

TOPPERS基礎実装セミナー

(LPC2388版:基本)

基礎2編:2日目

TOPPERSプロジェクト
教育ワーキング・グループ

2014/02/08

TOPPERSプロジェクト認定

1



本教材の利用条件

NEXCESS基礎コース01 組込みソフトウェア開発技術の基礎

Copyright (C) 2006-2007 by 名古屋大学 組込みソフトウェア技術者人材養成プログラム
Copyright (C) 2006-2007 by 本田晋也, 高田広章

上記著作権者は、以下の(1)~(4)の条件を満たす場合に限り、本コンテンツ(本コンテンツを改変・翻訳したものを含む、以下同じ)を使用・複製・改変・翻訳・再配布(以下、利用と呼ぶ)することを無償で許諾する。

- (1) この枠内の著作権表記等が、そのままの形でコンテンツ中に含まれていること。
- (2) 本コンテンツを再配布する場合には、再配布の形態等を、以下のウェブサイトから報告すること。
<http://www.nces.is.nagoya-u.ac.jp/NEXCESS/REPORT/>
- (3) 本コンテンツを改変・翻訳する場合には、コンテンツを改変・翻訳した旨の記述を、コンテンツ中に含めること。また、改変・翻訳者の著作権表記等は、この枠内の著作権表記等とは別に行うこと。
- (4) 本コンテンツ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、上記著作権者を免責すること。

※ 本コンテンツの一部は、文部科学省 科学技術振興調整費により、名古屋大学 組込みソフトウェア技術者人材養成プログラム(NEXCESS)の一環として作成しました。

※ 本コンテンツ中に記載されている商品名やサービス名などは、各社の商標または登録商標です。

2014/02/08

TOPPERSプロジェクト認定

2



本ドキュメントに関して

1. 著作権に関する表記

＜TOPPERS基礎実装セミナー(LPC2388版:基本2)2日目＞

Copyright (C) 2006-2008 by 名古屋大学 組込みソフトウェア技術者人材養成プログラム

Copyright (C) 2006-2008 by 本田晋也, 高田広章 名古屋大学

Copyright (C) 2008 -2013 by 竹内良輔 (株)リコー

上記著作権者は、以下の (1)～(3) の条件を満たす場合に限り、本ドキュメント (本ドキュメント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) 本ドキュメントを利用する場合には、上記の著作権表示、この利用条件および以下の無保証規定が、そのままの形でドキュメント中に含まれていること。

(2) 本ドキュメントを改変する場合には、ドキュメントを改変した旨の記述を、改変後のドキュメント中に含めること。ただし、改変後のドキュメントがTOPPERSプロジェクト指定の開発成果物である場合には、この限りではない。

(3) 本ドキュメント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。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

本ドキュメントは、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、本ドキュメントに関して、特定の使用目的に対する適合性も含めて、いかなる保障もしない。また、本ドキュメント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。

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 Real-time 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。

2014/02/08

TOPPERSプロジェクト認定

3



スケジュール

■ 2日目

- | | |
|----------------------|-------|
| 1. 実習開発環境の構築 | 0.5時間 |
| 2. RTOSの基礎プログラミング実習1 | 1.5時間 |
| 3. RTOSの基礎プログラミング実習2 | 2.0時間 |
| 4. 同期通信機能プログラミング実習 | 1.5時間 |
| 5. まとめ | 0.5時間 |

2014/02/08

TOPPERSプロジェクト認定

4



概要

マイコンボード上でのRTOSを用いたプログラミングを学ぶ

- 実行・開発環境の説明
 - ASPでのインクルードファイルと型設定
 - 実行・開発環境の使用方法
- RTOS基本プログラミング実習
 - RTOS基本プログラミングの概要
 - タスク生成方法
 - デバッグ方法
 - マルチタスクプログラミング
 - 周期ハンドラ
 - 割込みハンドラ

2014/02/08

TOPPERSプロジェクト認定

5



概要

- RTOS同期・通信機能プログラミング実習
 - セマフォによる排他制御
 - イベントフラグ, データキューによる事象通知
- カップラーメンタイマーの実装(宿題)
 - タスク設計
 - 実装

2014/02/08

TOPPERSプロジェクト認定

6



実習開発環境の構築

2014/02/08

TOPPERSプロジェクト認定

7



2日目の開発環境の構築作業

- 2日目の実習環境はROMモニタを使用します
 - 作成したプログラムはTeraTermを用いてダウンロードし、そのまま、実行確認ができます
- 2日目の実習環境を構築するために以下の作業を行う
 - 作業用のmy_programディレクトリを作成
 - カーネルライブラリを作成
 - 毎回、カーネルプログラムのコンパイルが必要なくなる
 - サンプル用のled1_taskプログラムをビルド、実行する
 - このプログラムを元にいろいろな改造を行う

2014/02/08

TOPPERSプロジェクト認定

8



開発・実行環境の使用方法

- 開発環境と2種類の実行環境使用方法について解説
- ROM使用版
 - ROMモニタを使用してプログラムをダウンロードして実行
 - ダウンロード後、ROMモニタのGOコマンドでプログラムが起動する
 - ASPがUART0を乗っ取る形で実行
 - 電源OFF、リセットでプログラムが消滅する
- 直接実行版
 - プログラムをフラッシュメモリに書き込み実行
 - 電源オンで毎回実行できる
 - タスクモニタを用いたデバッグは可能

2014/02/08

TOPPERSプロジェクト認定

9



開発・実行環境の使用方法

- TOPPERS/ASPカーネルは、コンソールの出力先としてUARTを使用する
- 出力先はカーネルのソースコード中にあるマクロで変更可能(初期設定はUART0)
 - target_syssvc.h
 - 1とするとUART0, 2とするとUART1が出力先となる

```
#define SIO_PORTID          UINT_C(1)
#define LOGTASK_PORTID     SIO_PORTID
```

- 直接実行版では、フラッシュメモリ書き込みソフトがUART0を使用するため、UART1で実行してもよい
- ASPは2つのポート対応しているため切り替え設定が可能

2014/02/08

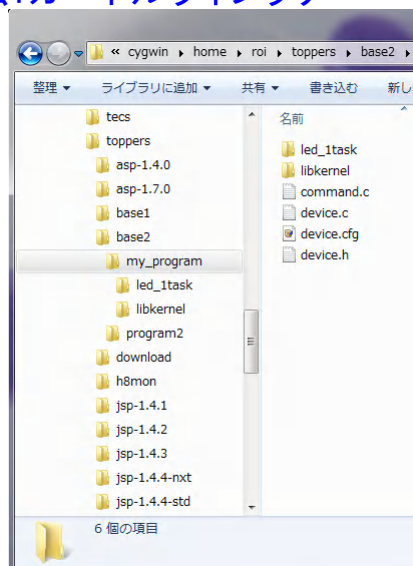
TOPPERSプロジェクト認定

10



開発・実行環境の使用法:カーネルライブラリ

- 学習用のワークスペースは asp-1.7.0に並列して開発環境のbase2をコピーする
- base2ディレクトリの中に my_programを作成して開発を行う
- my_program中にカーネルのプログラムをライブラリ化したカーネルライブラリをlibkernel中に作成する



2014/02/08

TOPPERSプロジェクト認定

11



開発・実行環境の使用法:カーネルライブラリ

- my_programディレクトリ中にlibkernelディレクトリを作成し、以下の手順でカーネルライブラリlibkernel.aを作成する
- Configureコマンド実行後は、3つのファイル Makefile,sample1.cfg,sample1.c,sample1.hが生成される

```
$ cd base2
$ mkdir my_program
$ cd my_program
$ mkdir libkernel
$ cd libkernel
$ ../../asp-1.7.0/configure -T lpc2388_gcc -f
```

2014/02/08

TOPPERSプロジェクト認定

12



開発・実行環境の使用方法:タスク時間の測定

- タスクの実行時間の測定が行えるようMakefileのコンパイルスイッチに、LOG_DSP_ENTERとLOG_DSP_LEAVEを追加する
- カーネルライブラリをビルドする

Makefile

```
130 #
131 # 共通コンパイルオプションの定義
132 #
133 COPTS := COPTS -g -DLOG_DSP_ENTER -DLOG_DSP_LEAVE
134 ifndef OMIT_WARNING_ALL
```

```
$ make depend
$ make libkernel.a
```

2014/02/08

TOPPERSプロジェクト認定

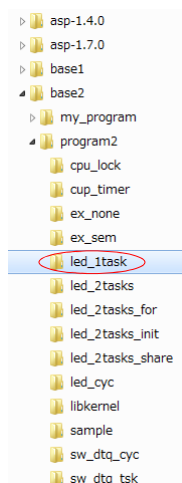
13



開発・実行環境の使用方法:ビルド

- kernelライブラリが正しく参照しているかの確認を行う
- 確認用にprogram2/led_1taskをビルドする
- led_1taskフォルダのコピー
 - ファイルを変更できるよう、led_1task のフォルダをコピーする
 - program2下の4つのファイルをmy_program/ にコピーする
 - device.cfg device.c device.h
 - command.c

```
$ cd ../led1_task
$ make depend
$ make
```



2014/02/08

TOPPERSプロジェクト認定

14



開発・実行環境の使用方法:ビルド

- led_1taskがカーネルプログラムをコンパイルせずビルドできる
- led_1task/Makefileの82行目にカーネルライブラリの指定変数(KERNEL_LIB)の定義があり、この設定があるとカーネルライブラリをビルドしない
- configureでsample1を生成した場合は、KERNEL_LIBの設定は空になっている

```
78 #
79 # カーネルライブラリ(libkernel.a)のディレクトリ名
80 # (カーネルライブラリもmake対象にする時は、空に定義する)
81 #
82 KERNEL_LIB = ../libkernel
```

2014/02/08

TOPPERSプロジェクト認定

15



開発・実行環境の使用方法:ダウンロード

- ROMモニタ使用版
 - ROMモニタ起動後、LDコマンドにてダウンロードに移行
 - asp.srecをTeraTermにドロップしてダウンロード
- 直接実行版
 - ROM実行モードでビルドを行う(デフォルトがROMモニタ版)
 - make depend
 - make DBGENV=ROM
 - 基礎1セミナーの”フラッシュメモリからの起動”を参考にasp.hexをFlash Magicを用いてフラッシュメモリに書き込む
 - 書き込みが終了したら、J8、J9をもとに戻してリセットを押す

