

TOPPERS基礎実装セミナー

(Raspberry Pi Pico(W)/Pico2(W)版:基本)

基礎2編:1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)1日目＞

Copyright (C) 2006-2009 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) 2006-2009 by 本田晋也、高田広章 名古屋大学

Copyright (C) 2008-2025 by 竹内良輔 (株)リコー

Copyright (C) 2013-2021 by 森田浩

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

■ 1日目

1. リアルタイムOSの基礎	0.3時間
2. ITRON基礎知識	1.5時間
3. 開発環境の構築	0.5時間
4. TOPPERS/ASPカーネルの導入	1.5時間
5. システム検証モジュールの導入	2.0時間
6. まとめ	0.5時間

概要

リアルタイムOSの基礎及び、 TOPPERS／ASPカーネルの導入と拡張を学ぶ

- リアルタイムOSの基礎
 - リアルタイムOSを用いた組込み開発
 - μ ITRON仕様
 - TOPPERS／ASPカーネル
- TOPPERS／ASPカーネルの導入
 - 開発環境の構築
 - サンプルプログラムのビルドと実行
- TOPPERS／ASPカーネルの拡張
 - システム検証モジュールの導入と改造

リアルタイムOSの基礎

1. リアルタイムOSを用いた組込み開発
2. タスク間の同期と通信

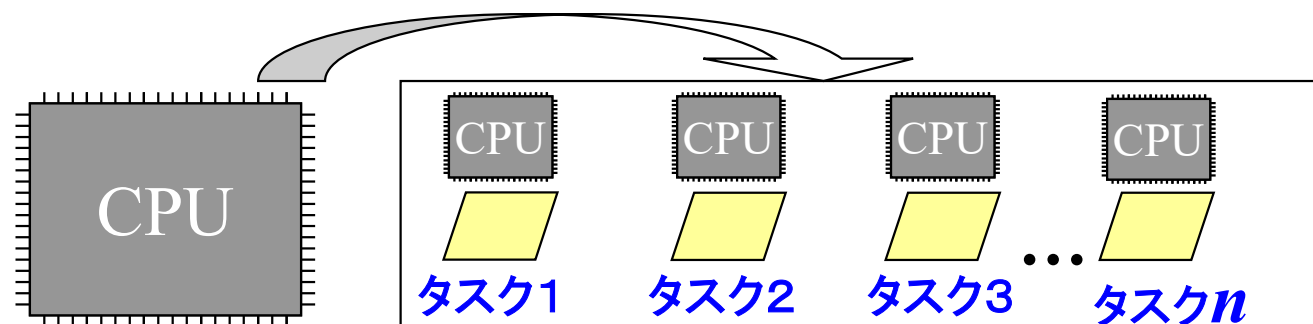
小規模組込み開発の限界

- 小規模システムでは小規模な機器管理が中心です。開発規模が大きくなると、複雑なUI部、通信機能、ファイルシステムの管理等種々のノウハウを持った開発者が分業してシステムを開発する必要があります
- 多くの機能を組み合わせた場合、main関数からの関数コールではハードリアルタイム性を保持するのが難しくなる
 - ある機能で100msの待ちが発生した場合、main関数からの関数呼び出しでは、その機能中でソフトウェアディレイで待つしかなく、不要なCPUの占有が発生する
 - ソフトウェアディレイ中に他の機能を動かそうとするとシステムが複雑化し、メンテナンス性が落ちる

➡ リアルタイムOSを用いれば問題解決可能

中規模組込みシステムとリアルタイムOSの必要性

- リアルタイムOSを用いることにより、個々の機能を独立したタスクとして動作させることが可能となります
 - リアルタイムOSの大きな特徴はマルチタスク機能
- マルチタスク機能とは
 - 複数のタスクを擬似並列に実行するための機能
 - 1CPUのシステムでは、実際には同時に実行できるのは1つのタスクのみです
 - あたかも複数のタスクを同時に実行しているようにみせる機能です



リアルタイムOSを利用するメリット

- ソフトウェアの構造化により生産性、保守性、信頼性が向上する(システム規模が大きくなると重要な問題となる)

ソフトウェアの構造化≡モジュール化

- 時間制約を持った多重処理システムの構築が容易になる
 - 論理的な処理順序と時間的な処理順序の分離により、時間制約を持ったソフトウェアの保守性・再利用性が向上

ソフトウェア部品を用いたソフトウェア開発
(コンポーネントベース開発)の基盤

- ハードウェア(特にプロセッサ)の違いを隠蔽できる
 - ハードウェアの細部を知らない技術者でも、アプリケーションが構築できる

リアルタイムOSを使用するデメリット

- リアルタイムOS自身にオーバーヘッドがある
 - オーバーヘッドによる実行性能の低下
 - RTOS自身のワークエリアによりメモリ消費
- マルチタスク機構による問題点
 - タスクのスタックエリアによるメモリ消費
 - 非決定性が増すことによるデバッグ・テストの困難化
行儀のよいタスク間インターフェイス設計が必要
- リアルタイムOSのブラックボックス化により、問題発生時の解析が困難となる
 - リアルタイムOSに対応したツールの利用で改善は可能
 - リアルタイムOSの原理、内部構造を知る技術者は必要

リアルタイムOSの基礎

1. リアルタイムOSを用いた組込み開発
2. タスク間の同期と通信

タスク間の通信と同期

- タスク間の通信・同期の必要性
 - タスクが協調して動作するためには、タスク間でデータをやりとりすること(=タスク間通信)が必要
 - タスク間通信の際には、タスク同士で動作タイミングをあわせること(=タスク間同期)が必要
 - 複数のタスクが同一の資源を取り合う場合にも、タスク間の同期が必要
- タスク間通信・同期のタイプ

タスクを仮想化されたプロセッサと捉えるなら、タスク間の通信・同期はプロセッサ間の通信・同期を仮想化したもの

共有メモリによる通信 or メッセージによる通信

共有メモリによる通信

複数のタスクからアクセスできるメモリ領域上に
受け渡しするデータ(共有データ)置く

- C言語のグローバル変数による実現
- 共有データを読み書きする際には、RTOSの機能を用いて、排他制御することが必要
 - 割込み/ディスパッチの禁止
 - セマフォ/ミューテックス
 - ⚠ デッドロックと優先度逆転に注意
 - タスクの優先度の変更
- 共有データを速やかに処理させたい場合には、事象の発生を知らせる機能を用いる
 - タスクの起動/起床
 - イベントフラグ/Condition Variable(条件変数)

メッセージによる通信

RTOSが持つメッセージ通信のための機能を用いて、
タスク間でメッセージを受け渡しする

- メッセージ通信機能のタイプ
 - コネクションあり or なし、単方向 or 双方向、
1対1(送信できるタスクが固定) or n対1 or n対n
 - 通信相手の指定方法
 - 通信オブジェクトを指定 or 相手タスクを指定 or
 - 同期メッセージ通信 or 非同期メッセージ通信
 - (非同期通信で)バッファにメッセージが無い場合の振る舞い
 - 待つ(ブロッキング) or 待たない(ノンブロッキング)
 - (非同期通信で)バッファがフルの時の振る舞い
 - 待つ(ブロッキング) or 待たない(ノンブロッキング)
or 上書きする

メッセージ通信機能のタイプ(続き)

- メッセージが固定長 or 可変長(任意長)
- (可変長の時に)パケットの単位がある or ない
- ポインタを渡す or コピーする
- (非同期通信で)メッセージのキューイング順序
 - FIFO順 or 優先度順
- 受信/送信待ちタスクのキューイング順序
 - FIFO順 or 優先度順
- その他特殊な機能
 - マルチキャスト/ブロードキャスト
 - 状態メッセージ(OSEK/VDX仕様の unqueued message)

RTOSの持つメッセージ通信機能(複数の機能を持つ場合も多い)の性質をよく理解することが必要

タスク間通信・同期の設計

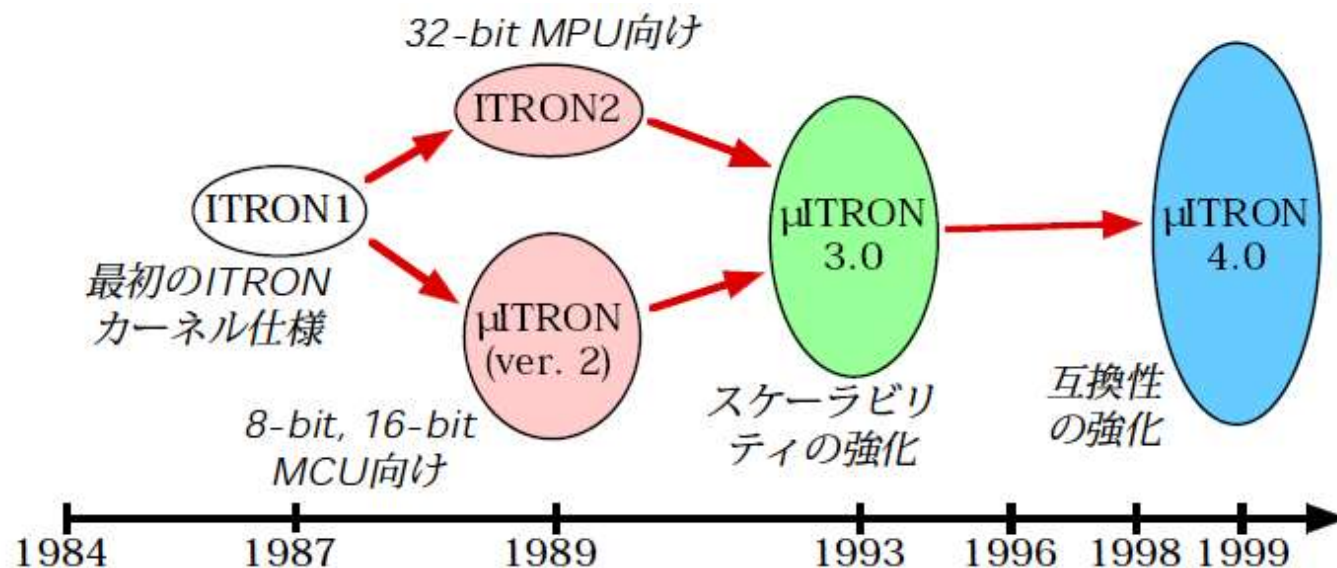
- 共有メモリ vs. メッセージ通信
 - 基本的には、一方で実現できることは、もう片方でも実現できる
 - 一般には共有メモリの方がオーバーヘッドが小さい
 - システム検証には、通信の粒度が大きくRTOSレベルで監視できるメッセージ通信の方が扱いやすい
 - 例えば、問題の切り分けが容易
 - 状況により、どちらかが有利/便利な状況がある
 - 情報の流れが 1:n の場合 (ブロードキャストを含む) や、情報が必要なタイミングが不定の場合には、共有メモリが有利
 - 情報のキューイングが必要な時には、メッセージ通信が便利

ITRONの基礎知識

1. [ITRON仕様](#)
2. μ ITRONの同期・通信機能
3. TOPPERS/ASPカーネル

ITRON仕様

- ITRONプロジェクト
 - TRONプロジェクトのサブプロジェクトの1つ
 - ITRON仕様
 - ITRONプロジェクトで策定されたリアルタイムカーネル仕様
 - これまでに4世代のITRON仕方を策定・公表
 - 現世代のリアルタイムカーネル技術の範囲では、完成度の高い仕方に
- ➡ 組み込みシステム開発のオープン化に対する基盤



ITRON仕様の特徴



- OSの小型軽量化が可能
 - ワンチップマイコンにも適用可能
- 仕様の理解が容易
 - 技術者教育のための標準化の側面を重視
- 完全にオープンな標準仕様
 - ロイヤリティなしで実装することができる
- 多種多様なプロセッサ用に実装できる/されている
 - 8bitワンチップマイコンから32bit RISCマイコンまで
 - 異なるプロセッサへの移行が容易に
- 多くの機器で使用実績がある
 - 組み込みシステム分野で最も広く使われているOS仕様
- 多くのメーカ/ベンダがサポート

ITRON仕様のカーネルの機能(μ ITRON4.0仕様)

- カーネルの機能
 - タスク管理機能
 - タスク付属同期機能
 - タスク例外処理機能
 - 同期・通信機能
 - 拡張同期・通信機能
 - メモリプール機能
 - 時間管理機能
 - システム状態管理機能
 - 割り込み管理機能
 - サービスコール管理機能

- サービスコール数
 - フルセット
 - サービスコール : 166
 - 静的API : 21
 - スタンダードプロファイル
 - サービスコール : 70
 - 静的API : 11
 - 自動車制御用プロファイル
 - サービスコール : 43
 - 静的API : 8
 - 最小セット



！ I/O操作のための機能は規定されていない

ITRON仕様のサービスコール(API)

- サービスコール名称のガイドライン

例) `act_tsk`

操作対象を表す. この場合はタスク(task)

操作方法を表す. この場合は起動(activate)

- サービスコールが成功するとE_OKが戻り値として返される

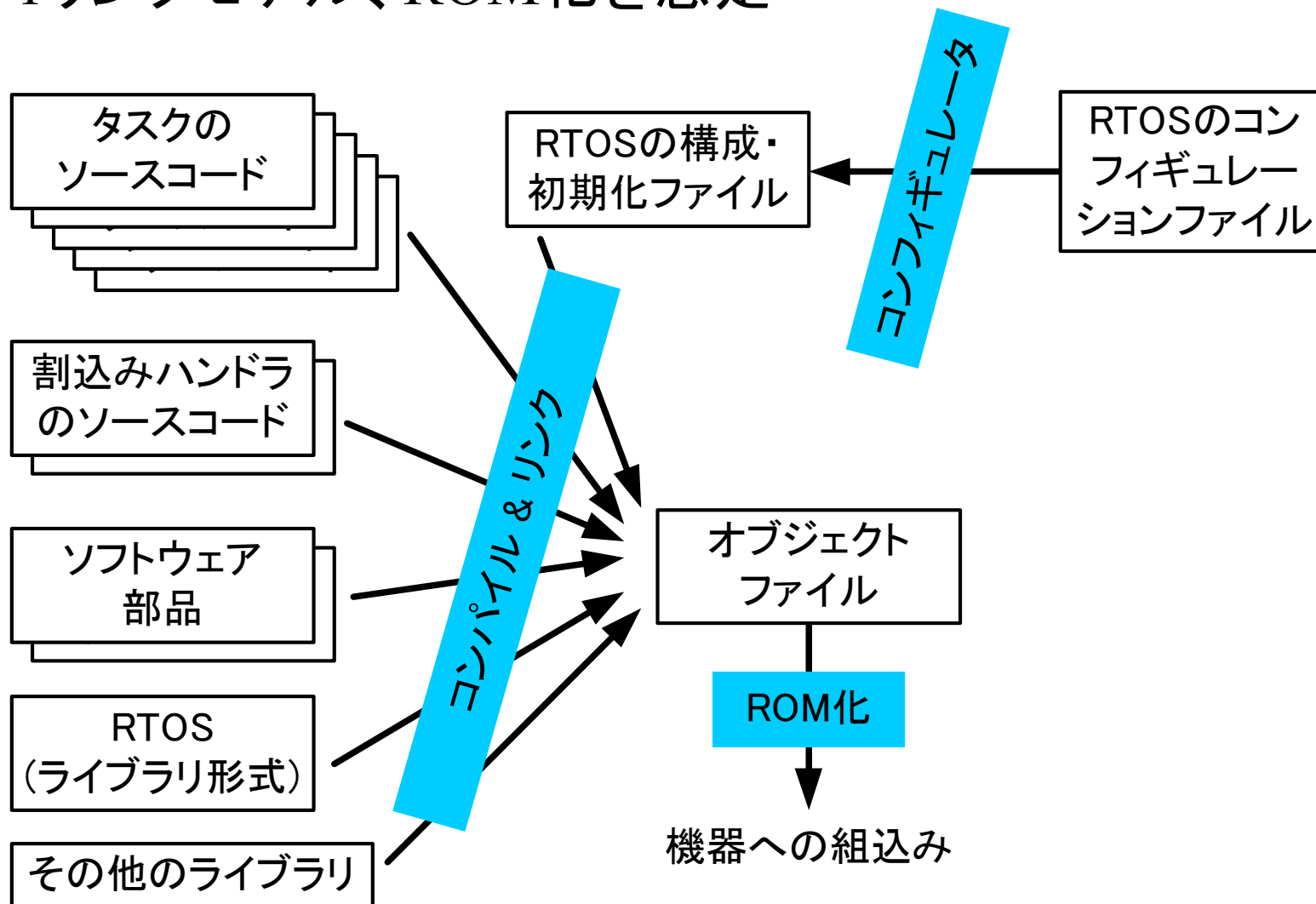
- 割込みハンドラ(厳密には)から呼び出せるサービスコールは"i"で始まる名称として区別

例) `iact_tsk`

- 割込みハンドラから呼び出せないAPIがある
 - 待ち状態に入るAPI

RTOSを用いた組込みソフトウェアの構築方法

- 1リンクモデル、ROM化を想定



RTOSのコンフィギュレーション

- RTOSの構成や初期状態を決める手順
- どのような構成が変更できるかはRTOS毎. 例えば、
 - 用いる機能/用いない機能
 - 各オブジェクト(タスクやセマフォ)などの最大値
 - タスク優先度の段数
 - 各オブジェクトの生成・定義情報
(オブジェクトを静的に生成する場合)
 - コンフィギュレーション機構
 - コンフィギュレーションによりRTOSのコードを変える方法(条件コンパイルなどを使う)
 - コンフィギュレーション結果より、1つ(または複数)のソースコードに生成する方法

μITRON4.0仕様におけるコンフィギュレーション

- カーネルオブジェクトは静的に生成
 - オブジェクト生成情報の、コンフィギュレーションファイルの中での記述方法を標準化(静的API)
- 静的APIの例

```
CRE_TSK(tskid, {tskatr, exinf, task, itskpri, stksz,stk});  
DEF_INH(inhno, {inhatr, inthdr});
```

- コンフィギュレータは静的APIを解析して、カーネルの内部情報を生成する

静的APIの詳細

CRE_TSK	タスク生成(静的API)	【スタンダード】
cre_tsk	タスク生成	【スタンダード外】

【静的API】

CRE_TSK (ID tskid, {ATR tskatr, VP_INT exinf, FP task,
PRI itskpri, SIZE, stksz, VP stk})

【C言語API】

ER ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk)

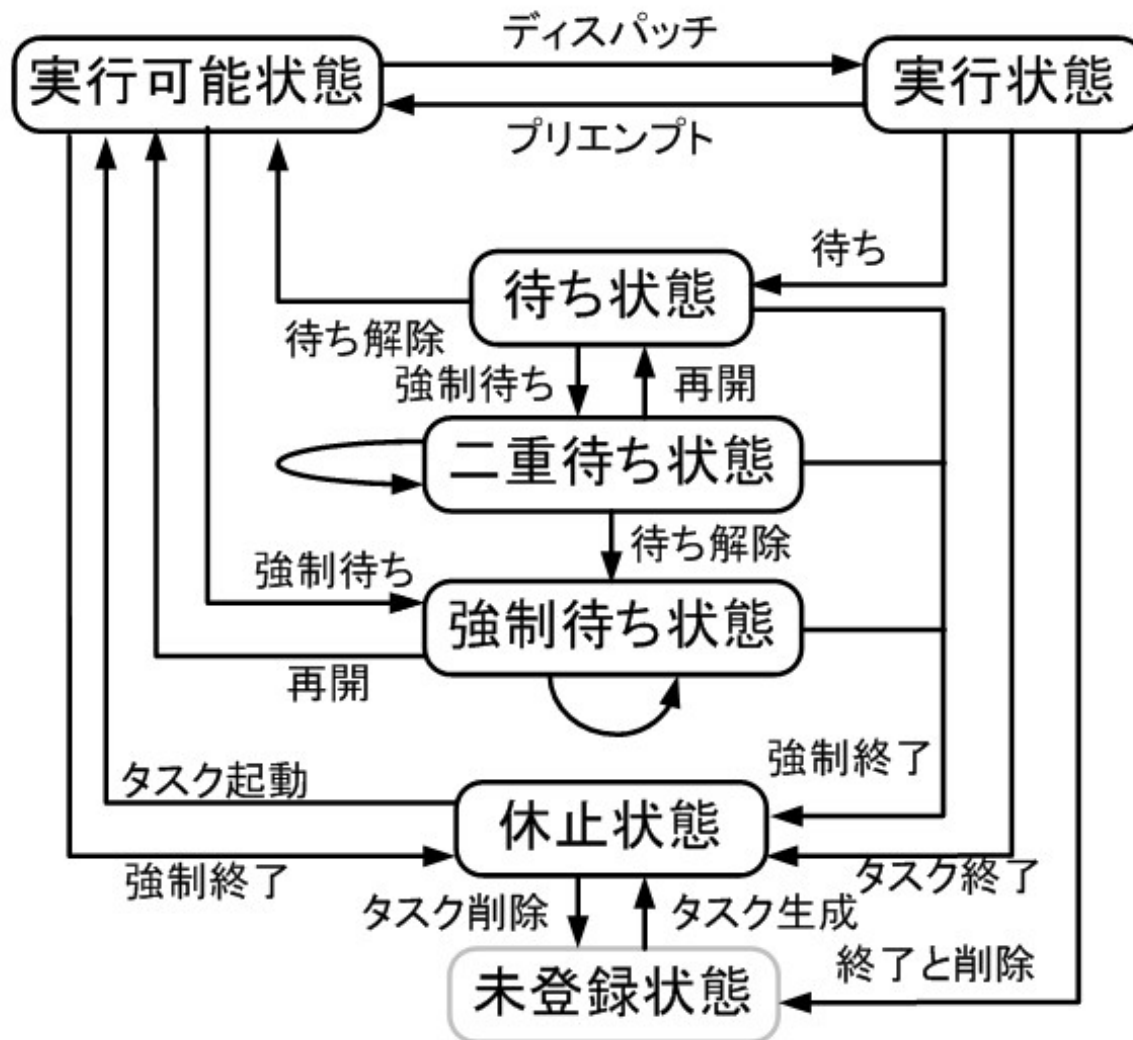
【パラメータ】

ID	tskid	生成対象のタスクのID番号
T_CTSK	*pk_ctsk	タスク生成情報を入れたパケット
ATR	tskatr	タスク属性(記述言語、初期状態)
VP_INT	exinf	タスクの拡張情報(タスクへのパラメータ)
FP	task	タスクの起動番地
PRI	itskpri	タスクの起動優先度
SIZE	stksz	タスクのスタック領域のサイズ(バイト数)
VP	stk	タスクのスタック領域の先頭番地

