

TOPPERS基礎実装セミナー

(Raspberry Pi Pico(W)/Pico2(W)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

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。

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(Raspberry Pi Pico(W)/Pico2(W)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) 2008-2025 by 竹内良輔 教育WG

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。

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

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

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

実習開発環境の構築

2日目の開発環境の構築作業

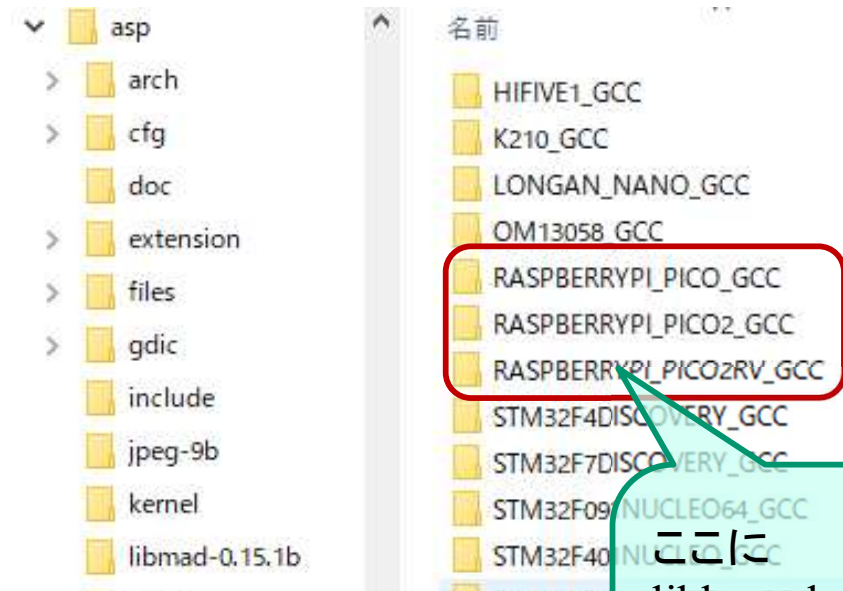
- 2日目の実習環境はROMモニタを使用します
 - 作成したプログラムはTeraTermを用いてダウンロードし、そのまま、実行確認ができます
- 2日目の実習環境を構築するために以下の作業を行う
 - 作業用のmy_programディレクトリを作成
 - カーネルライブラリを作成
 - 毎回、カーネルプログラムのコンパイル必要がなくなる
 - サンプル用のled1_taskプログラムをビルド、実行する
 - このプログラムを元にいろいろな改造を行う

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
- RAM実行版
 - 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UART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

カーネルライブラリのビルド

- これから作業を行うワークディレクトリ
(OBJ/RASPBERRYPI_PICO_GCC)の下にカーネルライブラリ用ディレクトリ: libkernelに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 asp/OBJ/RASPBERRYPI_PICO_GCC
$ mkdir libkernel
$ cd libkernel
$ ../../../../configure -T raspberrypi_pico_gcc -f
```

ここに
libkernel.aを作
成、これを使う
とビルド時間
が短縮できる

カーネルライブラリのビルド

- Pico2の場合、以下のディレクトリでカーネルライブラリを作成する

Pico2:Cortex-M33

```
$ cd asp/OBJ/RASPBERRYPI_PICO2_GCC  
$ mkdir libkernel  
$ cd libkernel  
$ ../../../../configure -T raspberrypi_pico2_gcc -f
```

Pico2:RISC-V

```
$ cd asp/OBJ/RASPBERRYPI_PICO2RV_GCC  
$ mkdir libkernel  
$ cd libkernel  
$ ../../../../configure -T raspberrypi_pico2rv_gcc -f
```

RISC-Vでの学習にはRISC-Vクロスコンパイラをインストールする必要があります

開発・実行環境の使用方法: タスク時間の測定

- asp/OBJ/RASPBERRYPI_PICO_GCC/libkernelを修正
- 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

Makefile

```
143 ifndef OMIT_OPTIMIZATION
144   COPTS := $(COPTS) -O2
145 endif
146 CDEFS := $(CDEFS) -DLOG_DSP_ENTER -DLOG_DSP_LEAVE
147 INCLUDES := -I. -I$(SRCDIR)/include -I$(SRCDIR)/arch ....
```

```
$ make libkernel.a
```

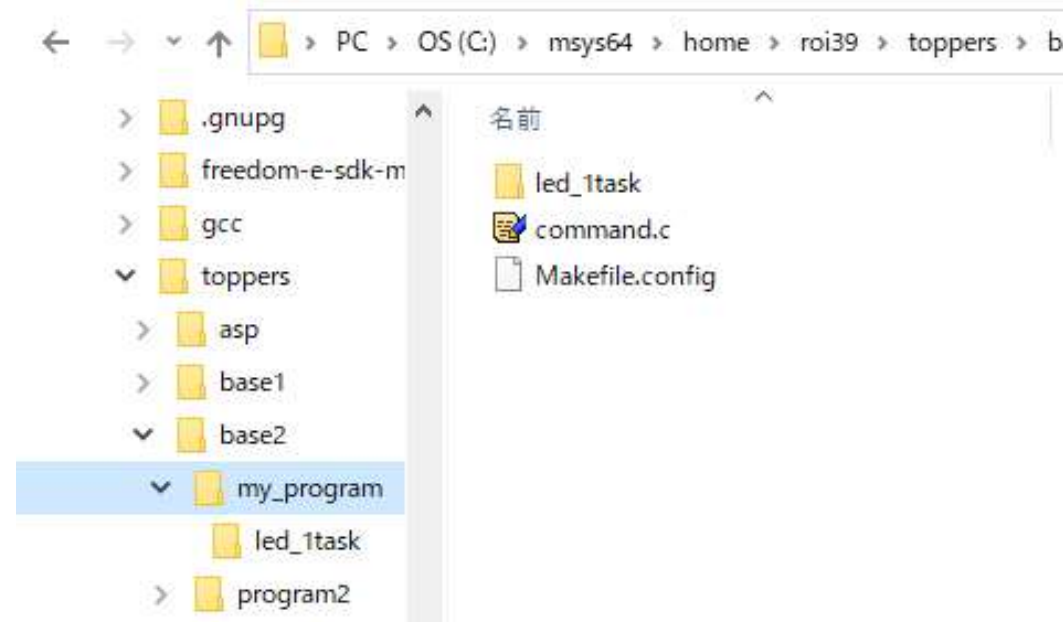
開発・実行環境の使用方法:ビルド

- 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

```
81 #  
82 # カーネルライブラリ(libkernel.a)のディレクトリ名  
83 # (カーネルライブラリもmake対象にする時は、空に定義する)  
84 #  
85 KERNEL_LIB = $(SRCDIR)/OBJ/$(TARGET2)/libkernel
```

開発・実行環境の使用方法:カーネルライブラリ

- 学習用のワークスペースはasp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
- カーネルライブラリは
asp/OBJ/RASPBERRYPI_PICO_GCC/libkernel1に作成

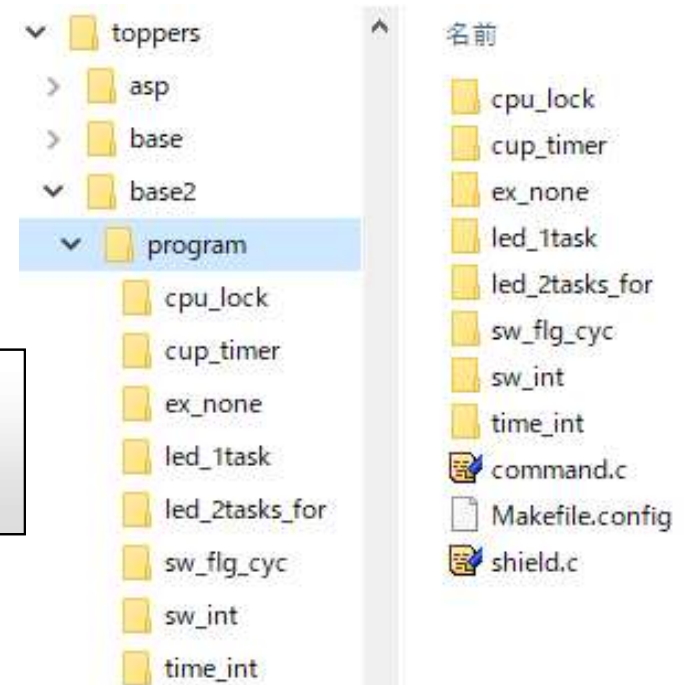


開発・実行環境の使用方法:ビルド

- 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program/led_1taskをビルドする
- led_1taskフォルダに移動
 - makeコマンドでビルドする

Pico2:Cortex-M33の場合、make BOARDNO=1
Pico2:RISC-Vの場合、make BOARDNO=2

```
$ cd ../../../../base2/program/led_1task  
$ make
```

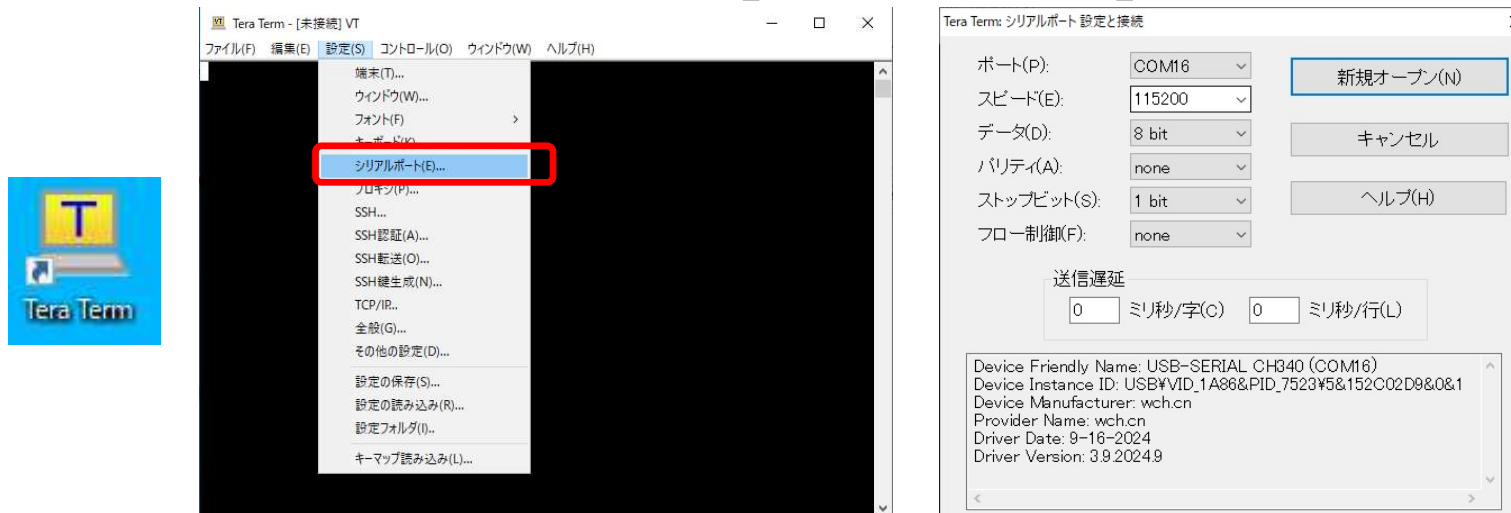


開発・実行環境の使用方法:ダウンロード

- RAM実行版
 - 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 DBGENV=ROM
 - BOOT-SWを押しながら、PCにUSB接続し、表示されたフォルダにasp.uf2をドロップし、フラッシュメモリに書き込む

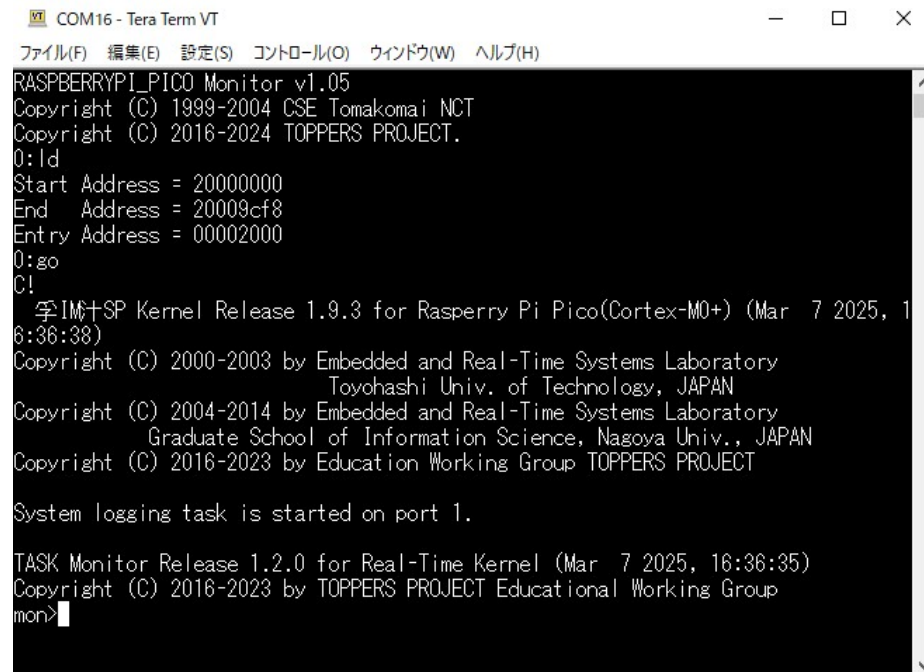
開発・実行環境の使用手法：TeraTermの実行

- 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 : 115200bps, 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



開発・実行環境の使用方法：実行

- RAM実行 : GOコマンドにより実行する
- 直接実行版 : asp.uf2を書き込む
- 実行されると、起動メッセージが TeraTerm に表示される
ROMモニタを再び書き込む場合、rommon_pico.uf2をフラッシュROMに書き込む



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
RASPBERRYPI_PICO Monitor v1.05
Copyright (C) 1999-2004 CSE Tomakomai NCT
Copyright (C) 2016-2024 TOPPERS PROJECT.
0:ld
Start Address = 20000000
End Address = 20009cf8
Entry Address = 00002000
0:so
C!
  孚IM+SP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  7 2025, 1
6:36:38)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
                          Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
                          Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

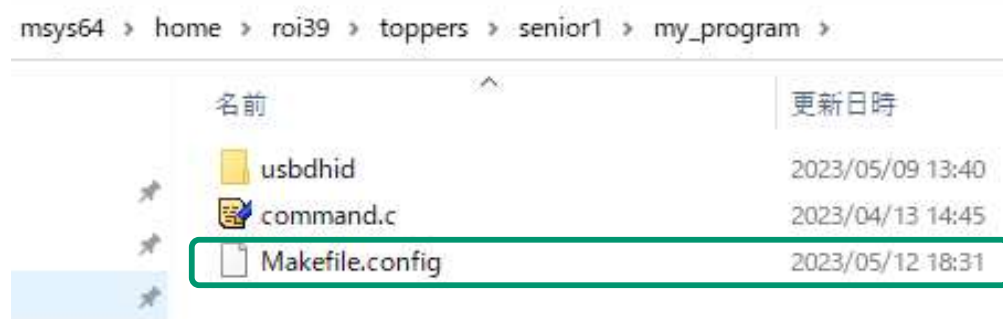
System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  7 2025, 16:36:35)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>
```