2014/02/08

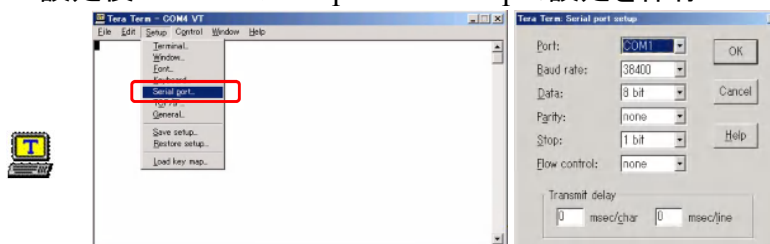
TOPPERSプロジェクト認定

16



開発・実行環境の使用方法：TeraTermの実行

- TeraTermは、UARTから送られてくるデータを表示する
- UARTによる通信は双方でスピード(BPS)やデータフォーマットを合わせる必要がある
- TeraTermのメニューのSetup→Serial port で設定
 - Port は接続しているCOMポートの番号に設定
 - Baud rate : 38400bps, Data : 8bit
 - Parity : none, Stop : 1bit, Flow control : なし
- 設定後メニューのSetup→Save setupで設定を保存



2014/02/08

TOPPERSプロジェクト認定

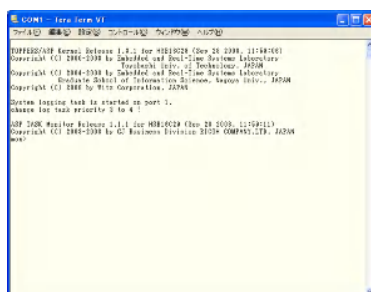
17



開発・実行環境の使用方法：実行

- ROMモニタ使用 : GOコマンドにより実行する
- 直接実行版 : ジャンパーピンを戻しリセットボタンを押す
- 実行されると、起動メッセージが TeraTerm に表示される
再びFlashMagicによりフラッシュメモリに

 プログラムを書き込む場合は、TeraTermを終了させる



2014/02/08

TOPPERSプロジェクト認定

18



RTOSの基礎プログラミング実習

1. 概要

2. タスク生成方法
3. デバッグ方法
4. マルチタスクプログラミング
5. 周期ハンドラ
6. 割込みプログラム

2014/02/08

TOPPERSプロジェクト認定

19



概要

- TOPPERS/ASPカーネルを用いてRTOSの基本プログラミングを学ぶ
- タスク生成方法
 - 1タスクによるLED点滅プログラムの解説
- デバッグ方法
 - デバッグ方法を解説し、1タスクによるLED点滅プログラムに適用
- マルチタスクプログラミング
 - 1タスクによるLED点滅プログラムをマルチタスクプログラムに拡張
- 周期ハンドラ
 - 周期ハンドラを用いてLED点滅プログラムを作成
- 割込みプログラム
 - スイッチ割込みによる割込みサービスルーチン、ハンドラを作成

2014/02/08

TOPPERSプロジェクト認定

20



RTOS基本プログラミング実習：プログラムファイル

• プログラムファイルの置き場所

- 教材ディレクトリ/program2

タスク生成

led_1task : 1タスクによるLED点滅プログラム

led_1task_syslog : 1タスクによるLED点滅プログラム (printf() デバッグ)

マルチタスクプログラミング

led_2tasks_for : 2タスクによるLED点滅プログラム (forループ版)

led_2tasks : 2タスクによるLED点滅プログラム (dly_tsk版)

led_2tasks_init : 2タスクによるLED点滅プログラム (初期化ルーチン)

led_2tasks_share : 2タスクによるLED点滅プログラム (コード共有)

周期ハンドラ

led_cyc : 周期ハンドラによるLED点滅プログラム

割込みプログラム

sw_int : 割込みサービスルーチンによるLED点滅プログラム

sw_int2 : 複数の割込みによるLED点滅プログラム

sw_int_api/2 : 割込みハンドラからのAPI呼び出し (タスク起動/起床)

cpu_lock : CPUロック状態

2014/02/08

TOPPERSプロジェクト認定

21



RTOSの基礎プログラミング実習

1. 概要
2. [タスク生成方法](#)
3. デバッグ方法
4. マルチタスクプログラミング
5. 周期ハンドラ
6. 割込みプログラム

2014/02/08

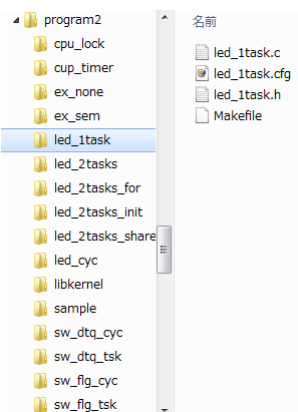
TOPPERSプロジェクト認定

22



1タスクによるLED点滅プログラム : led_1task 概要

- タスクによりLED1を一定周期で点滅させる
- 学習内容
 - プログラムの全体像の理解
 - デバイスドライバ
 - タスクの記述方法
 - コンフィギュレーションファイル
 - タスクの生成方法
- 構成ファイル
 - led_1task.h : ヘッダーファイル
 - led_1task.c : Cソースファイル
 - led_1task.cfg : コンフィギュレーションファイル
 - Makefile : メイクファイル



2014/02/08

TOPPERSプロジェクト認定

23



led_1task 解説 : デバイス操作関数群 LED

- 拡張I/OボードのLEDとスイッチ用のドライバ関数
 - my_programディレクトリ以下のドライバを使用します
 - device.c device.h device.cfg
- LED関連
 - マクロ

#define LED1	0x80
#define LED2	0x40
#define LED3	0x20
:	
#define LED8	0x01
 - 関数
 - void led_init(intptr_t exinf)
 - 初期化(引数は0とする)
 - void led_out(unsigned char led_data)
 - 引数にマクロで点灯パターンを指定

2014/02/08

TOPPERSプロジェクト認定

24



led_1task 解説 : ITRON用のデバイスドライバ

- ITRONではI/O関連の機能のデバイス関連のAPIを定めていないためOSを使用しない場合と同様の記述で作成

```
/* FIO2レジスタ */
#define BREG_FIO2_SET ((volatile unsigned long *)0x3FFFC058)
#define BREG_FIO2_CLR ((volatile unsigned long *)0x3FFFC05C)
/* LED接続ポート書き込み */
void
led_out(unsigned char led_data){
    *BREG_FIO2_SET = ~led_data & LED_MASK;
    *BREG_FIO2_CLR = led_data & LED_MASK;
}
```

- 割込み関連(後述)
 - 割込みハンドラの登録や呼び出しはTOPPERS/ASPで実現(ITRONではユーザー側での記載が必要)
 - 静的APIを用いて割込み設定を行います

2014/02/08

TOPPERSプロジェクト認定

25



led_1task 解説 : ディップスイッチ操作関数群

- LPC2388はユーザー使用可能なDIPSWはないため、UARTの設定で8つのDIPSWを設定する
- マクロ

```
#define DSW1      0x01
#define DSW2      0x02
#define DSW3      0x04
    :
#define DSW8      0x80
```

- 関数
 - unsigned char switch_dip_sense(void)
 - ディップスイッチの状態取得
 - 戻り値にON状態スイッチのビットを'1'とした値を返す

2014/02/08

TOPPERSプロジェクト認定

26



led_1task 解説 : プッシュスイッチ操作関数群

- マクロ

```
#define PSW2 0x001
```

- 関数

- void switch_push_init(intptr_t exinf)
 - 初期化(引数は使用しない)
- void switch_push_clear(void)
 - PUSHスイッチ割込みのクリアを行う

- 割込み設定

- 割込み設定は静的APIで行う
- (他のRTOSとの大きな違い)

2014/02/08

TOPPERSプロジェクト認定

27



led_1task 解説 : ヘッダーファイル(led_1task.h)

- タスクやハンドラの外部宣言を記述
- コンフィギュレーションファイルで参照する各種マクロとタスクやハンドラのプロトタイプ宣言を記述する
- TOPPERS_MACRO_ONLYはアセンブラ言語から参照された場合、解釈できないマクロ以外の記載を示すためのコンパイルスイッチで、ここでは必ずしも設定する必要はありません

```
#define STACK_SIZE 386 /* タスクのスタックサイズ */
#define DEFAULT_PRIORITY 8 /* タスクの優先度 */
```

```
#ifndef TOPPERS_MACRO_ONLY
extern void led1_task(intptr_t exinf);
```

```
#endif /* TOPPERS_MACRO_ONLY */
```

アセンブラ言語で解釈できない記載を定義

対応が分かり易いように

2014/02/08

TOPPERSプロジェクト認定

28



led_1task 解説 : Cソースファイル(led_1task.c)

- タスク等を記述
- TOPPERS/ASPの標準インクルードファイルと自動ID割付け結果ヘッダファイルの二つのファイルをインクルードする
- タスクは実行されると無限ループ内で, busy_wait()関数により一定時間毎にLED1を点滅させる

```
#include <kernel.h>
#include <t_syslog.h>
#include <t_stdlib.h>
#include "kernel_cfg.h"
#include "device.h"
```

```
void
led1_task(intptr_t exinf){
    led_init(0);
    for (;;) {
        led_out(LED1);
        busy_wait();
        led_out(0x00);
        busy_wait();
    }
}
```

点灯

消灯

2014/02/08

TOPPERSプロジェクト認定

29



led_1task 解説 : コンフィギュレーションファイル

- 静的APIを記述 (led_1task.cfg)
- led1_task 関数から開始されるタスクを生成
- ヘッダーファイルをインクルード
- サポートモジュールのコンフィギュレーションファイルをインクルード

```
INCLUDE("target_timer.cfg");
INCLUDE("syssvc/syslog.cfg");
INCLUDE("syssvc/banner.cfg");
INCLUDE("syssvc/serial.cfg");
INCLUDE("syssvc/logtask.cfg");
INCLUDE("monitor/monitor.cfg");
```

サポートモジュールのコンフィギュレーションファイル

```
#include "device.h"
#include "led_1task.h"
```

起動時に実行可能状態

タスクの起動番地

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, led1_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

2014/02/08

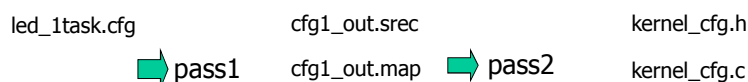
TOPPERSプロジェクト認定

30



led_1task 解説 : コンフィギュレータ

- コンフィギュレーションファイル进行处理するプログラム
- コンフィギュレータを実行 (ProjectEditorでcfg.oを選択して部分ビルド)
- コンフィギュレーションファイルはコンフィギュレータにより処理され、以下の二つのファイルが生成される
 - kernel_cfg.h : ID自動割付結果ファイル
 - kernel_cfg.c : カーネル構成・初期化ファイル
- コンフィギュレータはプロジェクトのビルド時に自動的に実行
- コンフィギュレータは2つのパスで実行される



2014/02/08

TOPPERSプロジェクト認定

31



led_1task 解説 : ID自動割付結果ファイル kernel_cfg.h

- カーネルオブジェクトのIDマクロを定義
- タスクや他のカーネルオブジェクトにはIDが割り付けられる
- IDはコンフィギュレータが自動的に割り付ける
- 静的APIにはオブジェクトIDをマクロ定義する名前を指定する
- kernel_cfg.hには名前をID番号に定義するマクロ定義が出力される
- プログラムではこのマクロを用いてIDを指定する
- コンフィギュレーションファイル **.hに定義がない**

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, led1_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

- kernel_cfg.h

```
#define LOGTASK      1
#define MONTASK      2
#define MAIN_TASK    3
#define SERIAL_RCV_SEM1 1
.....
```

- タスクのID指定

```
act_tsk(MAIN_TASK);
```

- あと2つタスクが存在する....