マルチタスク機構

- スケジューリング方式
 - μ ITRON4.0では、プリエンプティブな優先度ベーススケジューリングによるマルチタスク機構をサポート
 - 同優先度のタスクはFCFS (First Come First Served) 方式でスケジューリング. 同優先度のタスク内での実行順序を変更する機能を用意. これを用いて、同優先度内でのラウンドロビンスケジューリングも実現可能
 - 優先度付けは静的が基本だが、優先度を動的に変更する機能も用意
- タスク状態
 - 実行、実行可能、休止、待ち、強制待ち、二重待ち、未登録の7つの状態をサポート

タスクの状態遷移



- 実行状態 (RUNNING)
- 実行可能状態 (READY)
- 待ち状態 (WAITING)
 - タスクが自ら待ち状態に入る
- 強制待ち状態 (SUSPENDED)
 - 他のタスクにより待ち状態にされた
- 二重待ち状態 (WAITING-SUSPENDED)
 - 上の2つが重なった状態
- 休止状態 (DORMANT)
- 未登録状態 (NON-EXISTENT)

タスク管理機能

タスクの状態を直接的に操作するための機能

- タスク管理機能のサービスコール

cre_tsk	タスクの生成
del_tsk	タスクの削除
act_tsk, iact_tsk	タスクの起動
can_act	タスクの起動要求の無効化
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスクの優先度の変更
get_pri	タスクの優先度の参照
ref_tsk	タスクの状態参照

- タスクの起動要求 (act_tsk) のキューイングとその無効化 (can_act)

タスク付属同期機能

タスクを直接操作して同期を実現するための機能

- タスク付属同期機能のサービスコール

slp_tsk, tslp_tsk	自タスクを起床待ち状態に
wup_tsk, iwup_tsk	他タスクの起床
can_wup	起床要求の無効化
rel_wai, irel_wai	タスクの待ち状態の強制解除
sus_tsk	タスクを強制待ちに
rsm_tsk, frsm_tsk	タスクを強制待ちからの再開
dly_tsk	自タスクを遅延

- タスクの起床要求 (wup_tsk) のキューイングとその無効化 (can_wup)
- タイムアウト付きの起床待ち (tslp_tsk) と自タスクの遅延 (dly_tsk)

時間管理機能：概要

- システム時刻管理
 - システム時刻(カーネルが管理する現在時刻)を管理するための機能
- タイムイベントハンドラ
 - 時間の経過をきっかけに呼び出されるルーチン
 - μ ITRON4.0仕様では以下のタイマハンドラを用意
 - 周期ハンドラ(周期的に呼ばれる)
 - アラームハンドラ(指定した時刻に呼ばれる)
 - オーバランハンドラ(オーバラン時に呼ばれる)
- 時間同期のためのその他の機能
 - タスクの遅延(タスクを一定時間待たせる)
 - 待ちに入るサービスコールのタイムアウト機能

時間管理機能：サービスコール

- システム時刻管理のためのサービスコール

set_tim	システムクロックの設定
get_tim	システムクロックの読み出し
isig_tim	タイムティック供給

- 周期ハンドラ操作のためのサービスコール

cre_cyc	周期ハンドラ生成
del_cyc	周期ハンドラの削除
sta_cyc	周期ハンドラの起動
stp_cyc	周期ハンドラ停止

システム状態管理機能

システム状態を参照/変更するための機能

- システム状態管理機能のサービスコール

rot_rdq, irot_rdq	タスクの実行順序の回転
get_tid,iget_tid	実行状態のタスクのIDの取り出し
loc_cpu, iloc_cpu	CPUロック状態に
unl_cpu_iunl_cpu	CPUロック状態の解除
dis_dsp	ディスパッチ禁止
ena_dsp	ディスパッチ許可
sns_ctx	非タスクコンテキスト実行中か?
sns_loc	CPUロック状態か?
sns_dsp	ディスパッチ禁止状態か?
sns_dpn	ディスパッチ保留状態か?

割込み処理と割込み管理機能

- 割込み処理(外部割込み)
 - μ ITRON仕様では、割込みハンドラをアプリケーション側で記述できる
 - 割込みが発生すると、カーネル内の出入り口処理を経由して、割込みハンドラが呼び出される
 - 割込みハンドラの中でサービスコールを呼び出して、タスクの起動/起床などが可能
 - 割込みハンドラ処理はタスクよりも優先
 - ディスパッチが遅延する
- 割込み管理機能のサービスコール

def_inh

割込みハンドラの定義

- この他に、割込みを個別に禁止/許可する機能など

システム構成管理機能

- プロセッサレベルの例外処理(CPU例外)
 - 割込みと類似
 - CPU例外が発生すると、カーネル内の出入口処理を経由して、CPU例外ハンドラが呼び出される
 - CPU例外ハンドラの中でサービスコールを呼び出して、タスクに対して通知を行うことが可能
 - CPU例外ハンドラの処理はタスクよりも優先
 - ⚠ タスク例外との違いに注意
- 初期化ルーチンの定義
 - 初期化ルーチンは、システム初期化時に実行される
- システム構成管理機能のサービスコール

def_exc

CPU例外ハンドラの定義

ATT_INI

初期化ルーチンの追加

タスク例外処理

- タスクに対する割込み/例外に対応(仮想化された割込み)
- 明示的なサービスコール呼び出しにより、タスクに例外を発生させる
- 次にそのタスクが実行されるタイミングで、タスク例外処理ルーチンが呼び出される
- UNIXのシグナル処理/シグナルハンドラに相当
- タスク例外処理サービスコール

def_tex	タスク例外処理ルーチンの定義
ras_tex, iras_tex	タスク例外処理の要求
dis_tex	タスク例外処理の禁止
ena_tex	タスク例外処理の許可
sns_tex	タスク例外処理禁止状態か?

ITRONの基礎知識

1. ITRON仕様
2. [μITRONの同期・通信機能](#)
3. TOPPERS/ASPカーネル

μITRONの同期・通信機能

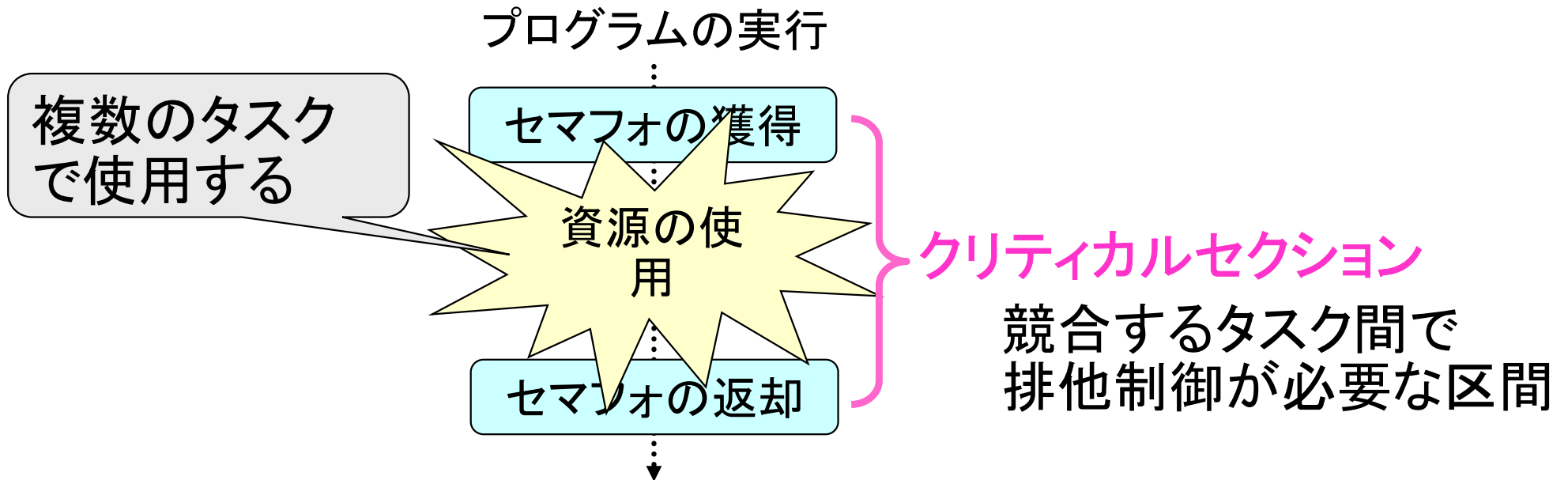
μITRON4.0では、タスク間同期・通信のために
以下の機能を用意

- (主に)共有メモリによる通信のための機能
 - セマフォ*(排他制御など)
 - イベントフラグ*(事象通知)
 - ミューテックス(排他制御)
- メッセージ通信のための機能
 - データキュー*(同期・非同期、1ワードのメッセージ)
 - メールボックス*(非同期、ポインタを送受信)
 - メッセージバッファ(同期・非同期、可変長メッセージ)
 - ランデブ(同期、双方向に通信、可変長メッセージ)

*スタンダードプロファイルに含まれる機能

セマフォ(Semaphore)

- 使用されていない資源の有無や数量をセマフォの資源数として管理することにより排他制御を実現
- 資源を使用する前にセマフォ資源を獲得し、資源を使用した後にセマフォ資源を返却する.
- 資源が全て使用されていれば、セマフォ資源獲得の時点で待ち状態になる.



セマフォ(Semaphore) : サービスコール&語源

- サービスコール

cre_sem

セマフォの生成

del_sem

セマフォの削除

sig_sem, isig_sem

セマフォ資源の返却

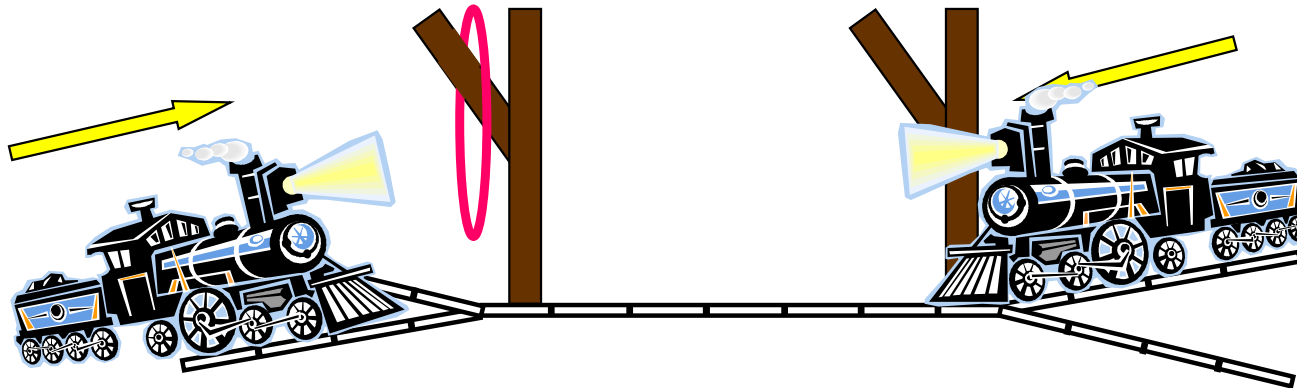
wai_sem, pol_sem, twai_sem

セマフォ資源獲得

- 語源

鉄道の単線区間で進入制御に用いていた「輪」.

単線区間に入る場合は, この輪(セマフォ)を取る

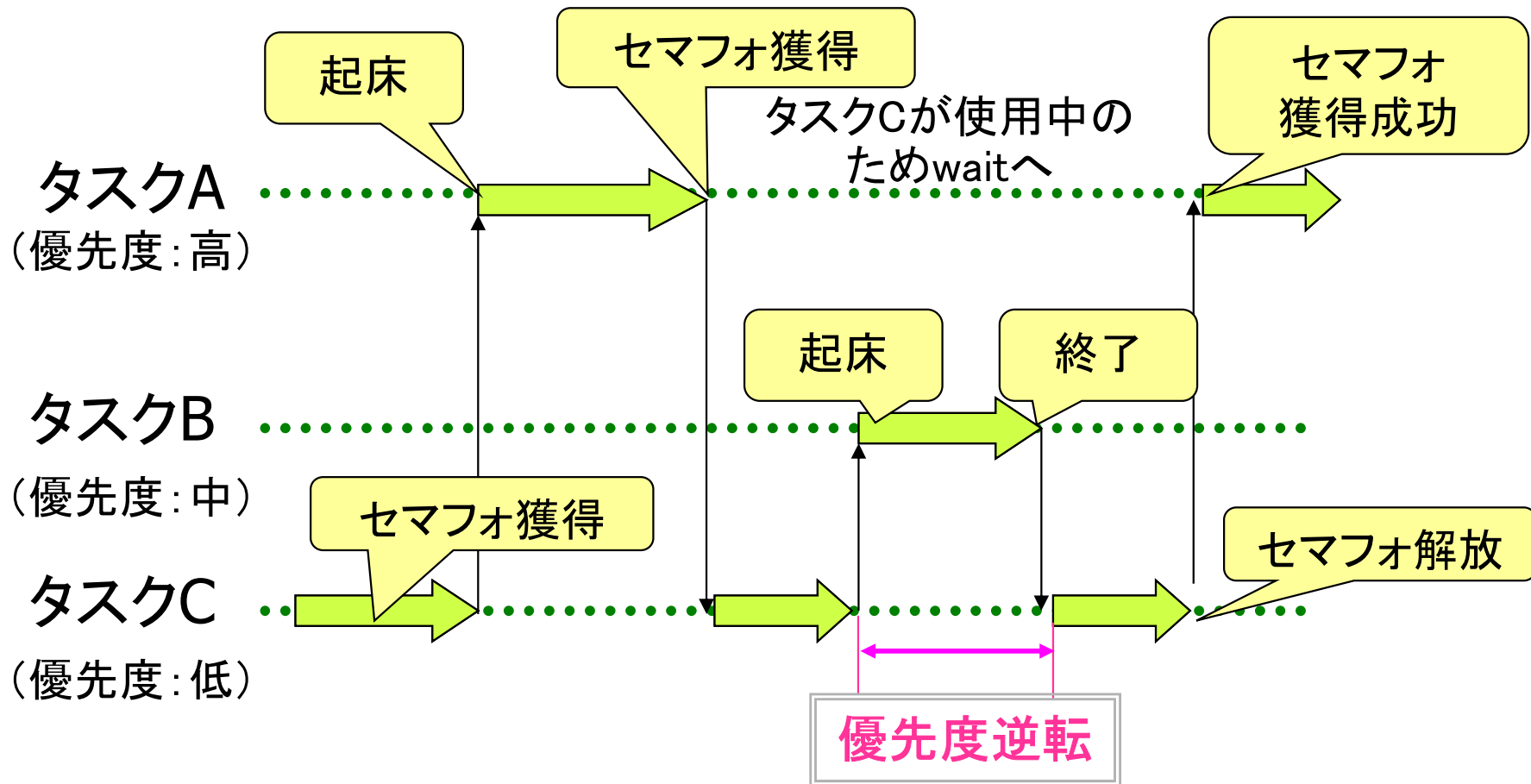


輪 : セマフォ
列車 : タスク
単線区間 : 資源

セマフォ(Semaphore): 優先度逆転

- 高優先度のタスクが低優先度のタスクの実行により待たされること

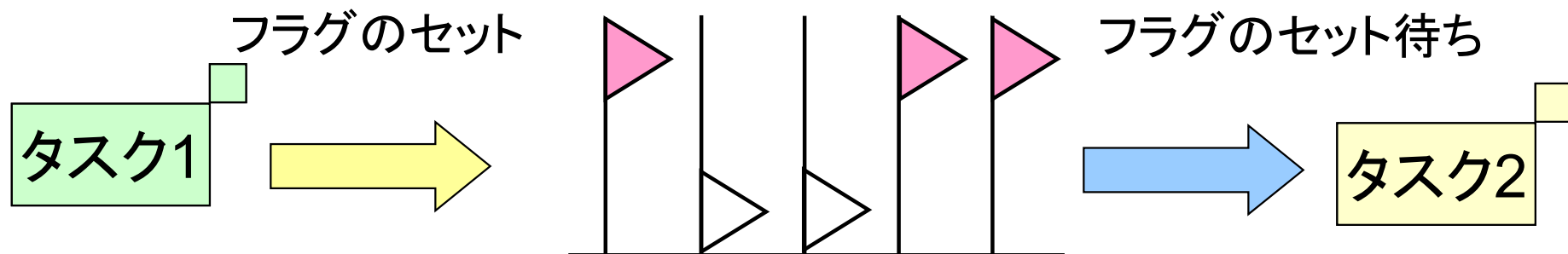
例) 高優先度のタスクAがそれより低優先度のタスクBの実行により待たされる. (注 タスクCに待たされるのは本質的)



イベントフラグ (EventFlag)

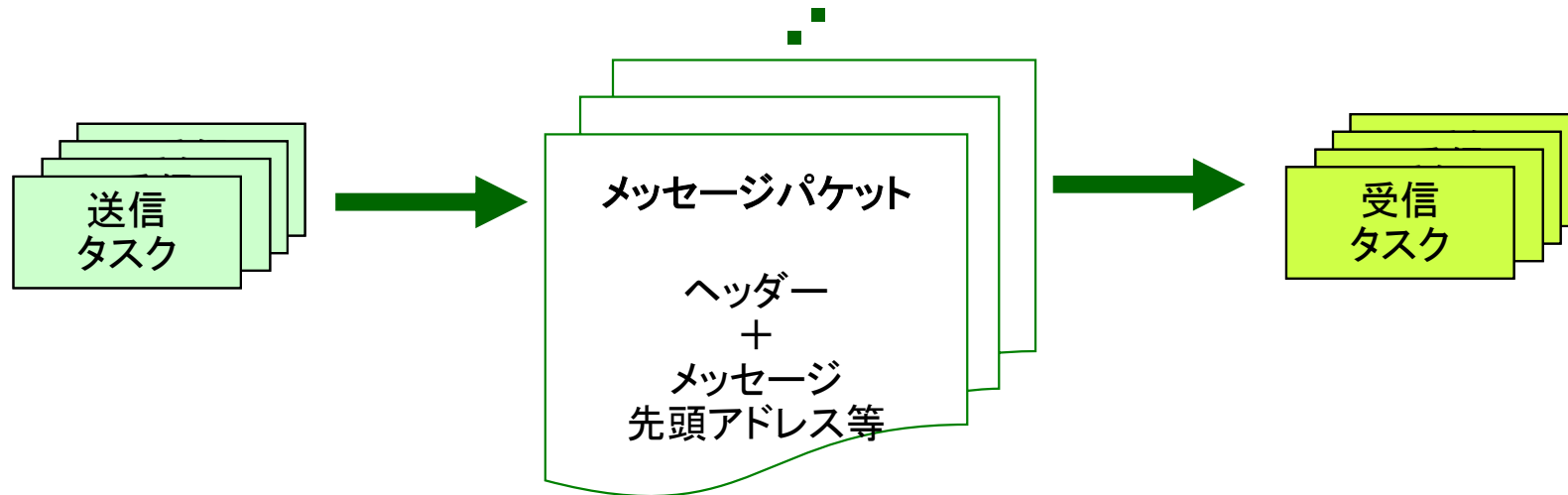
- タスク間でイベントの発生を伝えることで同期するための機構. 1つのイベントフラグは、複数のフラグで構成
- サービスコール

cre_flg	イベントフラグの生成
del_flg	イベントフラグの削除
set_flg, iset_flg	イベントフラグのセット
clr_flg	イベントフラグのクリア
wai_flg, pol_flg, twai_flg	イベントフラグ待ち