Pico W/Pico2 WをCPUボードとして使用している場合

- Pico W/Pico2 WをCPUボードとして使用している場合は、my_program/Makefile.configのボードタイプをPICO_Wに変更します
- この設定で、Raspberry Pi Pico W/Pico2 W用のドライバをビルドします

```
33 #  
34 # ボードタイプ  
35 #  
36 BOARDTYPE = PICO_W  
37
```



Pico/Pico2のビルド方法

- make時のオプション設定でPico/Pico2の切り替えが可能

make BOARDNO=0	BOARDNOは省略可能、Picoをビルド
make BOARDNO=1	Pico2:Cortex-M33をビルド
make BOARDNO=2	Pico2:RISC-Vをビルド

- make時のオプション設定が面倒な場合、Makefile.configにBOARDNO設定を追加することで変更が可能

```
1 #  
2 # ターゲット略称の設定  
3 #  
4 BOARDNO = 1
```

この行を追加
Pico2:Cortex-M33に設定

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

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

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/program

タスク生成

led_1task : 1タスクによるLED点滅プログラム

マルチタスクプログラミング

led_2task_for : 2タスクによるLED点滅プログラム(forループ版)
Makefileとコンフィギュレーションファイルのみ

割込みプログラム

time_int : 割込みタイマーによるLED点滅

sw_int : 割込みサービスルーチンによるLED点滅プログラム

cpu_lock CPUロック状態

排他制御

ex_none : 排他制御なし2タスク出力プログラム

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)

サンプル

cup_timer : カップ・ラーメン・タイマー

RTOSの基礎プログラミング実習

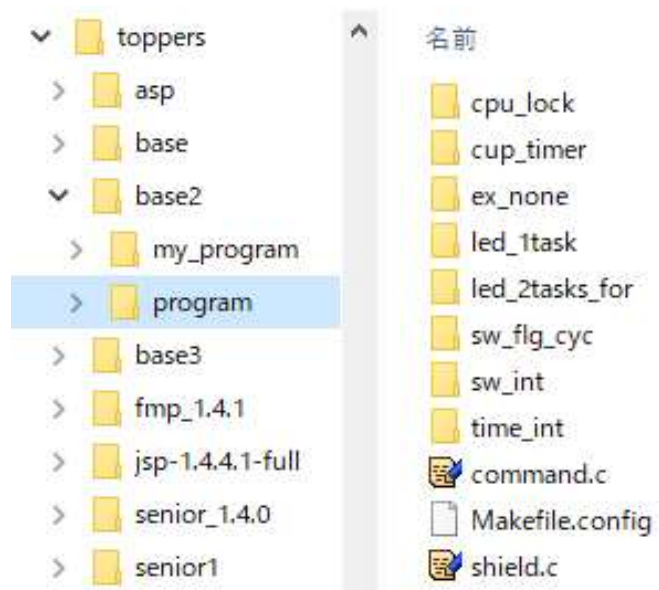
1. 概要
2. タスク生成方法
3. デバッグ方法
4. マルチタスクプログラミング
5. 周期ハンドラ
6. 割込みプログラム

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

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
 - pdic/rp2040またはpdic/rp2350ディレクトリとprogramディレクトリにのドライバを使用します
 - device.c device.h device.cfg
 - shield.c(教材シールド用のドライバ)
- LED関連
 - マクロ

```
#define LED01      (1<<0)
#define LED02      (1<<1)
#define LED03      (1<<2)
#define LED04      (1<<3)
#define LED_MASK (LED01|LED02|LED03|LED04)
```
 - 関数
 - void led_init(intptr_t exinf)
 - 初期化(引数は0とする)
 - __attribute__((weak)) void led_out(unsigned int led_data)
 - 引数にマクロで点灯パターンを指定

Weak属性:他に同一名の関数がリンクされた場合、そちらを優先してリンクする

led_1task 解説 : ITRON用のデバイスドライバ

- SILを用いてドライバを記載する

```
__attribute__((weak)) void
led_out(unsigned int led_data)
{
    const gpio_conf_t *pled = &led_confdata[0];
    uint32_t i;
    /* 設定値はLED接続ビット以外は変更しない */
    for(i = 0 ; i < NUM_LED ; pled++, i++){
        if((led_data & (1<<i)) != 0)
            sil_wrw_mem((uint32_t *) (TADR_SIO_BASE+TOFF_SIO_GPIO_OUT_SET), (1<<pled->pinpos));
        else
            sil_wrw_mem((uint32_t *) (TADR_SIO_BASE+TOFF_SIO_GPIO_OUT_CLR), (1<<pled->pinpos));
    }
}
```

Pico/Pico2上のLEDを制御する

- 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

led_1task 解説：ディップスイッチ操作関数群

- シールド上の2ピンのDIPSWを使用する。
- マクロ

#define DSW1	0x01
#define DSW2	0x02

- 関数
 - uint_t 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

led_1task 解説 : プッシュスイッチ操作関数群

- マクロ

```
#define PSW1      0x001
```

- 関数

- void switch_push_init(intptr_t exinf)
 - 初期化(引数を使用しない)
- void setup_sw_func(intptr_t exinf)
 - プッシュスイッチの割り込み関数を設定

- 割り込み設定

- 割り込み設定は静的APIで行う
- (他のRTOSとの大きな違い)

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

```
#include <itron.h>
```

```
#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);
```

```
extern void shield_init(intptr_t exinf);
```

```
extern void setup_sw2_func(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

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"
#include "led_1task.h"
```

```
void
led1_task(intptr_t exinf){
    shield_init(0);
    for (;;) {
        led_out(LED01);
        busy_wait();
        led_out(0x00);
        busy_wait();
    }
}
```

点灯

消灯

led_1task 解説 : Cソースファイル(shield.c)

- 教材シールドのデバイス制御を行う
- led_in / led_out / switch_dip_senseはdevice.cの記載のドライバより、優先してリンクされる

関数	機能
shield_init	教材シールドのデバイスの初期化を行う
led_in	教材シールドのLED接続ポートの読み出し
led_out	教材シールドのLED接続ポートの書き込み
setup_sw2_func	教材シールドのプッシュスイッチの割込み関数設定
switch_dip_sense	教材シールドのDIPSWの取り出し

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");
#if BOARDNO == 0
INCLUDE("pdic/rp2040/device.cfg");
#else
INCLUDE("pdic/rp2350/device.cfg");
#endif
INCLUDE("monitor/monotor.cfg");

#include "led_1task.h"
CRE_TSK(MAIN_TASK, { TA_ACT, 0, led_1task,
                    DEFAULT_PRIORITY, STACK_SIZE, NULL });
ATT_INI({ TA_NULL, 0, device_info_init });
```

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

起動時に実
行可能状態

タスクの
起動番地

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

led_1task.cfg

cfg1_out.srec

kernel_cfg.h



pass1

cfg1_out.map



pass2

kernel_cfg.c

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....

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

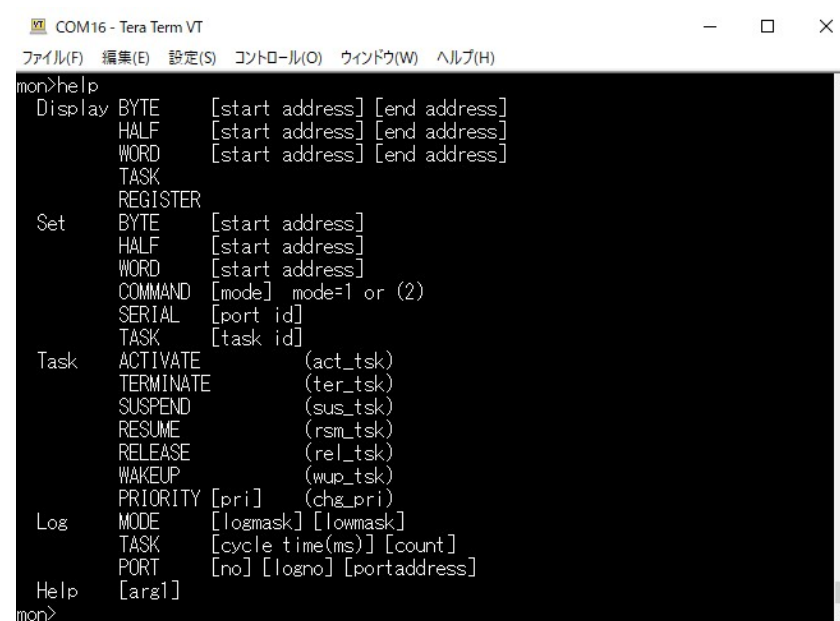
タスクモニタ：タスクモニタとは？

- 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付け, 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/monitor/monitor.cfg

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

演習：タスクモニタの使用法

- asp.srecをダウンロードして実行
- コンソールにコマンドを入力して使用する
- このプログラムを実行すると、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



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
mon>help
Display BYTE [start address] [end address]
        HALF [start address] [end address]
        WORD [start address] [end address]
        TASK
        REGISTER
Set      BYTE [start address]
        HALF [start address]
        WORD [start address]
        COMMAND [mode] mode=1 or (2)
        SERIAL [port id]
        TASK [task id]
Task     ACTIVATE (act_tsk)
        TERMINATE (ter_tsk)
        SUSPEND (sus_tsk)
        RESUME (rsm_tsk)
        RELEASE (rel_tsk)
        WAKEUP (wup_tsk)
        PRIORITY [pri] (chg_pri)
Log      MODE [logmask] [lowmask]
        TASK [cycle time(ms)] [count]
        PORT [no] [logno] [portaddress]
Help     [arg1]
mon>
```

演習：タスクモニタによるタスク操作

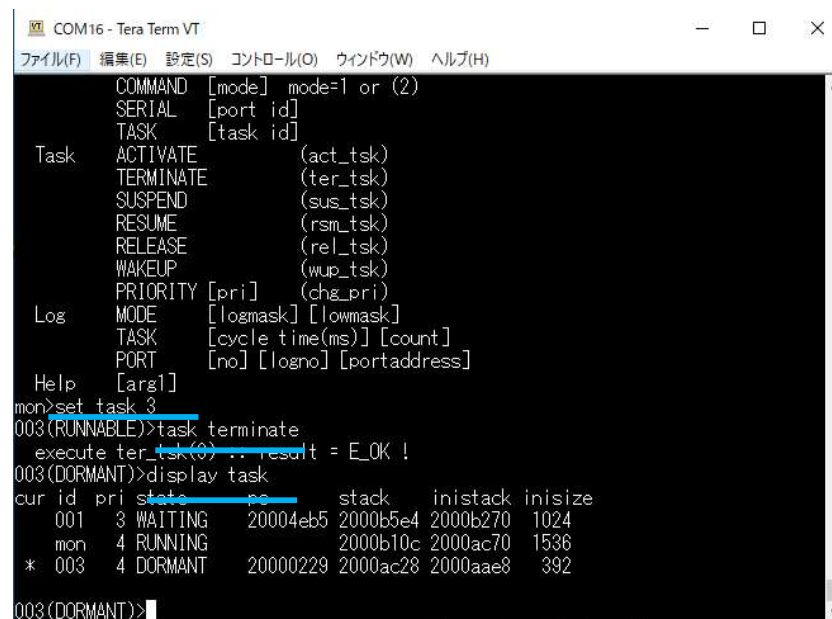
- タスクモニタではタスクに対して次の操作が可能
 - タスクの起動 (act_tsk) : タスクを実行可能状態に
 - タスクの終了 (ter_tsk) : タスクを休止状態に
 - タスクのサスペンド (sus_tsk) : タスクを強制待ち状態に
 - タスクのレジューム (rsm_tsk) : タスクを強制待ち状態から復帰
 - 優先度の変更 (chg_pri) : タスクの優先度を変更する
- タスクの操作する前に操作対象のタスクを指定する必要がある
 - タスクIDは kernel_cfg.h で定義されている値
 - 一度設定すると、それ以後のタスク操作は指定したタスクに対して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

```
mon> set task [タスクID]
```