2014/02/08

TOPPERSプロジェクト認定

32



led_1task 解説 : カーネル構成・初期化ファイル kernel_cfg.c

- カーネルオブジェクトの生成情報から生成されたカーネルの内部データ
- タスク管理ブロック(TCB)やスタック領域を確保

```
#include "led_1task.h"
...
static STK_T _kernel_stack_MAIN_TASK[COUNT_STK_T(STACK_SIZE)];
...
const TINIB _kernel_tinib_table[TNUM_TSKID] = {
    { TA_ACT, (intptr_t)(0), (led1_task),
      INT_PRIORITY(DEFAULT_PRIORITY), ROUND_STK_T(ST
    ....
```

- コンフィギュレーションファイルでのincludeまたはINCLUDE指定(静的API)

```
include "led_1task.h"
```

RTOSの基礎プログラミング実習

1. 概要
2. タスク生成方法
3. [デバッグ方法](#)
4. マルチタスクプログラミング
5. 周期ハンドラ
6. 割込みプログラム

タスクモニタ：タスクモニタとは？

- led_1taskには、led1_taskタスク以外に2つのタスクが存在する
- タスクモニタ(タスク)
 - led_1task.cfg からインクルードするmonitor.cfg 内で生成
 - UARTを経由してコマンドを受け、他のタスクの管理や監視を行うタスク
- ログタスク
 - led_1task.cfg からインクルードするlogtask.cfg 内で生成
 - カーネルログの出力を行うタスク
- asp-1.7.0/monitor/monitor.cfg

```
#include "monitor.h"
CRE_TSK(MONTASK, { TA_HLNG|TA_ACT,
                  MONITOR_PORTID,
                  monitor,MONITOR_PRIORITY,
                  MONITOR_STACK_SIZE, NULL });
```

2014/02/08

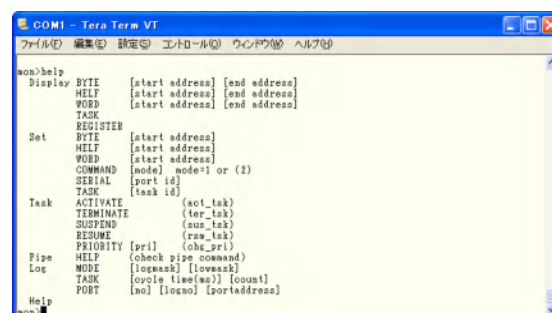
TOPPERSプロジェクト認定

35



演習：タスクモニタの使用方法

- コンソールにコマンドを入力して使用する
- led_1taskを実行すると、タスクモニタが起動されてコンソールにモニタの起動ログを表示し、コマンド待ちを示す mon> が表示される
- helpと入力するとコマンド一覧が表示される



2014/02/08

TOPPERSプロジェクト認定

36



演習：タスクモニタによるタスク操作

- タスクモニタではタスクに対して次の操作が可能
 - タスクの起動(act_tsk) : タスクを実行可能状態に
 - タスクの終了(ter_tsk) : タスクを休止状態に
 - タスクのサスペンド(sus_tsk) : タスクを強制待ち状態に
 - タスクのレジューム(rsm_tsk) : タスクを強制待ち状態から復帰
 - 優先度の変更(chg_pri) : タスクの優先度を変更する
- タスクの操作する前に操作対象のタスクを指定する必要がある
 - タスクIDは kernel_cfg.h で定義されている値
 - 一度設定すると、それ以後のタスク操作は指定したタスクに対して行われる

```
mon> set task [タスクID]
```

2014/02/08

TOPPERSプロジェクト認定

37



演習：led1_taskの停止

led1_task(ID番号3)を休止状態にする

- set task 3 : ID番号3のタスク指定
- task terminate : タスクの終了

```

mon:display task
cur id pri state ps stank inistack inimize
001 4 WAITING 000e05a0 0000125f 000000d0 1014
# mon 3 RUNNING 00001479 000012d0 512
002 3 DORMANT 000e0526 0000103a 000014d0 388
mon:task 3
002(DORMANT)>task terminate
terminate ter_tsk(3) ? result = E_OK !
002(DORMANT)>display task
cur id pri state ps stank inistack inimize
001 4 WAITING 000e05a0 0000125f 000000d0 1014
# mon 3 RUNNING 00001479 000012d0 512
002 3 DORMANT 000e7012 0000103a 000014d0 388
002(DORMANT)>

```

- LED1の点滅が停止する(実行可能状態から休止状態に)
- display task でタスクの状態を表示可能
- コマンドは最初の一文字で短縮可能

2014/02/08

TOPPERSプロジェクト認定

38



演習：タスクモニタによるled1_taskの起動

先ほど休止状態にしたled1_taskを起動させる

- 起動は task activate コマンドで行います
- 起動すると, LED1の点滅が再開される
 - タスクとして関数の先頭から実行される(休止状態から実行可能状態)

```

COM1 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(C) ウィンドウ(W) ヘルプ(H)

mon/display task
our id pri state pc stack inistack inisize
001 4 WAITING 000e05a0 0000125f 00000edc 1024
* mon 3 RUNNING 00001478 000012dc 512
* 003 8 RUNNABLE 000e0638 0000163a 000014dc 388

mon/set task 3
003(RUNNABLE)>task terminate
execute ter_task(3) :: result = E_OK !
003(DORMANT)>display task
our id pri state pc stack inistack inisize
001 4 WAITING 000e05a0 0000125f 00000edc 1024
* mon 3 RUNNING 00001478 000012dc 512
* 003 8 DORMANT 000e7812 0000163a 000014dc 388

003(DORMANT)>task activate
execute act_task(3) :: result = E_OK !
003(RUNNABLE)>display task
our id pri state pc stack inistack inisize
001 4 WAITING 000e05a0 0000125f 00000edc 1024
* mon 3 RUNNING 00001478 000012dc 512
* 003 8 RUNNABLE 000e0638 0000163a 000014dc 388

003(RUNNABLE)>
  
```

2014/02/08

TOPPERSプロジェクト認定

39



演習：タスクモニタ使用時の優先度設定に関する注意

優先度の設定によってはタスクモニタを使用できなくなる場合がある

- タスクモニタの優先度は3で通常はコンソールからの入力待ちで待ち状態になっており, コンソールに入力があると起床して指定されたコマンドを実行する
- タスクモニタでタスクの優先度を設定できるが, タスクモニタより高い優先度を設定すると, タスクモニタが動作しなくなるため, 注意が必要
 - led1_task(タスク3)はループで時間を作っているため

```

003(RUNNABLE)>task priority 6
execute chg_pri(3, 6) :: result = E_OK !
003(RUNNABLE)>display task
our id pri state pc stack inistack inisize
001 4 RUNNABLE 000e05a0 0000125f 00000edc 1024
* mon 3 RUNNING 00001478 000012dc 512
* 003 6 RUNNABLE 000e0638 0000163a 000014dc 388

003(RUNNABLE)>task priority 2
  
```

2014/02/08

TOPPERSプロジェクト認定

40



printf() デバッグ

- オートドックスなデバッグ手法
 - UNIX等のプログラム開発と同じ
- プログラム中で変数の値を表示させたり、パスの確認のために使用
- 出力先や出力方法が必要
 - PC上のプログラムの場合は、画面に文字を表示する
 - 組込みの場合、UART経由で文字を出力可能
 - UARTがない場合もしくは、コード中で出力系の関数を使えない箇所(割込みハンドラ等)は、LEDに値を表示してprintf()デバッグの代わりとする

2014/02/08

TOPPERSプロジェクト認定

41



printf() デバッグ : syslog()関数

TOPPERS/ASPでは、printf()の代わりにsyslog()関数を使用する

- 標準関数を実装するとリソース使用量が多くなるため、代替方法を用意している
- printf()と異なり第一引数はログ出力レベルを指定
- 引数の数に応じてx(0~6)の値を変更する

```
syslog_x(LOG_NOTICE, .....)
```

```
syslog_0(LOG_NOTICE, "test");
syslog_1(LOG_NOTICE, "test = %d", i);
syslog_2(LOG_NOTICE, "test = %d , 0x%x", i, e);
...
syslog_6(LOG_NOTICE, .....
```

2014/02/08

TOPPERSプロジェクト認定

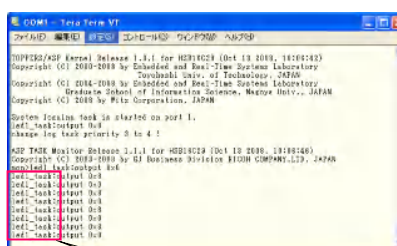
42



演習 : led_1taskへのprintf()デバッグ適用(led_1task_syslog)

- led_1taskをLED出力部分でLEDの出力値をsyslog()でコンソールに出力するように改造
- マルチタスク環境ではどのタスクのログか分かるように出力元のタスク名やタスクIDも一緒に出力

```
void
led1_task(intptr_t exinf){
    led_init(0);
    for (;;) {
        syslog_1(LOG_NOTICE,
            "led1_task : out_put 0x%x", LED1);
        led_out(LED1);
        busy_wait();
        syslog_1(LOG_NOTICE,
            "led1_task : out_put 0x%x", 0x00);
        led_out(0x00);
        busy_wait();
    }
}
```



出力元

2014/02/08

TOPPERSプロジェクト認定

43



RTOSの基礎プログラミング実習

1. 概要
2. タスク生成方法
3. デバッグ方法
4. マルチタスクプログラミング
5. 周期ハンドラ
6. 割込みプログラム

2014/02/08

TOPPERSプロジェクト認定

44



2タスクによるLED点滅プログラム : led_2tasks_for

led_1task にLED3を点滅させるタスク(led3_task)
を追加する

- 学習内容
 - 新規プロジェクトの作成方法
 - 待ち状態
 - 初期化ルーチン
 - タスクのコード共有

2014/02/08

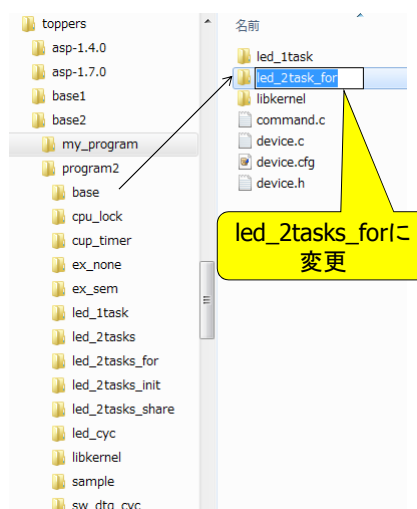
TOPPERSプロジェクト認定

45



演習 : 新規プロジェクトの作成方法

- 雛形のプロジェクトファイルをベースに作成
- プロジェクトのためのフォルダ(led_2tasks_for)をmy_program/に作成
- program2/baseにあるファイルを作成したフォルダにコピーする



