タスクでは、tslp_tskサービス・コールで自分の中に250msのプロセス実行を作ることができる
- entry_taskとtimer_taskのforループ文の中は、外から呼び出されなくても250msごとに実行している

リアルタイムOS上では、2つのプロセスはタスクとして実行します。ノンOSの場合と異なり、2つのプロセスはメインルーチンによってコールされて実行するのではなく、各々が別々のプロセッサ上動作するように実行します。これはentry_taskとtimer_taskが、それぞれがメインルーチンのように動作するということです。

switch_processの中味のプログラムをentry_taskのforループ文の中に、timer_processの中味のプログラムをtimer_taskのforループ文の中に入れ込めばメインルーチンがなくても実行します。

TOPPERS/JSPを用いたタイマーの実装2

2つのタスク間の通信は？

- ENTRY_TASKからTIMER_TASKに、タイマーの状態遷移要求の伝達が必要です
- どのような情報を伝えるのでしょうか？
 1. タイマー起動通知
 2. タイマーの時間延長通知
 3. タイマーの停止通知
- 伝達情報により、TIMER_TASKは3つの状態遷移が発生する

タイマー起動状態、タイムアウト状態、タイマー停止状態

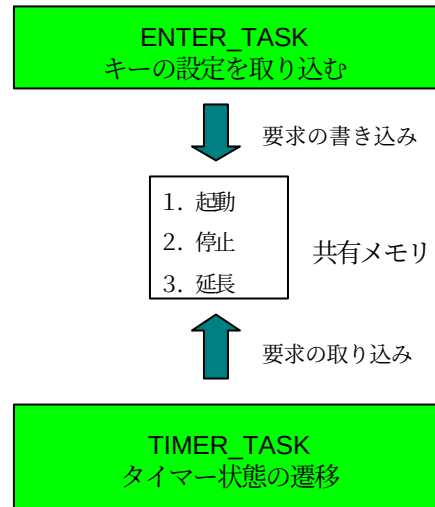
複数のタスクでデータを共有するメモリ領域を共有メモリと呼びます。どちらかのタスクがこの領域にデータを書き込み、もう一方のタスクがデータを読み取るようにすれば情報伝達ができます。

TIMER_TASKはデータを読み取ったデータをイベントとし、タイマーの状態状態を遷移するように、プログラムしてください。

TOPPERS/JSPを用いたタイマーの実装3

共有メモリの利用

- 共有メモリを介して情報のやりとりを行う
- 今回は片方が書き込み、片方が読み込み



TOPPERS/JSPを用いたタイマーの実装4

共有メモリーを用いて、データ通信

- 3つの状態を、共有メモリ BYTE型のビット領域)を用いてイベント通信する
- ENTRY_TASKは通信情報の書き込み、TIMER_TASKは通信情報の読み出し。
- タスクモニターを用いれば、2つのタスクを別々のデバックできます
- 共有メモリは、モニターのメモリ表示機能で参照ができます

TOPPERS/JSPを用いたタイマーの実装5

TOPPERSのシステムログを使ってみよう

- timer2.cのソース中のようにシステムログ関数にて、コンソールにログ表示ができます
- syslog_1(LOG_NOTICE, “Sample entry task starts (exinf=%d).”, exinf); ENTRY_TASKの起動時にexinfの値をログ表示します
- ログ情報には重要度の設定があり、デフォルトではLOG_INFOは表示レベルではありません。vmsk_log関数、または、タスクモニターの設定で表示レベルの切り替えが可能です

重要度はinclude/syslog.hを参照してください

syslog_n関数の第1引数で、ログ情報の重要度の設定を行います。この値とlogmask値の比較によって、表示を行うかどうかの判定を行って表示しますので、デバッグ後、不要なログの表示、非表示を切り替えることが可能です。TOPPERS/JSP1.4の設定では以下の関数の設定で切り替えが可能です。

ER syslog_setmask(UINT logmask, UINT lowmask);

また、モニターのLog MODE コマンドでも、変更が可能です。

/* ログ情報の重要度の定義(syslog.h) */

```
#define LOG_EMERG      0          /* シャットダウンに値するエラー */
#define LOG_ALERT      1
#define LOG_CRIT       2
#define LOG_ERR         3          /* システムエラー */
#define LOG_WARNING    4          /* 警告メッセージ */
#define LOG_NOTICE     5
#define LOG_INFO       6
#define LOG_DEBUG      7          /* デバッグ用メッセージ */
```

レビュー

1. カップラーメンタイマーの実装
2. レビュー
3. 同期と通信
4. イベントフラグを用いた改造
5. まとめ、宿題

できあがったプログラムをレビューして見ましょう
バランスのとれた実装を行うには、いくつかのテクニックがあります。

まず、モジュール分割について

- モジュールサイズの最適サイズ

1ページの半分程度（30行位）

実装が単純となるように

- システムの明解さ

均等型システム（左右均等、上下は論理－物理）

モジュールの関係だけでシステムが理解できる

- 重複の極小化

同じ機能を複数のモジュールに持たせない

- 情報隠蔽

モジュールで取り扱うデータを隠蔽する

レビュー

作成したプログラムに自信のある方は発表を

- プログラムには、正解はありません
- 設計品質自体は吟味ができます

モジュール分割

サイズの最適化、システムの明解さ
重複の極小化、情報隠蔽

モジュールの凝集度

凝集度は高いほど保守性が良い

モジュールの結合度

結合度は弱いほど保守性が良い

・モジュールの凝集度（コヒージョン）

凝集度は高い順に並べました。

- ① 機能的 :単一の目的を持った機能、情報、責務。
- ② 逐次的 :同一のデータに対する、関連が強く、必然的な順番を有す機能集合
- ③ 通信的 :同一のデータに対する、複数目的の機能集合
- ④ 手順的 :関連の薄い複数データに対する、順次機能
- ⑤ 一時的 :同一時期に発生する、互いに関係のない機能集合
- ⑥ 論理的 :外部から利用する時に便利な機能の寄せ集め
- ⑦ 偶発的 :まったく相関のない機能の寄せ集め

・モジュールの結合度（カップリング）

結合度の弱い順に並べました。

- ① データ結合 :構造を持たない単一データ
- ② 構造体結合 :同一の目的を持つデータメンバの集合
- ③ バンドリング結合 :複数のフィールドから構成される構造体
- ④ 制御結合 :相手のモジュールを制御するフラグ
- ⑤ ハイブリッド結合 :値の範囲により意味の異なるデータやフラグ
- ⑥ 共有結合 :グローバルデータ
- ⑦ 内容結合 :相手のモジュールの内部を参照 (アセンブラ)