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



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

COMMAND [mode] mode=1 or (2)
SERIAL [port id]
TASK [task id]

Task
  ACTIVATE (act_tsk)
  TERMINATE (ter_tsk)
  SUSPEND (sus_tsk)
  RESUME (rsm_tsk)
  RELEASE (rel_tsk)
  WAKEUP (wup_tsk)
  PRIORITY [pri] (chg_pri)
Log
  MODE [logmask] [lowmask]
  TASK [cycle time(ms)] [count]
  PORT [no] [logno] [portaddress]
Help [arg1]

mon>set task 3
003(RUNNABLE)>task terminate
execute ter_tsk(0) .. result = E_OK !
003(DORMANT)>display task
cur id pri state ps stack inistack inisize
001 3 WAITING 20004eb5 2000b5e4 2000b270 1024
mon 4 RUNNING 2000b10c 2000ac70 1536
* 003 4 DORMANT 20000229 2000ac28 2000aae8 392
003(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

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
 - タスクとした関数の先頭から実行される(休止状態から実行可能状態)

```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

WAKEUP (wup_tsk)
PRIORITY [pri] (chg_pri)
Log MODE [logmask] [lowmask]
TASK [cycle time(ms)] [count]
PORT [no] [logno] [portaddress]
Help [arg1]
mon>set task 3
003(RUNNABLE)>task terminate
execute ter_tsk(3) :: result = E_OK !
003(DORMANT)>display task
cur id pri state pc stack inistack inisize
001 3 WAITING 20004eb5 2000b5e4 2000b270 1024
mon 4 RUNNING 2000b10c 2000ac70 1536
* 003 4 DORMANT 20000229 2000ac28 2000aae8 392

003(DORMANT)>task activate
execute act_tsk(3) :: result = E_OK !
003(RUNNABLE)>display task
cur id pri state pc stack inistack inisize
001 3 WAITING 20004eb5 2000b5e4 2000b270 1024
mon 4 RUNNING 2000b10c 2000ac70 1536
* 003 8 RUNNABLE 20005061 2000ac28 2000aae8 392

003(RUNNABLE)>
```

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4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

```
mon 4 RUNNING 2000ad38 2000a680 2048
* 003 8 RUNNABLE 20004def 2000a62c 2000a4f8 392

003(RUNNABLE)>task priority 6
execute chg_pri(3, 6) :: result = E_OK !
003(RUNNABLE)>display task
cur id pri state pc stack inistack inisize
001 3 WAITING 20004c83 2000b1f0 2000ae80 1024
mon 4 RUNNING 2000ac3c 2000a680 2048
* 003 6 RUNNABLE 20004def 2000a62c 2000a4f8 392

003(RUNNABLE)>task priority 2
```

led1_taskの優先度を6に

コマンドを受付

led1_taskの優先度を2に

これ以上コマンドを受付られない

printf() デバッグ

- オーソ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が使えない箇所(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

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

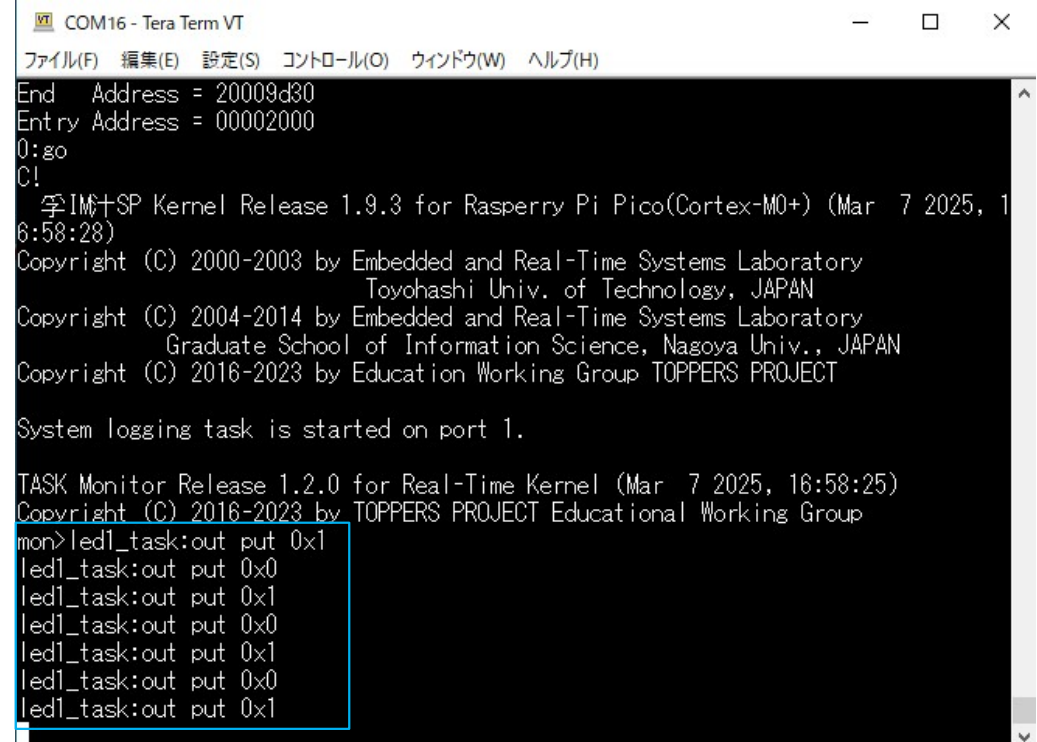
```
syslog_x(LOG_NOTICE, .....)
```

```
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, .....
```

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

- 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
- タスク関数名(led1_task)を表示するようコメントを作成

```
void  
led1_task(intptr_t exinf){  
    led_init(0);  
    for (;;) {  
        syslog_1(LOG_NOTICE,  
            "led1_task : out_put 0x%x", LED01);  
        led_out(LED01);  
        busy_wait();  
        syslog_1(LOG_NOTICE,  
            "led1_task : out_put 0x%x", 0x00);  
        led_out(0x00);  
        busy_wait();  
    }  
}
```



```
COM16 - Tera Term VT  
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)  
End Address = 20009d30  
Entry Address = 00002000  
0:go  
C!  
  孚IM+SP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  7 2025, 16:58:28)  
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory  
    Toyohashi Univ. of Technology, JAPAN  
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory  
    Graduate School of Information Science, Nagoya Univ., JAPAN  
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT  
  
System logging task is started on port 1.  
  
TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  7 2025, 16:58:25)  
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group  
mon>led1_task:out put 0x1  
led1_task:out put 0x0  
led1_task:out put 0x1  
led1_task:out put 0x0  
led1_task:out put 0x1  
led1_task:out put 0x0  
led1_task:out put 0x1
```

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

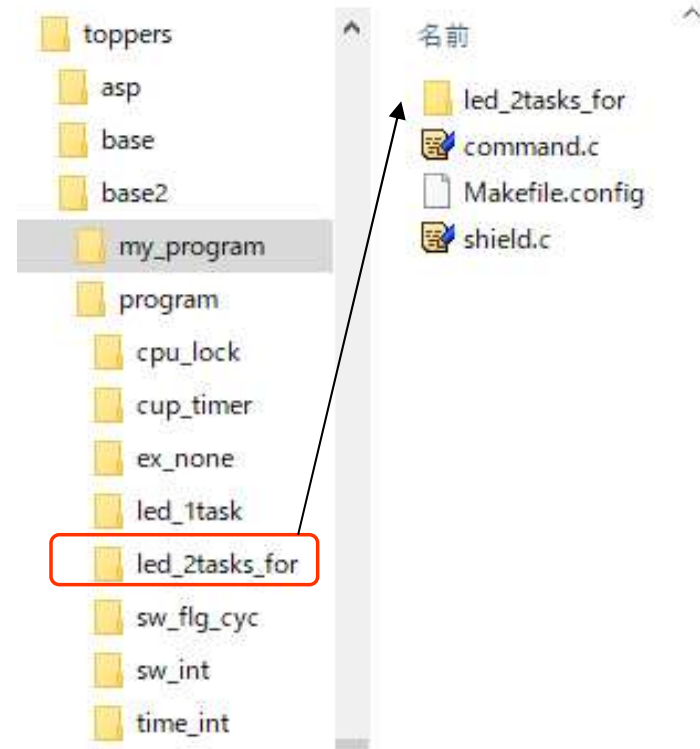
2タスクによるLED点滅プログラム : led_2tasks_for

led_1task を改造してLED2を点滅させるタスク(led2_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

開発・実行環境の使用方法:ビルド

- base2の下にmy_programディレクトリを作成
- program/led_2tasks_forをmy_programディレクトリにコピー
- program下の3つのファイルをmy_program/ にコピーする
 - Makefile.config
 - command.c
 - shield.c



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

```
#include <itron.h>

#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(intptr_t exinf);
extern void shield_init(intptr_t exinf);
extern void setup_sw2_func(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

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"
#include "device.h"
#include "led_2tasks.h"
```

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

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
- 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")
#if BOARDNO == 0
INCLUDE("pdic/rp2040/device.cfg");
#else
INCLUDE("pdic/rp2350/device.cfg");
#endif
INCLUDE("monitor/monitor.cfg")
#include "led_2tasks.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

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LED2を点滅するタスク(led2_task)を追加する
- 手順
 - LEDの状態は共有変数により保持する
 - led2_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
 - led2_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

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

• LED1点滅タスク

LEDのデータの
共有メモリ化を
行う

```
UW led_data= 0;
void
led1_task(intptr_t exinf){
    shield_init(0);
    for (;;) {
        led_data |= LED01;
        led_out(led_data);
        busy_wait();
        led_data &= ~LED01;
        led_out(led_data);
        busy_wait();
    }
}
```

• LED2点滅タスク

```
void
led2_task(intptr_t exinf){

    for (;;) {
        led_data |= LED02;
        led_out(led_data);
        busy_wait();
        led_data &= ~LED02;
        led_out(led_data);
        busy_wait();
    }
}
```

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 led2_task(intptr_t exinf);
extern void shield_init(intptr_t exinf);
extern void setup_sw2_func(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)

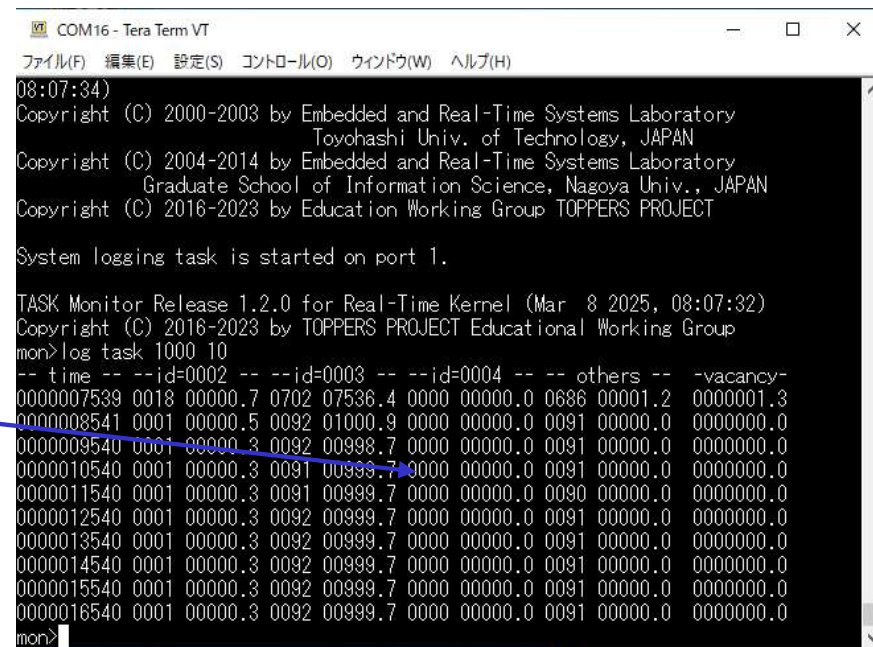
```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, led1_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(LED2_TASK, { TA_ACT, 0, led2_task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

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, led2_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 10コマンド
でタスクの実行状態を確認

task 4(led2_task)の実行
時間がゼロ



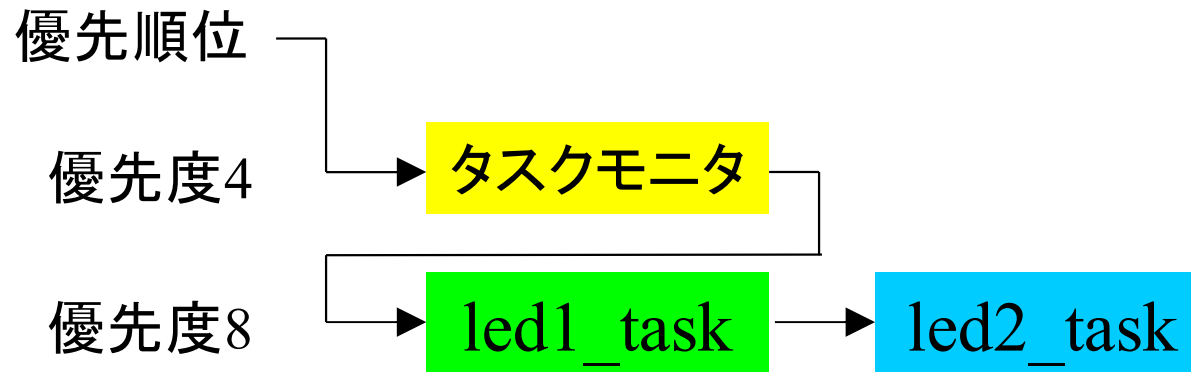
```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
08:07:34)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  8 2025, 08:07:32)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>log task 1000 10
-- time -- --id=0002 -- --id=0003 -- --id=0004 -- -- others -- -vacancy-
0000007539 0018 00000.7 0702 07536.4 0000 00000.0 0686 00001.2 0000001.3
0000008541 0001 00000.5 0092 01000.9 0000 00000.0 0091 00000.0 0000000.0
0000009540 0001 00000.3 0092 00998.7 0000 00000.0 0091 00000.0 0000000.0
0000010540 0001 00000.3 0091 00999.7 0000 00000.0 0091 00000.0 0000000.0
0000011540 0001 00000.3 0091 00999.7 0000 00000.0 0090 00000.0 0000000.0
0000012540 0001 00000.3 0092 00999.7 0000 00000.0 0091 00000.0 0000000.0
0000013540 0001 00000.3 0092 00999.7 0000 00000.0 0091 00000.0 0000000.0
0000014540 0001 00000.3 0092 00999.7 0000 00000.0 0091 00000.0 0000000.0
0000015540 0001 00000.3 0092 00999.7 0000 00000.0 0091 00000.0 0000000.0
0000016540 0001 00000.3 0092 00999.7 0000 00000.0 0091 00000.0 0000000.0
mon>
```

led_2tasks_for : タスクレベルの変更

- タスクモニタで led1_task と led2_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(led2_task)に変更
 - task priority 6 : タスク優先度を6に変更
- これによりLED2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, LED2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
 - led_2tasks_forをコピーしてled_2tasksに変更
- 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

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
- LED2点滅タスク