2014/02/08

TOPPERSプロジェクト認定

46



新規プロジェクトの作成方法：ヘッダーファイル

- ヘッダーファイル (led_2tasks.h) をフォルダ led_2tasks_for に作成
- ヘッダーファイルに記述する内容
 - タスクのプロトタイプ宣言
 - タスクのスタックサイズ
 - タスク優先度の設定
 また、マクロ以外のC言語の表記は#ifdef
TOPPERS_MACRO_ONLY ~#endifで囲む必要がある
- led_1taskのヘッダーファイル (led_1task.h) の内容をコピーする

```
#define STACK_SIZE      386 /* タスクのスタックサイズ */
#define DEFAULT_PRIORITY 8 /* タスクの優先度 */
#ifdef TOPPERS_MACRO_ONLY
extern void led1_task(VP_INT exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

2014/02/08

TOPPERSプロジェクト認定

47



プロジェクトの作成方法：Cソースファイル

- Cソースファイル (led_2tasks.c) を作成したフォルダに作成
- Cソースファイルの先頭にはTOPPERS/ASPの標準インクルードファイルと自動ID割付け結果ヘッダファイルの二つのファイルをインクルードする

```
#include <kernel.h>
#include <t_syslog.h>
#include <t_stdlib.h>
#include "kernel_cfg.h"
```

- led_1taskのCソースファイル (led_1task.c) の内容をコピーする

2014/02/08

TOPPERSプロジェクト認定

48



プロジェクトの作成方法 : コンフィギュレーションファイル

- led_2tasks.cfgがカーネルコンフィギュレーションファイル
- xxx.hの部分を作成するプロジェクトのヘッダーファイル (この例ではled_2tasks.h)に置き換える
- プロジェクトに必要な静的APIを追加する

```
INCLUDE("target_timer.cfg")
INCLUDE("syssvc/syslog.cfg")
INCLUDE("syssvc/banner.cfg")
INCLUDE("syssvc/serial.cfg")
INCLUDE("syssvc/logtask.cfg")
INCLUDE("monitor/monitor.cfg")
#include "device.h"
#include "xxx.h"
/* 静的API追加 */
```

- led_1taskのカーネルコンフィギュレーションファイルの静的API(タスクled1_taskの生成)をコピーする

2014/02/08

TOPPERSプロジェクト認定

49



演習 : 2タスクによるLED点滅プログラムの作成 (led_2tasks_for)

- led_1taskにLED3を点滅するタスク(led3_task)を追加する
- 手順
 - LEDの状態は共有変数により保持する
 - led3_task の内容は led1_task をベースにしてforループで時間を作成する
 - led3_task の優先度やスタックサイズは led1_task と同一にする
 - タスクが追加されていることは、タスクモニタで、display task を実行することにより確認できる
 - 正しくタスクを追加しても、LEDは片方しか点滅しないので、理由を考察する

2014/02/08

TOPPERSプロジェクト認定

50



led_2tasks_for 解説 : Cソースファイル(led_2tasks.c)

- LED1点滅タスク
- LED3点滅タスク

LEDのデータの
共有メモリ化を
行う

```
unsigned char led_data= 0;
void
led1_task(intptr_t exinf){
    led_init(0);
    for (;;) {
        led_data &= ~LED1;
        led_out(led_data);
        busy_wait();
        led_data |= LED1;
        led_out(led_data);
        busy_wait();
    }
}
```

```
void
led3_task(intptr_t exinf){
    for (;;) {
        led_data &= ~LED3;
        led_out(led_data);
        busy_wait();
        led_data |= LED3;
        led_out(led_data);
        busy_wait();
    }
}
```

2014/02/08

TOPPERSプロジェクト認定

51



led_2tasks_for 解説 : その他のファイル

- ヘッダーファイル(led_2tasks.h)

```
#ifndef TOPPERS_MACRO_ONLY
extern void led1_task(intptr_t exinf);
extern void led3_task(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

- コンフィギュレーションファイル(led_2tasks.cfg)

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, led1_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(LED3_TASK, { TA_ACT, 0, led3_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

2014/02/08

TOPPERSプロジェクト認定

52



led 2tasks for: 実行

- 意図した通りに動作したか?
- ➡ 片方のLEDしか点滅しない
- led1_task, led3_taskは同一優先度でかつ、待ち状態に入らないため、どちらかのタスクのみ(レディキューの先頭にある方)実行状態になっている
- 同一優先度のタスク間ではFCFS(First Come First Served)のルールで実行される

log task 1000コマンドでタスクの実行状態を確認

task 4(led3_task)の実行
時間がゼロ

GOMI - Toshiba VT

Copyright (C) 2004-2008 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN

Copyright (C) 2008 by Nite Corporation, JAPAN

System loading task is started on port 1.

change bus task priority 3 to 4

ASP TASK Monitor Release 1.1.1 for H2010029 (Oct 13 2006, 11:21:27)

Copyright (C) 2008-2009 by GJ Business Design Institute INC./COMPANY, LTD., JAPAN

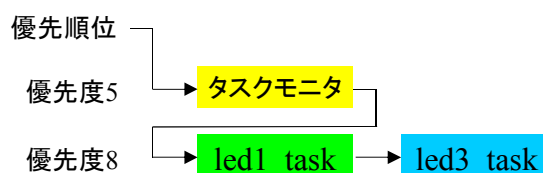
memdisplay task

```

port id  port state      bus  stack  initval  initval
00000000  00000000  00000000  00000000  00000000  00000000
00000001  00000001  00000000  00000000  00000000  00000000
00000002  00000002  00000000  00000000  00000000  00000000
00000003  00000003  00000000  00000000  00000000  00000000
00000004  00000004  00000000  00000000  00000000  00000000
00000005  00000005  00000000  00000000  00000000  00000000
00000006  00000006  00000000  00000000  00000000  00000000
00000007  00000007  00000000  00000000  00000000  00000000
00000008  00000008  00000000  00000000  00000000  00000000
00000009  00000009  00000000  00000000  00000000  00000000
0000000A  0000000A  00000000  00000000  00000000  00000000
0000000B  0000000B  00000000  00000000  00000000  00000000
0000000C  0000000C  00000000  00000000  00000000  00000000
0000000D  0000000D  00000000  00000000  00000000  00000000
0000000E  0000000E  00000000  00000000  00000000  00000000
0000000F  0000000F  00000000  00000000  00000000  00000000
00000010  00000010  00000000  00000000  00000000  00000000
00000011  00000011  00000000  00000000  00000000  00000000
00000012  00000012  00000000  00000000  00000000  00000000
00000013  00000013  00000000  00000000  00000000  00000000
00000014  00000014  00000000  00000000  00000000  00000000
00000015  00000015  00000000  00000000  00000000  00000000
00000016  00000016  00000000  00000000  00000000  00000000
00000017  00000017  00000000  00000000  00000000  00000000
00000018  00000018  00000000  00000000  00000000  00000000
00000019  00000019  00000000  00000000  00000000  00000000
0000001A  0000001A  00000000  00000000  00000000  00000000
0000001B  0000001B  00000000  00000000  00000000  00000000
0000001C  0000001C  00000000  00000000  00000000  00000000
0000001D  0000001D  00000000  00000000  00000000  00000000
0000001E  0000001E  00000000  00000000  00000000  00000000
0000001F  0000001F  00000000  00000000  00000000  00000000
00000020  00000020  00000000  00000000  00000000  00000000
00000021  00000021  00000000  00000000  00000000  00000000
00000022  00000022  00000000  00000000  00000000  00000000
00000023  00000023  00000000  00000000  00000000  00000000
00000024  00000024  00000000  00000000  00000000  00000000
00000025  00000025  00000000  00000000  00000000  00000000
00000026  00000026  00000000  00000000  00000000  00000000
00000027  00000027  00000000  00000000  00000000  00000000
00000028  00000028  00000000  00000000  00000000  00000000
00000029  00000029  00000000  00000000  00000000  00000000
0000002A  0000002A  00000000  00000000  00000000  00000000
0000002B  0000002B  00000000  00000000  00000000  00000000
0000002C  0000002C  00000000  00000000  00000000  00000000
0000002D  0000002D  00000000  00000000  00000000  00000000
0000002E  0000002E  00000000  00000000  00000000  00000000
0000002F  0000002F  00000000  00000000  00000000  00000000
00000030  00000030  00000000  00000000  00000000  00000000
00000031  00000031  00000000  00000000  00000000  00000000
00000032  00000032  00000000  00000000  00000000  00000000
00000033  00000033  00000000  00000000  00000000  00000000
00000034  00000034  00000000  00000000  00000000  00000000
00000035  00000035  00000000  00000000  00000000  00000000
00000036  00000036  00000000  00000000  00000000  00000000
00000037  00000037  00000000  00000000  00000000  00000000
00000038  00000038  00000000  00000000  00000000  00000000
00000039  00000039  00000000  00000000  00000000  00000000
0000003A  0000003A  00000000  00000000  00000000  00000000
0000003B  0000003B  00000000  00000000  00000000  00000000
0000003C  0000003C  00000000  00000000  00000000  00000000
0000003D  0000003D  00000000  00000000  00000000 
```

led 2tasks for : タスクレベルの変更

- タスクモニタで led1_task と led3_task の優先度を変更してスケジューリング方法を確認する
 - set task 4 : タスク4(led3_task)に変更
 - task priority 6 : タスク優先度を6に変更
- これによりLED3のみ点滅する



演習 : 2タスクによるLED点滅プログラム改(led_2tasks)

- LED1を1秒間隔, LED3を0.25秒間隔で点滅するようにプログラムを書き換える
- 方式
 1. 定期的に2個のタスクを切り替える方式
 2. 時間生成を待ち状態で実現する方式
 – 本プログラムは時間を作るのが本質なので2を採用
- ITRON仕様には、任意の時間自タスクを待ち状態にするAPI `dly_tsk()` がある
 - ミリ秒単位で待ち時間を指定可能

2014/02/08

TOPPERSプロジェクト認定

55



led_2task : Cソースファイル(led_2tasks.c)