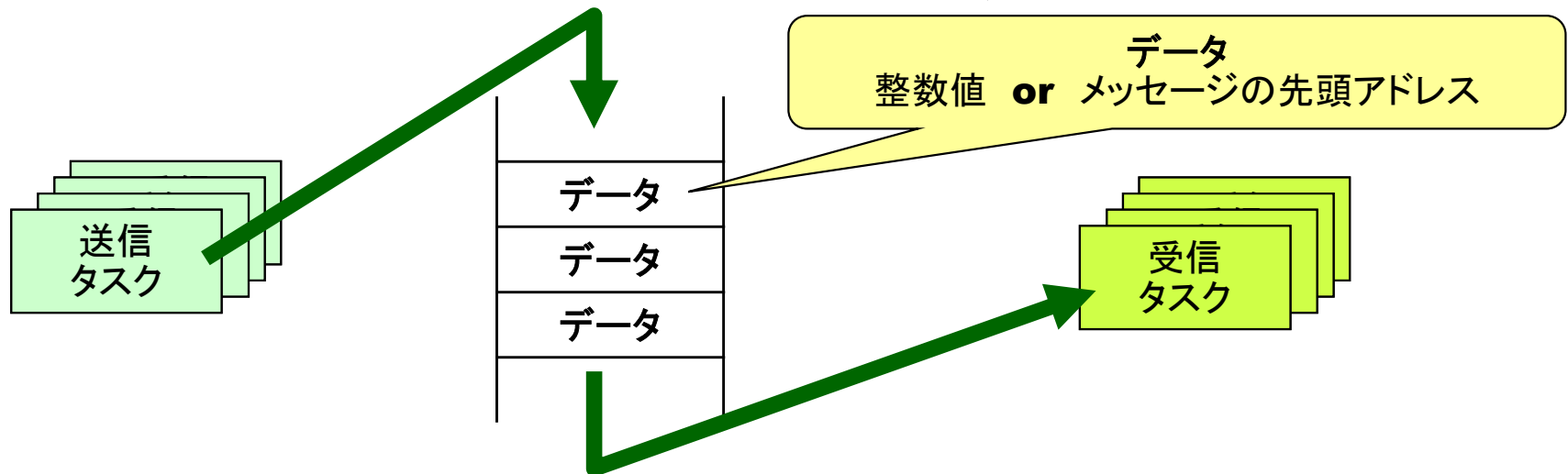


メールボックス と データキュー: 概要

- メールボックス : 共有メモリ上に置かれたデータをやり取りする



- データキュー : 1ワードのメッセージをやり取りする

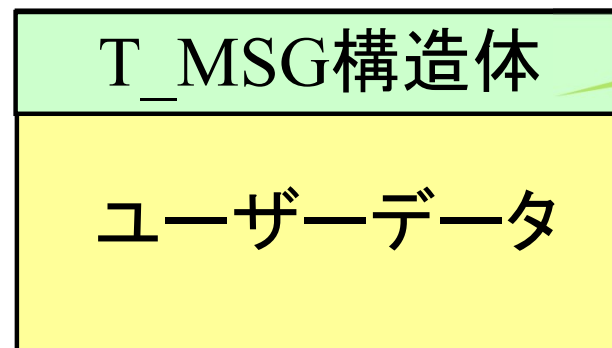


メールボックス (Mail Box)

- 可変長メッセージの非同期メッセージ通信機構
 - カーネルがリンクに用いるための領域を、メッセージの先頭にアプリケーション側で用意
- サービスコール

cre_mbx	メールボックスの生成
del_mbx	メールボックスの削除
snd_mbx	メールボックスへの送信
rcv_mbx, prcv_mbx, trcv_mbx	メールボックスからの受信

受け渡す
メッセージ



カーネルが
使用する領域

データキュー(Data Queue)

- 1ワードメッセージの非同期メッセージ通信機構
- メッセージは, 内部バッファにコピーされる
- メッセージが無い/フルの時は, 待ち合わせる
- データキュー領域のサイズを0に設定すると、同期メッセージ通信も実現可能
- サービスコール

cre_dtq	データキューの生成
del_dtq	データキューの削除
snd_dtq, psnd_dtq, tsnd_dtq, isnd_dtq	データキューへの送信
rcv_dtq, prcv_dtq, trcv_dtq	データキューからの受信
fsnd_dtq, ifsnd_dtq	データキューへの上書き送信

まとめ

- μ ITRONの同期・通信機能
 - セマフォ
 - イベントフラグ
 - メールボックス
 - データキュー
- それぞれの機能の性質を理解して、使用目的に最適な機能を選択することが重要.
- 細かいパラメータの設定で、振る舞いが変わるので、プログラミングの際には注意する.

ITRONの基礎知識

1. ITRON仕様
2. μ ITRONの同期・通信機能
3. [TOPPERS/ASPカーネル](#)

TOPPERS/JSPカーネルについて

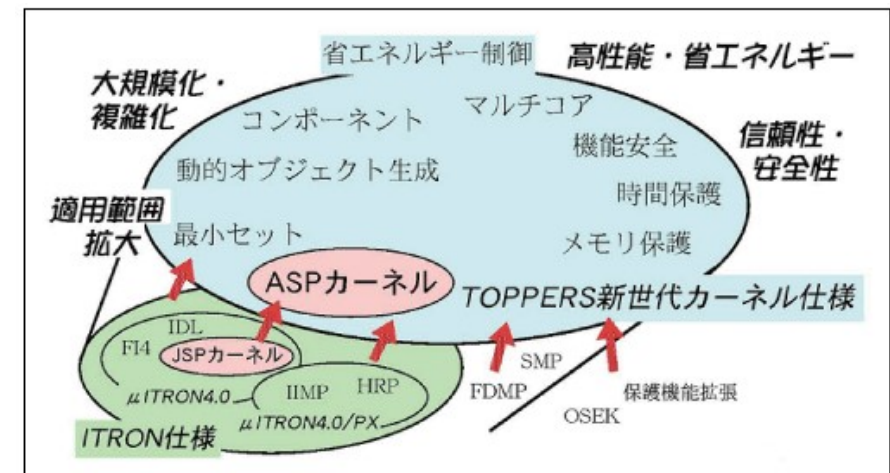
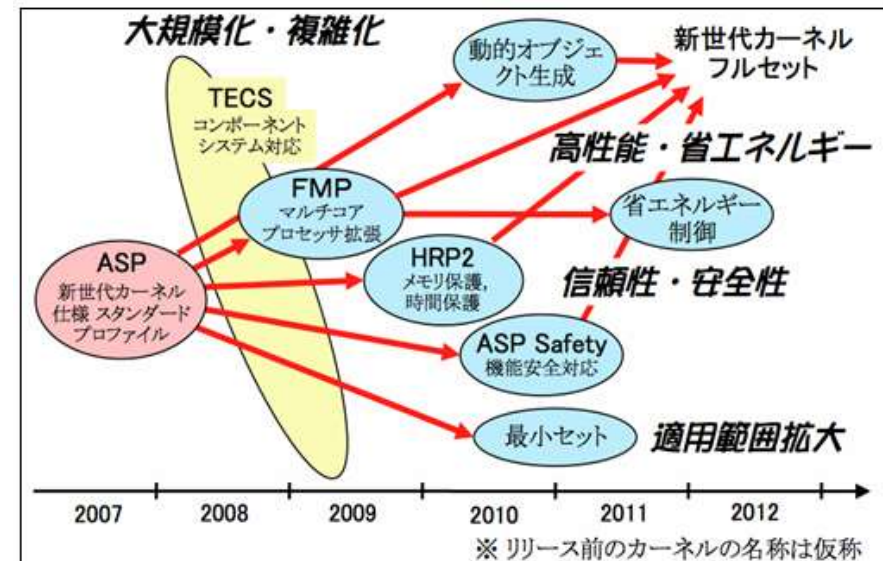
- JSPとは
 - JSP = Just Standard Profile
 - TOPPERSプロジェクトで開発されているμITRON4.0仕様のスタンダードプロファイルに準拠したオープンソースのリアルタイムOS.
最新版は Release 1.4.4.1-full
- 開発の目的
 - μITRON4.0仕様の評価、リファレンス実装
 - 研究・教育機関における研究・教育のプラットフォーム
 - ソフトウェア部品(IP)開発のプラットフォーム
 - 評価目的・プロトタイプ開発への利用
 - 実製品への適用
- ターゲット環境
 - SH1/2, SH3, H8, ARMv4, MIPS, PowerPC

TOPPERS/JSPカーネルの特徴

- 読みやすく改造しやすいソースコード
 - 定量的な評価は難しいが、読みやすさには自身あり
 - 可能な限り #ifdef を追放
 - 様々な改造・拡張の実績あり
- 他のターゲットへのポーティングが容易な構造
 - 実行性能を落とさないプロセッサの抽象化
 - 新しいプロセッサへのポーティングを2日間で行った例
- 高い実行性能と小さいRAM使用量
 - **！** 大部分をC言語で記述したカーネルとしては
- LinuxおよびWindows上でのシミュレーション環境
 - プロトタイプ開発、教育用に最適
- オープンソースソフトウェアのみで開発環境まで

TOPPERS/ASPカーネルについて

- これまでの経験を反映して、次世代リアルタイムOSの普及版の開発を行う
- 開発方針
 - μ ITRON4.0仕様をベースに拡張・改良を行う。
 - ソフトウェアの再利用性を重視する
 - 高信頼性・安全なシステム構築を支援する
 - アプリケーションシステム構築に必要な機能は積極的に取り組む



TOPPERS/ASPカーネルの概要

- 仕様の概要

TOPPERS/JSPカーネル(μ ITRON4.0仕様スタンダードプロファイル準拠)に対して、信頼性・安全性・ソフトウェアポータビリティ向上のための各種の拡張・改良

- μ ITRON4.0仕様からの主な拡張・改良点

- ・割込み処理機能を「TOPPERS準拠割込み処理モデル」によりプロセッサによらず標準化
- ・いくつかの独自の拡張機能(同期・通信オブジェクトの再初期化、優先度データキューなど)
- ・システムコンフィギュレーションの仕組みの見直し
- ・信頼性・安全性の向上については細かな改良の積み重ね
- ・TOPPERS組込みコンポーネント仕様の導入(将来拡張)

- μ ITRON4.0仕様の上位互換ではない

TOPPERS/ASP仕様拡張の詳細1

- スタンダードプロファイル外の機能の一部導入
 - イベントハンドラに複数待ち機能(TA_WMUL)
 - アラームハンドラ
 - 割込みサービスルーチン
 - 割込み管理機能
 - オブジェクトの状態参照機能
- μ ITRON4.0仕様に対する変更
 - ITRON標準データの見直し
 - 非タスクコンテキストからのext_tsk
 - CPU例外ハンドラで行える操作
 - カーネルの用いる管理領域の分離
 - 処理単位とメモリ領域のデータ型の見直し
 - 値がゼロの定数(オブジェクト属性等)の見直し
 - 強制待ち要求ネストの廃止
 - システム時刻の設定機能(set_tim)の廃止

TOPPERS/ASP仕様拡張の詳細2

- JSPカーネルにおける独自の拡張機能
 - 性能評価用システム時刻参照機能(get_utm)
 - 終了処理ルーチン機能(ATT_TER)
 - カーネル動作状態の参照(sns_ker)
- ASPカーネルにおける独自の拡張
 - 割込み要求ラインの属性の設定(CFG_INT)
 - 同期・通信オブジェクトの再初期化機能
 - 優先度付きデータキューの新設
 - 自タスクの拡張情報の参照(get_inf)
 - カーネルの終了(ext_ker)
 - 非タスクコンテキスト用のスタック領域の設定(DEF_ICS)

TOPPERS/ASP仕様拡張の詳細3

- JSPにおける実装定義／実装依存部規定からの変更
 - アプリケーション向けのインクルードファイルの構成の整理
 - 割込み処理／例外関連の型の定義変更
 - 処理単位の実行開始／リターン時のシステム状態の規定
 - iset_timの廃止
 - カーネルの用いる領域の指定方法
 - カーネル管理外の割込みの扱いの規定
- システムコンフィギュレーション処理の見直し
 - システムコンフィギュレーション処理を全面的に見直した
 - 自動生成される標準的なファイルは以下の2つとなった
 - kernel_cfg.c:カーネル構成初期化ファイル
 - kernel_cfg.h:カーネル構成ヘッダファイル

TOPPERS/ASPカーネルとμITRON4.0仕様

- ASPカーネルは理想的なμITRON4.0仕様
 - ASPカーネルで拡張されたアラームハンドラや割込みの静的APIはμITRON4.0仕様に記載された仕様
 - アプリ側でitron.hをインクルードすれば、μITRON4.0仕様の型等の定義は全て使用可能と
 - カーネル内部の機能拡張は、APIを規定するμITRONの仕様の適用外
- ASPカーネルで失ったRTOSのポータビリティ
 - JSPカーネルでは変動部はC言語、アセンブラ記述であった
 - ASPカーネルで割込みアーキテクチャをRTOSで規定したことにより、ターゲット依存部の割込み機能をポーティングするために、割込みアーキテクチャ部構築のために、コンフィギュレーション部の改造が必須となった
 - これにより、サンプル実装されたARM系以外のポーティングの難易度が、格段に上がってしまった

TOPPERS/ASPでの型宣言

- TOPPERS新世代カーネルでは、ITRON仕様のいくつかの型を廃止し、明示的に型内容の分かる型に変更した
- ITRON仕様型、定数についても、itron.hインクルードファイルをインクルードすれば使用が可能です
 - 本実習では、アプリケーションをμITRON4.0レガシー・コートとして実習する(新しい型はアプリケーションでは使用しない)

ITRON型

新しい型

B	int8_t	W	int32_t	VP	void *
UB	uint8_t	UW	uint32_t	INT	int_t
VB	uint8_t	VW	uint32_t	UINT	uint_t
H	int16_t	D	int64_t	BOOL	bool_t
UH	uint16_t	UD	uint64_t	VP_INT	intptr_t
VH	uint16_t	VD	uint64_t		

その他の型とTOPPERS/ASPでの定数の変更

- 従来通り使用するITRON型は以下のとおりです。これらはRTOSの仕様に依存した重要な型です

FN	機能コード	SIZE	メモリ領域のサイズ
ER	エラーコード	TMO	タイムアウト設定
ID	オブジェクトID番号	RELTIM	相対時間
ATR	オブジェクトの属性	SYSTIM	システム時刻
STAT	オブジェクトの状態	SYSUTM	性能測定用システム時間
MODE	サービスコールの動作モード	FP	プログラムの起動番地
PRI	優先度		

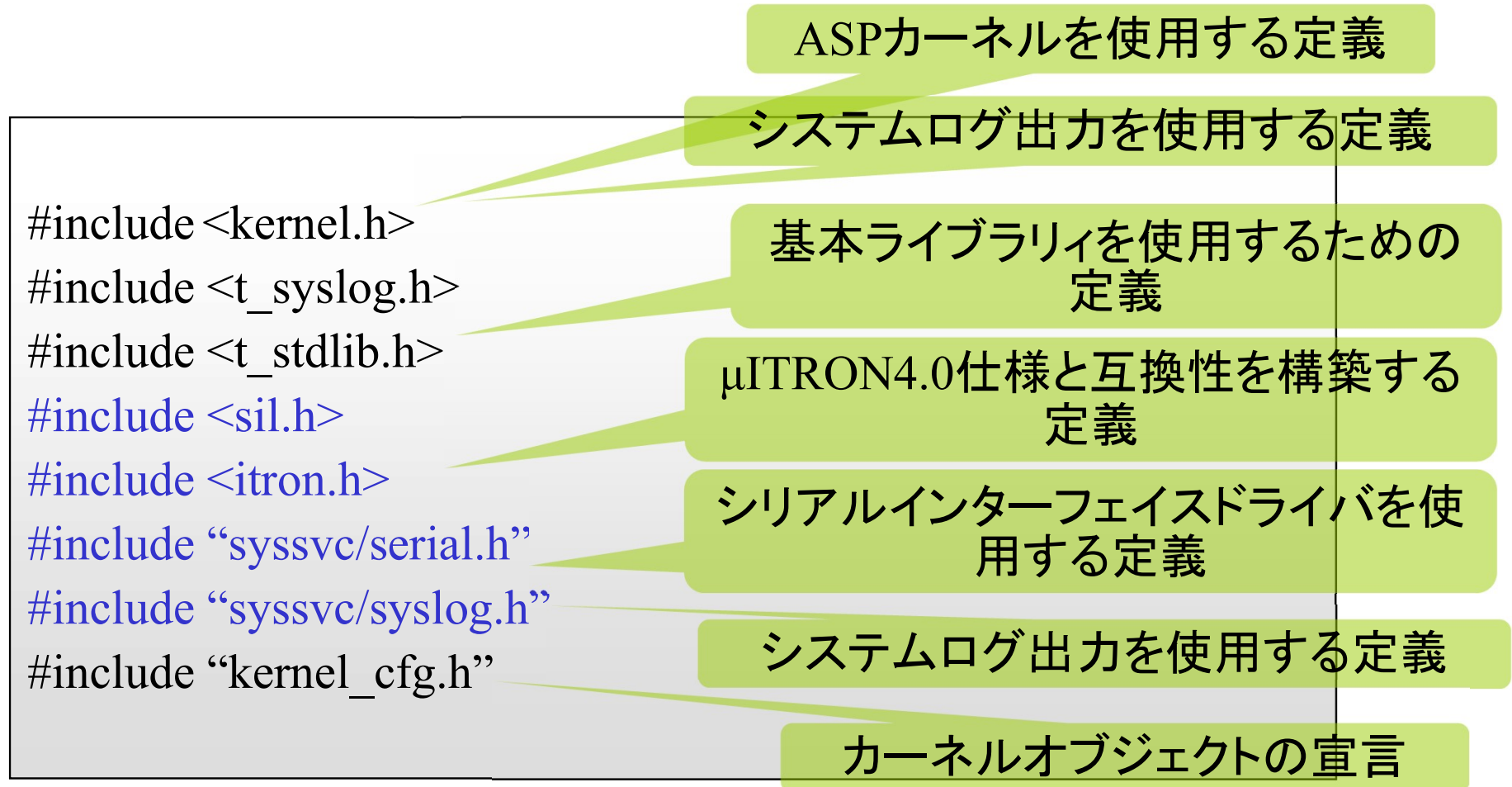
- 真偽判定定数は以下のように変更します

TRUE ➡ true

FALSE ➡ false

TOPPERS/ASPの環境インクルード手順

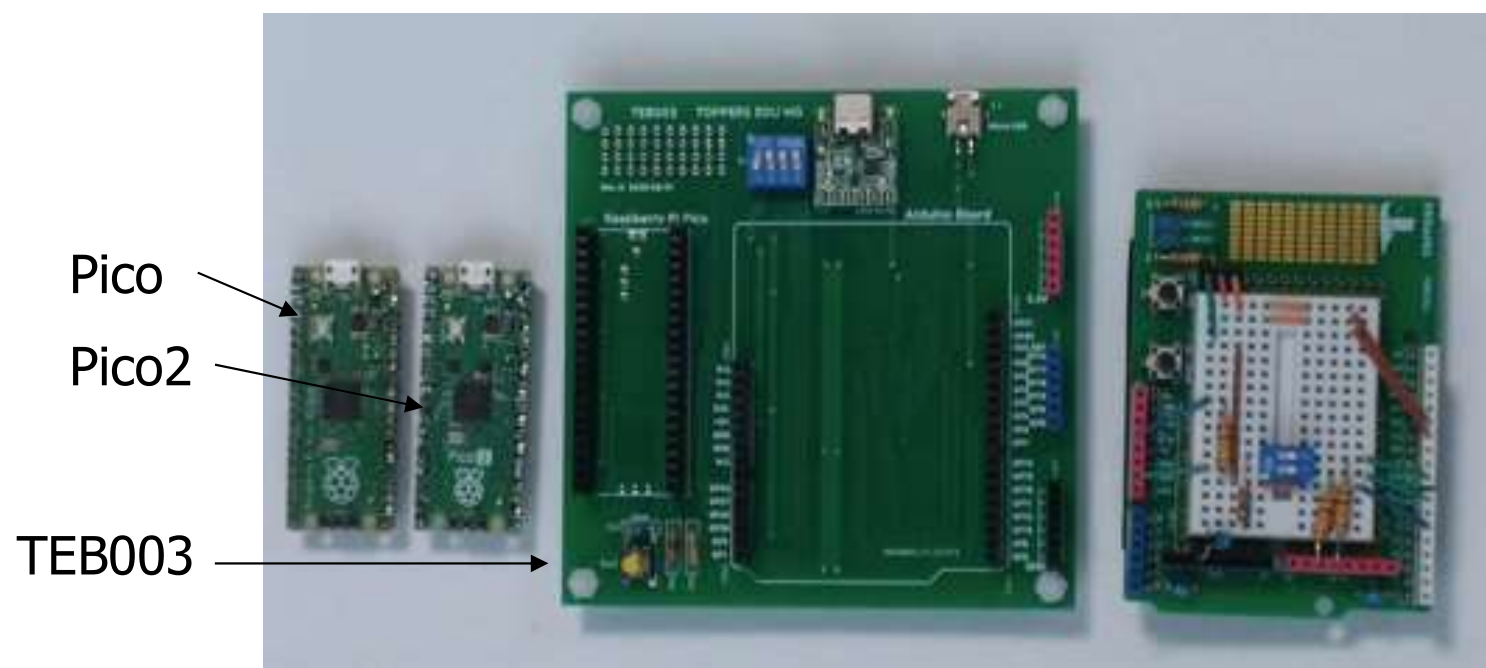
- 以下にTOPPERS/ASPを使用するためのインクルードファイルを示します



開発環境の構築

ボードの確認とソフトウェアのセットアップ

- パソコン上に開発環境がセットアップされていない場合は、開発環境をセットアップする必要がある
- 開発環境のセットアップ手順は開発環境編、開発環境の確認「ソフトウェア環境の構築」の記載してあります