レビューとテスト

その留意点

- レビュー、デバッグ、テストは、どの順番で、どの作業から行うのが良いかは、開発目標、開発環境、開発者の技能によります
- レビューの為にプログラムを作ったり、デバッグ、テストの為にプログラムをすることもありますが
- レビュー前にコード構成用のツールで整形したり、文法チェッカや-Wallで警告を表示させて内容を吟味し、より分かりやすいコードにしましょう
- 開発の目的を達成することが重要で、技術を駆使するのは、そのための手段です

プログラムの静的、動的ツールとしては高価のものがいろいろ発売されています。
簡単に使えるものとしては、C、C++、Java言語用のコード整形ツールとしてUNIXコマンドの
astyle、C言語用の文法チェッカとして、UNIXコマンドのsplintなどがあります。

同期と通信

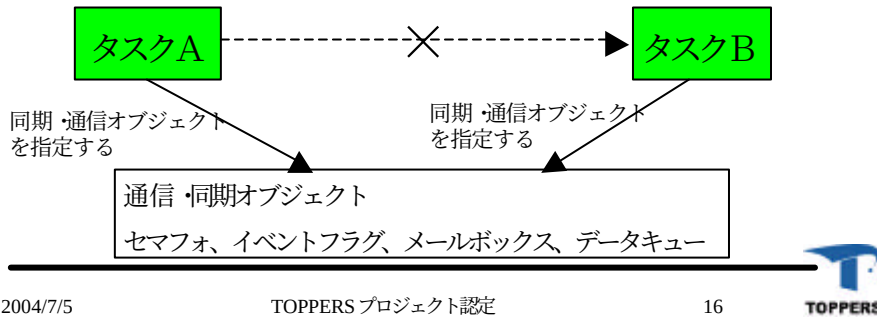
1. カップラーメンタイマーの実装
2. レビュー
3. 同期と通信
4. イベントフラグを用いた改造
5. まとめ、宿題

外部のオブジェクトとして供給されるタスク間の通信や同期について勉強しましょう。基本的な同期と通信機能として、セマフォ、イベントフラグ、メールボックス、データキューがあります。

タスク間の通信・同期の概念

タスクは仮想化されたプロセッサ

- タスク間の通信・同期はプロセッサ間の通信・同期を仮想化したもの
- 相手タスクを指定せず、通信・同期オブジェクトを介して同期通信を行う

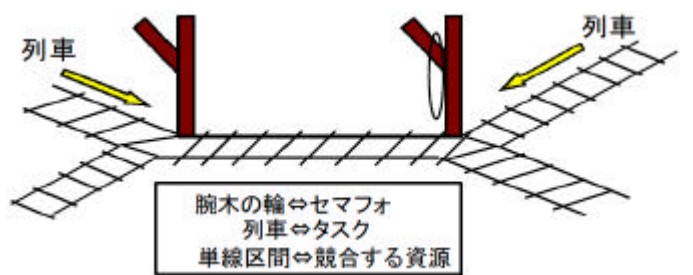


セマフォ 1(Semaphore)

OSが提供する排他制御のしくみ

■ 語源

昔、鉄道の単線区間で進入制御に用いていた「**腕木**」、腕木にかけてある「**輪**」をとった列車の**み**が単線区間に進入できる手順



セマフォは、使用されていない資源の有無や数量をカウンタで表現することにより、その資源を使用する際の排他制御や同期を行うオブジェクトです。資源を解放する場合、セマフォのカウンタを+1し、資源を取得する場合、セマフォのカウンタを-1します。

セマフォはこのカウンタと資源の獲得待ちのタスクの待ち行列で構成されています。資源の獲得を繰り返すカウンタがゼロになると、使用されていない資源が無い為、資源の取得を要求したタスクはセマフォ資源の待ち行列につながれ、待ち状態に入ります。

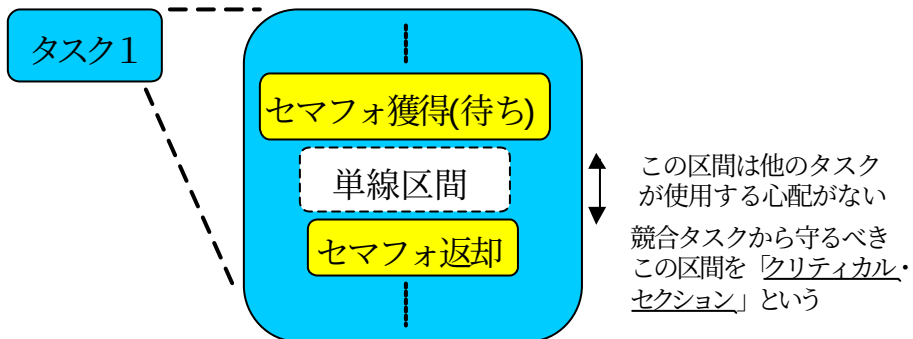
セマフォには資源の初期値と、資源に対して資源が返却され過ぎるのを防ぐ為に最大資源数が設定できます。最大資源数を越えて資源の返却を行おうとした場合、エラーが発生します。

セマフォ IDは個々のセマフォを識別するための IDです。

セマフォ 2(Semaphore)

使い方

- 1つの資源と1つのセマフォを対応つける



・必要な区間のみを効率良く保護できる

・ソースコードの保守が向上

μITRON4.0仕様のセマフォのサービス・コールは以下の通りです。

1. セマフォの生成

```
CRE_SEM(ID semid, {ATR sematr, UINT isemcnt, UINT maxsem});
```

ID semid セマフォ ID 番号

ATR sematr セマフォ属性(TA_FIFO || TA_TPRI)

UINT isemcnt セマフォ資源数の初期値

UNIT maxsem セマフォの最大資源数

2. セマフォ資源の返却

```
ER ercd = sig_sem(ID semid);    ER ercd = iseg_sem(ID semid);
```

ID semid 資源返却対象のセマフォ番号

3. セマフォ資源の取得

```
ER ercd = wai_sem(ID semid);    ER ercd = pol_sem(ID semid);
```

```
ER ercd = twai_sem(ID semid, TMO tmout);
```