- 共有メモリの排他制御は必要ない
 - タスクが切り替わるポイントを考える
- LED1点滅タスク
- LED3点滅タスク

```
#define LED1_INTERVAL 1000
void
led1_task(intptr_t exinf){
    led_init(0);
    for (;;) {
        led_data &= ~LED1;
        led_out(led_data);
        dly_tsk(LED1_INTERVAL);
        led_data |= LED1;
        led_out(led_data);
        dly_tsk(LED1_INTERVAL);
    }
}
```

```
#define LED3_INTERVAL 250
void
led3_task(intptr_t exinf){
    for (;;) {
        led_data &= ~LED3;
        led_out(led_data);
        dly_tsk(LED3_INTERVAL);
        led_data |= LED3;
        led_out(led_data);
        dly_tsk(LED3_INTERVAL);
    }
}
```

led_dataの
排他制御は
必要ない

2014/02/08

TOPPERSプロジェクト認定

56



初期化ルーチン

- LED接続ポートの初期化関数(led_init(0))は led1_task が実行している
 - led1_task が実行されるまでは初期化されず, 先にled3_task が実行された場合、出力関数(led_out())が正しく動作しない
- 対処方法
 1. 初期化済みかどうかチェック変数を用意
 2. 両方のタスクで初期化
 3. 初期化関数としてカーネル起動時に実行

➡ 3の方法が最もスマート
- ITRON仕様には初期化ルーチンを登録する静的API (ATT_INI)がある

属性(Cかアセンブラか)

拡張情報
(初期化ルーチンへの引数)

初期化ルーチンの
起動番地

```
ATT_INI({ATR intatr, intptr_t exinf, FP intrtn});
```

2014/02/08

TOPPERSプロジェクト認定

57



演習 : 2タスクによるLED点滅プログラム改(led_2tasks_init)

- led_2tasks をベースにLEDの初期化関数をカーネルの初期化ルーチンとして実行するようプログラムを変更する
- led_2tasksからは初期化ルーチンを外す
- コンフィギュレーションファイル
 - デバイスドライバのヘッダーファイルのインクルード
 - 初期化ルーチンの登録

```
#include "device.h"
ATT_INI({TA_NULL, 0, led_init});
```



- device.cfgの取り込み

```
INCLUDE("../device.cfg");
```

2014/02/08

TOPPERSプロジェクト認定

58



演習：タスクのコード共有 (led_2tasks_share)

- 2タスクによるLED点滅プログラムの
led1_task と led3_task のコードを共有化する
- 個別に持つ情報
 - LEDの設定ビット
 - 点滅周期
 - 構造体の配列に入れ, そのインデックスを exinf で渡す
- 正しく設定するLED情報と点滅周期を得ているかタスク
起動時にコンソールに出力すること

```
typedef struct{
    unsigned char led_bit;
    int          interval;
}LED_TASK_INFO;

LED_TASK_INFO task_info[] = {{LED1, LED1_INTERVAL},
                              {LED3, LED3_INTERVAL}};
```

2014/02/08

TOPPERSプロジェクト認定

59



led_2tasks_share : Cソースファイル (led_2tasks.c)

- 実行されたらローカル変数にLEDの設定ビットと点滅周期をコピーする

```
void
led_task(intptr_t exinf){
    unsigned char led_bit = task_info[(int)exinf].led_bit;
    int          interval = task_info[(int)exinf].interval;
    syslog_3(LOG_NOTICE, "led_task : exinf = %d, led_bit = 0x%x,
                        interval = %d msec", exinf, led_bit, interval);

    for (;;) {
        led_data &= ~led_bit;
        led_out(led_data);
        dly_tsk(interval);
        led_data |= led_bit;
        led_out(led_data);
        dly_tsk(interval);
    }
}
```

2014/02/08

TOPPERSプロジェクト認定

60



led 2tasks share : コンフィギュレーションファイル

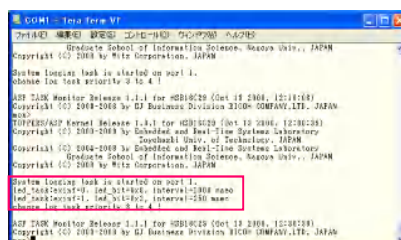
- 同一関数 led task を指定

```

CRE_TSK(MAIN_TASK, { TA_ACT, 0, led_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(LED3_TASK, { TA_ACT, 1, led_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });

```

- ・ コンソール出力



RTOSの基礎プログラミング実習

1. 概要
2. タスク生成方法
3. デバッグ方法
4. マルチタスクプログラミング
5. 周期ハンドラ
6. 割込みプログラム

演習：周期ハンドラによるLED点滅プログラム(led_cyc)

- LED1をタスクにより1秒周期で, LED3を周期ハンドラにより0.5秒周期で点滅させるプログラムを作成
- 手順
 - led_1taskをベースにする
 - LED3点滅タスク(led_3task)は生成しない
 - Cファイルに周期ハンドラを追加
 - コンフィギュレーションファイルに生成情報を追加

led_cyc : Cソースファイル(led_cyc.c)

- 周期ハンドラはタスクと同様に関数として記述する
- LED3点滅周期ハンドラ

```
void
led3_cyc_handler(intptr_t exinf){
    if((led_data & LED3) == LED3){
        led_data &= ~LED3;
    }else{
        led_data |= LED3;
    }
    led_out(led_data);
}
```

led_cyc : その他のファイル

- ヘッダーファイル (led_cyc.h)

```
#define LED3_INTERVAL 500
#ifdef TOPPERS_MACRO_ONLY
.....
extern void led3_cyc_handler(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

- コンフィギュレーションファイル (led_cyc.cfg)

```
CRE_CYC(LED3_CYC_HANDLER, {TA_STA,
0, led3_cyc_handler, LED3_INTERVAL, 0});
```

RTOSの基礎プログラミング実習

1. 概要
2. タスク生成方法
3. デバッグ方法
4. マルチタスクプログラミング
5. 周期ハンドラ
6. 割込みプログラム

割込みプログラミング：割込みルーチン登録

- TOPPERS/ASPの割込みルーチンの使い方を学ぶ
 - ASPでは割込みハンドラと割込み要求ライン属性の設定にて割込みを登録する

ハンドラ番号

属性(実装定義)

割込みハンドラの
起動番地

```
DEF_INH(INHNO inhno, {ATR inhatr, FP inthdr});
```

- LPC2388では、ハンドラ番号として”ソフトウェア割込み番号”を指定する
- DEF_INHは割込みハンドラの登録及び、ハンドラの前後処理は行いますが、割込み関連の属性の設定はCFG_INTで行う
- タスク実行時は、プロセッサの状態は割込み許可状態で割込みマスクはしない状態です

2014/02/08

TOPPERSプロジェクト認定

67



割込みプログラミング：割込み要求ライン属性設定

- 割込み属性の設定を行う
 - 割込み属性を登録する静的API

ハンドラ番号

属性(実装定義)

割込み優先度

```
CFG_INT(INTNO inhno, {ATR intatr, PRI intpri});
```

- intatrは割込み属性([TA_ENAINT] | [TA_EDGE]等)
- intpriは割込み優先度を指定する
 - 優先度は-1~-15で設定する

2014/02/08

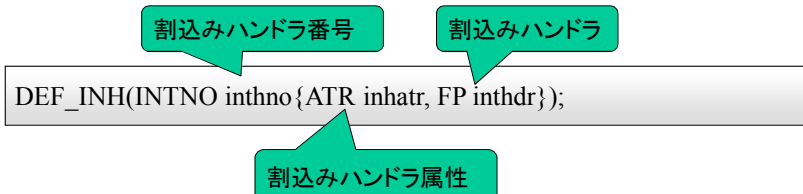
TOPPERSプロジェクト認定

68



割込みプログラム: 割込みハンドラ

- TOPPERS/ASPの割込みハンドラを用いて割込み処理を行う
 - ITRON準拠の割込みハンドラの登録を行う



- inhnoで指定された割込みハンドラ番号に対してinthdrで指定する割込みハンドラを設定する
- inhatrで指定する割込み属性は通常は実装依存でTA_NULLを指定する

2014/02/08

TOPPERSプロジェクト認定

69



演習: 割込みによるLED点滅プログラム(SW_INT)

- 周期ハンドラによるLED点滅プログラム(led_cyc)をベースに, SW2により発生するINT0割込みでLED3を反転するプログラムを作成する
- 手順
 - 割込みサービスルーチンを使用して作成する
 - SW2の初期化ルーチンを登録する
 - switch_push_init(void)
 - プッシュスイッチ初期化
 - SW2の割込みサービスルーチンを作成、静的APIで登録
 - SW2(INT0)の割込み番号は14(INTNO_EINIT0)
 - device.hで定義(INTNO_SW2/INHNO_SW2)
 - SW2の割込み属性を静的APIで登録
 - エッジ割込み、優先度は-4
 - device.hで定義(INTATR_SW2,INTPRI_SW2)

2014/02/08

TOPPERSプロジェクト認定

70



sw_int : Cソースファイル(sw_int.c)

- SW2の割込みサービスルーチン
 - 割込みサービスルーチンはタスクと同様に関数として記述する

```
void
sw_int(void){
    unsigned int led = (unsigned int)LED3;
    switch_push_clear(); /* 割込みクリア */
    if ( (led_data & led) == led ) {
        led_data &= ~led;
    } else {
        led_data |= led;
    }

    led_out(led_data);
    syslog_1(LOG_NOTICE,"sw_int : led_data 0x%x",led_data);
}
```

2014/02/08

TOPPERSプロジェクト認定

71



sw_int : その他のファイル

- ヘッダーファイル(sw_int.h)

```
#ifndef TOPPERS_MACRO_ONLY
extern void led1_task(intptr_t exinf);
extern void sw_int(void);
#endif /* TOPPERS_MACRO_ONLY */
```

- コンフィギュレーションファイル(sw_int.cfg)
 - SW2の割込み番号は device.h で定義済み

```
ATT_INI({ TA_NULL, 0, switch_push_init });
DEF_INH(INHNO_SW2, { TA_NULL, sw_int});
CFG_INT(INTNO_SW2, {TA_ENAINT|INTATR_SW2,INTPRI_SW2});
```

2014/02/08

TOPPERSプロジェクト認定

72



割り込みプログラミング：割り込みサービスルーチン登録

- TOPPERS/ASPの割り込みルーチンの使い方を学ぶ
 - ASPでは割り込みサービスルーチンを登録する静的APIを用意している



- 割り込みサービスルーチン優先度は1、拡張情報にて割り込みサービスルーチンに引数を渡せます。その他はDEF_INH、CFG_INTの設定と同じです

演習：割り込みによるLED点滅プログラム(sw_int2)

- sw_int を割り込みサービスルーチンを用いて記述する
 - 割り込みサービスルーチンは同一属性の割り込みを共通化できる
- 手順
 - sw_intをベースにする

sw_int2 : Cソースファイル (sw_int.c)

- SW2の割込みサービスルーチン
 - 割込みハンドラをATT_ISRを用いて割込みサービスルーチンに書き換える
 - 割込みサービスルーチンではLED3を引数として渡す
- インクルードファイル (sw_int.h)