TOPPERS/ASPカーネルの導入

1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

μITRON導入実習：実習内容

- TOPPERS/ASPカーネルの導入実習を行う
- TOPPERS/ASPカーネルのビルドについての説明
- カーネルのダウンロードと環境づくり
 - ダウンロードとツールを作成します
- 開発・実行環境の使用方法についての説明
 - 開発環境と2種類の実行環境の使用方法について解説
- カーネルのビルドと実行(デバグガ使用版)
 - Raspberry Pi Pico/Pico2用カーネルをビルドし、サンプルプログラムを実行します
- サンプルプログラムの説明
 - サンプルプログラムの説明と実行確認

TOPPERS/ASPカーネルの導入

1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

TOPPERS/ASPカーネルのセットアップ

- パソコンにTOPPERS/ASP開発環境がセットアップされていない場合は、セットアップを行う必要がある
- TOPPERS/ASP開発環境のセットアップは、開発環境編、「TOPPERS/ASPカーネルの導入」に記載してあります

TOPPERS/ASPカーネルの導入

1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

TOPPERS/ASPカーネルのダウンロード

- TOPPERSプロジェクトのWebからasp-1.9.3の個別パッケージをダウンロードします
- 同様にARM Coretex-M0アーキテクチャ・GCC依存部パッケージasp_arch_arm_m0_gcc-1.9.8をダウンロードします

TOPPERS/ASPカーネル個別パッケージのダウンロード

TOPPERS/ASPカーネルの最新リリースの個別パッケージを配布しています。各ターゲットに対応するカーネル非依存部パッケージ、依存部パッケージ（アーキテクチャ依存部、ターゲット依存部）、シリアルドライバ依存部(PDIC)パッケージを個別にダウンロードできます。

ターゲットごとに必要なソースコードを一つにまとめた簡易パッケージは、[こちら](#)からダウンロードできます。

一般公開以前のバージョン(Release 1.3.0以前)に対応した個別パッケージは、会員向け早期リリースとしての扱いですが、[こちら](#)からダウンロードできます。

ターゲット非依存部

TOPPERS/ASPカーネルターゲット非依存部パッケージ(相当:名古屋大学)

パッケージ	リリース日
asp-1.9.3.tar.gz	2017-04-29
release 1.9.2との違い	
asp-1.9.2.tar.gz	2015-05-30
release 1.9.1との違い	

ARM Cortex-M0アーキテクチャ・GCC依存部パッケージ (担当: TOPPERSプロジェクト教育WG)			
パッケージ	ディレクトリ	対応する非依存部のバージョン	リリース日
p_arch_arm_m0_gcc-1.9.8.tar.gz	arch/arm_m_gcc target/om13058_gcc target/raspberrypi_pico_gcc target/stm32f091nucleo64_gcc target/stm32l073nucleo64_gcc target/stm32g071nucleo64_gcc target/stm32g0b1nucleo64_gcc target/stm32loradiscovery_gcc	1.9.*	2025-02-25
p_arch_arm_m0_gcc-1.9.7.tar.gz	arch/arm_m_gcc target/om13058_gcc target/raspberrypi_pico_gcc target/stm32f091nucleo64_gcc target/stm32l073nucleo64_gcc target/stm32g071nucleo64_gcc target/stm32g0b1nucleo64_gcc target/stm32loradiscovery_gcc	1.9.*	2024-09-24

TOPPERS BASE PLATFORMのダウンロード

- TOPPERSプロジェクトのウェブからASPカーネル用の最新のTOPPERS BASE PLATFORM(RP)をダウンロードします

TOPPERS BASE PLATFORM とは

TOPPERS BASE PLATFORMは、組み込み教育用にTOPPERSプロジェクト教育ワーキンググループが開発した組み込みソフトウェアプラットフォームです。実装にあたっては、μITRON4.0仕様研究会のデバイスドライバ設計ガイドラインを参考に実装しました。プラットフォームとして、STマイクロエレクトロニクス社のSTM32マイクロコントローラとRISC-Vをターゲットに作成した[TOPPERS BASE PLATFORM\(ST/RV\)](#)と、上級のハードウェアをターゲットとしたIntel社のCyclone V SoCをターゲットとした[TOPPERS BASE PLATFORM\(CV\)](#)があります。以下に2つのプラットフォームの概要を記載します。

対応教育教材

TOPPERS BASE PLATFORM(ST)を使用した組み込み実装教材は以下の通りです。

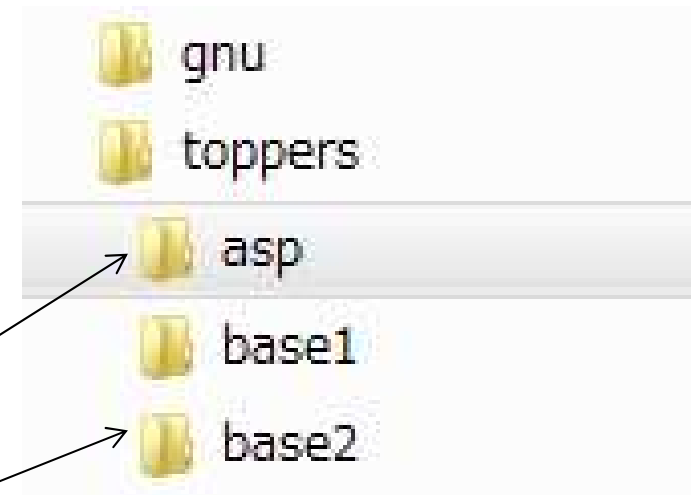
1. [基礎1](#)、[基礎2](#)、[基礎3](#)

開発成果物のダウンロード

TOPPERS BASE PLATFORM(ST/RV/RP) : 担当TOPPERS教育WG		
パッケージ	マニュアル	リリース日
asp_baseplatform-1.5.0.tar.gz	V1.5.0 Reference manual(ST) V1.5.0 Reference manual(RV) V1.5.0 Reference manual(RP)	2025-07-29
asp_baseplatformv1.4.6_052224.tar.gz	V1.4.6 Reference manual(ST) V1.4.6 Reference manual(RV) V1.4.6 Reference manual(RP)	2024-05-22

TOPPERS/ASPを構築するディレクトリの位置

- TOPPERS/ASPの開発ディレクトリは基礎2、基礎3で共通に使用します
- 2日目の実習プログラムのディレクトリ(base2)と並列の位置にaspのディレクトリを作成してください



base2の実習ディレクトリとaspは並列に作成

TOPPERS/ASP :Picoの解凍

- 開発用のディレクトリにasp-1.9.3.tar.gz、asp_arch_arm_m0_gcc-1.9.8.tar.gzと、教材中のBASE PLATFORM(RP/RV)V1.5.0(asp_baseplatform-1.5.0.tar.gz)を置きます
- asp-1.9.3.tar.gzとasp_arch_arm_m0-gcc-1.9.8.tar.gzの解凍後、asp_baseplatform-1.5.0.tar.gzを解凍します

```
$ ls
asp_arch_arm_m0_gcc-1.9.8.tar.gz asp-1.9.3.tar.gz
asp_baseplatform-1.5.0.tar.gz
$ tar zxvf asp-1.9.3.tar.gz
$ tar zxvf asp_arch_arm_m0_gcc-1.9.8.tar.gz
$ tar zxvf asp_baseplatform-1.5.0.tar.gz
```

TOPPERS/ASP Pico2の場合

- Pico2で実習を行う場合、ASPカーネル依存部は以下の通りです

Pico2の場合、表のarchを解凍してください

Pico2:Cortex-M33

asp_arch_arm_m33_gcc-1.9.3.tar.gz

Pico2:RISC-V

asp_arch_riscv_gcc-1.9.6.tar.gz

Pico2:Cortex-M33

```
$ tar zxvf asp_arch_arm_m33_gcc-1.9.3.tar.gz
```

Pico2:RISC-V

```
$ tar zxvf asp_arch_riscv_gcc-1.9.6.tar.gz
```

コンフィギュレータの構築

- aspディレクトリの下にcfg/cfgディレクトリを作成する
- Webより、コンフィギュレータのWindowsバイナリ版をダウンロードしcfg.exeを取り出す
- asp/cfg/cfgにcfg.exeを置く

TOPPERS新世代カーネル用コンフィギュレータ

TOPPERS新世代カーネル用コンフィギュレータの最新リリースを配布しています。コンフィギュレータ自体については[こちら](#)を、構築方法等についてはアーカイブに含まれる README.txt をご覧ください。

下記32bit Linux用バイナリは、次の環境でライブラリを静的リンクしてビルドしたものです。

- Ubuntu 12.04 LTE 32bit
- boost 1.46.1
- Xerces C++ 3.1.1

64bitのLinux用のboostライブラリに現状問題があるため、64bitのLinuxでビルドしたcfgは正しく動作しません。64bitのLinuxでcfgを使用する場合は、下記の32bit Linux用バイナリを使用するか、32bitのLinuxでソースからビルドしてスタティックリンクしたバイナリを使用下さい。

最新のリリース			
リリース名	タイプ	サイズ	リリース日
コンフィギュレータ Release 1.9.6	tar.gz(EUC,LF)	150KB	2017-03-31
コンフィギュレータ Release 1.9.6 (Windows用バイナリ)	zip	10.5MB	2017-03-31
コンフィギュレータ Release 1.9.6 (32bit Linux用バイナリ)	gz	7.0MB	2017-03-31

[Release 1.9.5との違い](#)

TOPPERS/ASPのワークスペースの作成

- aspディレクトリの下に
OBJ/RASBERRYPI_PICO_GCC/SAMPLE1ディレクトリ
を作成します
- SAMPLE1ディレクトリに入り、configureコマンドを使用し
てSAMPLE1プログラムを生成します
- 以下のファイルが生成されます
 - Makefile,sample1.cfg,sample1.c,sample1.h

Pico:Cortex-M0+

```
$ cd RASPBERRYPI_PICO_GCC
$ mkdir SAMPLE1
$ cd SAMPLE1
$ ../../../../configure -T raspberrypi_pico_gcc
$ ls
Makefile sample1.c sample1.cfg sample1.h
```

Pico2用のワークスペース

- Pico2を使用する場合は別のワークスペースとなる
 - Pico2:RISC-VはRASPBerryPI_PICO2RV_GCCがないので作る必要がある

Pico2:Cortex-M33

RASPBerryPI_PICO2_GCC

Pico2:RISC-V

RASPBerryPI_PICO2RV_GCC

Pico2:Cortex-M33

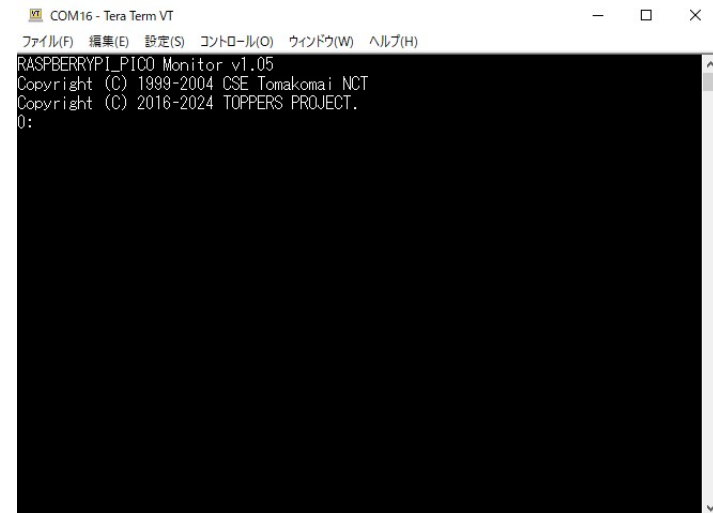
```
$ cd RASPBerryPI_PICO2_GCC
$ mkdir SAMPLE1
$ cd SAMPLE1
$ ../../../../configure -T raspberrypi_pico2_gcc
```

Pico2:RISC-V

```
$ mkdir RASPBerryPI_PICO2RV_GCC
$ cd RASPBerryPI_PICO2RV_GCC
$ mkdir SAMPLE1
$ cd SAMPLE1
$ ../../../../configure -T raspberrypi_pico2rv_gcc
```

開発環境の確認

- 基礎1講座で、ROMモニタをBOOT-SWオンでPCに接続し、表示したフォルダにuf2ファイルをドロップして書き込みました
- TEB003のUSBシリアルにUSBケーブルを接続
- TeraTermを以下の設定で起動します
 - COMn:nはUSBシリアルのCOM番号
 - 115200bps
 - データ8bit
 - パリティなし
 - ストップビット1
 - フロー制御なし
- PicoまたはPico2のUSBにUSBケーブルを接続して給電



UF2ファイルへのコンバージョン

- FLASH-ROM書き込み用にUF2ファイルに変換するコマンド: bin2uf2は基礎1で実装済みとする
- 作成したsample1のMakefileにUF2変換用のコマンドを追加する

```
#
# 全体のリンク
#
$(OBJFILE): $(APPL_CFG) kernel cfg.timestamp $(ALL_OBJS) $(LIBS_DEP)
$(LINK) $(CFLAGS) $(LDFLAGS) -o $(OBJFILE) $(START_OBJS) ¥
$(APPL_OBJS) $(SYSSVC_OBJS) $(CFG_OBJS) $(ALL_LIBS) $(END_OBJS)
$(NM) -n $(OBJFILE) > $(OBJNAME).syms
$(OBJCOPY) -O srec -S $(OBJFILE) $(OBJNAME).srec
ifeq ($(DBGENV),ROM)
$(OBJCOPY) -O binary -S $(OBJFILE) $(OBJNAME).bin
bin2uf2 $(OBJNAME).bin -family 0
endif
$(CFG) --pass 3 --kernel asp $(INCLUDES) ¥
--rom-image $(OBJNAME).srec --symbol-table $(OBJNAME).syms ¥
-T $(TARGETDIR)/target_check.tf $(CFG_TABS) $<
```

Pico2の場合は、
bin2uf2 \$(OBJNAME).bin) -family 1 としてください

SAMPLE1のビルド

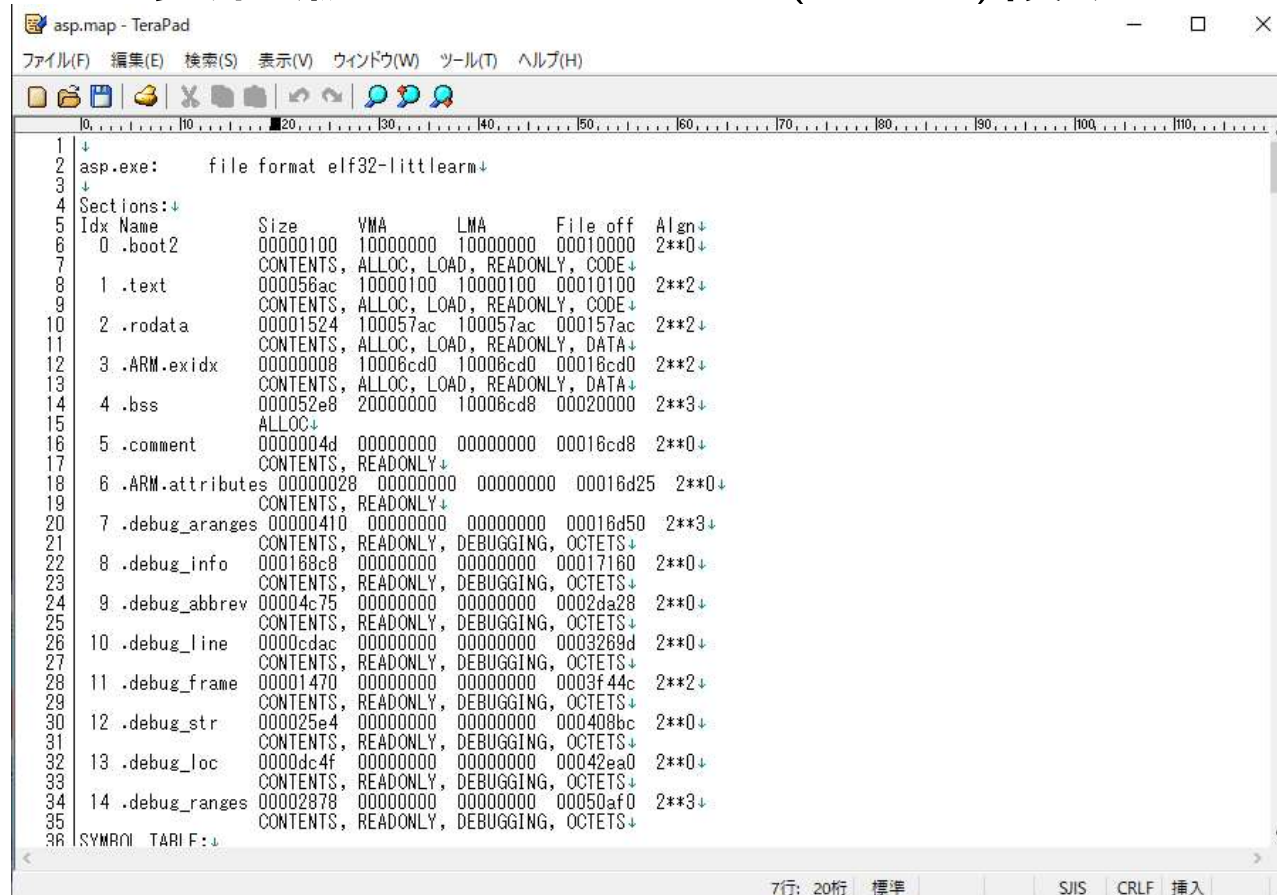
- ワークスペースで、SAMPLE1プログラムのビルドを行います
- Pico/Pico2用のTOPPERS/ASPはDBGENV環境スイッチの指定で実行環境を変えられます
 - DBGENV=RAMまたはなしでRAM実行版
 - DBGENV=ROMでROM実行版
- 今回はROM実行版を作成します
- 確認のためにROMファイルを作成します

```
$ make DBGENV=ROM  
$ arm-none-eabi-objdump -h -t asp.exe > asp.map
```