```
#define LED1_INTERVAL 1000
void
led1_task(intptr_t exinf){
    shield_init(0);
    for (;;) {
        led_data |= LED01;
        led_out(led_data);
        dly_tsk(LED1_INTERVAL);
        led_data &= ~LED01;
        led_out(led_data);
        dly_tsk(LED1_INTERVAL);
    }
}
```

```
#define LED2_INTERVAL 250
void
led2_task(intptr_t exinf){

    for (;;) {
        led_data |= LED02;
        led_out(led_data);
        dly_tsk(LED2_INTERVAL);
        led_data &= ~LED02;
        led_out(led_data);
        dly_tsk(LED2_INTERVAL);
    }
}
```

led_dataの
排他制御は
必要ない

初期化ルーチン

- LED接続ポートの初期化関数(`shield_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 `led2_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});
```

演習：2タスクによるLED点滅プログラム改(led_2tasks)

- 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
 - 初期化ルーチンの登録

```
ATT_INI({TA_NULL, 0, shield_init});
```

```
void  
led1_task(intptr_t exinf){  
    shield_init(0);  
    for (;;) {  
        led_data |= LED01;  
        led_out(led_data);  
        dly_tsk(LED1_INTERVAL);  
        led_data &= ~LED01;  
        led_out(led_data);  
        dly_tsk(LED1_INTERVAL);  
    }  
}
```

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 led2_task のコードを共有化する
- led_2tasksをコピーして、led_2tasks_shareを作成して変更
- 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{
    UW led_bit;
    W  interval;
}LED_TASK_INFO;

LED_TASK_INFO task_info[] = { {LED01, LED1_INTERVAL},
                               {LED02, LED2_INTERVAL}};
```

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){
    UW led_bit = task_info[(UW)exinf].led_bit;
    W  interval = task_info[(UW)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);
    }
}
```

led_2tasks_share : その他のファイル

- ヘッダーファイル (led_2tasks.h)

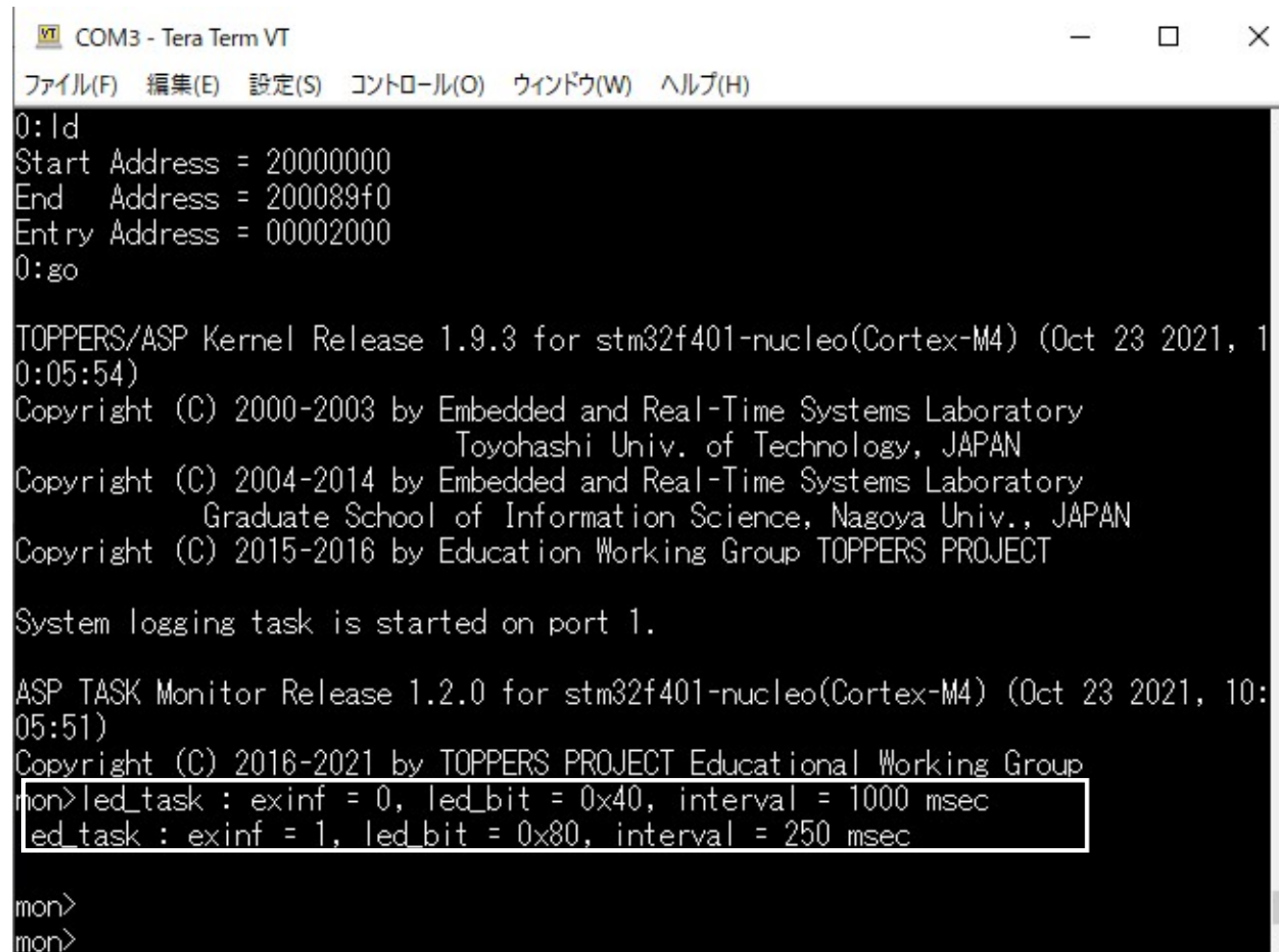
```
#ifndef TOPPERS_MACRO_ONLY
extern void led_task(intptr_t exinf);
extern void shield_init(intptr_t exinf);
extern void setup_sw2_func(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

- コンフィギュレーションファイル (led_2tasks.cfg)
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(LED2_TASK, { TA_ACT, 1, led_task,  
                    DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

led_2tasks_share : コンフィギュレーションファイル

- コンソール出力



```
COM3 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウインドウ(W) ヘルプ(H)
0:ld
Start Address = 20000000
End Address = 200089f0
Entry Address = 00002000
0:go

TOPPERS/ASP Kernel Release 1.9.3 for stm32f401-nucleo(Cortex-M4) (Oct 23 2021, 10:05:54)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
                        Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
                        Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2015-2016 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

ASP TASK Monitor Release 1.2.0 for stm32f401-nucleo(Cortex-M4) (Oct 23 2021, 10:05:51)
Copyright (C) 2016-2021 by TOPPERS PROJECT Educational Working Group
mon>led_task : exinf = 0, led_bit = 0x40, interval = 1000 msec
led_task : exinf = 1, led_bit = 0x80, interval = 250 msec

mon>
mon>
```

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, LED2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
 - LED2点滅タスク(led2_task)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

- 周期ハンドラはタスクと同様に関数として記述する
- LED2点滅周期ハンドラ

LEDのデータの
共有メモリ化を
行う

```
UW led_data= 0;

void
led2_cyc_handler(intptr_t exinf){
    if((led_data & LED02) == LED02){
        led_data &= ~LED02;
    }
    else{
        led_data |= LED02;
    }
    led_out(led_data);
}
```

led_cyc : Cソースファイル

- LED1点滅タスクを共有型に変更

```
void  
led1_task(intptr_t exinf){  
  
    shield_init();  
  
    for (;;) {  
        led_data |= LED01;  
        led_out(led_data);  
        busy_wait();  
        led_data &= ~LED01;  
        led_out(led_data);  
        busy_wait();  
    }  
}
```

led_cyc : その他のファイル

- ヘッダーファイル (led_1task.h)

```
#define LED2_INTERVAL 500
#ifndef TOPPERS_MACRO_ONLY
.....
extern void led2_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_1task.cfg)

```
CRE_CYC(LED2_CYC_HANDLER, {TA_STA,
                          0, led2_cyc_handler, LED2_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 inhattr, FP inthdr});
```

- Pico/Pico2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(target_config.hで定義)
- 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

割り込みプログラミング：割り込み要求ライン属性設定

- 割り込み属性の設定を行う
 - 割り込み属性を登録する静的API

ハンドラ番号

属性(実装定義)

割り込み優先度

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

- intatrは割り込み属性([TA_ENAINT] | [TA_EDGE]等)
- intpriは割り込み優先度を指定する
 - 優先度は以下の通り
 - Pico : -1 ~ -3
 - Pico2 : -1 ~ -15

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
 - μ ITRON4.0仕様準拠の割込みハンドラの登録を行う

割込み番号

割込みハンドラ

```
DEF_INH(INTNO inthno {ATR inhatr, FP inthdr});
```

割込みハンドラ属性

- inthno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

割込みによるLED点滅プログラム (time_int)

- 割込みによるLED点滅プログラムtime_intを検証する
- プログラムの説明
 - タイマー割込みを起動して、LED1を点滅させる
 - タイマーモジュールひとつで4つの割込み設定が可能
 - 割込み設定毎に、次回のタイムアップカウントを設定
 - TIMER0の0番割込みハンドラを起動する
 - 割込みハンドラ内で、LED点灯、消灯を反転して設定する
 - TIMER0の割込み属性と優先度を設定
 - INTATR_UTIMER 0U(属性)
 - INTPRI_UTIMER -2(優先度)
 - TIMER0の割込みハンドラ番号と割込み番号
 - INHNO_UTIMER0(割込みハンドラ番号)
 - INTNO_UTIMER0(割込み番号)

timer_int : Cソースファイル(time_int.c)

- TIMER0の0番の割込みハンドラ
 - 割込みハンドラはタスクと同様に関数として記述する
- TIMER_INTS(割込みステータス)を、TIMER_INTR(割込みレジスタ)に書くことで、割込み要因をクリアできる

```
void
timer_int(void)
{
    UW led = LED01;
    UW delay_time;
    UW ints = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTS));
    /*
     * 割込みクリア
     */
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTR), ints);
}
```

timer_int : Cソースファイル(time_int.c)

- TIMER_ALARMに次の割込みカウンタを書き込み、割込み要因を設定
- LED01を反転設定する

```
/*
 * リロード時間設定
 */
delay_time = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE+
    TOFF_TIMER_TIMERAWL)) + WRAP_TIME0;
sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_ALARM0
    +TIMER0*4), delay_time);
if ( (led_data & led) == led ) {
    led_data &= ~led;
}
else {
    led_data |= led;
}
led_out(led_data);
syslog_1(LOG_NOTICE,"timer_int : led_data 0x%x",led_data);
}
```

timer_int : Cソースファイル(time_int.c)