```
extern void led1_task(intptr_t exinf);
extern void sw_int(intptr_t exinf);
```

- コンフィギュレーションファイル (sw_int.cfg)

```
ATT_ISR({TA_NULL, LED3, INTNO_SW2, sw_int, 1 });
CFG_INT(INTNO_SW2, {TA_ENAINT|INTATR_SW2, INTPRI_SW2});
```

2014/02/08

TOPPERSプロジェクト認定

75



sw_int2 : Cソースファイル (int_sw.c)

- 割込みサービスルーチンを用いれば、PUSHスイッチが複数ある場合、引数にてLEDの設定を行うことができる

```
void
sw_int(intptr_t exinf){
    unsigned int led = (unsigned int)exinf;
    switch_push_clear(); /* 割込みクリア */
    if ( (led_data & led) == led ) {
        led_data &= ~led;
    } else {
        led_data |= led;
    }

    led_out(led_data);
    syslog_1(LOG_NOTICE, "sw_int : led_data 0x%x", led_data);
}
```

2014/02/08

TOPPERSプロジェクト認定

76



演習：割り込みハンドラからのAPI呼び出し (sw_int_api)

- 割り込みハンドラからのAPI呼び出しを行うプログラムを作成
- program2/led_2task_initをmy_program/にコピーしてディレクトリ名をsw_int_apiに変更してください
- プログラム仕様
 - 割り込みハンドラを用いて作成
 - タスクは休止状態で生成 (CRE_TSKの引数の属性にTA_ACTを付加しない)
 - タスクは起動時と終了時にコンソールにその旨を出力
 - SW2割り込みハンドラからのタスクを起動
 - タスクはLED1を0.5秒間隔で点滅を2回繰り返したのちext_tsk() で終了

2014/02/08

TOPPERSプロジェクト認定

77



sw_int_api : Cソースファイル (sw_int_api.c)

- タスクの終了
 - ext_tskは明示的に呼び出さない場合でも、タスクの終了時に自動的に呼び出される

```
void
led1_task(intptr_t exinf){
    int i;
    syslog(LOG_NOTICE,"led1_task : start!");
    for (i = 0; i < 4; i++) {
        if((led_data & LED1) == LED1){
            led_data &= ~LED1;
        } else {
            led_data |= LED1;
        }
        led_out(led_data);
        dly_tsk(LED1_INTERVAL);
    }
    syslog(LOG_NOTICE,"led1_task : done!");
    ext_tsk(); ←ITRON仕様書：P82(PDF P100)
}
```

2014/02/08

TOPPERSプロジェクト認定

78



sw_int_api : 割込みハンドラ

- SW2割込みハンドラ

```
void
sw2_handler(void){
    switch_push_clear();
    syslog(LOG_NOTICE,"sw2_handler : start!");
    SVC_ERROR(act_tsk(MAIN_TASK));
}
```

- 割込みハンドラや周期ハンドラ等の非タスクコンテキストではiが先頭に付くAPIのみが使用できる
- 通常のAPI(act_tsk)を使うとAPIは実行されず、コンテキストエラー(E_CTX)が返される
- APIの使用ミスを防ぐためには、戻り値のチェック(正常終了のE_OKかどうか)が必要
 SVC_ERROR()マクロにより実現

2014/02/08

TOPPERSプロジェクト認定

79



sw_int_api : サービスコールチェックマクロ

- t_perror関数を用いたマクロでエラーのログを表示

```
/*
 * サービスコールのエラーのログ出力
 */
void
svc_perror(const char *file, int_t line, const char *expr, ER ercd)
{
    if (ercd < 0) {
        t_perror(LOG_ERROR, file, line, expr, ercd);
    }
}
#define SVC_ERROR(expr) svc_perror(__FILE__, __LINE__, #expr, (expr))
```

- t_perror関数はercdからエラー内容を変換して表示する関数です
- SVC_ERRORマクロを作成しサービスコールのエラーチェックができます

2014/02/08

TOPPERSプロジェクト認定

80



sw_int_api : その他の修正

- ヘッダーファイル (sw_api_int.h)

```
#ifndef TOPPERS_MACRO_ONLY
extern void led1_task(intptr_t exinf);
extern void sw2_handler(void);
#endif /* TOPPERS_MACRO_ONLY */
```

- コンフィギュレーションファイル (sw_api_int.cfg)
 - SW2の割り込みハンドラ番号は device.h で定義済み

```
CRE_TSK(MAIN_TASK, { TA_NULL, 0, led1_task,
                     DEFAULT_PRIORITY,
                     STACK_SIZE, NULL });
割込みハンドラ番号
DEF_INH(INHNO_SW2, {TA_NULL, sw2_handler });
CFG_INT(INTNO_SW2, {TA_ENAINT|INTATR_SW2,INTPRI_SW2});
```

2014/02/08

TOPPERSプロジェクト認定

81

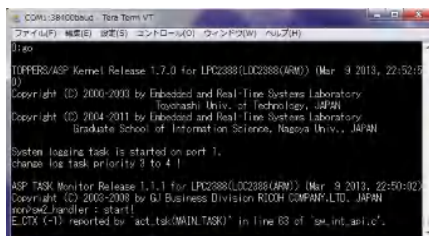


sw_int_api : SVC_PERRORによる引数チェック

- エラーが発生するサービスコールに書き換えて、戻り値が E_OK以外で正しくエラーログが表示されるかの確認を行います

```
void
sw2_handler(void){
    switch_push_clear();
    syslog(LOG_NOTICE,"sw2_handler : start!");
    SVC_PERROR(act_tsk(MAIN_TASK));
}
```

SW2プッシュ



2014/02/08

TOPPERSプロジェクト認定

82



sw_int_api : 起動要求のキューイング

- 起動要求はキューイングされます
- SW2割込みハンドラのiact_tsk()をSVC_PERROR付きに書き換える
- LED1点滅タスクが起動している間にもう一度SW2割込みを入れてLED1点滅タスクに起動要求を出す

起動要求1回目

起動要求2回目

LED1点滅タスク再起動

LED1点滅タスク起動

- 実行可能状態のタスクに、起動要求を出すとその要求はキューイングされる
- 起動要求がキューイングされていると、タスクが終了した後、再びタスク先頭から実行される

2014/02/08

TOPPERSプロジェクト認定

83



sw_int_api : キューイング回数

- SW2を連続して3回押して起動要求を3回発行する
- SW2割込みハンドラを3回以上起動して起動要求を発行すると起動要求は無視され、キューイングオーバーフローエラー(E_QOVR)が返される
- キューイング回数には上限がある
 - ➡ TOPPERS/ASPカーネルではタスク起動のキューイング回数は1回

起動要求1回目

起動要求2回目

起動要求3回目

LED1点滅タスク起動

エラー

2014/02/08

TOPPERSプロジェクト認定

84



演習: 割込みハンドラからのAPIの呼び出し (sw_int_api2)

- タスクの起動要求と同様に, タスクの起床要求もキューイングされる
- タスクの起動要求のキューイングを確認するため, プログラムを変更して, SW2割込みハンドラからタスクを起床させることで, LED1を点滅させるように変更
- タスクは実行可能状態で生成する必要がある

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, led1_task,
                     DEFAULT_MAIN_PRIORITY,
                     STACK_SIZE, NULL });
```

2014/02/08

TOPPERSプロジェクト認定

85



sw_int_api2 : Cソースファイル (sw_int_api.c)

- 起動されると slp_tsk により起床待ち状態となる
- 起床すると, LED1を2回点灯させる

```
void
led1_task(intptr_t exinf){
    int i;
    for (;;) {
        syslog(LOG_NOTICE, "led1_task : sleep!");
        slp_tsk();
        syslog(LOG_NOTICE, "led1_task : wakeup!");
        for (i = 0; i < 4; i++) {
            if ((led_data & LED1) == LED1) {
                led_data &= ~LED1;
            } else {
                led_data |= LED1;
            }
            led_out(led_data);
            dly_tsk(LED1_INTERVAL);
        }
    }
}
```

2014/02/08

TOPPERSプロジェクト認定

86



sw_int_api2 : 割込みハンドラ

- iwup_tsk() により led1_task を起床させる

```
void
sw2_handler(void){
    switch_push_clear();
    syslog(LOG_NOTICE,"sw2_handler : start!");
    SVC_PERRÖR(iwup_tsk(MAIN_TASK));
}
```

- 起床要求のキューイング回数は起動要求と同じく1回



```
COM1:38400baud - Tera Term VT
Graduate School of Information Science, Nagoya Univ., JAPAN

System loading task is started on port 1.
led1_task : sleep!
change log task priority 3 to 4 !

ASP TASK Monitor Release 1.1.1 for LPC2388 (LPC2388 (ARM)) (Mar  9 2013, 23:03:45)
Copyright (C) 2008-2009 by GJ Business Division RICOH COMPANY,LTD. JAPAN
sw2_handler : start!
led1_task : wakeup!
sw2_handler : start!
sw2_handler : start!
E_OVR (-1) reported by "iwup_tsk(MAIN_TASK)" in line 63 of "sw_int_api.c".
led1_task : sleep!
led1_task : wakeup!
led1_task : sleep!
```

2014/02/08

TOPPERSプロジェクト認定

87



CPUロック状態(割込みの禁止・許可)

- 全ての割り込みとディスパッチを禁止する状態
- タスク間, タスク-割込みハンドラ間, 割込みハンドラ間の排他制御を実現する
 - タスク間は, セマフォで排他制御した方がよい
- カーネル起動時は, CPUロック解除状態
- 時間制約のある処理のため一定時間プロセッサを占有するためにも用いられる
 - 例) デバイスの設定
- CPUロック状態では, 一部の例外(CPUロック解除, 状態参照)を除いてシステムコールを呼び出せない
- システムコール

- CPUロック状態への移行

```
loc_cpu()
```

- CPUロック状態の解除

```
unl_cpu()
```

2014/02/08

TOPPERSプロジェクト認定

88



演習 : CPUロック状態 (cpu_lock)

- CPUロック状態の機能を確認するプログラムを作成する
- プログラム仕様
 - タスク(dip_task)は実行可能状態で生成し, 5秒単位に割り込み禁止状態と割り込み許可状態を繰り返す
 - 割り込み禁止状態では「INT DISABLE !」を表示:LED1オン
 - 割り込み許可状態では「INT ENABLE !」を表示:LED1オフ
 - 一定時間はビジーループにより生成する
 - タスクは常に実行状態とする
 - SW2割り込みにより割り込みハンドラ(sw2_handler)を実行する
 - 割り込みハンドラは起動したことが分かるようにログを出力
- 確認事項
 - CPUロック状態では割り込みハンドラが実行されないことを確認
 - CPUロック中に発生した割り込みはCPUロック解除後にどうなるか

2014/02/08

TOPPERSプロジェクト認定

89



cpu_lock : タスク(dip_task)