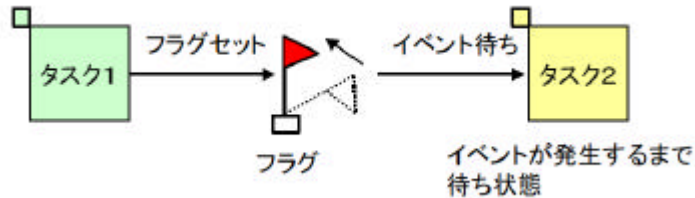
ID semid 資源獲得対象のセマフォ番号

TMO tmout タイムアウト指定 (twai_semのみ)

イベントフラグ 1(Event Flag)

しくみは？

- イベントの発生を他のタスクに通知する
イベントとフラグを対応つける



- 複数ビットのフラグに対して、ANDやORの待ち条件を設定することも可能

イベントフラグは、イベントの有無をビットごとのフラグで表すことにより、タスク間の同期を行うためのオブジェクトです。

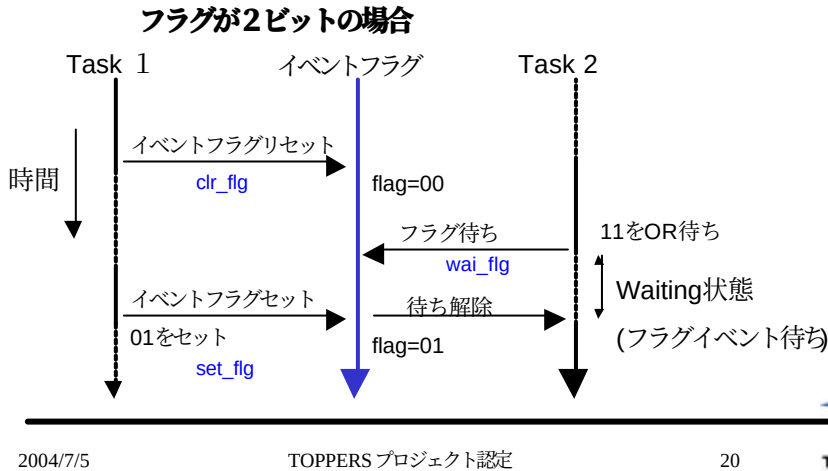
イベントフラグには、対応するイベントの有無をビット毎に表現するビットパターンと、そのイベントフラグの設定を待つタスクの待ち行列で構成されています。イベントを通知する側は、イベントフラグのビットパターンを指定をしたビットをセットまたはクリアすることが可能です。一方、イベントを待つ側は、イベントフラグのビットパターンの指定したビットのすべてまたは一部がセットされるまで、タスクをイベント待ち状態にすることができます。イベント待ち状態を要求したタスクは、要求したイベントの待ち行列につながれます。

イベントフラグ機能には、イベントフラグを生成、削除する機能、イベントフラグをセット、リセットする機能、イベントフラグを設定を待つ機能、イベントフラグの状態を参照する機能で構成されています。

イベントフラグ2(Event Flag)

タスクの実行状態

- イベントの有無をフラグのビットパターンで通知する



μITRON4.0のイベントフラグ機能のサービス・コールは以下の通りです。

1. イベントフラグの生成

```
CRE_FLG(ID flgid, {ATR flgatr, FLGPTN iflgptn});
```

ID flgid イベントフラグの ID 番号

ATR flgatr イベントフラグ属性((TA_TFIFO||TA_PRI) |
(TA_WSGL||TA_WMUL)| [TA_CLR])

FLGPTN iflgptn イベントフラグのビットパターンの初期値

2. イベントフラグのセット

```
ER ercd = set_flg(ID flgid, FLGPTN setptn);
```

```
ER ercd = iset_flg(ID flgid, FLGPTN setptn);
```

3. イベントフラグのクリア

```
ER ercd = clr_flg(ID flgid, FLGPTN clrpntn);
```

4. イベントフラグ待ち

```
ER ercd = wai_flg(ID flgid, FLGPTN waipntn, MODE wfmode, FLGPTN *p_flgptn);
```

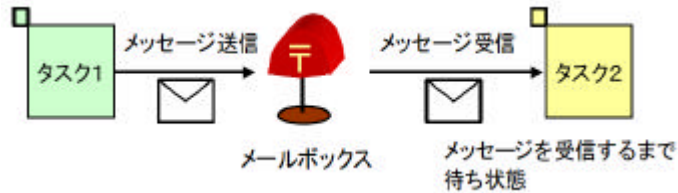
```
ER ercd = pol_flg(ID flgid, FLGPTN waipntn, MODE wfmode, FLGPTN *p_flgptn);
```

```
ER ercd = twai_flg(ID flgid, FLGPTN waipntn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);
```

メールボックス1(Mail Box)

しくみは？

- メッセージを送受信する(同期と通信を同時に行う)



- メールのデータ構造

メッセージ
1通分の
データ



OSが使用する
ヘッダー部
ユーザーのデータ領域
(OSは感知しない)

アプリケーション毎にメールのデータ構造を定義して使用する

```
struct MAIL_tag {  
    T_MSG header;  
    int data;  
}
```

メールボックスは、共有メモリ上に置かれたメッセージを受け渡しすることにより、同期を通信を行うためのオブジェクトです。メールボックスは、送信されたメッセージを入れるためのメッセージキューと、メッセージの受信を待つタスクの待ち行列で構成されます。メッセージを送信するタスクは、送信したいメッセージをメッセージキューにいれます。一方、メッセージを受信するタスクは、メッセージキューに入っているメッセージを1つ取り出します。メッセージキューにメッセージが入っていない場合は、次にメッセージが送られてくるまでメールボックスからの受信待ち状態となります。待ち状態になったタスクはそのメールボックスの待ち行列につながれます。