- メイン・ルーチンでTIMER0の0番割込みの初期化を行う
- その後、スリープ

```
/*
 * メインタスク
 */
void
main_task(intptr_t exinf)
{
    UW delay_time;
    UW inte = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTE));

    inte |= (1<<TIMER0);
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTE), inte);
    delay_time = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE
        +TOFF_TIMER_TIMERAWL)) + WRAP_TIME0;
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_ALARM0
        +TIMER0*4), delay_time);
    slp_tsk();
}
```

time_int : その他のファイル

- ヘッダーファイル(time_int.h)

```
#if BOARDNO == 0
#define IRQ_VECTOR_TIMER IRQ_VECTOR_TIMER0
#else
#define IRQ_VECTOR_TIMER IRQ_VECTOR_TIMER0_0
#endif
#define TIMER0      0
#define INHNO_UTIMER0 (IRQ_VECTOR_TIMER+TIMER0)
#define INTNO_UTIMER0 (IRQ_VECTOR_TIMER+TIMER0)
#define INTPRI_UTIMER (-2)      /* 割込み優先度 */
#define INTATR_UTIMER  0U      /* 割込み属性 */

#define WRAP_TIME0 (250*1000)
#ifndef TOPPERS_MACRO_ONLY
extern void main_task(intptr_t exinf);
extern void time_int(void);
:
#endif /* TOPPERS_MACRO_ONLY */
```

PicoとPico2で
TIMER0の割込
み名称と番号が
異なる

割込み周期クロックの定義

time_int : その他のファイル

- コンフィギュレーションファイル (time_int.cfg)

```
#include "device.h"  
#include "time_int.h"
```

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

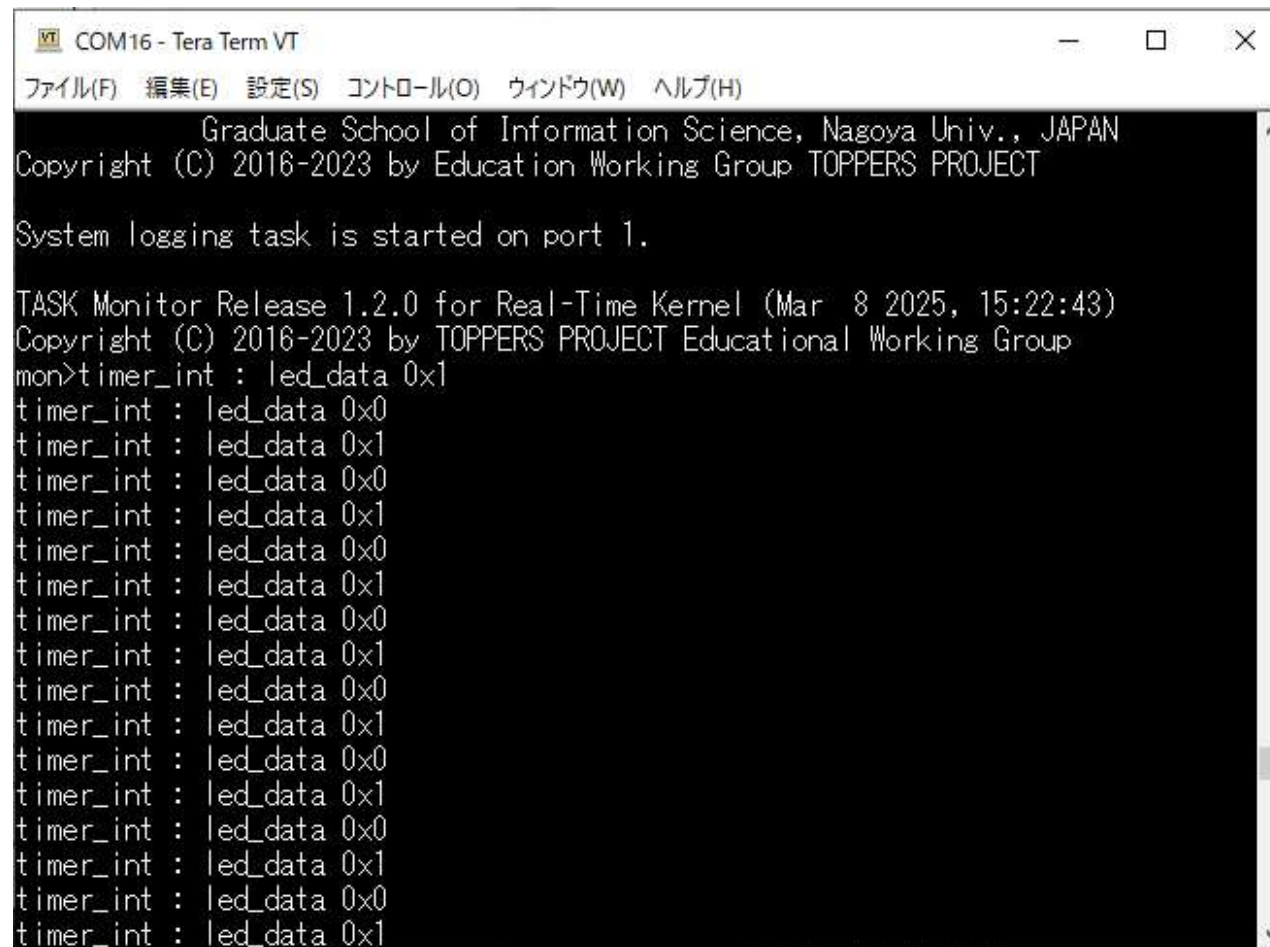
```
ATT_INI({ TA_NULL, 0, shield_init });  
ATT_INI({ TA_NULL, 0, device_info_init });
```

TIMER0割込みの定義

```
DEF_INH(INHNO_UTIMER0, { TA_NULL, timer_int });  
CFG_INT(INTNO_UTIMER0, { TA_ENAINT|INTATR_UTIMER,  
    INTPRI_UTIMER });
```

演習: time_intをビルド実行する

- time_intをビルド、ダウンロードしてLED1が250ms毎に点滅することを確認する



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT
System logging task is started on port 1.
TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar 8 2025, 15:22:43)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
timer_int : led_data 0x0
timer_int : led_data 0x1
```

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
μITRON4.0仕様準拠の静的APIを用意している

割り込みサービス
ルーチン属性

割り込み番号

割り込みサービス
ルーチン優先度

```
ATT_ISR({ATR isratr, intptr_t exinf, INTNO inhno, ISR isr, PRI isrpri});
```

サービスルーチン
拡張情報

割り込みサービスルー
チンの起動番地

- 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：割込みによるLED点滅プログラム(time_int2)

- time_int を割込みサービスルーチンを用いて記述する
割込みサービスルーチンは同一属性の割込みを
共通化できる
- 手順
 - time_intをベースにする
 - time_intをmy_program上でコピーして、名前を
time_int2にして演習を行ってください

time_int2 : Cソースファイル(time_int.c)

- TIMER0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TIMER0を引数として渡
- インクルードファイル(time_int.h)

```
extern void main_task(intptr_t exinf);  
extern void timer_isr(intptr_t exinf);
```

- コンフィギュレーションファイル(time_int.cfg)

```
ATT_ISR({TA_NULL, TIMER0, INTNO_UTIMER0, timer_isr, 1 });  
CFG_INT(INTNO_UTIMER0, {TA_ENAINT|INTATR_UTIMER, INTPRI_UTIMER});
```

time_int2: Cソースファイル(time_int.c)

- 割込みサビスルーチンを用いれば、タイマー割込み値が複数ある場合、引数にて適切にタイマー設定を行うことができる

```
void
timer_isr(intptr_t exinf){
    UW no = (UW)exinf;
    UW led = LED01;
    UW delay_time;
    UW ints = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTS))
        & (1<<no);

    /*
     * 割込みクリア
     */
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTR), ints);
    /*
     * リロード時間設定
     */
    delay_time = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE
        +TOFF_TIMER_TIMERAWL)) + WRAP_TIME0;
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_ALARM0
        +no*4), delay_time);
```

演習：2つ割込みによるLED点滅プログラム

- TIMER0の1番割込みを発生させ、LED2を点滅させる
- 割込みサービスは、引数により共通化して2つのLEDに対応するように改造する
- LED2点滅は500msとする
- 手順
 - time_int2をベースに改造をおこなう
 - time_int.hに1番割込みの定義を追加する
 - time_int.cfgに1番割込みの割込みサービスルーチンの静的APIを追加する
 - time_int.cの割込みサービスルーチンを2つの割込みに対応するように修正する
 - time_int.cのメイン・タスクに1番割込みの初期化を追加する

time_int2 : ヘッダーファイル

- time_int.hにTIMER0の1番割込みの定義を追加

```
#define TIMER0      0
#define TIMER1      1
#define INHNO_UTIMER0 (IRQ_VECTOR_TIMER+TIMER0)
#define INTNO_UTIMER0 (IRQ_VECTOR_TIMER+TIMER0)
#define INHNO_UTIMER1 (IRQ_VECTOR_TIMER+TIMER1)
#define INTNO_UTIMER1 (IRQ_VECTOR_TIMER+TIMER1)
#define INTPRI_UTIMER (-2)      /* 割込み優先度 */
#define INTATR_UTIMER  0U       /* 割込み属性 */

#define WRAP_TIME0 (250*1000)
#define WRAP_TIME1 (500*1000)
```

time_int2 : コンフィギュレーションファイル

- time_int2.cfgにTIMER0の1番割込みの割込みサービスルーチンを追加

```
ATT_ISR({ TA_NULL, TIMER0, INTNO_UTIMER0, timer_isr, 1 });  
CFG_INT(INTNO_UTIMER0, { TA_ENAINT|INTATR_UTIMER,  
    INTPRI_UTIMER });  
ATT_ISR({ TA_NULL, TIMER1, INTNO_UTIMER1, timer_isr, 1 });  
CFG_INT(INTNO_UTIMER1, { TA_ENAINT|INTATR_UTIMER,  
    INTPRI_UTIMER });
```

time_int2: Cソースファイル(time_int.c)

- led_2task2_shareで使した構造体を追加し、timer_intを改造する

```
typedef struct{
    UW led_bit;
    W  interval;
}TIMER_INT_INFO;
```

```
TIMER_INT_INFO timer_info[] ={{LED01, WRAP_TIME0},
                                {LED02, WRAP_TIME1}};
```

```
void
timer_isr(intptr_t exinf){
    UW no = (UW)exinf;
    UW led = timer_info[no].led_bit;
    W  interval = timer_info[no].interval;
    UW delay_time;
    UW ints = sil_rew_mem((uint32_t *) (TADR_TIMER_BASE+TOFF_TIMER_INTS))
                & (1<<no);
```

time_int2: Cソースファイル(time_int.c)

- WRAP_TIME0をintervalに書き換えれば、割込みタイミングを変更できる

```
/*
 * 割込みクリア
 */
sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTR), ints);
/*
 * リロード時間設定
 */
delay_time = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE
                                     +TOFF_TIMER_TIMERAWL)) + interval;
sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_ALARM0
                         +no*4), delay_time);
:
led_out(led_data);
syslog_2(LOG_NOTICE,"timer_int(%d) : led_data 0x%x", no, led_data);
}
```

timer_int2 : Cソースファイル(time_int.c)

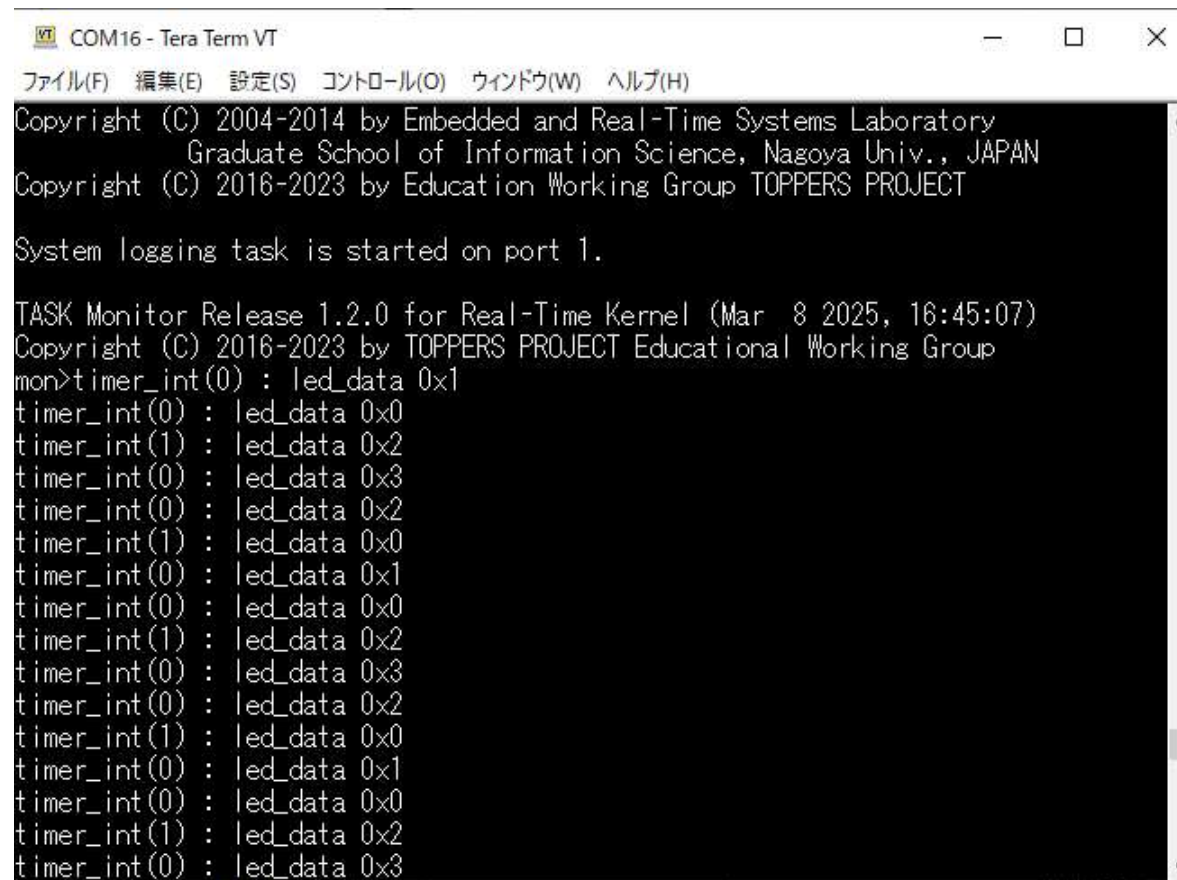
- メイン・ルーチンでTIMER0の1番割込みの初期化を追加