- 5秒毎に割り込み禁止と許可を繰り返すプログラムを作成

```
void dip_task(intptr_t exinf){
  dly_tsk(500);
  for(;;){
    syslog(LOG_EMERG, "INT_ENABLE !");
    led_out(LED1);
    loc_cpu();
    int_delay(D5SEC);
    led_out(0);
    unl_cpu();
    syslog(LOG_EMERG, "INT_DISABLE !");
    int_delay(D5SEC);
  }
}
```

INT DISABLE ! を表示
5秒間割り込み禁止

INT ENABLE ! を表示
5秒間割り込み許可

2014/02/08

TOPPERSプロジェクト認定

90



cpu_lock : 割込みハンドラ

- 起動するとログを出力

```
void
sw2_handler(void){
    switch_push_clear(); /* 割込みクリア */
    syslog_0(LOG_NOTICE,"sw2_handler : start!");
}
```

cpu_lock : コンフィギュレーションファイル

- タスク生成と割込みハンドラの登録

```
CRE_TSK(MAIN_TASK, { TA_ACT, (VP_INT) 0, dip_task,
                    DEFAULT_MAIN_PRIORITY,
                    STACK_SIZE, NULL });
DEF_INH(INHNO_SW2, { TA_HLNG, sw2_handler });
CFG_INT(INTNO_SW2, {TA_ENAINT|INTATR_SW2, INTPRI_SW2});
```

- 初期化ルーチン
 - ディップスイッチとプッシュスイッチの初期化は device.cfgにて登録済です

RTOS基本プログラム実習：まとめ

- タスク生成
- デバッグ方法
 - タスクモニタ, printf() デバッグ
- プロジェクトの作成方法
- マルチタスクプログラミング
 - forループ, dly_tsk(), コード共有, 初期化ルーチン
- 周期ハンドラ
- 割込みハンドラ
 - 割込みの禁止
 - 戻り値のチェック (SVC_PERROR()), 起動・起床要求のキューイング
- 組込みプログラム開発では、コード修正から実行までが長い場合が多いため、間雲なprintf()デバッグより、コードを注意深くチェックした方が効率的

2014/02/08

TOPPERSプロジェクト認定

93



同期通信機能プログラミング実習

1. 実習内容の説明
2. セマフォによる排他制御
3. イベントフラグによる事象通知
4. データキューによる事象通知

2014/02/08

TOPPERSプロジェクト認定

94



RTOS同期・通信機能プログラミング実習

- TOPPERS/ASPカーネルを用いてRTOSの同期・通信機能を用いたプログラミングを学ぶ
- セマフォ
 - タスク間の排他制御をしないプログラムの解説
 - 上記プログラムへのセマフォによる排他制御追加
- イベントフラグ
 - イベントフラグによる事象通知を行うプログラムの作成
- データキュー
 - イベントプラグを用いたプログラムをデータキューにより書き換え

2014/02/08

TOPPERSプロジェクト認定

95



プログラムファイル

- プログラムファイルの置き場所
 - /program2

ex_none	排他制御なし2タスク出力プログラム
ex_sem	セマフォによる排他制御あり2タスクプログラム
sw_flg_cyc	イベントフラグによる通知プログラム (周期ハンドラから)
sw_flg_tsk	イベントフラグによる通知プログラム(タスクから)
sw_dtq_cyc	データキューによる通知プログラム (周期ハンドラから)
sw_dtq_tsk	データキューによる通知プログラム(タスクから)

2014/02/08

TOPPERSプロジェクト認定

96



同期通信機能プログラミング実習

1. 実習内容の説明
2. [セマフォによる排他制御](#)
3. イベントフラグによる事象通知
4. データキューによる事象通知

2014/02/08

TOPPERSプロジェクト認定

97



演習：排他制御なし2タスク出力プログラム(ex_none)

- 排他制御を行っていないプログラムを実行する
- 同じコードを共有する2つのタスク(タスク1とタスク2)がforループで時間を作りながらグローバル変数countを関数busy_wait_inc()の引数として渡し、その戻り値をcountに入れる
- 同時にローカル変数local_countをインクリメントする
- 2つのタスクは、一定時間で実行を切り替える(方法は後述)
- 2つのタスクでインクリメントするcount は local_count の倍となる

```
int count = 0;
void
task(intptr_t exinf){
    int local_count = 0;
    for (;;) {
        count = busy_wait_inc(count);
        local_count = busy_wait_inc(local_count);
        syslog_3(LOG_NOTICE, "Task %d : count = %d, local_count = %d",
                  exinf, count, local_count);
    }
}
```

2014/02/08

TOPPERSプロジェクト認定

98



ex_none : busy_wait_inc

- busy_wait_incは引数をインクリメントして、一定時間経過してから返す(作為的な)関数

```
#define DELAY_LOOP 0x40000L
busy_wait_inc(int e){
    long i;
    e++;
    for(i = 0; i < DELAY_LOOP; i++);
    return e;
}
```

- タスクは関数 task のタスクを2個作成

```
CRE_TSK(MAIN_TASK, { TA_ACT, 1, task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_ACT, 2, task,
                 DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

2014/02/08

TOPPERSプロジェクト認定

99

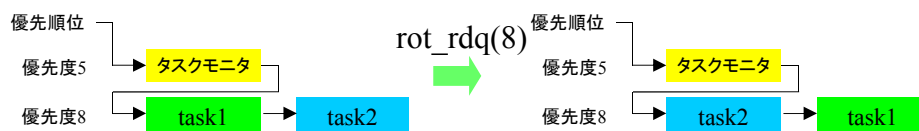


ex_none : irot_rdq()

- このままでは、片方のタスクしか動作しないため、10m周期で起動する周期ハンドラでタスクの優先順位の回転を行う irot_rdq() を実行することで、10m毎にタスクを切り替える

```
void
rot_cyc_handler(void){
    SVC_ERROR(irot_rdq(DEFAULT_PRIORITY));
}
```

優先度8



2014/02/08

TOPPERSプロジェクト認定

100



ex_none : 実行

- 二つのタスクでインクリメントする count が local_count と同じ増加量となってる
 - busy_wait_inc()実行中に周期ハンドラが動作してirot_rdq()によりタスク切り替えが発生した場合、片方のタスクがcountを更新する前にcountを読み込むため
- countの増加を正しくするためには, countへのアクセス(読み込みと更新の間)を排他制御する必要がある

```

COM1:38400baud - Tera Term VT
Copyright (C) 2003-2008 by GI Business Division RICOH COMPANY, LTD. JAPAN
mon>task 2 : count = 1, local_count = 1
Task 1 : count = 1, local_count = 1
Task 2 : count = 2, local_count = 2
Task 1 : count = 2, local_count = 2
Task 2 : count = 3, local_count = 3
Task 1 : count = 3, local_count = 3
Task 2 : count = 4, local_count = 4
Task 1 : count = 4, local_count = 4
Task 2 : count = 5, local_count = 5
Task 1 : count = 5, local_count = 5
Task 2 : count = 6, local_count = 6
Task 1 : count = 6, local_count = 6
Task 2 : count = 7, local_count = 7
Task 1 : count = 7, local_count = 7
Task 2 : count = 8, local_count = 8
Task 1 : count = 8, local_count = 8
  
```

2014/02/08

TOPPERSプロジェクト認定

101



演習 : セマフォによる排他制御プログラム(ex_sem)

- 共有変数 count の更新をセマフォで排他制御する
- セマフォ生成API
 - 属性はTA_NULL(FIFO順)を指定します
 - 初期値, 最大値は排他制御対照の資源の性質によって適切に設定する

属性

初期値

最大数

```
CRE_SEM(ID semid, {ATR sematr, unit_t isement, uint_t maxsem});
```

- セマフォ操作API

```

wai_sem(ID semid) : セマフォ資源の獲得
sig_sem(ID semid) : セマフォ資源の返却
  
```

2014/02/08

TOPPERSプロジェクト認定

102



ex_sem : Cソースファイルとコンフィギュレーションファイル

- Cソースファイル(ex_sem.c)

```
void
task(VP_INT exinf){
    int local_count = 0;
    for (;;) {
        wai_sem(SEM_COUNT);
        count = busy_wait_inc(count);
        sig_sem(SEM_COUNT);
        local_count = busy_wait_inc(local_count);
        ....
    }
}
```

- コンフィギュレーションファイル(ex_sem.cfg)
 - 初期状態では資源は使用可能なので初期値は1
 - 資源は最大で1個なので最大値は1

```
CRE_SEM(SEM_COUNT, {TA_NULL, 1, 1});
```

2014/02/08

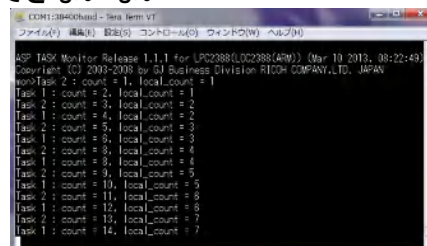
TOPPERSプロジェクト認定

103



ex_sem : 実行結果

- 排他制御によりcountがlocal_countの倍の頻度で更新されることが確認できる
- count更新中にタスクが切り替わっても、他のタスクはセマフォを取得できない



- 他の方
 - ディスパッチ禁止状態やCPUロック状態でも排他制御が可能
 - ディスパッチ禁止 : dis_dsp()/ena_dsp()
 - CPUロック : loc_cpu()/unl_cpu()
- ➡ セマフォを使う場合との相違点は?

2014/02/08

TOPPERSプロジェクト認定

104



ex_sem : 排他制御の実現方法

- 各排他制御方法の違いを理解して使用する必要がある
- 割込みハンドラとの排他制御
 - タスク・割込みハンドラ間, 割込みハンドラ間の排他制御はCPUロック状態でなければ実現できない
- 影響範囲
 - CPUロック状態やディスパッチ禁止状態は, 資源に関係ないタスクや割込みハンドラへ影響を及ぼす
 - セマフォは資源に関連するタスク間でのみの排他制御を実現する
- 割込み応答時間
 - CPUロック状態を長時間使用すると割込み応答性が悪くなる
- プロセッサの占有
 - デバイス等の初期化で, プログラムの実行を中断なく実行したい場合には, CPUロック状態を用いる必要がある

2014/02/08

TOPPERSプロジェクト認定

105



同期通信機能プログラミング実習

1. 実習内容の説明
2. セマフォによる排他制御
3. [イベントフラグによる事象通知](#)
4. データキューによる事象通知

2014/02/08

TOPPERSプロジェクト認定

106



演習：イベントフラグによる通知プログラム (sw_flg_cyc)

- イベントフラグと周期ハンドラを使ってディップスイッチの状態をLEDに反映させるプログラムを作成する
- プログラムの仕様
 - 周期ハンドラ
 - SW1,2,3,4,~8の状態を周期的に監視し、状態が変化したらタスクにイベントフラグで通知する
 - イベントフラグをセットするタイミングはスイッチの状態が変化した場合のみが望ましいですが、常に周期ハンドラでスイッチの状態を設定しても問題ない
 - タスク
 - イベントフラグ待ちとなりイベントフラグによる通知があれば、その通知に従ってLEDの状態を変更する
 - 初期状態
 - 初期化ルーチンで設定する

2014/02/08

TOPPERSプロジェクト認定

107



PIPE DIPコマンド

- LPC2388にはユーザーが使用可能なDIPSWがないためPIPEコマンドを拡張してRAM上でDISPSWを管理し、sw_dip_sense関数で取り出せるようにプログラミングした
 - PIPE DIP でDIPSWの状態表示
 - PIPE DIP 1 1 でDIPSW1をオン
 - PIPE DIP 1 0 でDIPSW1をオフ