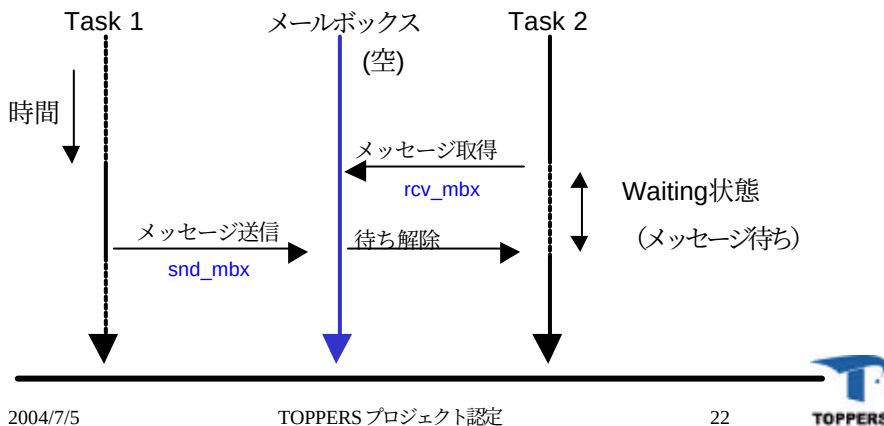
メールボックスの機能は、メールボックスの生成、削除の機能、メールボックスに対してメッセージを送信、受信する機能、メールボックスの状態を参照する機能があります。

メールボックス2(Mail Box)

タスク状態遷移

- メッセージを渡すことにより、同期と通信を同時に行う
送るのはメッセージの先頭アドレスのみ

受信タスクが先に待っている場合



2004/7/5

TOPPERS プロジェクト認定

22



μITRON4.0のメールボックス機能のサービス・コールは以下の通りです。

1. メールボックスの生成

```
CRE_MBX(ID mbxid, {ATR mbxatr, PRI maxpri, VP mprihd});
```

ID mbxid メールボックスの ID 番号

ATR mbxatr メールボックス属性

((TA_TFIFO||TA_TPRI)||(TA_MFIFO||TA_MPRI))

PRI maxpri 送信されるメッセージの優先度の最大数

VP mprihd 優先度別のメッセージキューヘッダー領域の先頭番地

2. メールボックスへの送信

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

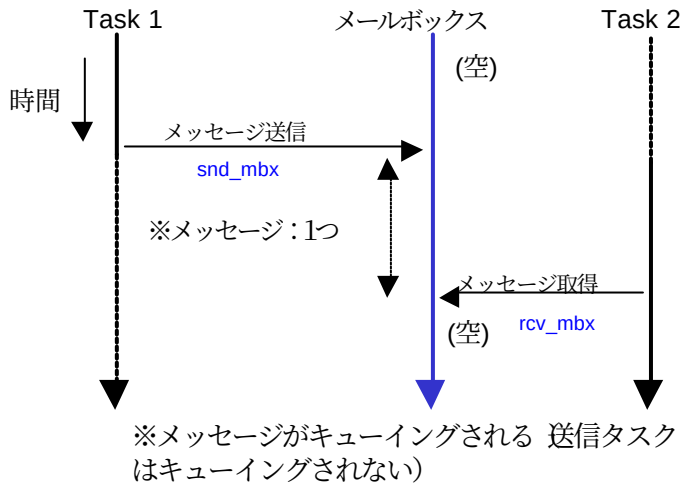
ID mbxid 送信対象のメールボックスの ID 番号

T_MSG * pk_msg メールボックスへ送信するメッセージパケットの先頭番地

メールボックス3(Mail Box)

タスク遷移状態

メッセージが先に来て待っている場合



3. メールボックスからの受信

```
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);
```

ID mbxid 受信対象のメールボックスの ID 番号

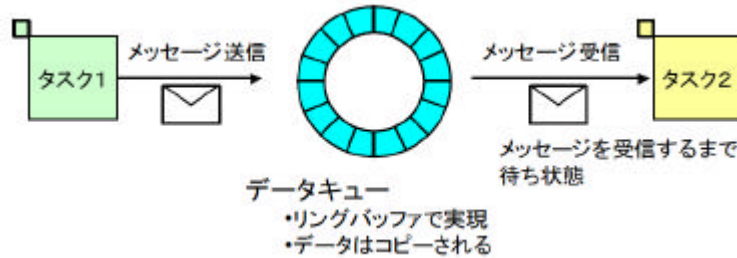
TMO tmout タイムアウト指定(trcv_mbxのみ)

T_MSG ** ppk_msg メールボックスから受信したメッセージの格納領域

データキュー 1(Data Queue)

しくみは？

- 1ワードメッセージ (データ)を送受信する
同期と通信を同時に行う



データキューは1ワードのメッセージを受け渡しすることにより、同期と通信を行う為のオブジェクトです。データキューは、データの送信を持つタスクの待ち行列 (送信待ち行列) とデータの受信を待つタスクの待ち行列 (受信待ち行列)、および、送信されたデータを格納する為のデータキュー領域で構成されています。

データを送信する側は送信したいデータをデータキューに入れます。データキューに空きが無い場合は、データキュー領域に空きができるまでデータキューへの送信待ち状態になります。データキューへの送信待ちになったタスクは、そのデータキューの送信待ち行列につながれます。一方、データを受信するタスクは、データキューに入っているデータを1つ取り出します。データキューにデータが入っていない場合は、次にデータが送られてくるまでデータキューからの受信待ち状態となります。データキューからの受信待ち状態となったタスクは、そのデータキューの受信待ち行列につながれます。

データキュー機能は、データキューを生成、削除する機能、データキューに対してデータを送信、強制送信、受信する機能、データキューの状態を参照する機能があります。

データキュー 2(Data Queue)

メールボックスとデータキューの違い

■ メールボックス

- データ構造にリンクリストを用いる
- メッセージ長は任意
- メッセージ数 (最大数) は任意
送信タスクが待たされない)
- ヘッダーが必要
- メッセージの先頭アドレスのみを送受信し、メッセージ本体はコピーされない
- 受信されるまではデータ領域を保護する必要がある

■ データキュー

- メッセージ長は1ワード固定
- メッセージ数 (最大数) は固定
送信タスクが待たされる場合がある)
- ヘッダーが不要
- メッセージ本体がコピーされる
- 受信されるまで元のデータの領域を保護する必要がない

μITRON4.0のデータキュー機能のサービス・コールは以下の通りです。

1. データキューの生成

```
CRE_DTQ(ID dtqid, {ATR dtqatr, UINT dtqcnt, VP dtq});
```

ID dtqid	データキューの ID 番号
ATR dtqatr	データキュー属性(TA_TFIFO TA_TPRI)
UINT dtqcnt	データキュー領域の容量 (データの個数)
VP dtq	データキュー領域の先頭番地