Pico2:RISC-Vの場合は、arm-none-eabiの部分がRISC-Vコンパイラ名となる

実行アドレスの確認

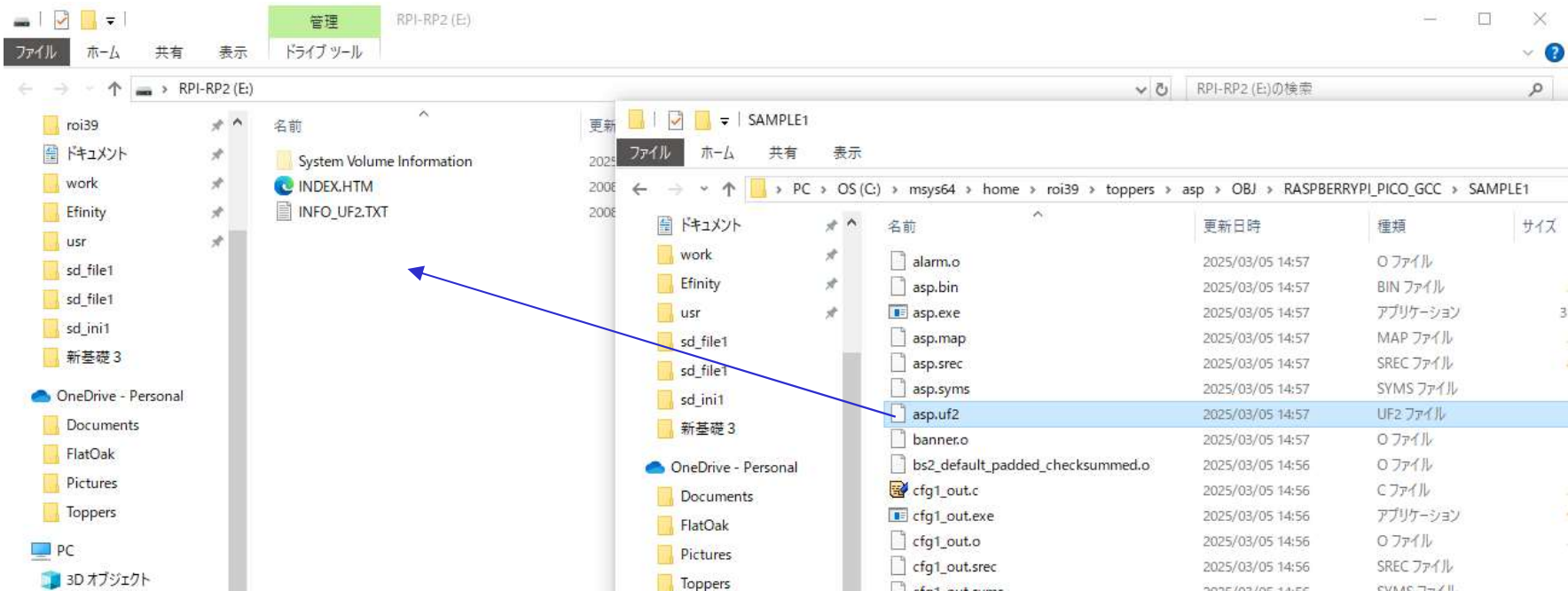
- asp.mapファイルをエディタを使って開いてください
- .textのアドレスが0x10000000 (ROM) 領域となっています
 - RAM実行版は0x20000000 (RAM) 領域



```
1 ↓
2 asp.exe:   file format elf32-littlearm↓
3 ↓
4 Sections:↓
5 Idx Name      Size      YMA      LMA      File off  Algn↓
6 0 .boot2      00000100  10000000 10000000 00010000  2**0↓
7 CONTENTS, ALLOC, LOAD, READONLY, CODE↓
8 1 .text       000058ac  10000100 10000100 00010100  2**2↓
9 CONTENTS, ALLOC, LOAD, READONLY, CODE↓
10 2 .rodata     00001524  100057ac 100057ac 000157ac  2**2↓
11 CONTENTS, ALLOC, LOAD, READONLY, DATA↓
12 3 .ARM.exidx  00000008  10006cd0 10006cd0 00016cd0  2**2↓
13 CONTENTS, ALLOC, LOAD, READONLY, DATA↓
14 4 .bss        000052e8  20000000 10006cd8 00020000  2**3↓
15 ALLOC↓
16 5 .comment    0000004d  00000000 00000000 00016cd8  2**0↓
17 CONTENTS, READONLY↓
18 6 .ARM.attributes 00000028  00000000 00000000 00016d25  2**0↓
19 CONTENTS, READONLY↓
20 7 .debug_aranges 00000410  00000000 00000000 00016d50  2**3↓
21 CONTENTS, READONLY, DEBUGGING, OCTETS↓
22 8 .debug_info  000188c8  00000000 00000000 00017160  2**0↓
23 CONTENTS, READONLY, DEBUGGING, OCTETS↓
24 9 .debug_abbrev 00004c75  00000000 00000000 0002da28  2**0↓
25 CONTENTS, READONLY, DEBUGGING, OCTETS↓
26 10 .debug_line  0000cdac  00000000 00000000 0003269d  2**0↓
27 CONTENTS, READONLY, DEBUGGING, OCTETS↓
28 11 .debug_frame 00001470  00000000 00000000 0003f44c  2**2↓
29 CONTENTS, READONLY, DEBUGGING, OCTETS↓
30 12 .debug_str   000025e4  00000000 00000000 000408bc  2**0↓
31 CONTENTS, READONLY, DEBUGGING, OCTETS↓
32 13 .debug_loc   0000dc4f  00000000 00000000 00042ea0  2**0↓
33 CONTENTS, READONLY, DEBUGGING, OCTETS↓
34 14 .debug_ranges 00002878  00000000 00000000 00050af0  2**3↓
35 CONTENTS, READONLY, DEBUGGING, OCTETS↓
36 SYMDEF TAB F:↓
```

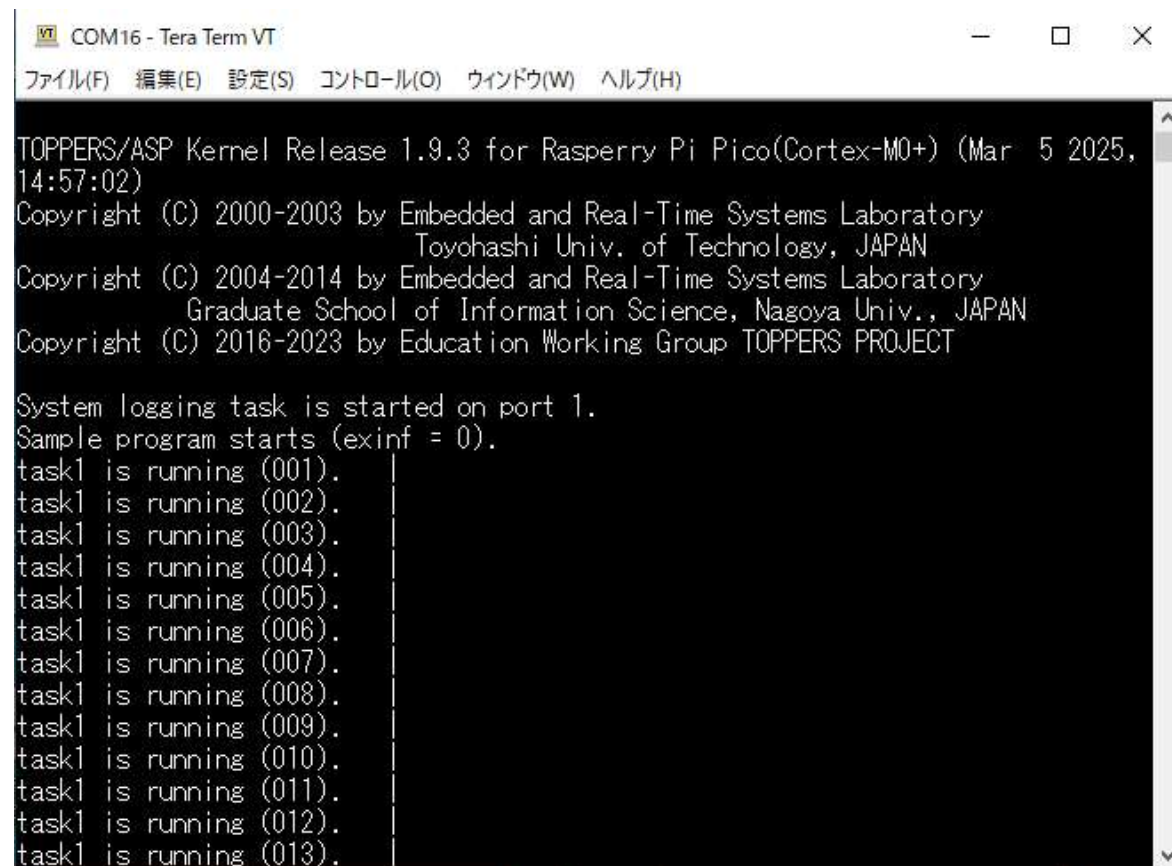
プログラムをFLASH-ROMに書き込む

- BOOT-SWを押しながら、USBケーブルを接続
- フォルダが表示されたら、asp.uf2をドロップする



ROM実行版の実行

- 書込みが終了すると、SAMPLE1が実行します
- この後の実習はROM版で行いますが、以後の実習はRAM版で行いますのでRAM実行版を確認します



```
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)

TOPPERS/ASP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  5 2025,
14:57:02)
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

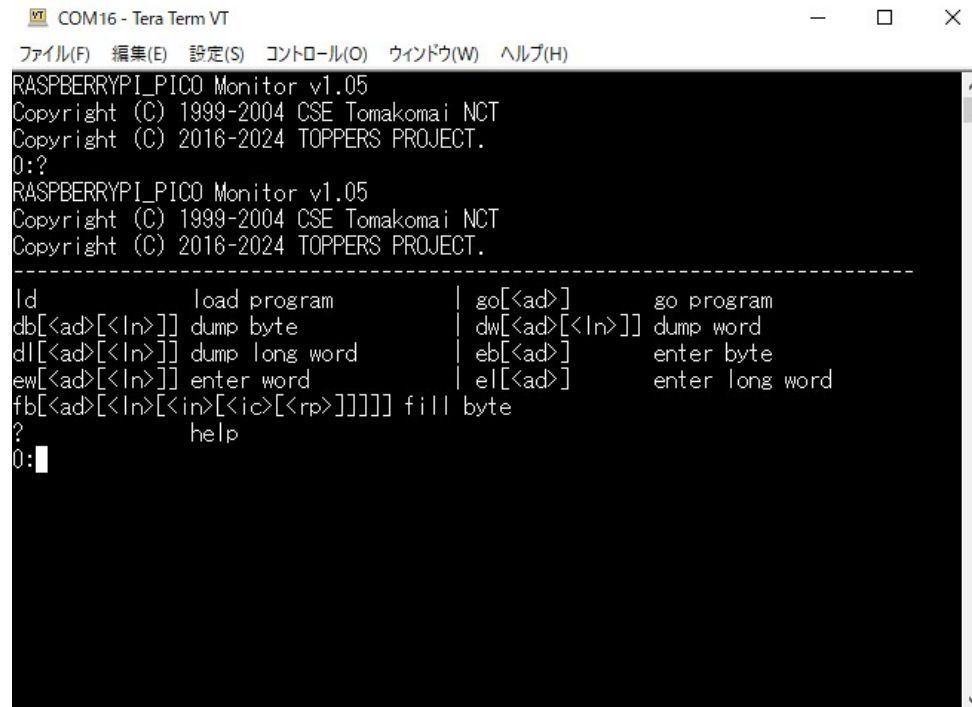
System logging task is started on port 1.
Sample program starts (exinf = 0).
task1 is running (001).
task1 is running (002).
task1 is running (003).
task1 is running (004).
task1 is running (005).
task1 is running (006).
task1 is running (007).
task1 is running (008).
task1 is running (009).
task1 is running (010).
task1 is running (011).
task1 is running (012).
task1 is running (013).
```

ROMモニタの書き込みと開発環境の確認

- もう一度、BOOT-SWを押して、ROMモニタを書き込みます
- ROMモニタの確認を行う

Pico:Cortex-M0+	rommon_pico.uf2
Pico2:Cortex-M33	rommon_pico2.uf2
Pico2:RISC-V	rommon_pico2rv.uf2

ROMモニタは
tools/rommonにある



```
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)
RASPBERRYPI_PICO Monitor v1.05
Copyright (C) 1999-2004 CSE Tomakomai NCT
Copyright (C) 2016-2024 TOPPERS PROJECT.
0:?
RASPBERRYPI_PICO Monitor v1.05
Copyright (C) 1999-2004 CSE Tomakomai NCT
Copyright (C) 2016-2024 TOPPERS PROJECT.
-----
ld          load program          go[<ad>]      go program
db[<ad>[<ln>]] dump byte          dw[<ad>[<ln>]] dump word
dl[<ad>[<ln>]] dump long word      eb[<ad>]      enter byte
ew[<ad>[<ln>]] enter word           el[<ad>]      enter long word
fb[<ad>[<ln>[<ln>[<ic>[<rp>]]]]] fill byte
?          help
0:█
```

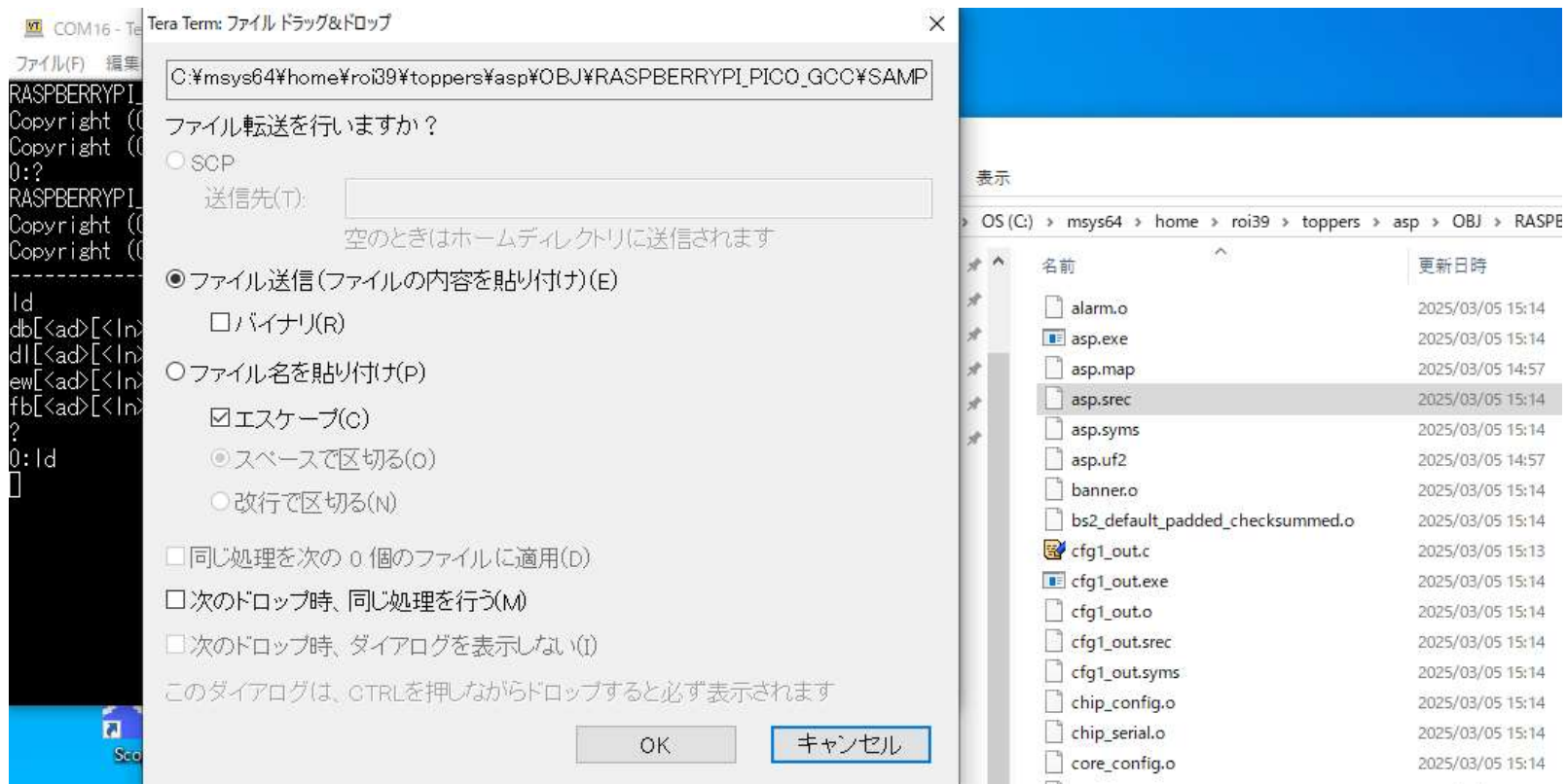
RAM版のビルド

- Makeコマンドで再ビルドして、RAM実行するasp.srecを作成する

```
$ make clean  
$ make
```

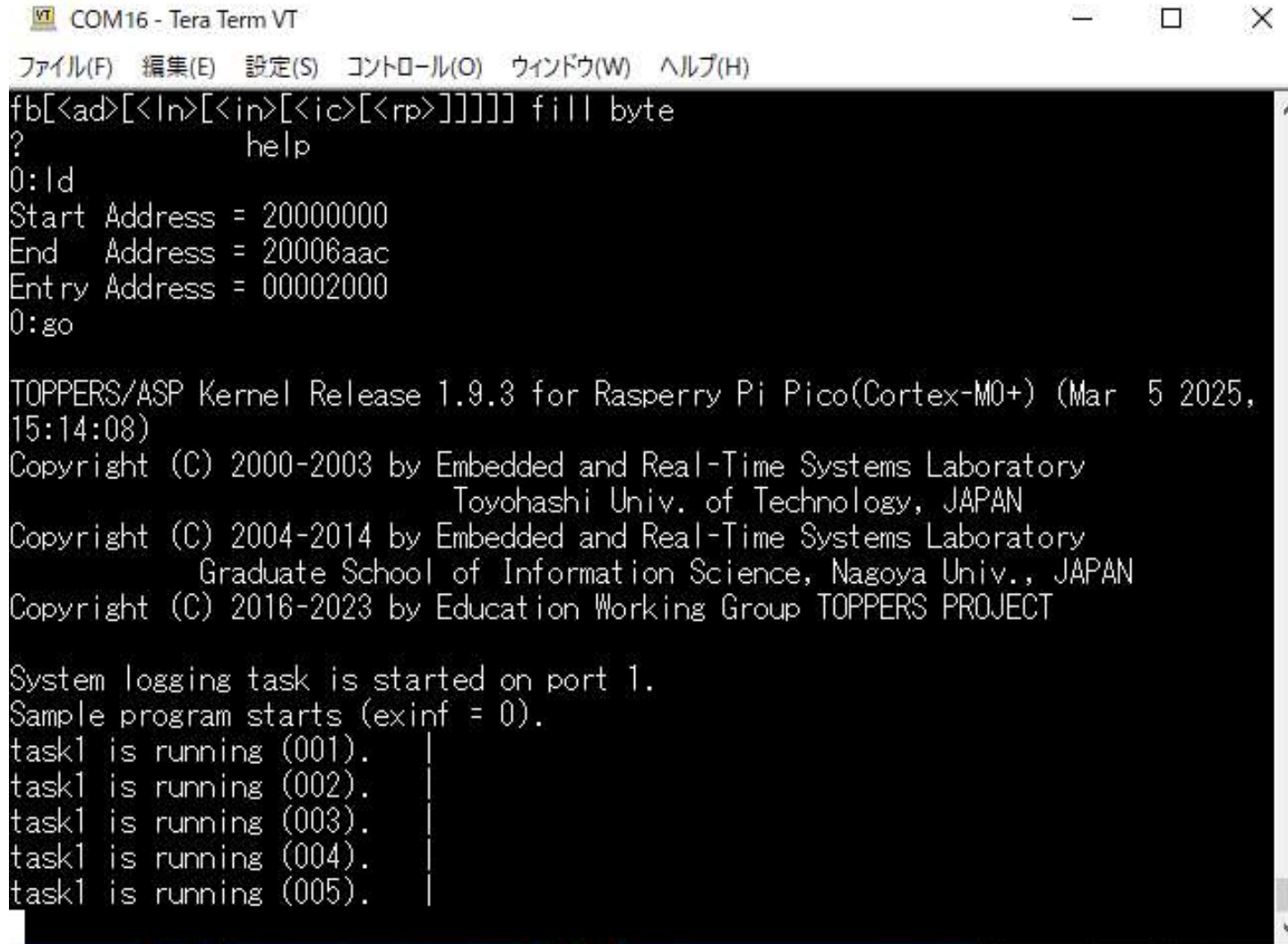
ダウンロード

- ROMモニタ上でldコマンドを実行して、ダウンロードモードにします
- ビルドされたasp.srecをTeraTermに重ねてダウンロードしてください



実行

- ダウンロードが終了したら、goコマンドで実行してください



```
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)
fb[<ad>[<ln>[<in>[<ic>[<rp>]]]]] fill byte
? help
0:ld
Start Address = 20000000
End Address = 20006aac
Entry Address = 00002000
0:go

TOPPERS/ASP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar 5 2025,
15:14:08)
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.
Sample program starts (exinf = 0).
task1 is running (001).
task1 is running (002).
task1 is running (003).
task1 is running (004).
task1 is running (005).
```