```
void
main_task(intptr_t exinf)
{
    UW delay_time;
    UW inte = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTE));

    inte |= (1<<TIMER0) | (1<<TIMER1);
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_INTE), inte);
    delay_time = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE
        +TOFF_TIMER_TIMERAWL)) + WRAP_TIME0;
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_ALARM0
        +TIMER0*4), delay_time);
    delay_time = sil_rew_mem((uint32_t*)(TADR_TIMER_BASE
        +TOFF_TIMER_TIMERAWL)) + WRAP_TIME1;
    sil_wrw_mem((uint32_t*)(TADR_TIMER_BASE+TOFF_TIMER_ALARM0
        +TIMER1*4), delay_time);
    slp_tsk();
}
```

演習: time_intをビルド実行する

- time_intをビルド、ダウンロードしてLED1が240ms毎に、LED2が500ms毎に点滅することを確認する



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar 8 2025, 16:45:07)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>timer_int(0) : led_data 0x1
timer_int(0) : led_data 0x0
timer_int(1) : led_data 0x2
timer_int(0) : led_data 0x3
timer_int(0) : led_data 0x2
timer_int(1) : led_data 0x0
timer_int(0) : led_data 0x1
timer_int(0) : led_data 0x0
timer_int(1) : led_data 0x2
timer_int(0) : led_data 0x3
timer_int(0) : led_data 0x2
timer_int(1) : led_data 0x0
timer_int(0) : led_data 0x1
timer_int(0) : led_data 0x0
timer_int(1) : led_data 0x2
timer_int(0) : led_data 0x3
```

割込みによるLED点滅プログラム (sw_int)

- 割込みによるLED点滅プログラムsw_intを使って検証
- プログラムの説明
 - プッシュスイッチの割込みは割込みハンドラを使用して作成している
 - プッシュスイッチの初期化はshield_initで行っている
 - 割込みをハンドリングするには、setup_sw2_funcを使って、割込みのコールバック・ルーチンを登録する
 - コンフィギュレーションファイル(sw_int.cfg)にコールバック・ルーチン(sw_int)は登録済み

```
ATT_INI({ TA_NULL, 0, shield_init });  
ATT_INI({ TA_NULL, sw_int, setup_sw2_func });  
ATT_INI({ TA_NULL, 0, device_info_init });
```

sw_int : Cソースファイル (sw_int.c)

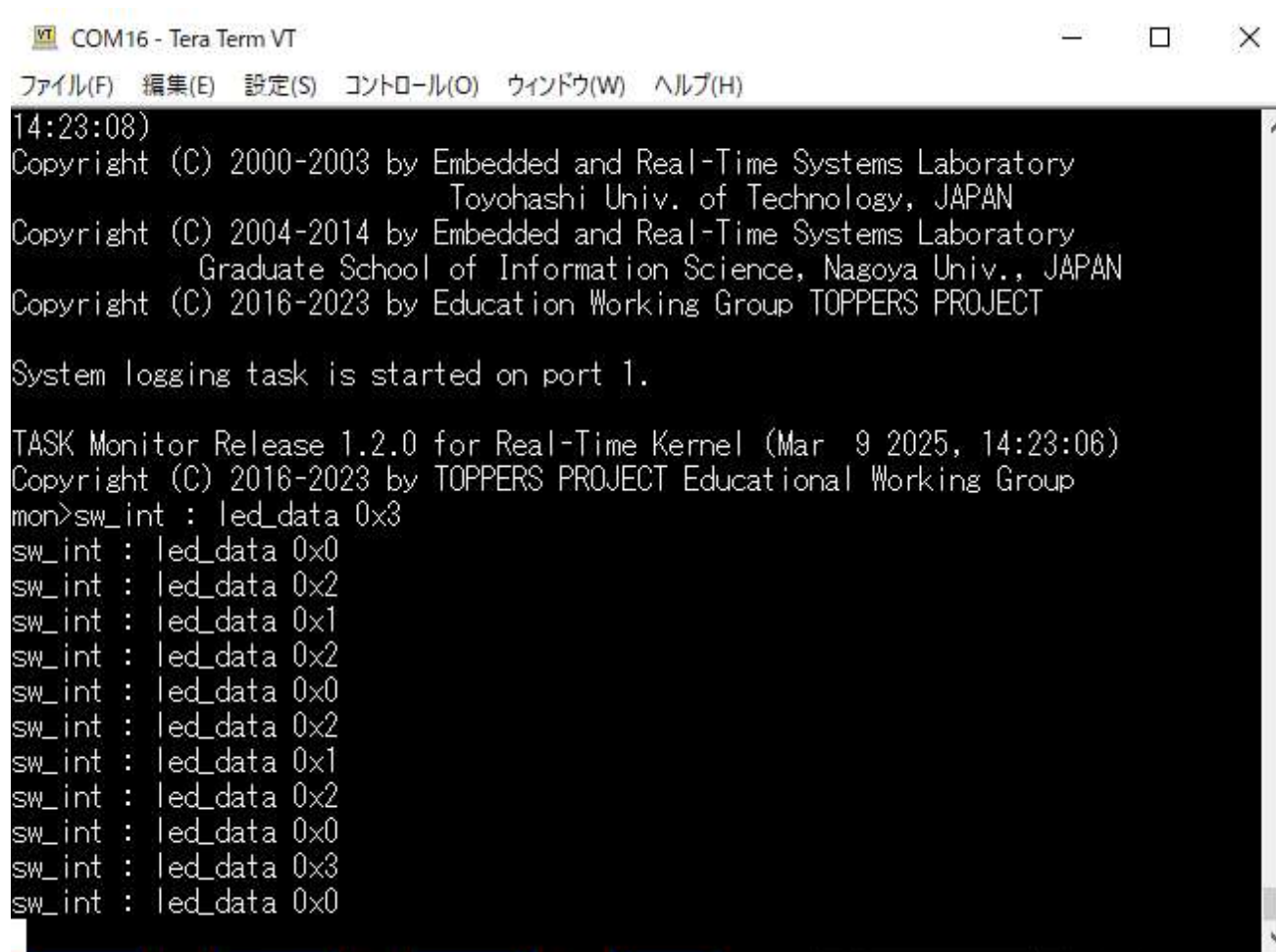
- プッシュスイッチのコールバック・ルーチン

```
void
sw_int(void){
    UW = (unsigned int)LED02;
    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);
}
```

演習: sw_intをビルド実行する

- sw_intをビルド、ダウンロードしてプッシュスイッチ押下で割込みが発生し、LED2の発光が変わることを確認しよう



```
VT COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
14:23:08)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
                        Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
                        Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 14:23:06)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>sw_int : led_data 0x3
sw_int : led_data 0x0
sw_int : led_data 0x2
sw_int : led_data 0x1
sw_int : led_data 0x2
sw_int : led_data 0x0
sw_int : led_data 0x2
sw_int : led_data 0x1
sw_int : led_data 0x2
sw_int : led_data 0x0
sw_int : led_data 0x3
sw_int : led_data 0x0
```

演習：割込みハンドラからのAPI呼び出し(`sw_int_api`)

- 割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
- `sw_int`をベースに作り変えます
- `sw_int`を`my_program`上でコピーして、名前を`sw_int_api`に変更
- プログラム仕様
 - 割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
 - プッシュスイッチ割込みコールバック・ルーチン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

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){
    W i;
    syslog(LOG_NOTICE,"led1_task : start!");
    for (i = 0; i < 4; i++) {
        if((led_data & LED01) == LED01){
            led_data &= ~LED01;
        } else {
            led_data |= LED01;
        }
        led_out(led_data);
        dly_tsk(LED1_INTERVAL);
    }
    syslog(LOG_NOTICE,"led1_task : done!");
    ext_tsk(); ←μITRON仕様書 : P82(PDF P100)
}
```

sw_int_api : 割り込みコールバック関数

- SW1割り込みコールバック関数

```
void  
sw_int(void){  
    switch_push_clear();  
    syslog(LOG_NOTICE,"sw1_int: start!");  
    SVC_PERROR(act_tsk(MAIN_TASK));  
}
```

- 割り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_PERROR()マクロにより実現

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_PERROR(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_PERROR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

sw_int_api : その他の修正

- インターバル設定を1000から500へ

```
#define LED1_INTEVAL 500
```

- コンフィギュレーションファイル (sw_int.cfg)
 - SW1の割込みコールバック関数は sw_int.h で定義済み

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

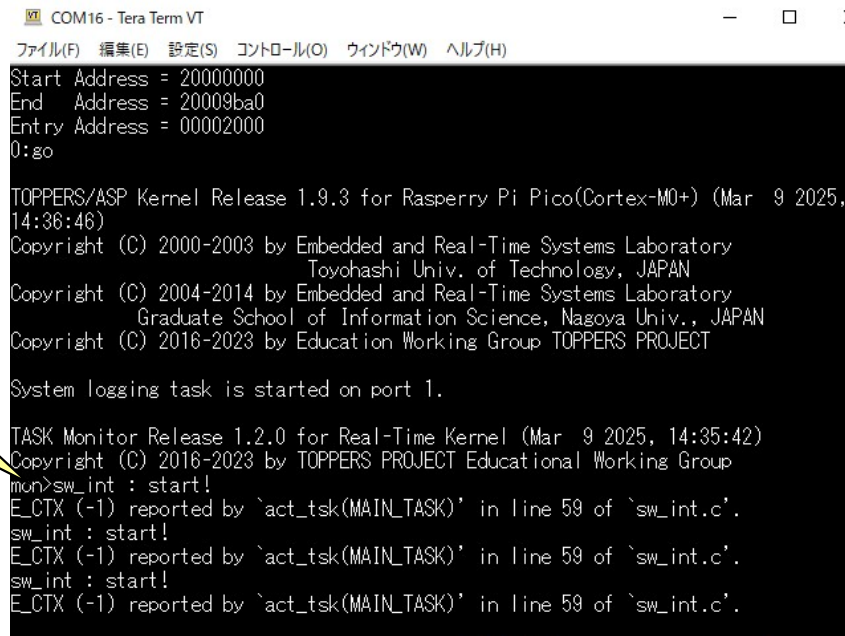
```
ATT_INI({ TA_NULL, 0, shield_init });  
ATT_INI({ TA_NULL, sw_int, setup_sw2_func });  
ATT_INI({ TA_NULL, 0, device_info_init });
```

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  
sw1_int(void){  
    switch_push_clear();  
    syslog(LOG_NOTICE,"sw1_int : start!");  
    SVC_PERROR(act_tsk(MAIN_TASK));  
}
```

プッシュスイッチをプッシュ



```
COM16 - Tera Term VT  
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)  
Start Address = 20000000  
End Address = 20009ba0  
Entry Address = 00002000  
0:go  
  
TOPPERS/ASP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  9 2025, 14:36:46)  
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory  
Toyohashi Univ. of Technology, JAPAN  
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory  
Graduate School of Information Science, Nagoya Univ., JAPAN  
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT  
  
System logging task is started on port 1.  
  
TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 14:35:42)  
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group  
mon>sw_int : start!  
E_CTX (-1) reported by 'act_tsk(MAIN_TASK)' in line 59 of 'sw_int.c'.  
sw_int : start!  
E_CTX (-1) reported by 'act_tsk(MAIN_TASK)' in line 59 of 'sw_int.c'.  
sw_int : start!  
E_CTX (-1) reported by 'act_tsk(MAIN_TASK)' in line 59 of 'sw_int.c'.
```

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
- プッシュスイッチ割込みコールバックの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プッシュスイッチ割込みを入れて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

```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
0:ld
Start Address = 20000000
End Address = 20009c18
Entry Address = 00002000
版!o
  学IM+SP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  9 2025
14:42:03)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT
System logging task is started on port 1.
TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 14:35:42)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>sw_int : start!
led1_task:start!
sw_int : start!
led1_task:done!
led1_task:start!
led1_task:done!
```

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

sw_int_api : キューイング回数

- SW1を連続して3回押して起動要求を3回発行する
- SW1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

```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Entry Address = 00002000
0xgo
TOPPERS/ASP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  9 2025,
14:42:03)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT
System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 14:35:42)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>sw_int : start!
led1_task:start!
sw_int : start!
sw_int : start!
E_QOVR (-1) reported by `iact_tsk(MAIN_TASK)` in line 59 of `sw_int.c`.
led1_task:done!
led1_task:start!
led1_task:done!
```

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、プッシュスイッチ割込みコールバック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
- sw_int_apiをmy_program上でコピーして名前をsw_int_api2に変更して作成する