2. データキューへの送信

```
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);
```

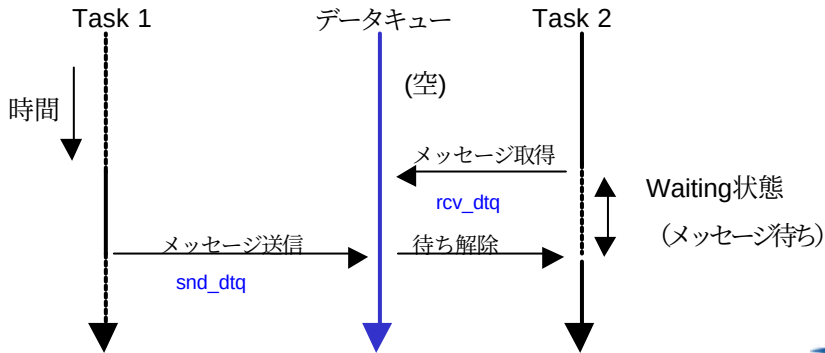
ID dtqid	送信対象のデータキューの ID 番号
VP_INT data	データキューへの送信するデータ
TMO tmout	タイムアウト指定(tsnd_dtqのみ)

データキュー 3(Data Queue)

タスクの状態遷移

- メッセージを渡すことにより、同期と通信を同時に行う
送るのは1ワードメッセージのみ

受信タスクが先に来て待っている場合



2004/7/5

TOPPERS プロジェクト認定

26



3. データキューへの強制通信

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

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

ID dtqid 送信対象のデータキューの ID 番号

VP_INT data データキューへ送信するデータ

4. データキューからの受信

```
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);
```

ID dtqid 受信対象のデータキューの ID 番号

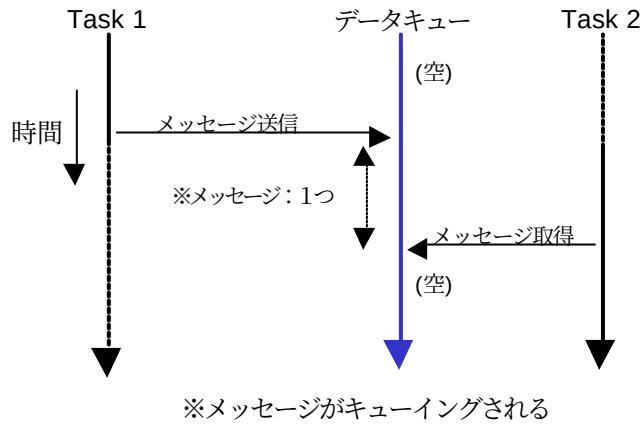
TMO tmout タイムアウト設定(trcv_dtq)

VP_INT *p_data 受信データの格納領域

データキュー 4(Data Queue)

タスクの状態遷移

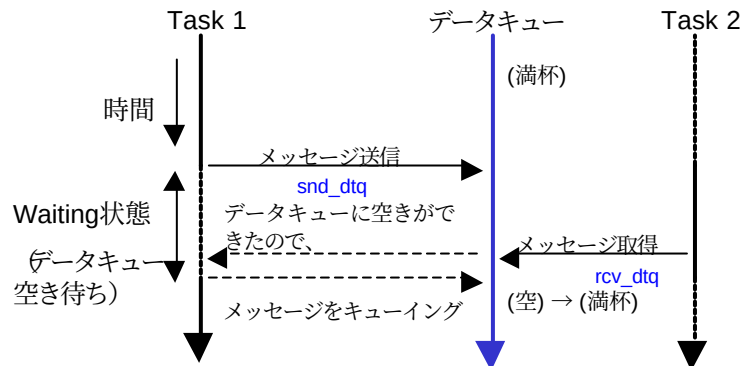
メッセージが先に来て待っている場合



データキュー 5(Data Queue)

タスクの状態遷移

データキューが満杯で空きを待っている場合



満杯時、送信タスクがキューイングされる

μITRON4.0仕様で提供する通信・同期機能

■ どのような通信・同期機能があるか

	セマフォ	イベント	メールボックス	データキュー
機能	排他制御・同期	イベント通知	同期・通信	同期・通信
情報量	少	中	多	多
オーバーヘッド	小	小	中～大	大
キューイングされるオブジェクト	資源待ちタスク	イベント待ちタスク	受信タスク メッセージ	送信タスク 受信タスク メッセージ
サービスコール	CRE_SEM, sig_sem, wai_sem, twai_sem, pol_sem	CRE_FLG, set_flg, clr_flg, wai_flg, pol_flg, twai_flg	CRE_MBX, snd_mbx, rcv_mbx, prcv_mbx, trcv_mbx	CRE_DTQ, snd_dtq, psnd_dtq, tsnd_dtq, fsnd_dtq, rcv_dtq, prcv_dtq, trcv_dtq

システム設計を行う場合、同じシステムに種々の通信・同期機能を混在して使用すると、システムが複雑になってしまう為、特別な事情がある場合を除いて避けた方が良いでしょう。なるだけ、システム自体がシンプルとなるように通信・同期機能を選択することをお勧めします。

イベントフラグを用いた改造

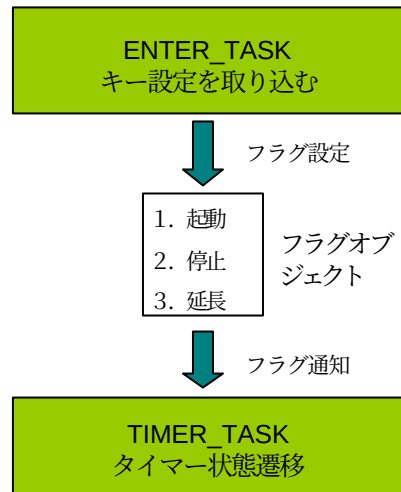
1. カップラーメンタイマーの実装
2. レビュー
3. 同期と通信
4. イベントフラグを用いた改造
5. まとめ 宿題

作成したTOPPERS/JSP版カップラーメン・タイマーをイベントフラグを用いてタスク間通信を行うように改造しよう。イベントフラグを用いることにより、モジュールの凝集度が弱くなる。

イベントフラグによるタスク間通信1

共有メモリで作成した部分を改造しよう

- フラグオブジェクトを用いて、ENTER_TASKからTIMER_TASKへコマンドの通知を行う構成を形成する
- イベントフラグ・サービス・コールを用いてコマンドの伝達を行う