TOPPERS/ASPカーネルの導入

1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

sample1プログラムの内容

- sample1プログラムはひとつのメインタスクと3つの並列実行タスクで構成される
- メインタスクは種々のRTOS評価用メッセージを並列実行タスクに送り、並列実行タスクが表示するメッセージにてTOPPERS/ASPのサービスコールと動作の評価を行う
- 並列実行タスクは同一優先度で、待ち状態を作らずに実行する

タスクID	優先度	タスク関数	引数	機能
TASK1	10	task	1	並列実行されるタスクは、task_loop 回空ループを実行する度に、タスク* が実行中であることをあらわすメッセージを表示する。
TASK2	10	task	2	
TASK3	10	task	3	
MAIN_TASK	5	main_task	0	コマンドの読み込みと並列タスクへの通知

sample1用コマンド1

- sample1のコマンドは、並列実行タスクに対するコマンドと、全体の制御を行うコマンドがあります
- 以下に全体の制御に関するコマンドを記載します

コマンド	機能
1	以後のコマンドの対称を並列実行タスク1とする
2	以後のコマンドの対称を並列実行タスク2とする
3	以後のコマンドの対称を並列実行タスク3とする
v	発行したシステムコールを表示する(デフォルト)
q	発行したシステムコールを表示しない
r	三つの優先度(9、10、11)のレディキューを回転させる
Q	プログラムを終了する
Ctrl-C	プログラムを終了する

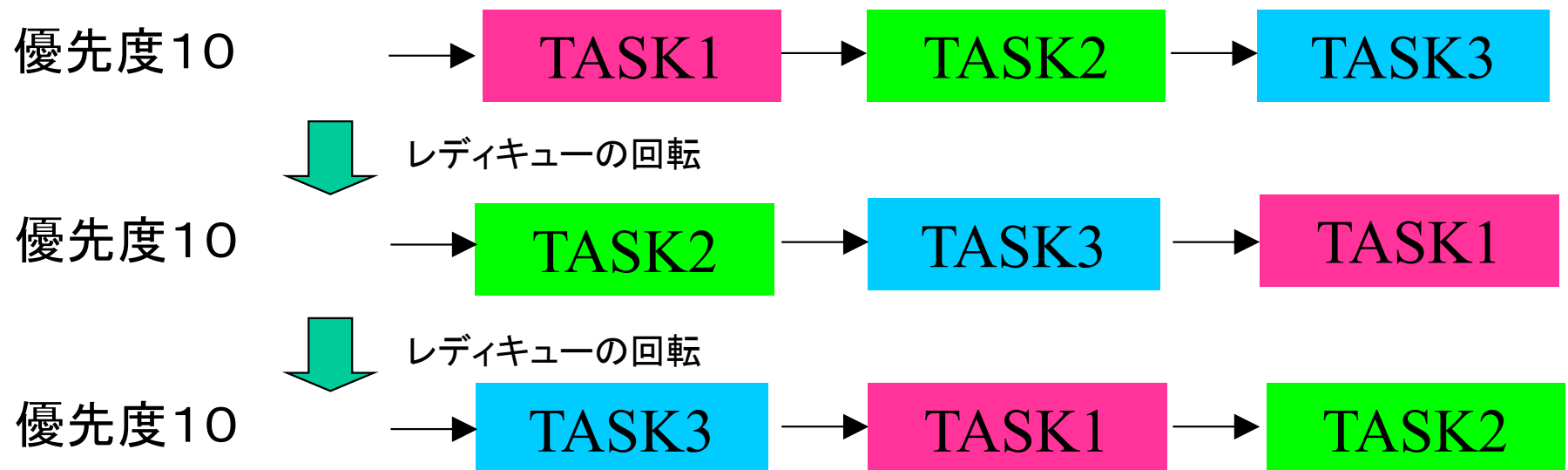
sample1コマンド2

- 並列実行タスクに対する簡単なコマンドは以下の通り

コマンド	機能
a	タスクをact_tskにより起動する
A	タスクに対する起動要求をcan_actによりキャンセルする
e	タスクにext_tskを呼び出させ、終了させる
t	タスクをter_tskにより強制終了させる
s	タスクにslp_tskを呼び出させ、起床待ちにさせる
S	タスクにtslp_tsk(10秒)を呼び出させ、起床待ちにさせる
w	タスクをwup_tskにより起床させる
W	タスクに対する起床要求をcan_wupによりキャンセルする
l	タスクをrel_waiにより強制的に待ち状態にする
>	タスクの優先度を9(HIGH_PRIORITY)にする
=	タスクの優先度を10(MID_PRIORITY)にする
<	タスクの優先度を11(LOW_PRIORITY)にする
d	タスクにdly_tsk(10秒)を呼び出させ、時間経過待ちにさせる

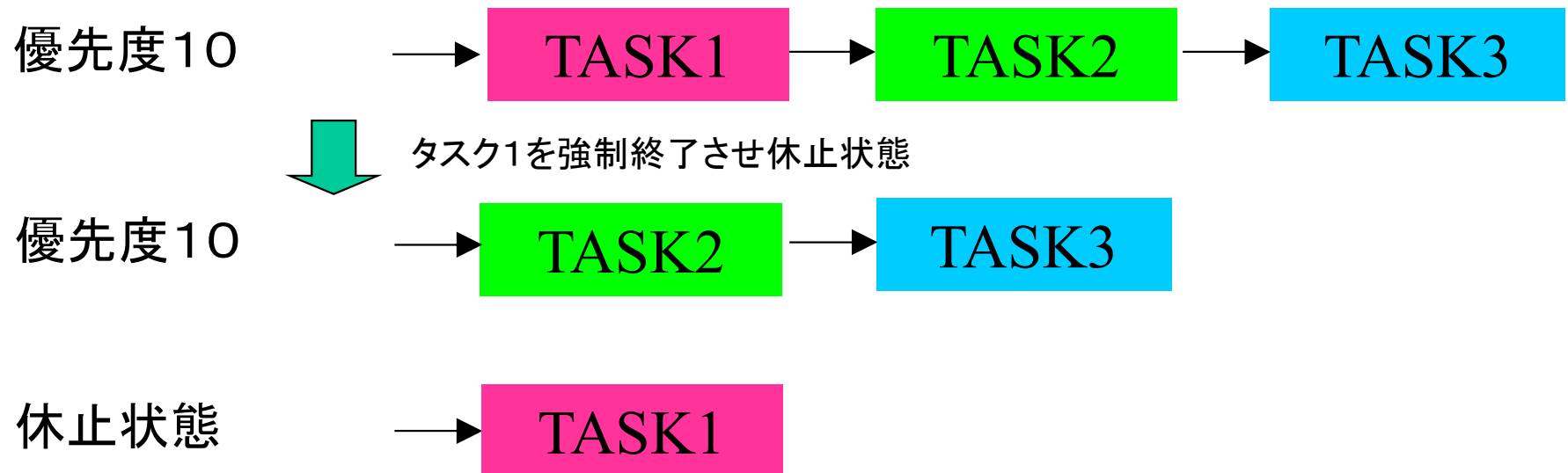
演習:レディキューの回転

- 起床時、3つの並列実行タスクは優先度10でレディ状態ですが、並列実行タスクには待ち状態がないため、レディキューの先頭のTASK1が常に実行されます
 - rコマンドでレディキューを回転させ、他のタスクの実行を確認しましょう
- 起動時の優先度10(MID_PRIORITY)のレディキューの状態



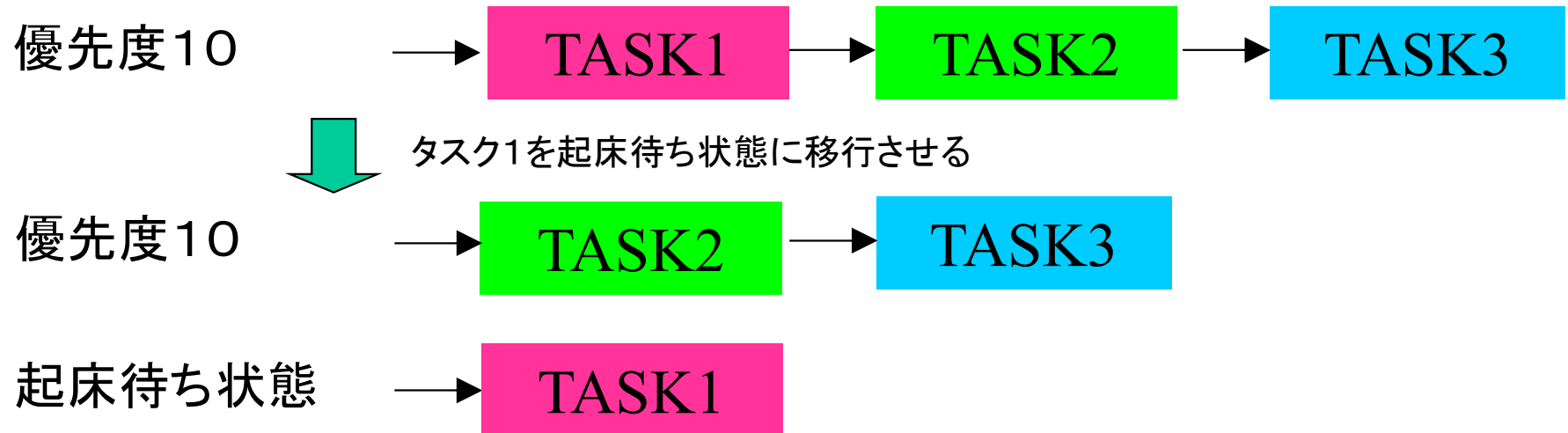
演習：タスクの起床と終了

- tコマンドを用いて、TASK1を強制終了させ、レディキューの回転を行っても、TASK1が実行しないことを確認してください
- aコマンドでTASK1を起動させ、レディキューの回転で、TASK1が実行されることを確認してください



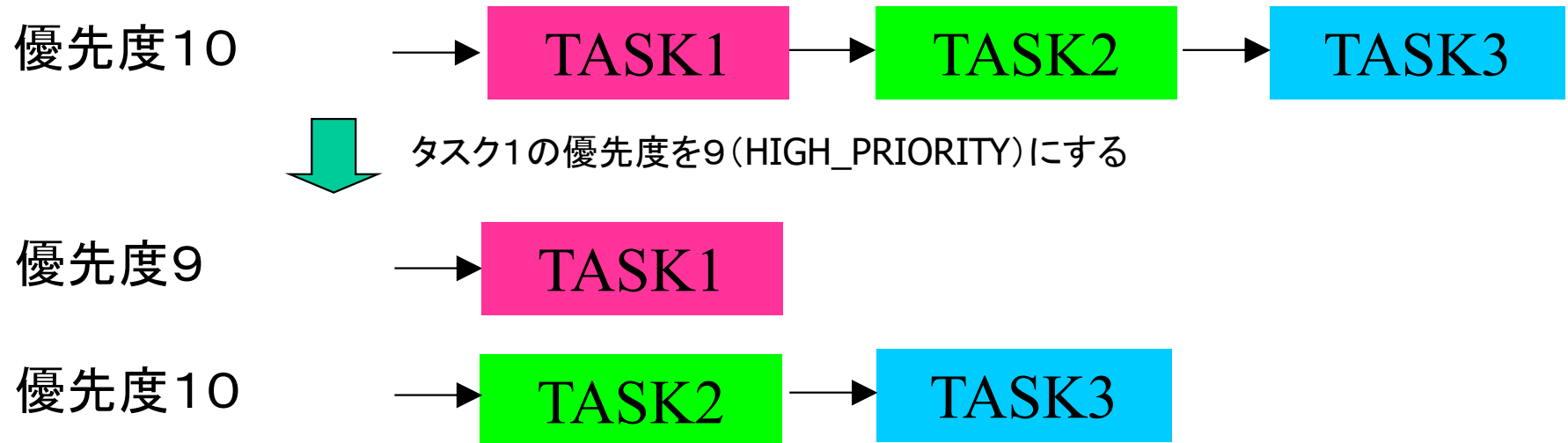
演習:タスクの起床待ち

- sコマンドでTASK1を起床待ち状態に、レディキューの回転でTASK1が実行されないことを確認してください
- wコマンドでTASK1を起床させ、レディキューの回転でTASK1が実行されることを確認してください
- dコマンドでTASK1を時間待ち状態にさせ、10秒間はレディキューの回転を行っても実行しないことを確認してください



演習：優先順位の変更

- >コマンドでTASK1の優先度を高くして、レディキューの回転を行っても、TASK1しか実行しないことを確認してください
- <コマンドでTASK1の優先度を低くして、レディキューの回転を行っても、TASK1が実行されないことを確認してください



システム検証モジュールの導入

1. タスクモニタの導入

2. タスクモニタの改造

2-1. タスク実行時間の測定

2-2. LED用拡張コマンド

タスクモニタの導入

- RTOSを使用した組込みシステムのメンテナンス性を向上するために、タスクモニタを使用することが有用な手段である
- TOPPERS/ASPにタスクモニタを導入する手順を、開発環境編、「タスクモニタの導入」に記載している
- 以下の演習は、タスクモニタの導入後に行ってください

タスクモニタの取り込み1

- 教育WGで開発したタスクモニタ付のサンプルプログラムを作成する
- デバイスドライバ取り込みを行います
 - モニタプログラムの生成
 - Makefileを修正します
 - コンフィグレーションファイルを修正します

タスクモニタの取り込み2(デバイスドライバ)

- この演習で使用するデバイス(LED、ディップスイッチ、プッシュスイッチ)のデバイスドライバを取り込みます
- 基礎1でを使用したデバイスドライバのASP対応版を
pdic/rp2040またはrp2350ディレクトリに用意してあります

関数名	型	引数	機能
led_init	void	inptr_t exinf	LEDデバイスの初期化
led_in	uint8_t	void	LEDの設定値の取り出し
led_out	void	uint8_t led_data	LED点灯、消灯を設定
switch_dip_init	void	inptr_t exinf	ディップスイッチの初期化
switch_dip_sense	uint8_t	void	ディップスイッチの状態取得
switch_push_init	void	inptr_t exinf	プッシュスイッチの初期化
switch_push_state	uint8_t	void	プッシュスイッチの状態取得

モニタプログラムの生成

- aspディレクトリの下にOBJ/RASPBERRYPI_PICO_GCC上の新たにMON2ディレクトリを作成します
- MON2ディレクトリに入り、-t オプション付きのconfigureコマンドを使用してモニタプログラムを生成します
- 以下のファイルが生成されます
 - Makefile,sample1.cfg,sample1.c,sample1.h

Pico2の場合は、各々のワークディレクトリに作成する

```
$ cd ..  
$ mkdir MON2  
$ cd MON2  
$ ../../../../configure -T raspberrypi_pico_gcc -t ../../../../monitor  
$ ls  
Makefile sample1.c sample1.cfg sample1.h
```

新たにMON2ディレクトリを作成

デバイスドライバを取り込む

- デバイスドライバのプログラム(device.c)をシステムサービスとして取り込む
- 青字の部分を追加(Pico2の場合、armv7m.oの記載は不要)

```
178
179 #
180 # システムサービスに関する定義
181 #
182 SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/syssvc
                $(SRCDIR)/library $(SRCDIR)/pdic/rp2040
183 SYSSVC_ASMOBJS := $(SYSSVC_ASMOBJS)
184 SYSSVC_COBJS := $(SYSSVC_COBJS) banner.o syslog.o serial.o logtask.o armv7m.o
185                device.o $(CXXRTS)
186 SYSSVC_CFLAGS := $(SYSSVC_CFLAGS)
187 SYSSVC_LIBS := $(SYSSVC_LIBS)
188 INCLUDES := $(INCLUDES) -I$(SRCDIR)/pdic/rp2040
```

Pico2の場合は、rp2350

Pico2の場合は不要

タスクモニタの取り込み(CFGファイルの修正)

- sample1.cfgにdevice.cfgを取り込む
- サンプルプログラムが自動実行しないようにTA_ACTを変更

```
INCLUDE("target_timer.cfg");  
INCLUDE("syssvc/syslog.cfg");  
INCLUDE("syssvc/banner.cfg");  
INCLUDE("syssvc/serial.cfg");  
INCLUDE("syssvc/logtask.cfg");  
INCLUDE("pdic/rp2040/device.cfg");  
INCLUDE("monitor/monitor.cfg");
```

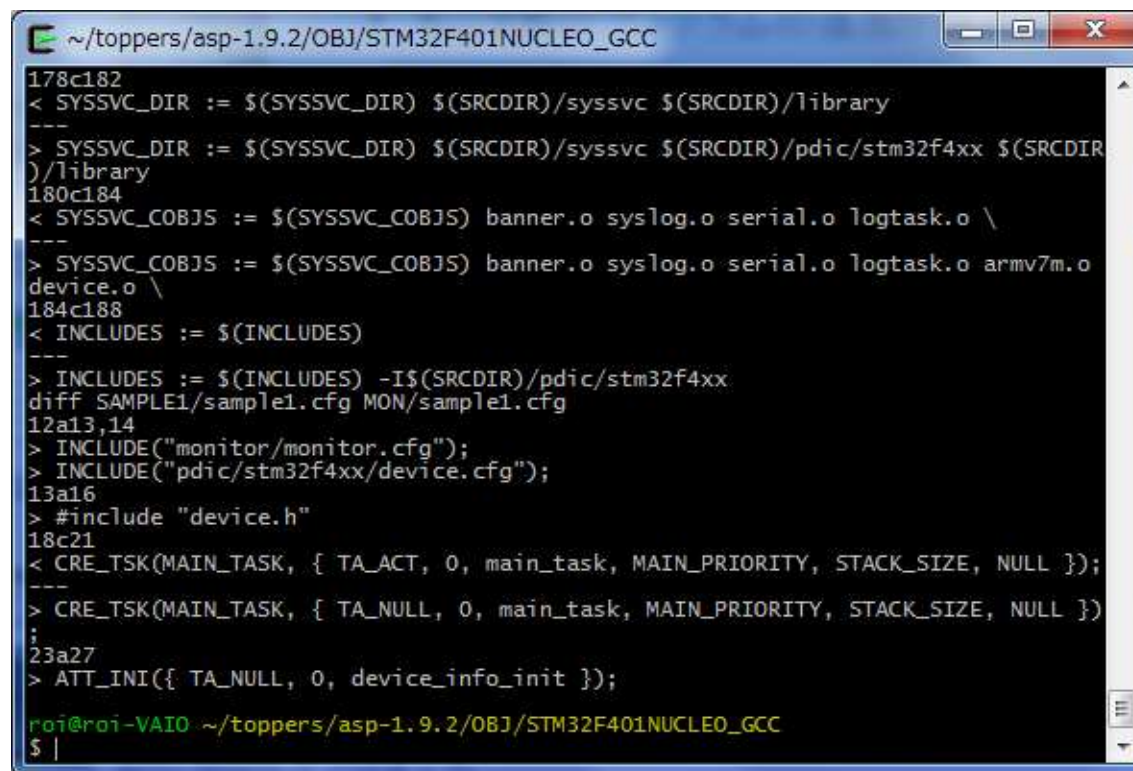
追加
Pico2の場合は、rp2350

```
#include "sample.h"  
CRE_TSK(TASK1, { TA_NULL, 1, task, MID_PRIORITY, STACK_SIZE, NULL });  
CRE_TSK(TASK2, { TA_NULL, 2, task, MID_PRIORITY, STACK_SIZE, NULL });  
CRE_TSK(TASK3, { TA_NULL, 3, task, MID_PRIORITY, STACK_SIZE, NULL });  
CRE_TSK(MAIN_TASK, { TA_NULL, 0, main_task, MAIN_PRIORITY, STACK_SIZE, NULL });  
DEF_TEX(TASK1, { TA_NULL, tex_routine });  
DEF_TEX(TASK2, { TA_NULL, tex_routine });  
DEF_TEX(TASK3, { TA_NULL, tex_routine });  
CRE_CYC(CYCHDR1, { TA_NULL, 0, cyclic_handler, 2000, 0 });  
CRE_ARM(ALMHDR1, { TA_NULL, 0, alarm_handler });
```