```

COM1:38400baud - Tera Term V1
ASP TASK Monitor Release 1.1.1 for LPC2388(LPC2388(ARM)) (Mar.10.2013, 08:33:02)
Copyright (C) 2003-2006 by GJ Business Division RICOH COMPANY,LTD. JAPAN
mon>PIPE DIP
dipsw
1 2 3 4 5 6 7 8
OFF OFF OFF OFF OFF OFF OFF OFF
mon>PIPE DIP 1 1
dipsw
1 2 3 4 5 6 7 8
ON OFF OFF OFF OFF OFF OFF OFF
mon>led_task : eventflag pattern = 0x1
mon>PIPE DIP 1 0
dipsw
1 2 3 4 5 6 7 8
OFF OFF OFF OFF OFF OFF OFF OFF
mon>led_task : eventflag pattern = 0x2

```

2014/02/08

TOPPERSプロジェクト認定

108



sw_flg_cyc : イベントフラグ関連のAPI

- イベントフラグ生成API
 - 属性はTA_CLRを指定する

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

- イベントフラグ操作API

- フラグ待ち

待ちパターン TWF_ORW ビットパターン
`wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)`

- フラグのセット

設定ビットパターン(論理和となる)
`set_flg(ID flgid, FLGPTN setptn )`

2014/02/08

TOPPERSプロジェクト認定

109



sw_flg_cyc : イベントへの割り付け

- フラグの各ビットにイベントを割り付ける

```
#define LED1_ON    0x0001
#define LED1_OFF   0x0002
#define LED2_ON    0x0004
#define LED2_OFF   0x0008
#define LED3_ON    0x0010
#define LED3_OFF   0x0020
:
#define LED8_OFF   0x8000
#define LED_EV_MASK 0xffff
```

- スイッチの状態変化でイベントフラグをセットするためには
前回スキャンしたスイッチ状態を記憶しておく必要がある

2014/02/08

TOPPERSプロジェクト認定

110



sw_flg_cyc : 初期化ルーチン

- 起動時のスイッチの状態をLEDに設定
- 初期化ルーチンをATT_INIでコンフィギュレーションファイルに登録する

```
unsigned short led_data;
unsigned int  sw_state;
void
state_init(intptr_t exinf){
    led_data = 0;
    sw_state = switch_dip_sense();
    if ((sw_state & DSW1) != DSW1) {
        led_data |= LED1;
    }
    if ((sw_state & DSW2) != DSW2) {
        led_data |= LED2;
    }
    ...省略...
    led_out(led_data);
}
```

2014/02/08

TOPPERSプロジェクト認定

111



sw_flg_cyc : 周期ハンドラ

- ディップスイッチの状態をチェックし、変更があればイベントフラグにより通知する

```
void sw_cyc_handler(intptr_t exinf){
    unsigned int tmp;
    FLGPTN flgptn = 0;
    tmp = switch_dip_sense();
    if ((sw_state & DSW1) != (tmp & DSW1)) {
        if ((tmp & DSW1) == DSW1) {
            flgptn |= LED1_ON;
        } else {
            flgptn |= LED1_OFF;
        }
    }
    ... 省略 ...
    if (flgptn != 0) {
        SVC_PERROR(iset_flg(SW_FLG, flgptn));
        sw_state = tmp;
    }
}
```

- 状態に変更があればフラグで通知

2014/02/08

TOPPERSプロジェクト認定

112



sw_flg_cyc : タスク

- イベントフラグによりスイッチの変更を受け取り, LEDを設定する

```
void
led_task(intptr_t exinf){
    FLGPTN flgptn;
    for(;;) {
        wai_flg(SW_FLG, SW_EV_MASK, TWF_ORW, &flgptn);
        syslog_1(LOG_NOTICE, "led_task : eventflag pattern = 0x%x", flgptn);
        if ((flgptn & LED1_ON) == LED1_ON) {
            led_data |= LED1;
        }
        if ((flgptn & LED1_OFF) == LED1_OFF) {
            led_data &= ~LED1;
        }
        ... 省略 ...
        led_out(led_data);
    }
}
```

全イベントOR

いずれかのイベントがセットされたら起床

2014/02/08

TOPPERSプロジェクト認定

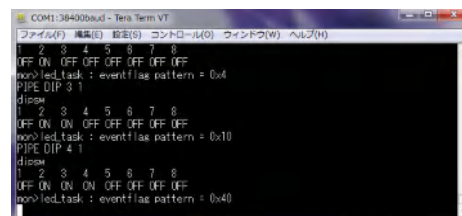
113



sw_flg_cyc : コンフィギュレーションファイル

- フラグの生成
 - 初期状態は何もセットされていないとする
- プログラム実行
 - SWの状態が変更されるとイベントフラグで通知される

```
CRE_FLG(SW_FLG, {TA_CLR, 0});
```



宿題: スイッチのスキャンを周期ハンドラではなくタスク (sw_task) で行うようにプログラムを変更する (sw_flg_task)

2014/02/08

TOPPERSプロジェクト認定

114



同期通信機能プログラミング実習

1. 実習内容の説明
2. セマフォによる排他制御
3. イベントフラグによる事象通知
4. データキューによる事象通知

2014/02/08

TOPPERSプロジェクト認定

115



演習：データキューによる通知プログラム (sw_dtq_cyc)

- イベントフラグによる通知プログラムをデータキューにより通知するように書き換える

- データキュー生成静的API

- 属性はTA_NULL
- バッファ場所はNULL

バッファ数

バッファ場所

```
CRE_DTQ(ID dtqid, {ATR dtqatr, uint_t dtqcnt, void *dtq});
```

- データキュー操作API

属性

- 送信

```
snd_dtq(ID dtqid, intptr_t data)
```

送信データ

- 受信

```
rcv_dtq(ID dtqid, intptr_t *p_data)
```

受信データ入れる変数のポインタ

2014/02/08

TOPPERSプロジェクト認定

116



sw_dtq_cyc : 周期ハンドラ

- スwitchの状態が変化したらデータキューにデータを送る

```
void sw_cyc_handler(intptr_t exinf){
    unsigned int tmp = switch_dip_sense();
    if ((sw_state & DSW1) != (tmp & DSW1)) {
        if ((tmp & DSW1) == DSW1) {
            SVC_PERROR(ipsnd_dtq(SW_DTQ, LED1_ON));
        } else {
            SVC_PERROR(ipsnd_dtq(SW_DTQ, LED1_OFF));
        }
    }
    if ((sw_state & SW2) != (tmp & SW2)) {
        if ((tmp & DSW2) == DSW2) {
            SVC_PERROR(ipsnd_dtq(SW_DTQ, LED2_ON));
        } else {
            SVC_PERROR(ipsnd_dtq(SW_DTQ, LED2_OFF));
        }
    }
    ... 省略...
    sw_state = tmp;
}
```

2014/02/08

TOPPERSプロジェクト認定

117



sw_dtq_cyc : タスク

- データキューからデータを受け取り, LEDをセットする

```
void led_task(intptr_t exinf){
    int sw_data;
    for (;;) {
        rcv_dtq(SW_DTQ, &sw_data);
        syslog_1(LOG_NOTICE, "led_task : switch changed = 0x%x", sw_data);
        if (sw_data == LED1_ON) {
            led_data |= LED1;
        } else if (sw_data == LED1_OFF) {
            led_data &= ~LED1;
        } else if (sw_data == LED2_ON) {
            led_data |= LED2;
        } else if (sw_data == LED2_OFF) {
            led_data &= ~LED2;
        }
        ... 省略 ...
        led_out(led_data);
    }
}
```

2014/02/08

TOPPERSプロジェクト認定

118



sw_dtq_cyc : コンフィギュレーションファイル

- sw_dtq_cyc.cfg

```
CRE_DTQ(SW_DTQ,{TA_NULL, 2, NULL});
```

宿題:

イベントフラグによる通知プログラムと同様に
スイッチのスキャンを周期ハンドラではなくタスク(sw_task)で行
うようにプログラムを変更する(sw_dtq_task)

RTOS同期・通信機能プログラミング実習 : まとめ

- ITRONの同期・通信機能を用いたプログラミング
 - セマフォによる排他制御
 - タスクの優先順位の回転 irot_rdq()
 - イベントフラグによる事象通知
 - 周期ハンドラ、タスク間
 - タスク間
 - データキューによる事象通知
 - 周期ハンドラ、タスク間
 - タスク間

まとめ

2014/02/08

TOPPERSプロジェクト認定

121



TOPPERS/ASP導入実習のまとめ

- コンパイルやリンク等の開発環境とターゲットボードがあれば、比較的簡単にTOPPERS/ASPの動作確認が行える
- ソースが供給されているので、RTOSのしくみを理解しやすい、問題発生時もソースを用いた問題解析が可能である
- RTOSを使用する中規模組込み開発では、多人数の開発者が分業して多くのプログラム開発やミドルウェアの導入を行うことが多い、タスクモニタをシステム用に機能拡張し、問題発生時に、問題の要因を簡単に判別できる開発環境を用意していくことが、開発効率や品質の向上につながる

2014/02/08

TOPPERSプロジェクト認定

122



著者リスト

- 実装開発環境
 - 竹内 良輔((株)リコー)
- RTOSの基礎プログラミング実習1
- RTOSの基礎プログラミング実習2
- 同期通信機能プログラミング実習
 - 本田 晋也(名古屋大学)
 - 竹内 良輔((株)リコー)
- LPC2388ボード対応
 - 竹内 良輔((株)リコー)

謝辞

本教材の開発において、NPO法人組込みソフトウェア管理者・技術者育成研究会およびNPO法人TOPPERSプロジェクトの作成した以下の教材を参考としました。

- OpenSESSAME Seminar組込みソフトウェア技術者・管理者向けセミナー初級者向けテキスト第5版
- TOPPERS初級実装セミナー教材

両団体および上記教材の作者の皆様へ感謝します。

本教材の開発において、NEXCESS初級コースおよび指導者養成コースの受講者の皆様のコメントを参考としました。両コースの受講者の皆様へ感謝します。