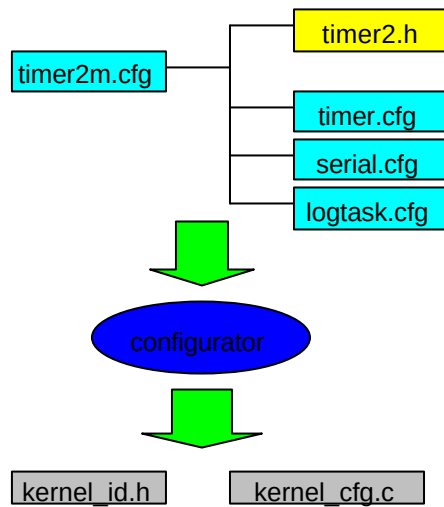
共有メモリで作成した場合と、どのように違うでしょうか？

1. 共有メモリの場合、グローバルデータを2つのタスクで参照、設定しあう為、モジュールの変更が発生した場合、2つのタスク（モジュール）で、この領域の実装を考慮しなければなりません。それに比べ、イベントフラグ等を用いた実装では専用のオブジェクトによる管理となり、インターフェイスの構築手順が明確になります。この例では、大きな差異がないように思われますが、もっとインターフェイスが複雑なシステムでは大きな恩恵があります。

2. イベントフラグを用いると、タスク管理機能としてサポートされる為、イベントの発生を受けてすぐに、受信タスクの起動が可能です。それに比べ、共有メモリのみの設定では、受信側のタスクはイベント発生のためのポーリングを行わなければならない。オーバーヘッドが発生します。

イベントフラグによるタスク間通信2 コンフィギュレーションの設定

- TOPPERS/JSPでは、タスクリソースは、コンフィギュレーションファイルで定義する
- リソースはコンフィギュレータを用いて、kernel_id.hとkernel_cfg.cの2つのソースに変換される
- timer2m.cfgにCRE_FLGを定義する



コンフィギュレーションファイルをエディタを用いて変更しましょう。TM中のmakeの設定によりbuildを行えばコンフィギュレータ、..¥..¥cfg¥cfg.exe)が起動してtimer2m.cfgからkernel_id.hとkernel_cfg.cを生成します。

CRE_FLGについては、イベントフラグ2のノートを参照してください。

このとき、イベントフラグID番号は自分で好きなラベルを設定してください。イベントフラグ属性は、(TA_TFIFO|TA_WSGL)。イベントフラグビットパターンの初期値は0でOKです。TA_TFIFOはFIFO順のキューイングを行う設定で、TA_WSGLはイベントフラグで複数のタスクが待ち状態となることを許さないという意味です。

イベントフラグによるタスク間通信3

送り側タスクの実装

- イベントを送信する場合、`set_flg` サービス・コールを用いて送信を行う
- 3つのイベントはそのまま使用できる

```
event |= EVT_TIMER_START;
```



```
set_flg(イベントフラグID, EVT_TIMER_START);
```

EVT_TIMER_START	タイマーの開始
EVT_TIMER_STOP	タイマーの停止
EVT_TIMER_COUNT	タイマー時間延長

まず、通信を行うビット配列の定義を行わなければなりません。

今回は、共有メモリ `event` で用いた定義をそのまま、使用します。

また、コンフィギュレーションファイルで指定した、イベントフラグIDがコンフィギュレータにより、`kernel_id.h`内の定義として自動的に変換されますので、そのラベルがイベントフラグIDとして使用することができます。

イベントの送信時、`set_flg` サービスフラグを用いて、イベントを送信してください。

```
ER set_flg(ID flgid, FLGPTN setprn);
```

ID	flgid	セット対象のイベントフラグのID番号
FLGPTN	setprn	セットするビットパターン

この場合、エラーは発生しませんので、エラーのチェックの必要がありません。

イベントフラグによるタスク間通信4

受け側タスクの実装

- 時間待ち機能付きイベント・フラグ待ち (`wai_flg`)、または、ポーリングによるイベント・フラグ待ち (`pol_flg`)を使用する
- サービス・コールの結果が `E_OK` のとき、`flgptn` に設定のビットがオンになる
- フラグを取り出した後、`clr_flg` にて、フラグをクリアする

```
#define EV_TIME      (起動|停止|延長)
/* ここから下は、タイマータスクの中の記述*/
FLGPTN flgptn;      /* フラグ取り出し領域*/
TMO tmout;          /* 待ち時間 (ms)*/
ER ercd;            /* サービスコールの結果*/
:
tmout = 待ち時間;
ercd =
twai_flg(EVT_FLGID, EV_TIME, TWF_ORW, &flgptn, tmout);
if(ercd == E_OK){
    clr_flg(EVT_FLGID, ~flgptn);
    :
}
```

`EV_TIME`は3つの待ちビットのOR値です。

`twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *flgptn, TMO tmout);`

ID flgid 待ち対象のイベントフラグID

FLGPTN waiptn 待ちビットパターン

MODE wfmode 待ちモード

FLGPTN flgptn 受信パターンの格納領域

TMO tmout タイムアウト時間(ms)

待ちモードは `TFW_ANDW` は待ちビットのすべてのビットがオンで起床することを指定し、`TFW_ORW` は待ちビットのどれかがオンになれば起床することを指定する。

イベントフラグ待ちサービス・コールでフラグを取り出しても、設定されたフラグはクリアされません、`clr_flg` サービス・コールを用いてクリアを行なう必要があります。タイムアウトの設定ですが、`tmout` の値は値によって待ち条件が変わります。

`TMO_POL(=0)` `pol_flg` と同等でポーリング設定

`TMO_FEVR(=-1)` `wai_flg` と同等で永久待ち設定

また、`tmout` の値はミリ秒値です。

まとめ、宿題

1. カップラーメンタイマーの実装
2. レビュー
3. 同期と通信
4. イベントフラグを用いた改造
5. まとめ、宿題

最後にまとめを行います。OAKS16 miniを購入して、宿題や鹿威しにも、チャレンジしてください。

タスク間の通信と同期の設計 1

共有メモリによる通信

- タスク間で共有化するメモリ領域上に共有データを置く
- 排他制御が必要 共有データが1度に読み書きできる場合は例外)
 - ・割り込み/デスパッチ禁止
 - ・セマフォ/ミューテックスを使用
 - ・タスク優先度の変更
- 事象の発生を知らせる仕組みが別途必要な場合も
 - ・タスクの起動/起床
 - ・イベントフラグ/Condition Variableなどを使用