SAMPLE1との違いの確認

- SAMPLE1との違いを確認します
- モニタ用のプログラムの追加が確認できます

```
$ diff ../SAMPLE1 ../MON2
```



```
~/toppers/asp-1.9.2/OBJ/STM32F401NUCLEO_GCC
178c182
< SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/syssvc $(SRCDIR)/library
---
> SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/syssvc $(SRCDIR)/pdic/stm32f4xx $(SRCDIR)
)/library
180c184
< SYSSVC_COBJS := $(SYSSVC_COBJS) banner.o syslog.o serial.o logtask.o \
---
> SYSSVC_COBJS := $(SYSSVC_COBJS) banner.o syslog.o serial.o logtask.o armv7m.o
device.o \
184c188
< INCLUDES := $(INCLUDES)
---
> INCLUDES := $(INCLUDES) -I$(SRCDIR)/pdic/stm32f4xx
diff SAMPLE1/sample1.cfg MON/sample1.cfg
12a13,14
> INCLUDE("monitor/monitor.cfg");
> INCLUDE("pdic/stm32f4xx/device.cfg");
13a16
> #include "device.h"
18c21
< CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, MAIN_PRIORITY, STACK_SIZE, NULL });
---
> CRE_TSK(MAIN_TASK, { TA_NULL, 0, main_task, MAIN_PRIORITY, STACK_SIZE, NULL })
;
23a27
> ATT_INI({ TA_NULL, 0, device_info_init });

roi@roi-VAIO ~/toppers/asp-1.9.2/OBJ/STM32F401NUCLEO_GCC
$ |
```

タスクモニタのソースの確認

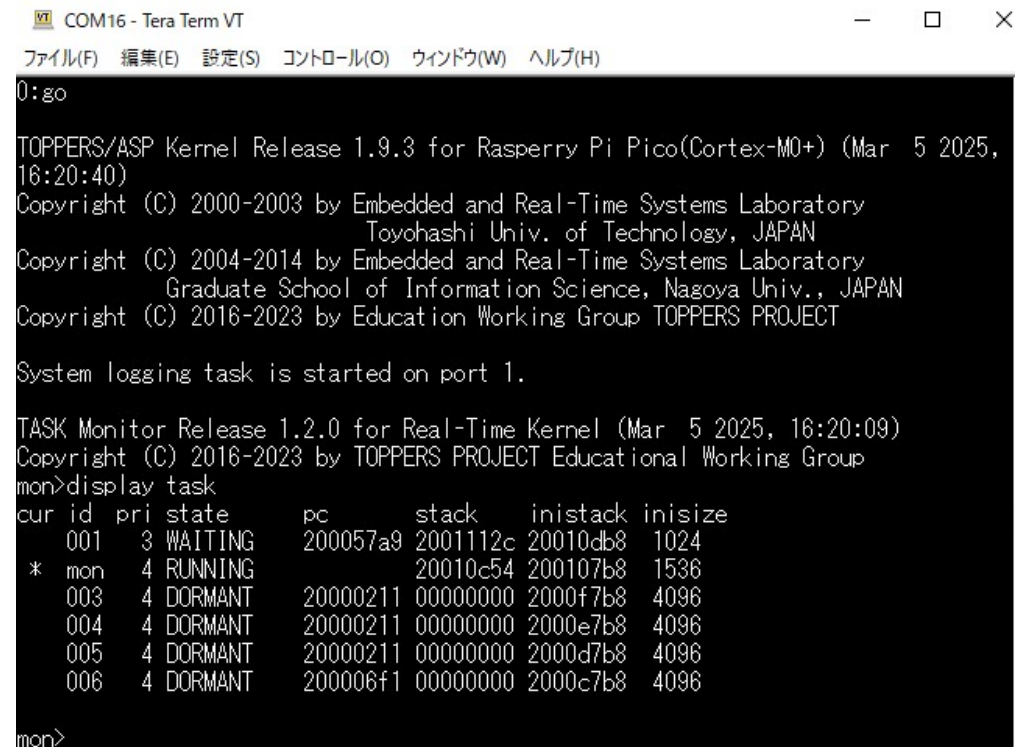
- 以下のファイルはmonitor/Makefile.configによりビルド時に自動的に取り込まれます
- タスクモニタの入出力先はstddev.cで関数設定されていますので、関数を入れ替えれば、種々のデバイスに対応が可能です

標準入出力の追加	モニタ基本部の追加	ARCH依存部の追加
stdio/stddev.c	monitor.c	arch/armv7m.c(Pico)
stdio/printf.c	task_expansion.c	arch/armv8m(Pico2)
stdio/sprintf.c		arch/riscv32(Pico2)
stdio/scanf.c		
stdio/sscanf.c		

ビルドとタスクモニタの起動を確認

- makeコマンドにて、再ビルドします
- ダウンロードして、goコマンドでタスクモニタが起動します

```
$ make
```



```
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:go

TOPPERS/ASP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Mar  5 2025,
16:20:40)
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  5 2025, 16:20:09)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>display task
cur id pri state      pc      stack  inistack inisize
  001   3 WAITING      200057a9 2001112c 20010db8 1024
* mon   4 RUNNING      20010c54 200107b8 1536
  003   4 DORMANT      20000211 00000000 2000f7b8 4096
  004   4 DORMANT      20000211 00000000 2000e7b8 4096
  005   4 DORMANT      20000211 00000000 2000d7b8 4096
  006   4 DORMANT      200006f1 00000000 2000c7b8 4096
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]
```

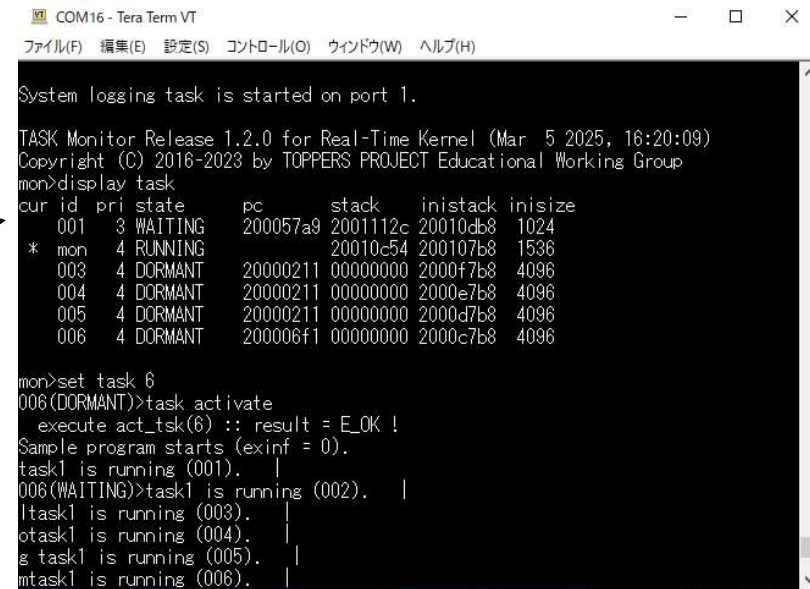
演習：サンプルプログラムの実行

- タスクモニタからサンプルプログラムを実行する
- 入力はタスクモニタが取り込むため、操作はできない
- display task : タスクの状態を表示
- set task 6 : ID番号6(メインタスク)のタスクを指定
- task activate : タスクの起動
- コマンドは最初の一文字で短縮可能

display task

set task 6

task activate



```
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.

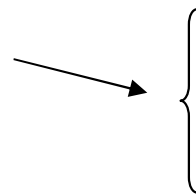
TASK Monitor Release 1.2.0 for Real-Time Kernel (Mar 5 2025, 16:20:09)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>display task
cur id pri state pc stack inistack inysize
001 3 WAITING 200057a9 2001112c 20010db8 1024
* mon 4 RUNNING 20010c54 200107b8 1536
003 4 DORMANT 20000211 00000000 2000f7b8 4096
004 4 DORMANT 20000211 00000000 2000e7b8 4096
005 4 DORMANT 20000211 00000000 2000d7b8 4096
006 4 DORMANT 200006f1 00000000 2000c7b8 4096

mon>set task 6
006(DORMANT)>task activate
execute act_task(6) :: result = E_OK !
Sample program starts (exinf = 0).
task1 is running (001).
006(WAITING)>task1 is running (002).
ltask1 is running (003).
otask1 is running (004).
g task1 is running (005).
mtask1 is running (006).
```

演習: サンプルプログラムの表示の停止

- サンプルプログラムはsyslogのLOG_NOTICEで表示を行っている、タスクモニタからsyslogの表示レベルを変更できる
- 表示を無視して、log mode 4 (return)を入力すると、表示レベルがLOG_WARNINGとなってログ表示しなくなる

log mode 4



```
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)

cur id pri state pc stack inistack inisize
001 3 WAITING 200057a9 2001112c 20010db8 1024
* mon 4 RUNNING 20010c54 200107b8 1536
003 4 DORMANT 20000211 00000000 2000f7b8 4096
004 4 DORMANT 20000211 00000000 2000e7b8 4096
005 4 DORMANT 20000211 00000000 2000d7b8 4096
006 4 DORMANT 200006f1 00000000 2000c7b8 4096

mon>set task 6
006(DORMANT)>task activate
execute act_tsk(6) :: result = E_OK !
Sample program starts (exinf = 0).
task1 is running (001). |
006(WAITING)>task1 is running (002). |
ltask1 is running (003). |
otask1 is running (004). |
g task1 is running (005). |
mtask1 is running (006). |
otask1 is running (007). |
detask1 is running (008). |
4task1 is running (009). |

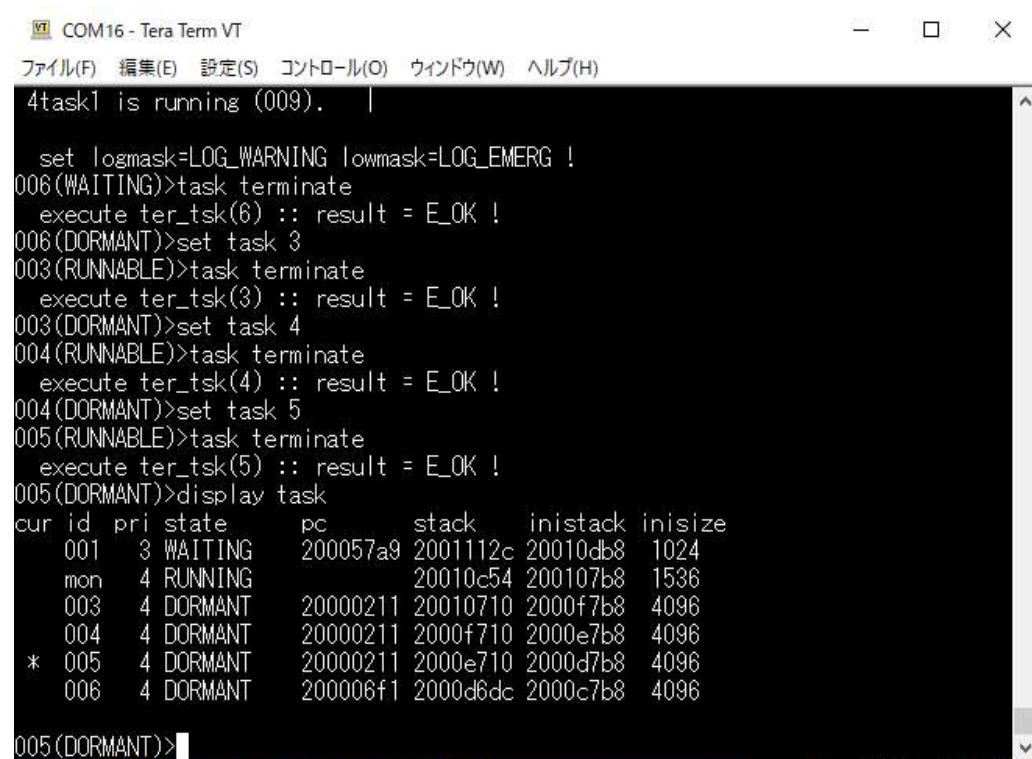
set logmask=LOG_WARNING lowmask=LOG_EMERG !
006(WAITING)>
```

演習: サンプルプログラムの停止

- サンプルプログラムは3～6までの4つのタスクで実行している
- task terminate : 現在選ばれているタスク(6)を終了
- set task とtask terminateで、3～5のタスクを終了させる

リセットは直接実行版では、
USBケーブルをオンオフ。

ROMモニタが再起動するので、
ダウンロードから再実行させる

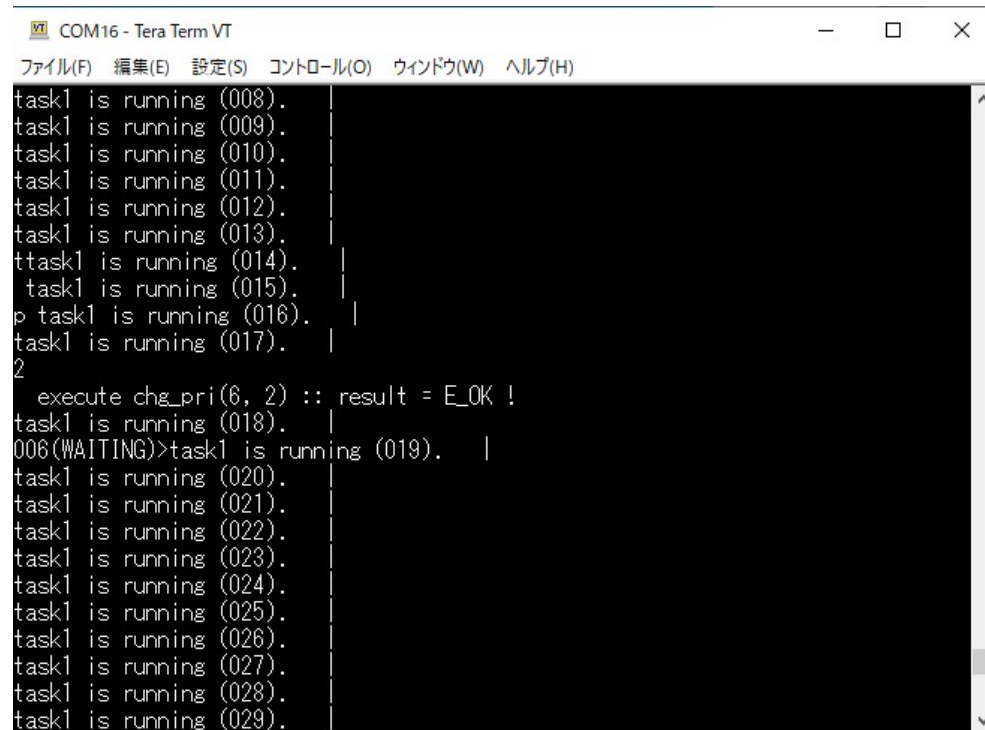


```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
4task1 is running (009). |
set logmask=LOG_WARNING lowmask=LOG_EMERG !
006(WAITING)>task terminate
execute ter_tsk(6) :: result = E_OK !
006(DORMANT)>set task 3
003(RUNNABLE)>task terminate
execute ter_tsk(3) :: result = E_OK !
003(DORMANT)>set task 4
004(RUNNABLE)>task terminate
execute ter_tsk(4) :: result = E_OK !
004(DORMANT)>set task 5
005(RUNNABLE)>task terminate
execute ter_tsk(5) :: result = E_OK !
005(DORMANT)>display task
cur id pri state pc stack inistack inisize
001 3 WAITING 200057a9 2001112c 20010db8 1024
mon 4 RUNNING 20010c54 200107b8 1536
003 4 DORMANT 20000211 20010710 2000f7b8 4096
004 4 DORMANT 20000211 2000f710 2000e7b8 4096
* 005 4 DORMANT 20000211 2000e710 2000d7b8 4096
006 4 DORMANT 200006f1 2000d6dc 2000c7b8 4096
005(DORMANT)>
```

タスクモニタでのタスク優先度の注意

- 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
- メインタスクの優先度2以上に設定すると、メインタスクが優先的にサンプルプログラム操作を行い、タスクモニタの操作ができなくなる

選択されているタスクがメインタスクの状態、task priority 2 (return)の短縮コマンド t p 2 (return)を発行して優先度2に変更



```
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)
task1 is running (008).
task1 is running (009).
task1 is running (010).
task1 is running (011).
task1 is running (012).
task1 is running (013).
task1 is running (014).
task1 is running (015).
p task1 is running (016).
task1 is running (017).
2
execute chg_pri(6, 2) :: result = E_OK !
task1 is running (018).
006(WAITING)>task1 is running (019).
task1 is running (020).
task1 is running (021).
task1 is running (022).
task1 is running (023).
task1 is running (024).
task1 is running (025).
task1 is running (026).
task1 is running (027).
task1 is running (028).
task1 is running (029).
```

演習：タスクモニタからのメモリ操作

- タスクモニタから直接、Pico上のLEDを操作する。デバイスドライバの取り込みで初期化は不要
- set wordコマンドでLEDが接続されている25ピンのポートレジスタ(0xD0000014/0xD0000018)に直接書き込み、動作を確認する
- 25ピンは0x020000000(1<<25)で操作

```
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 5 2025, 16:20:09)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>d w d0000000
d0000000 00000000 00000000 00000000 00000000
d0000010 00000000 00000000 00000000 00000000
d0000020 02000000 00000000 00000000 00000000
d0000030 00000000 00000000 00000000 00000000
d0000040 00000000 00000000 00000000 00000000
d0000050 00000000 00000000 00000000 00000000
d0000060 00000000 00000000 00000000 00000000
d0000070 00000000 00000000 00000001 00000000
mon>s w d0000014
d0000014 00000000 =02000000 00000000
d0000018 00000000 =02000000 00000000
d000001c 00000000 =.
mon>
```

0xD0000014に0x02000000を書き込み：LED点灯

0xD0000018に0x02000000を書き込み：LED消灯

". "で終了

注意：
Pico2のLEDポート
は、
(0xD0000018/0xD
0000020)です

注意：
Pico W/Pico2 Wで
は、この演習はでき
ません
本体LEDがGPIOに
接続していないため

システム検証モジュールの導入

1. タスクモニタの導入
2. タスクモニタの改造
 - [2-1.タスク実行時間の測定](#)
 - 2-2. LED用拡張コマンド

タスクごとの実行時間の測定

- タスクモニタは、簡単な方法でタスク毎の実行時間を測定する機能を持っています
- タスク毎の実行時間がわかると、システムがハードリアルタイムを満たしているかの確認が行えます
- タスクの優先順位設定や、高速化が必要なタスクの性能目標を立てるのに役立ちます
- このような機能の実装は、オーバーヘッドの増大や性能の低下を招きますので、必要のない場合は機能の無効化等の管理が必要です

タスク実行時間の取得

- タスク時間測定は、スケジューラ中でタスクのスイッチングを行う時に実行ログを取る形で測定を行います
- スケジューラソースcore_support.SのLOG_DSP_ENTERとLOG_DSP_LEAVEコンパイルスイッチが設定されたとき、log_dsp_enter/log_dsp_leaveがディスパッチの前後に呼び出されます

```
/*
 * ディスパッチャ本体
 */
ALABEL(dispatcher)
/*
 *
 */
#ifdef TOPPERS_FPU_CONTEXT
    mrs r3, control /* ディスパッチャ実行時はFPCAを一律クリアする */
    bic r3, r3, #CONTROL_FPCA
    msr control, r3
#endif /* TOPPERS_FPU_CONTEXT */
#ifdef LOG_DSP_ENTER
    ldr r1, =p_runtsk /* p_runtskをパラメータに */
    ldr r0, [r1]
    bl log_dsp_enter
#endif /* LOG_DSP_ENTER */
```