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

sw_int_api2 : Cソースファイル(sw_int.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

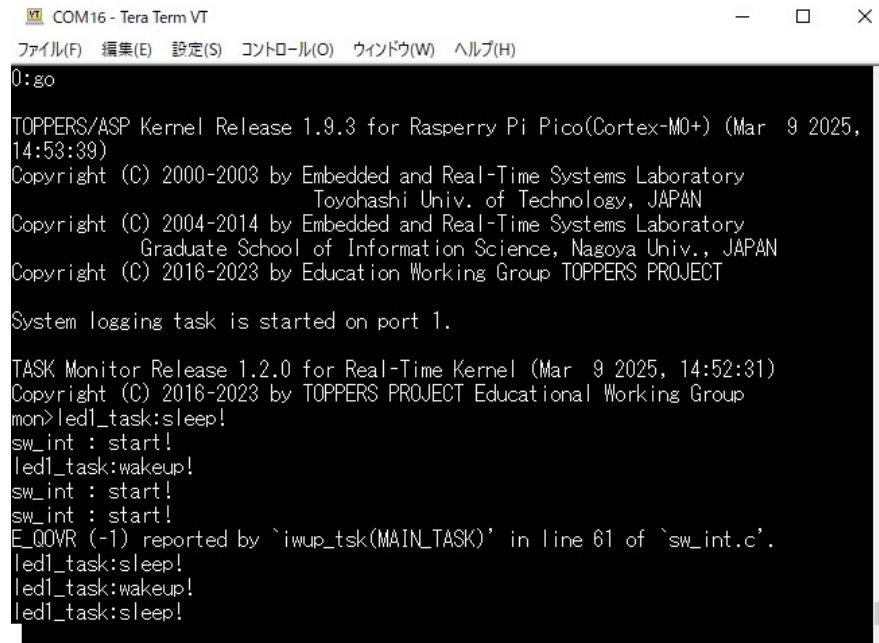
```
void  
led1_task(intptr_t exinf){  
    W i;  
    for (;;) {  
        syslog(LOG_NOTICE,"led1_task : sleep!");  
        slp_tsk();  
        syslog(LOG_NOTICE,"led1_task : wakeup!");  
        for (i = 0; i < 4; i++) {  
            if ((led_data & LED01) == LED01) {  
                led_data &= ~LED01;  
            } else {  
                led_data |= LED01;  
            }  
            led_out(led_data);  
            dly_tsk(LED1_INTERVAL);  
        }  
    }  
}
```

sw_int_api2 : 割込みコールバック・ルーチン

- iwup_tsk() により led1_task を起床させる

```
void  
sw_int(void){  
    switch_push_clear();  
    syslog(LOG_NOTICE,"sw_int : start!");  
    SVC_PERROR(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



The screenshot shows a terminal window titled "COM16 - Tera Term VT". The output text is as follows:

```
TOPPERS/ASP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  9 2025, 14:53:39)  
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory  
                                Toyohashi Univ. of Technology, JAPAN  
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory  
                                Graduate School of Information Science, Nagoya Univ., JAPAN  
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT  
  
System logging task is started on port 1.  
  
TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 14:52:31)  
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group  
mon>led1_task:sleep!  
sw_int : start!  
led1_task:wakeup!  
sw_int : start!  
sw_int : start!  
E_QOVR (-1) reported by `iwup_tsk(MAIN_TASK)' in line 61 of `sw_int.c'.  
led1_task:sleep!  
led1_task:wakeup!  
led1_task:sleep!
```

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
 - CPUロック状態の解除

`loc_cpu()`

`unl_cpu()`

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秒単位に割込み禁止状態と割込み許可状態を繰り返す
 - 割込み禁止状態では「INT DISABLE !」を表示:LED1オン
 - 割込み許可状態では「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
 - プッシュスイッチ割込みによりコールバック(sw_int)を実行する
 - sw_int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込みはCPUロック解除後にどうなるか

cpu_lock : タスク(dip_task)

- 5秒毎に割込み禁止と許可を繰り返すプログラムを作成
- fput_stringはASPカーネル内のポーリング・シリアル出力関数:target_fput_logを用いた文字列出力関数で割込み禁止でも、シリアル出力できる

```
void dip_task(intptr_t exinf){
    dly_tsk(500);
    for(;;){
        fput_string( "INT_DISABLE !\n");
        led_out(LED01);
        loc_cpu();
        int_delay(D5SEC);
        led_out(0);
        unl_cpu();
        fput_string("INT_ENABLE !\n");
        int_delay(D5SEC);
    }
}
```

INT_DISABLE !を表示
5秒間割込み禁止

INT_ENABLE !を表示
5秒間割込み許可

cpu_lock : 割込みコールバック・ルーチン

- 起動するとログを出力

```
void  
sw_int(void){  
    syslog_0(LOG_NOTICE,"sw_int : start!");  
}
```

cpu_lock : コンフィギュレーションファイル

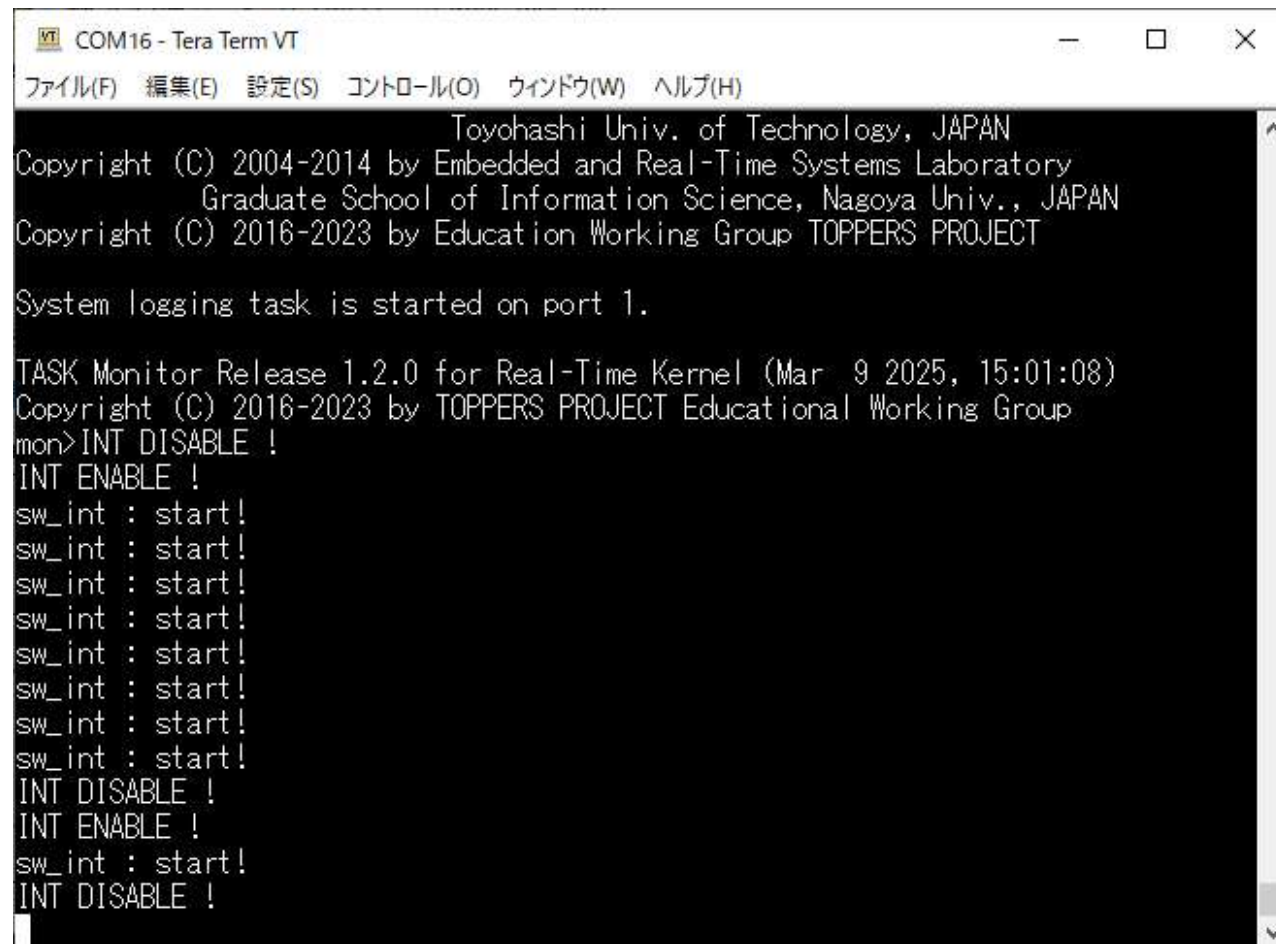
- タスク生成と割込みハンドラの登録

```
CRE_TSK(MAIN_TASK, { TA_ACT, (VP_INT) 0, dip_task,  
                    DEFAULT_MAIN_PRIORITY,  
                    STACK_SIZE, NULL });  
ATT_INI({ TA_NULL, 0, shield_init });  
ATT_INI({ TA_NULL, sw_int, setup_sw2_func });  
ATT_INI({ TA_NULL, 0, device_info_init });
```

- 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
shield_initにて行います

割込み禁止中は、割込み処理

- 「INT_DISABLE!」中にプッシュスイッチを押しても「sw_int : start!」は表示されない



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar 9 2025, 15:01:08)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>INT DISABLE !
INT DISABLE !
sw_int : start!
sw_int : start!
sw_int : start!
sw_int : start!
sw_int : start!
sw_int : start!
sw_int : start!
sw_int : start!
INT DISABLE !
INT ENABLE !
sw_int : start!
INT DISABLE !
```

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、闇雲な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

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

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
 - イベントフ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

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

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);
    }
}
```

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 (250*1000) /* 250ms */
busy_wait_inc(int e){
    long i;
    e++;
    for(i = 0; i < DELAY_LOOP; i++){
        sil_dly_nse(1000);
    }
    return e;
}
```

- タスクは関数 task のタスクを2個作成

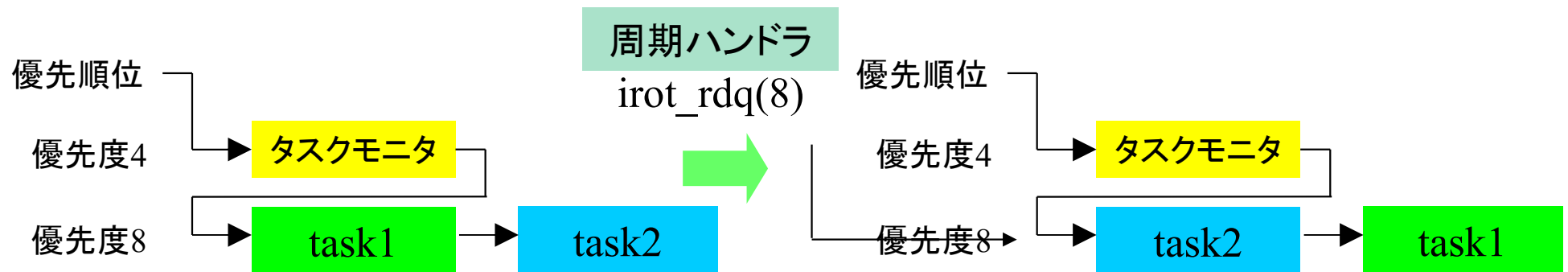
```
CRE_TSK(MAIN_TASK, { TA_ACT, 1, task,
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_ACT, 2, task,
                DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

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

優先度8

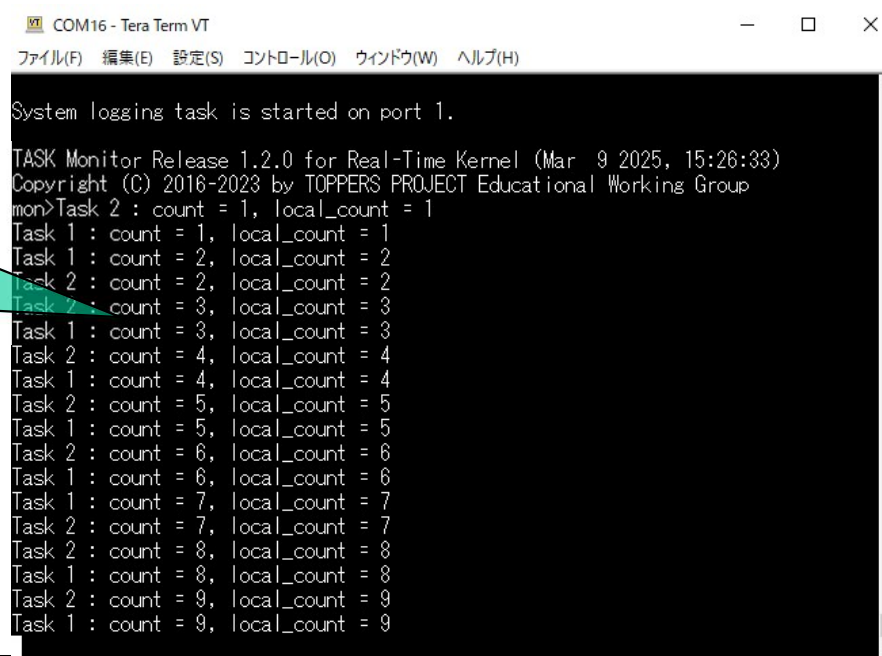
```
void  
rot_cyc_handler(void){  
    SVC_ERROR(irot_rdq(DEFAULT_PRIORITY));  
}
```



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

countが正しく
インクリメント
されない



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 15:26:33)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>Task 2 : count = 1, local_count = 1
Task 1 : count = 1, local_count = 1
Task 1 : count = 2, local_count = 2
Task 2 : 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
Task 2 : count = 9, local_count = 9
Task 1 : count = 9, local_count = 9
```

演習：セマフォによる排他制御プログラム(ex_sem)

- ex_noneを改造してセマフォによる排他制御を行うプログラムを作成する
- 作成方法
 - ex_noneをmy_programにコピーして名前をex_semに変更
 - ex_sem内の以下のプログラムをリネーム
 - ex_none.c → ex_sem.c
 - ex_none.cfg → ex_sem.cfg
 - ex_none.h → ex_sem.h
 - Makefileの166行目ex_noneをex_semに書き換え
 - ex_sem.cfgの15行目ex_noneをex_semに書き換え
 - ex_sem.cの10行目ex_none.hをex_sem.hに書き換え
- ex_semディレクトリ上でmakeを行って確認

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象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
```

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){
    W 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 });
```

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

```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 15:35:33)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>Task 2 : count = 1, local_count = 1
Task 1 : count = 3, local_count = 1
Task 2 : count = 4, local_count = 2
Task 1 : count = 4, local_count = 2
Task 2 : count = 6, local_count = 3
Task 1 : count = 6, local_count = 3
Task 2 : count = 8, local_count = 4
Task 1 : count = 9, local_count = 4
Task 2 : count = 9, local_count = 5
Task 1 : count = 11, local_count = 5
Task 2 : count = 11, local_count = 6
Task 1 : count = 13, local_count = 6
Task 2 : count = 14, local_count = 7
Task 1 : count = 14, local_count = 7
Task 2 : count = 16, local_count = 8
```

- 他の方法
 - 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?