実際のシステム設計で、タスク間通信を機能を構築する為のノウハウを説明します。通信方式は、大別して「共有メモリによる通信」と「メッセージによる通信」に分けることができます。

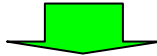
「共有メモリによる通信」では、タスク間の受け渡しデータをグローバルメモリ領域やシェアードメモリ上に確保します。この領域上に受け渡しデータを読み書きすることにより、タスク間通信を行います。各タスクや割り込みは非同期に動作しますので、あるタスクがデータを書き終わらないうちに、別のタスクがデータを読み取ると間違った情報を受け取ってしまう可能性があります。そこで、セマフォなどを使って排他制御することによって、このような、状態が発生することを防ぐことができます。

また、事象が変化した場合、すぐに対応を行う必要がある場合、タスクの起動や起床、イベントフラグなどを使って受け取り側タスクを、すぐに動作させる必要があります。

タスク間の通信と同期の設計 2

メッセージによる通信

- タスク間の通信をメッセージ渡しによる実現
- メッセージ通信のための機構が必要
- メッセージ通信機構の種類
 - ・同期メッセージ通信or非同期通信メッセージ
 - ・ポインタを渡すorコピーする
 - ・キューイングの順序 (FIFO順、優先度順)
 - ・メッセージが無い/フル時の振る舞い
 - ・パケットの単位があるorない



RTOSの持つメッセージ通信機構 (複数の機構を持つ場合もある)の性質を良く理解することが必要



「メッセージによる通信」では、メールボックスやデータキューなどのメッセージ機能を用いて構築します。メッセージの内容はシステム設定時よく、吟味する必要があります。メッセージ機構は、排他制御をキチンと行えば、自作することもできます。システム全体を考慮して最適な方法を構築してください。

タスク間の通信と同期の設計 3

共有メモリvs.メッセージ通信

- 基本的には、片方の方式で実現できることは、もう片方の方式でも実現できる 但し、タスクの待ち条件に注意
 - ・一般的には、共有メモリ方式の方がオーバーヘッドが小さい
 - ・システム検証には、メッセージ方式の方が扱いやすい。
例えば、問題の切り分けが容易
 - 状況により、片方の方式が有利/便利な状況がある
 - ・情報の流れが1:nの場合や、情報が必要なタイミングが不定の場合には、共有メモリ方式が有利
 - ・情報のキューイングが必要な時は、メッセージ方式が便利
- 両方式の混在は、システム設計が複雑となり注意が必要

セマフォを用いた排他制御

サンプルプログラムの紹介

- ディレクトリ `MUTEX1` にサンプルプログラムがあります
- `ENTRY_TASK`: 一定周期でタスク情報とシステム時間をコンソールに表示します
- `DISPLAY_TASK`: 一定周期でタスク情報とシステム時間をコンソール表示します
- 2つのタスクで非同期に表示を行うため、情報が混ざり合って表示されます
- セマフォによる排他制御で正しく表示させてください

TOPPERS/JSPは、優先度ベース、プリエンプティブなタスクスケジューリングを行うため、優先度が低いタスクが実行中でも優先度の高いタスクの実行条件が整えば、実行が中断さて、優先度の高いタスクに移行します。`mutex1`は2つのタスクで定期的にシリアルデバイスに対して表示を行います。`ENTRY_TASK`は`DISPLAY_TASK`より優先度が高いため、`DISPLAY_TASK`の表示中でも、表示を開始するため2つの表示が混ざり合って表示されます。

セマフォを用いて、排他制御を行い表示が混ざりあわないようにプログラムを修正しましょう。セマフォの生成は以下の静的APIを使います。

```
CRE_SEM(ID semid, {ATR sematr, UINT isemcnt, UINT maxsem});
```

【パラメータ】

ID semid	セマフォID
ATR sematr	セマフォ属性 (TA_TFIFO TA_TPRI)
UINT isemcnt	セマフォの資源数初期値(1)
UINT maxsem	セマフォの最大資源数(1)

TOPPERS/JSPの特徴

μITRON4.0準拠のスタンダードプロファイル

- 読みやすく改造しやすいソースコード
- 他のターゲットへのポータリングが容易な構造
- 高い実行性能と小さいRAM使用量
- LinuxおよびWindowsでのシミュレーション環境
- 開発環境まで含めたフリーソフトウェアのみで構築可能

TOPPERS/JSPカーネルは、μITRON4.0に準じたリアルタイムカーネルで、TOPPERSプロジェクトの最初の開発成果です。JSPはJust Standard Profileの略で、この名前が示すとおりμITRON4.0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ます。JSPカーネルの最新リリースは以下の (<http://www.toppers.jp/index.html>) からダウンロード可能です。

主な特徴は以下の通りです。

・読みやすく改造しやすいソースコード

研究・開発への利用を想定して開発を行ったことから、ソースコードの読みやすさや改造しやすさに重点を置いて実装しました。ただし、安易な読みやすさを追求して、効率の悪い平易なアルゴリズムを採用することはしていません。むしろ、タイムイベントの管理にヒープ構造を用いるなど、複雑であっても効果的なアルゴリズムは積極的に採用しています。

・他のターゲットへのポータリングが容易な構造

カーネルのできる限り多くの部分をC言語で記述しています。ターゲット独立部とターゲット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やシステムへのポータリングが容易な構造となっています。実際、ターゲットプロセッサのアーキテクチャを熟知していれば、3日間程度でポータリングできたという報告もあります。

・高い実行性能と小さいRAM使用量

大部分がC言語で記述されているカーネルとしては、高い実行性能と小さいRAM使用量を実現しています。

・LinuxやWindows上でのシミュレーション環境

JSPカーネルを、LinuxやWindows上の動作させるためのシミュレーション環境を用意しています。これらのシミュレーション環境は、LinuxおよびWindowsの1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、組込みシステムのプロトタイプ開発やロジックレベルでの検証、リアルタイム学習用途などに最適なものです。

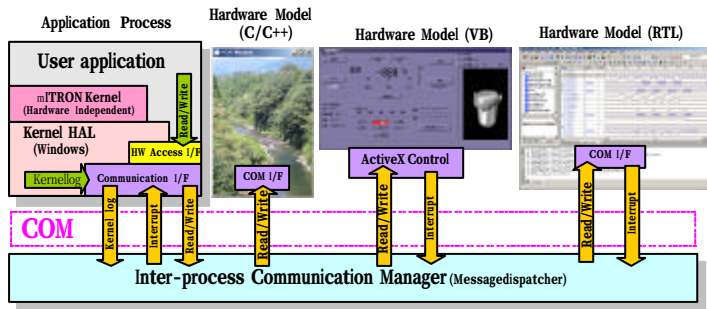
・開発環境まで含めてフリーソフトウェアのみで構築可能

GCCやGNU開発環境を標準のソフトウェア開発環境としています。そのためユーザは、カーネル本体のみならず開発環境もフリーで入手し、システム開発を行うことが可能です。

実機レスシミュレーション1

Windowsシミュレータを用いた開発

- Windows上での実装、シミュレーション、試験が可能です
- C++言語によるソースコードが公開されています



Windowsシミュレータは、TOPPERS/JSP1.4のWindows版カーネルに付加し、デバイスとのインターフェイスとシステムログのインターフェイスを提供します。

これらのインターフェイスを構築する部分は「デバイスドライバー」と呼ばれ、COMのインターフェイスを通してWindows上に常駐します。デバイスドライバーとVisual BASICとのインターフェイスは、デバイスとのインターフェイスについては「デバイスコントロール」、システムログについては「ログウォッチャコントロール」プログラムが担当します。これらのプログラムのインストール方法については、TOPPERS/JSP 1. 4内のdoc/windows.txtを参照してください。

実機レスシミュレーション2

カップラーメン・タイマーの開発

- 実行環境が異なっても、同じソースプログラムで開発実行が可能です

OAKS16-MINI	シミュレータ		OAKS16-MINI	シミュレータ	
timer3.cfg	timer3.cfg	×	monitor.h		
timer3.h	timer3.h	×	monsil.h	simsil.h	×
timer3,.c	timer3.c	○	oaks16.c		
oaks16mini.h	oaks16mini.h	○	task_expansion.c		
sim_device.h	sim_device.h	○	task_expansion.h		
monitor.c			oaks16_device.c	oaks16_device.c	○
monitor.cfg					

実行環境が異なる為、コンフィギュレーションファイルとヘッダーファイルは共通化できません。また、モニター機能は、VC++のデバック機能やデバイスドライバーで代用できるため、シミュレーション環境では削除しました。IOポートへのインターフェイスは環境によって入れ替えが必要です。

- 1) sil.h 実機環境
- 2) monsil.h モニターとのインターフェイスを持った実機環境
- 3) simsil.h Windowsシミュレーター環境

その他のソースやインクルードファイルはそのまま、実機、シミュレーションを問わず共通に使用できます。

標準的なTOPPERS/JSPの開発環境 1

LINUXとCygwin

- configディレクトリの下には、TOPPERS/JSPに対応したCPUや開発ボードの依存部のソースが収められている
- それらは、以下のネーミング・ルールで収められている
CPU名－ツール名／ボード名
以下のように、ツール名が省略されているものはGNUの開発環境である

m16c-renesas/oaks16_mini ルネサステクノロジ社の環境

sh3/ms7727cp1 GNUの環境

GNUの開発環境はLINUXやCygwin上で開発できる

このセミナーでは、ルネサステクノロジ社の開発ツールを使って、TOPPERS/JSPの構築を行いました。TOPPERS/JSPの基本的な開発環境はGNUを使った開発環境です。

configディレクトリ以下の種々のCPUやボードの対応はネーミング・ルールがあり、ツール名が省略されているものはGNUの環境で構築することを指定しています。

GNUのコンパイラやリンカーを使用するためには、LINUX等のUnixの実行環境が必要です。また、Windows上では、Cygwin等のUnix実行環境を必要とします。

この場合、構築にはPerl言語やGnu-makeを用いますので、開発の管理を行う技術者は、このような、知識を習得することも、有用でしょう。

標準的なTOPPERS/JSPの開発環境 2

Cygwin上での開発

- WindowsにCygwinをインストールする
- Cygwin上に、ターゲットのGNUの開発環境のインストールを行う
 - BINUTIL,GCC,GDB,NEWLIBのインストール
- コンフィギュレーションツールのビルド
- コンフィギュレーションスクリプトを使って、ターゲット用のMakefileやサンプルプログラムを生成する
- 依存関係を構築後、ターゲットをビルドする

開発環境の説明

開発環境の構築

- 開発ツールは自分のパソコンにも構築可能
付属のCD-ROMには、開発環境
(TM,KC30WA,KD30,FlashSta)が入っています、CD-ROMのインストール手順に従って自分のパソコンにもインストールが可能です
- フロッピーにサンプルソースがあります
フロッピー中にこのセミナーのサンプルソースが入っています
これにより、教材の環境の構築が可能です

OAKS16 mini版の開発環境設定方法

1. TOPPERS プロジェクトのWeb より、jsp-1.4.lzh(Windows版TOPPERS) ダウンロードして解凍してください。解凍したディレクトリはjsp-1.4¥jspですが、ディレクトリ名をTOPPERS¥jsp-1.4に変更してください。

2. 添付したファイルを解凍して1. で作成したディレクトリに添付のディレクトリとファイルを追加してください

TOPPERS¥jsp-1.4	¥monitor	<-追加
	¥OBJ	<-追加

3. TMでtmkファイルをオープンすれば、プロジェクトとしてコンパイル、リンク可能な状態になります。但し、*.tmiファイルと*.tmkファイル中に絶対パスが設定されており、解凍を行ったディレクトリと異なる場合はエディタで修正を行ってください。フロッピー上のパスはd:¥usr¥TOPPERS¥jsp-1.4となっています。

教材の作成

■ 教材の作成に協力して頂いたにお礼を申し上げます

株)東陽テクニカ

二上貴夫

株)ヤマハ

穴田啓樹

株)リコー

森本亮太

名古屋市工業研究所

小川清、斉藤直希

■ 参考資料

「組込みシステムとリアルタイムスケジューリング入門」

名古屋大学

高田広章

「リアルタイムOS入門-徹底解説 μITRON4.0仕様-」

宮城県産業技術センター 今井和彦

SESSAME組込みソフトウェア技術者初級セミナーテキスト」

組込みソフトウェア管理者・技術者育成研究会