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

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

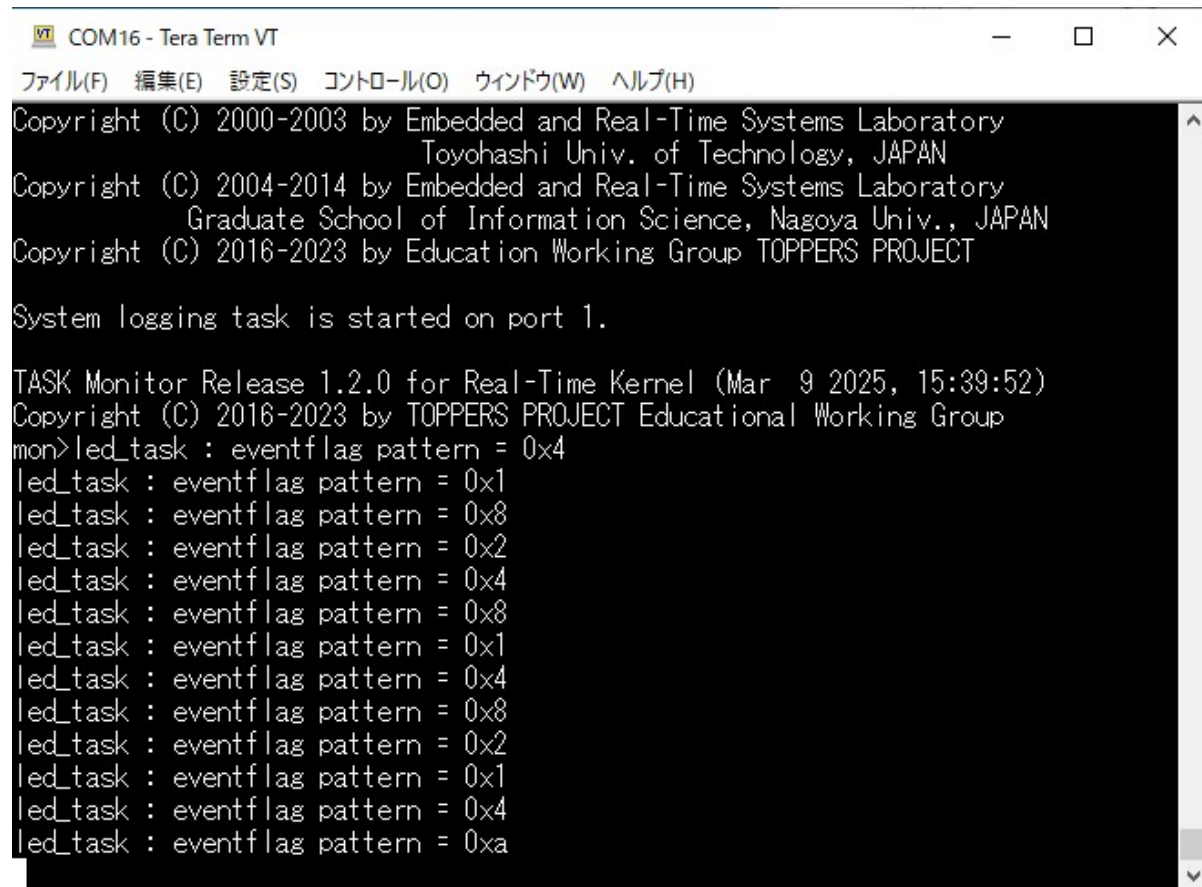
演習：イベントフラグによる通知プログラム (sw_flg_cyc)

- 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
 - DIPSW1,2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

演習：イベントフラグによる通知プログラム

- program上のsw_flg_cycをビルドする
- asp.srecを、ROMモニタのldコマンドでダウンロード
- goコマンドで実行する

DIP SWを操作する
とイベントフラグのロ
グを表示する



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT
System logging task is started on port 1.
TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar  9 2025, 15:39:52)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>led_task : eventflag pattern = 0x4
led_task : eventflag pattern = 0x1
led_task : eventflag pattern = 0x8
led_task : eventflag pattern = 0x2
led_task : eventflag pattern = 0x4
led_task : eventflag pattern = 0x8
led_task : eventflag pattern = 0x1
led_task : eventflag pattern = 0x4
led_task : eventflag pattern = 0x8
led_task : eventflag pattern = 0x2
led_task : eventflag pattern = 0x1
led_task : eventflag pattern = 0x4
led_task : eventflag pattern = 0xa
```

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 wmode, 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 )
```

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 LED_EV_MASK 0x000f
```

- 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

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

```
UW led_data;  
UW sw_state;  
void  
state_init(intptr_t exinf){  
    led_data = 0;  
    sw_state = switch_dip_sense();  
    if ((sw_state & DSW1) == DSW1) {  
        led_data |= LED01;  
    }  
    if ((sw_state & DSW2) == DSW2) {  
        led_data |= LED02;  
    }  
  
    led_out(led_data);  
}
```

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_ERROR(iset_flg(SW_FLG, flgptn));
        sw_state = tmp;
    }
}
```

- 状態に変更があればフラグで通知

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 |= LED01;  
        }  
        if ((flgptn & LED1_OFF) == LED1_OFF) {  
            led_data &= ~LED01;  
        }  
        ... 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

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

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

- プログラム実行
 - DIPSWの状態が変更されると
 - イベントフラグで通知される

演習 : イベントフラグによる通知プログラム

- `sw_flg_cyc`を改造してタスクからイベントフラグを通知するプログラムを作成する
- 作成方法
 - `sw_flg_cyc`を`my_program`にコピーして名前を`sw_flg_tsk`に変更
 - `sw_flg_tsk`内の以下のプログラムをリネーム
 - `sw_flg_cyc.c` → `sw_flg_tsk.c`
 - `sw_flg_cyc.cfg` → `sw_flg_tsk.cfg`
 - `sw_flg_cyc.h` → `sw_flg_tsk.h`
 - Makefileの166行目`sw_flg_cyc`を`sw_flg_tsk`に書き換え
 - `sw_flg_tsk.cfg`の15行目`sw_flg_cyc`を`sw_flg_tsk`に書き換え
 - `sw_flg_tsk.c`の10行目`sw_flg_cyc.h`を`sw_flg_tsk.h`に書き換え
- `sw_flg_tsk`ディレクトリ上で`make`を行って確認

演習：イベントフラグによる通知プログラム (sw_flg_tsk)

- イベントフラグとタスク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sw_flg_cycをもとに作成する
- プログラムの仕様
 - 監視タスク
 - DIPSW1,2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
 - 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

sw_flg_tsk : 監視タスク

- タスク実行で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_task(intptr_t exinf){
    UW tmp;
    FLGPTN flgptn;
    for(;;){
        tmp = switch_dip_sense();
        flgptn = 0;
        if ((sw_state & DSW1) != (tmp & DSW1)) {
            if ((tmp & DSW1) == DSW1) {
                flgptn |= LED1_ON;
                ... 省略 ...
            }
            if (flgptn != 0) {
                SVC_ERROR(set_flg(SW_FLG, flgptn));
                sw_state = tmp;
            }
            dly_tsk(SW_SCAN_INTERVAL);
        }
    }
}
```

- 状態に変更があればフラグで通知

sw_flg_tsk : その他の修正

- インクルードファイル (sw_flg_tsk.h)

```
#ifndef TOPPERS_MACRO_ONLY
extern void led_task(intptr_t exinf);
extern void sw_task (intptr_t exinf);
extern void state_init(intptr_t exinf);
extern void shield_init(intptr_t exinf);
extern void setup_sw2_func(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

- コンフィギュレーションファイル (sw_flg_tsk.cfg)
 - sw_taskをCRE_TASKで定義

```
#include "device.h"
#include "sw_flg_tsk.h"
CRE_TSK(MAIN_TASK, {TA_ACT, 1, led_task,
    DEFAULT_PRIORITY, STACK_SIZE, NULL});
CRE_TSK(SW_TASK, {TA_ACT, 0, sw_task,
    DEFAULT_PRIORITY, STACK_SIZE, NULL});
ATT_INI({ TA_NULL, 0, shield_init });
ATT_INI(TA_NULL, 0, state_init);
```

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

演習 : データキューによる通知プログラム (sw_dtq_cyc)

- sw_flg_cycを改造してタスクからイベントフラグを通知するプログラムを作成する
- 作成方法
 - sw_flg_cycをmy_program|にコピーして名前をsw_dtq_cycに変更
 - sw_dtq_cyc内の以下のプログラムをリネーム
 - sw_flg_cyc.c → sw_dtq_cyc.c
 - sw_flg_cyc.cfg → sw_dtq_cyc.cfg
 - sw_flg_cyc.h → sw_dtq_cyc.h
 - Makefileの166行目sw_flg_cycをsw_dtq_cycに書き換え
 - sw_dtq_cyc.cfgの15行目sw_flg_cycをsw_dtq_cycに書き換え
 - sw_dtq_cyc.cの10行目sw_flg_cyc.hをsw_dtq_cyc.hに書き換え
- sw_dtq_cycディレクトリ上でmakeを行って確認

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});
```

属性

- データキュー操作API

- 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

sw_dtq_cyc : 周期ハンドラ

- スイッチ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){
    UW tmp = switch_dip_sense();
    if ((sw_state & DSW1) != (tmp & DSW1)) {
        if ((tmp & DSW1) == DSW1) {
            SVC_ERROR(ipsnd_dtq(SW_DTQ, LED1_ON));
        } else {
            SVC_ERROR(ipsnd_dtq(SW_DTQ, LED1_OFF));
        }
    }
    if ((sw_state & SW2) != (tmp & SW2)) {
        if ((tmp & DSW2) == DSW2) {
            SVC_ERROR(ipsnd_dtq(SW_DTQ, LED2_ON));
        } else {
            SVC_ERROR(ipsnd_dtq(SW_DTQ, LED2_OFF));
        }
    }

    sw_state = tmp;
}
```

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){
    UWsw_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 |= LED01;
        } else if (sw_data == LED1_OFF) {
            led_data &= ~LED01;
        } else if (sw_data == LED2_ON) {
            led_data |= LED02;
        } else if (sw_data == LED2_OFF) {
            led_data &= ~LED02;
        }

        led_out(led_data);
    }
}
```

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_tsk)

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 `irotd_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

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
- 本講座では μ ITRON4.0仕様準拠のAPIでの実習を行っており、他の μ ITRON4.0仕様のカーネルとも互換性を持つ
- 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

講座終了の確認

- my_programには、以下のディレクトリができ正しくプログラムが実行されているはずです

実習済プログラム	(sw_dtq_tskは無くても良い)
ex_sem	: LEDカウント排他制御あり
led_2tasks	: 2タスクLED点滅プログラム完成版
led_2tasks_for	: 2タスクLED点滅改造初版
led_2tasks_share	: 2タスクLED点滅タスク共有プログラム完成版
led_cyc	: 1タスクと周期ハンドラによるLED点滅
sw_dtq_cyc	: SW制御を周期ハンドラからデータキュー通信
sw_flg_tsk	: SW制御をタスクからイベントフラグ通信
sw_int_api	: プッシュスイッチ割込みタスク起動
sw_int_api2	: プッシュスイッチ割込みタスク起床
sw_int2	: プッシュスイッチ割込み
timer_int2	: 割込みタイマーによるLED1と2の点滅

著者リスト

- 実装開発環境
 - 竹内 良輔(教育WG)
- 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(教育WG)
- Raspberry Pi Pico/Pico2ボード対応
 - 竹内 良輔(教育WG)

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。