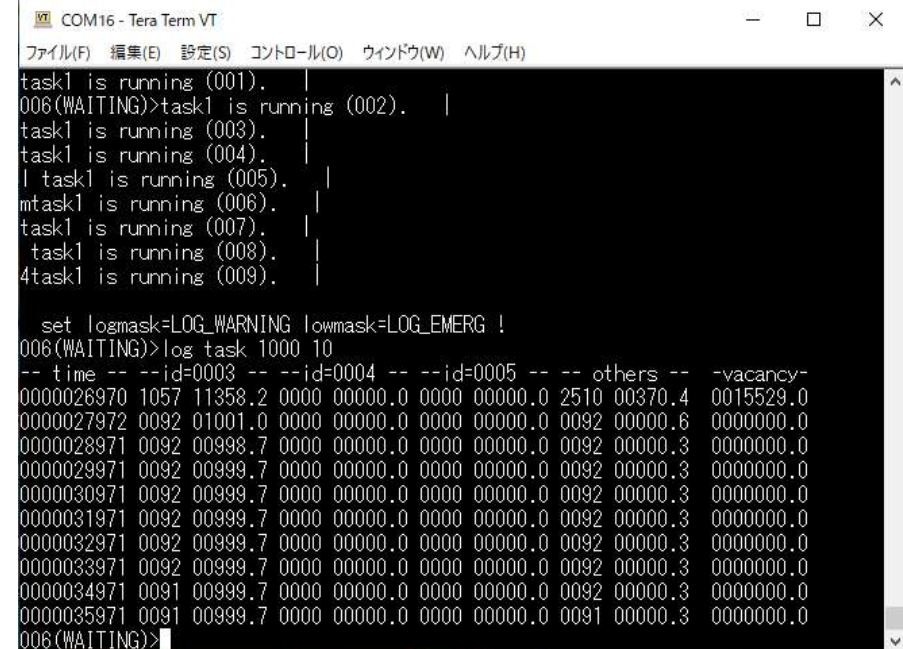
タスク時間の測定

- タスクモニタで、これらの関数の呼び出しで実行時間の測定を行う

```
ALABEL(dispatcher_0)
    ldr r0, =p_schedtsk /* p_schedtskをp_runtskに */
    ldr r1, [r0]
    ldr r2, =p_runtsk
    str r1, [r2]
    cmp r1, #0 /* p_runtskがNULLならdispatcher_1へ */
    beq dispatcher_1
    ldr sp, [r1, #TCB_sp] /* タスクスタックを復帰 */
#ifdef LOG_DSP_LEAVE
    mov r0, r1 /* p_runtskをパラメータに */
    mov r4, r1 /* r1はスクラッチレジスタなので保存 */
    bl log_dsp_leave
    mov r1, r4
#endif /* LOG_DSP_LEAVE */
    ldr pc, [r1, #TCB_pc] /* 実行再開番地を復帰 */
ALABEL(dispatcher_1)
```

タスク実行時間の表示

- スケジューラの切り替え時に保存されたタスク実行時間は、LOG TASKコマンドで設定された時間単位に表示を行い、表示後は実行時間をリセットします
- タスク実行時間の収集はmonitor/task_expansion.c中のlog_tsk_enterとlog_tsk_leave関数で行います。収集したタスク実行時間の取り出しはget_tsklog関数をタスクモニタが周期的に行うことにより実行されます



```
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)
task1 is running (001).
006(WAITING)>task1 is running (002).
task1 is running (003).
task1 is running (004).
task1 is running (005).
mtask1 is running (006).
task1 is running (007).
task1 is running (008).
4task1 is running (009).

set logmask=LOG_WARNING lowmask=LOG_EMERG !
006(WAITING)>log task 1000 10
-- time -- --id=0003 -- --id=0004 -- --id=0005 -- -- others -- --vacancy--
0000026970 1057 11358.2 0000 00000.0 0000 00000.0 2510 00370.4 0015529.0
0000027972 0092 01001.0 0000 00000.0 0000 00000.0 0092 00000.6 0000000.0
0000028971 0092 00998.7 0000 00000.0 0000 00000.0 0092 00000.3 0000000.0
0000029971 0092 00999.7 0000 00000.0 0000 00000.0 0092 00000.3 0000000.0
0000030971 0092 00999.7 0000 00000.0 0000 00000.0 0092 00000.3 0000000.0
0000031971 0092 00999.7 0000 00000.0 0000 00000.0 0092 00000.3 0000000.0
0000032971 0092 00999.7 0000 00000.0 0000 00000.0 0092 00000.3 0000000.0
0000033971 0092 00999.7 0000 00000.0 0000 00000.0 0092 00000.3 0000000.0
0000034971 0091 00999.7 0000 00000.0 0000 00000.0 0092 00000.3 0000000.0
0000035971 0091 00999.7 0000 00000.0 0000 00000.0 0091 00000.3 0000000.0
006(WAITING)>
```

LOG TASK <時間(単位MS)> <回数>

演習：タスク実行時間ログ機能の追加

- Makefileに「LOG_DSP_ENTER」と「LOG_DSP_LEAVE」を追加する
- core_support.Sはカーネルソースなので、クリア後再ビルドします

```
136 #
137 # 共通コンパイルオプションの定義
138 #
139 COPTS := $(COPTS) -g
140 ifndef OMIT_WARNING_ALL
141 COPTS := $(COPTS) -Wall
142 endif
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$(SRCDIR)/arch -I$(SRCDIR) $(INCLUDES)
148 LDFLAGS := $(LDFLAGS)
149 LIBS := $(LIBS) -lc $(CXXLIBS)
150 CFLAGS = $(COPTS) $(CDEFS) $(INCLUDES)
```

演習: TASKの実行時間の確認

- sample1の実行中、TASK1のみ実行していることをタスク実行時間ログ機能を使って確認してみよう
- TASK1がID=3、TASK2がID=4、TASK3がID=5に設定され、その他のタスクの実行時間はothersに計測されます
- MAIN_TASKはID=6なので、実行後set task 6で対象タスクを6に切り替え、task activateでMAIN_TASKを実行します
- 表示が始まった後、log mode 4でログ表示を止めます
- log task 1000 10で1秒単位に実行時間ログを計測します
- ID=3(TASK1)に実行が割り当てられ、ID=4(TASK2)とID=5(TASK3)に実行時間が割り当てられていないことを確認してください

システム検証モジュールの導入

1. タスクモニタの導入
2. タスクモニタの改造
 - 2-1. タスク実行時間の測定
 - 2-2. LED用拡張コマンド

演習：タスクモニタのコマンドの拡張を行う

- タスクモニタを組み込みシステム用に拡張する
- タスクモニタはカテゴリとコマンドを追加できる
- コマンド拡張機能でPico上のLEDコマンドを追加して、LEDの点灯、消灯を行う
- カテゴリ: DEVICE、コマンド: LEDを設ける
- LEDコマンドには2つのパラメータがあり、第1パラメータはLEDの番号(1-4)をセットし、第2パラメータは消灯: 0、点灯: 1を指定する
- LEDコマンドを用いて、LEDを制御する
 - 以下、DEVICEカテゴリにLEDコマンド等々を追加

DEVICE LED (LED番号1-4) (消灯-0、点灯-1)
DIP (DIPSW番号1-4) (OFF-0、ON-1)

演習: カテゴリとコマンドの追加

- コマンドリンク構造体を作成し、`setup_commnad`関数を使用して新たなカテゴリとコマンドの追加ができる

以下、`monitor.h`のコマンドリンク構造体記述

```
/*
 * コマンドデスパッチ用の構造体定義
 */
typedef struct _COMMAND_INFO {
    const char *command;          /* コマンド文 */
    int_t      (*func)(int argc, char **argv); /* 実行関数 */
} COMMAND_INFO;

typedef struct _COMMAND_LINK COMMAND_LINK;
struct _COMMAND_LINK {
    COMMAND_LINK *pcnext;
    int          num_command;
    const char *command;          /* 主コマンド文 */
    int_t      (*func)(int argc, char **argv); /* 実行関数 */
    const char *help;             /* ヘルプ文 */
    const COMMAND_INFO *pcinfo;
};
```

演習 : command.cの取り込み

- 拡張コマンドのソースファイルcommand.cはbase2/programディレクトリにあります、command.cをOBJ/RASPBERRYPI_PICO_GCC/MON2ディレクトリにコピーする
- コンパイルの対象となるようにMakefileを書き換えます

```
153 #
154 # アプリケーションプログラムに関する定義
155 #
156 APPLNAME = sample1
157 APPLDIR =
158 APPL_CFG = $(APPLNAME).cfg
159
160 APPL_DIR = $(APPLDIR) $(SRCDIR)/library
161 APPL_ASMOBS =
162 ifdef USE_CXX
163   APPL_CXXOBS = $(APPLNAME).o
164   APPL_COBS = command.o
165 else
166   APPL_COBS = $(APPLNAME).o command.o
167 endif
```

追加

演習 : command.cの実装

- DEVICEカテゴリを実行するcommand.cを参照する

```
#include <kernel.h>
#include <t_syslog.h>
#include <t_stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include "syssvc/serial.h"
#include "syssvc/syslog.h"
#include "monitor.h"
#include "device.h"
```

モニタのコマンド型とプロ
トタイプ宣言を取り出す

デバイス関数のプロトタイ
プ宣言を取り出す

```
static int_t led_func(int argc, char **argv);
static int_t rst_func(int argc, char **argv);
static int_t cup_func(int argc, char **argv);
```

```
static const COMMAND_INFO device_command_info[] = {
    {"LED",          led_func},
    {"RST",          rst_func},
    {"CUP",          cup_func}
};
```

CUPは省略して
よい

```
#define NUM_DEVICE_CMD ¥  
    (sizeof(device_command_info)/sizeof(COMMAND_INFO))
```

演習 : command.cの実装

- DEVICEカテゴリを実行するcommand.cを参照する

```
static const char device_name[] = "DEVICE";
static const char device_help[] =
" Device LED (no) (on-1,off-0) led control¥n"
"      RST          soft reset¥n"
"      CUP command      cup control¥n";

static const uint16_t led_pattern[4] = {
    LED01, LED02, LED03, LED04
};

static COMMAND_LINK device_command_link = {
    NULL,
    NUM_DEVICE_CMD,
    device_name,
    NULL,
    device_help,
    &device_command_info[0]
};

static uint8_t cup_command;
```

コマンドリンク
構造体

演習: command.cの参照

- 個々のコマンドは引数をargc(引数の数), argv(引数の文字列のポインタ配列)の形式で渡される

```
/*
 * LED設定コマンド関数
 */
static int_t led_func(int argc, char **argv)
{
    int_t arg1=0, arg2;
    uint16_t reg;

    if(argc < 3)
        return -1;
    arg1 = atoi(argv[1]);
    arg2 = atoi(argv[2]);
    if(arg1 >= 1 && arg1 <= 4){
        reg = led_in();
        if(arg2 != 0)
            reg |= led_pattern[arg1-1];
        else
            reg &= ~led_pattern[arg1-1];
        led_out(reg);
    }
    return 0;
}
```

引数1, 2から数値を
取り出す

デバイスドライバ
関数を呼び出す

演習 : command.c/CUPコマンド

- CUPコマンドは、タスクモニタ経由でSAMPLE1のメインタスクにコマンドを渡すために使用するため、この記載はなくてもよい

```
/*
 * CUPコマンド関数
 */
static int_t cup_func(int argc, char **argv)
{
    if(argc > 1)
        cup_command = *argv[1];
}
/*
 * 入力文字取り出し関数
 */
int8_t
get_cup_command(void)
{
    uint8_t cmd = cup_command;
    cup_command = 0;
    return cmd;
}
```

引数1の文字を
cup_commandに保存

アプリから
get_cup_command()関数
を呼び出すことでCUPコマ
ンドでセットした文字を取り
出せる

演習: command.cを参照

- コマンドリンク構造体をsetup_command関数を用いて、タスクモニタに登録すれば新たなコマンドが追加できます
- コマンドの追加はATT_INIサービスコールを使用する
- ATT_INIサービスコールは初期化ルーチンを実行する
- コマンドの追加処理を初期化で実行する

```
/*  
 * DEVICEコマンド設定関数  
 */  
void device_info_init(intptr_t exinf)  
{  
    setup_command(&device_command_link);  
}
```

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: sample.hの書き換え

- ATT_INIサービスコールに初期化関数を認識させるために、device_info_init関数のプロトタイプ宣言をsample.hに追加する

```
/*  
 * 関数のプロトタイプ宣言  
 */  
#ifndef TOPPERS_MACRO_ONLY  
  
extern void task(intptr_t exinf);  
extern void main_task(intptr_t exinf);  
extern void tex_routine(TEXPTN texptn, intptr_t exinf);  
#ifdef CPUEXC1  
extern void cpuexc_handler(void *p_excinf);  
#endif /* CPUEXC1 */  
extern void cyclic_handler(intptr_t exinf);  
extern void alarm_handler(intptr_t exinf);  
extern void device_info_init(intptr_t exinf);  
extern int8_t get_cup_command(void);  
  
#endif /* TOPPERS_MACRO_ONLY */
```

追加

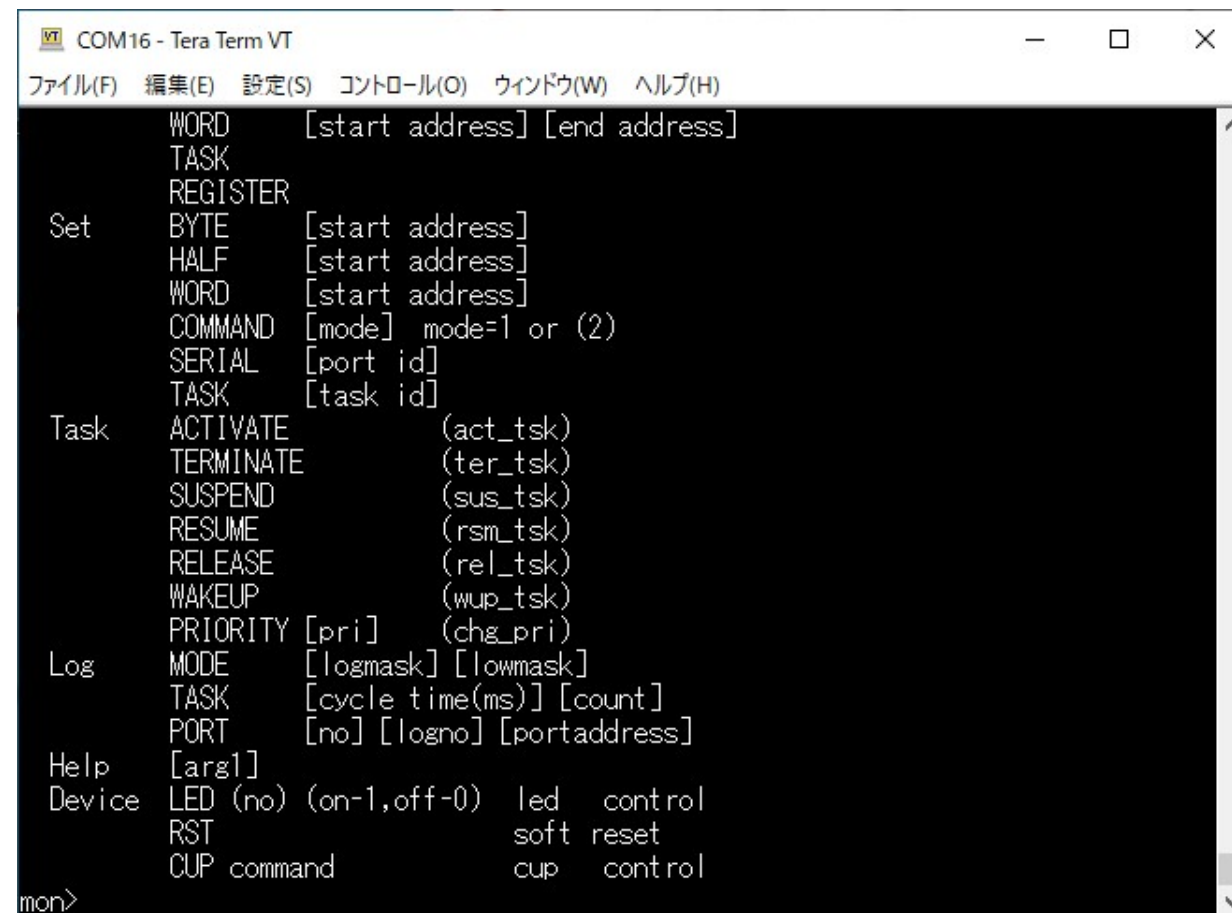
演習 : sample1.cfgの修正

- コンフィギュレーションファイルsample1.cfgを修正し device_info_init関数を初期化関数と実行するように ATT_INIサービスコールを追加する

```
#include "sample1.h"
CRE_TSK(TASK1, { TA_NULL, 1, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_NULL, 2, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK3, { TA_NULL, 3, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(MAIN_TASK, { TA_NULL, 0, main_task, MAIN_PRIORITY, STACK_SIZE, NULL });
DEF_TEX(TASK1, { TA_NULL, tex_routine });
DEF_TEX(TASK2, { TA_NULL, tex_routine });
DEF_TEX(TASK3, { TA_NULL, tex_routine });
CRE_CYC(CYCHDR1, { TA_NULL, 0, cyclic_handler, 2000, 0 });
CRE_ALM(ALMHDR1, { TA_NULL, 0, alarm_handler });
ATT_INI({ TA_NULL, 0, device_info_init });
#ifdef CPUEXC1
DEF_EXC(CPUEXC1, { TA_NULL, cpuexc_handler });
#endif /* CPUEXC1 */
```

演習:ビルドとダウンロード実行

- ビルドを行い、ldコマンドダウンロード実行
- 実行後、helpコマンドを実行すると、DEVICEカテゴリのLEDコマンドが追加されている



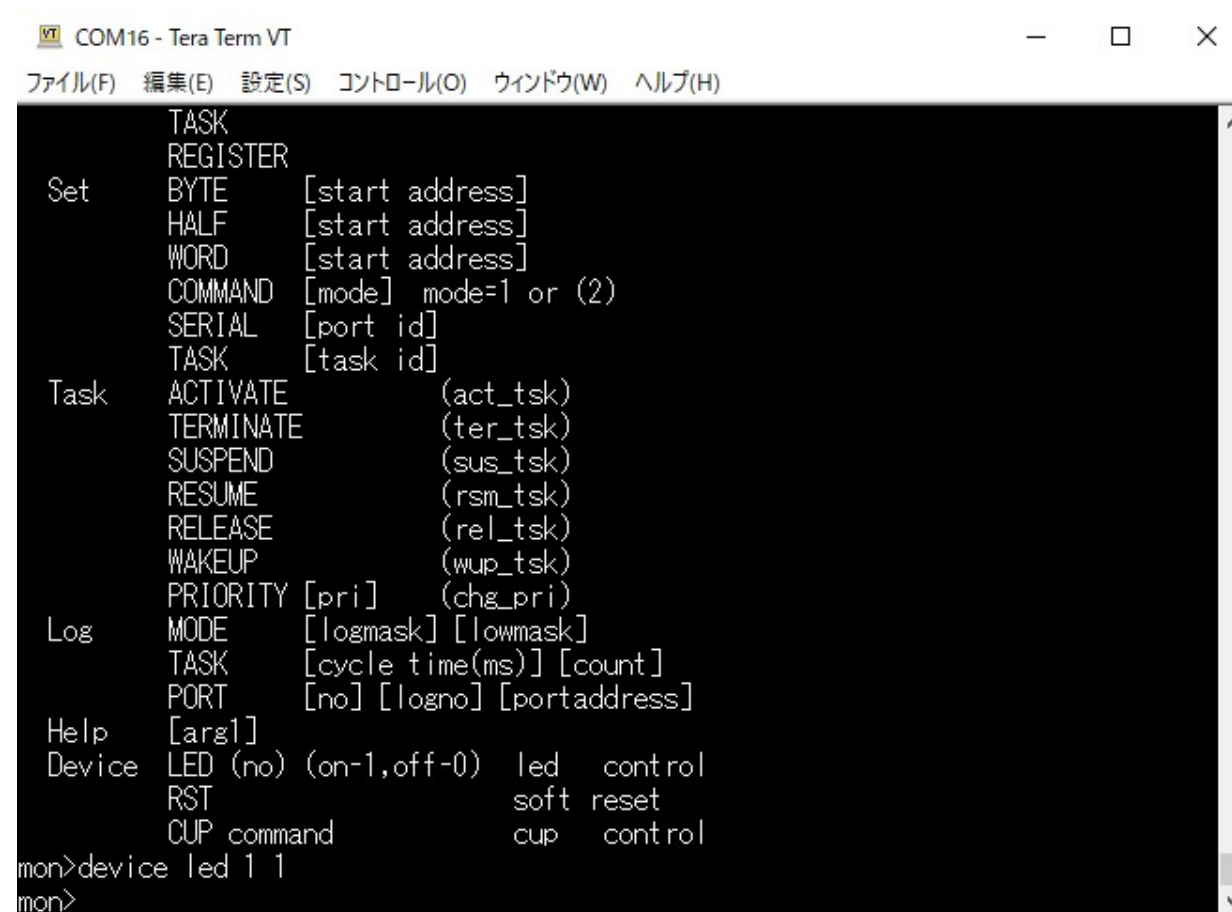
```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

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]
Device LED (no) (on-1,off-0) led control
      RST          soft reset
      CUP command  cup control

mon>
```

演習: LEDコマンドの実行確認

- DEVICE LED 1 1(return)でPico/Pico2のLED1が点灯することを確認してください



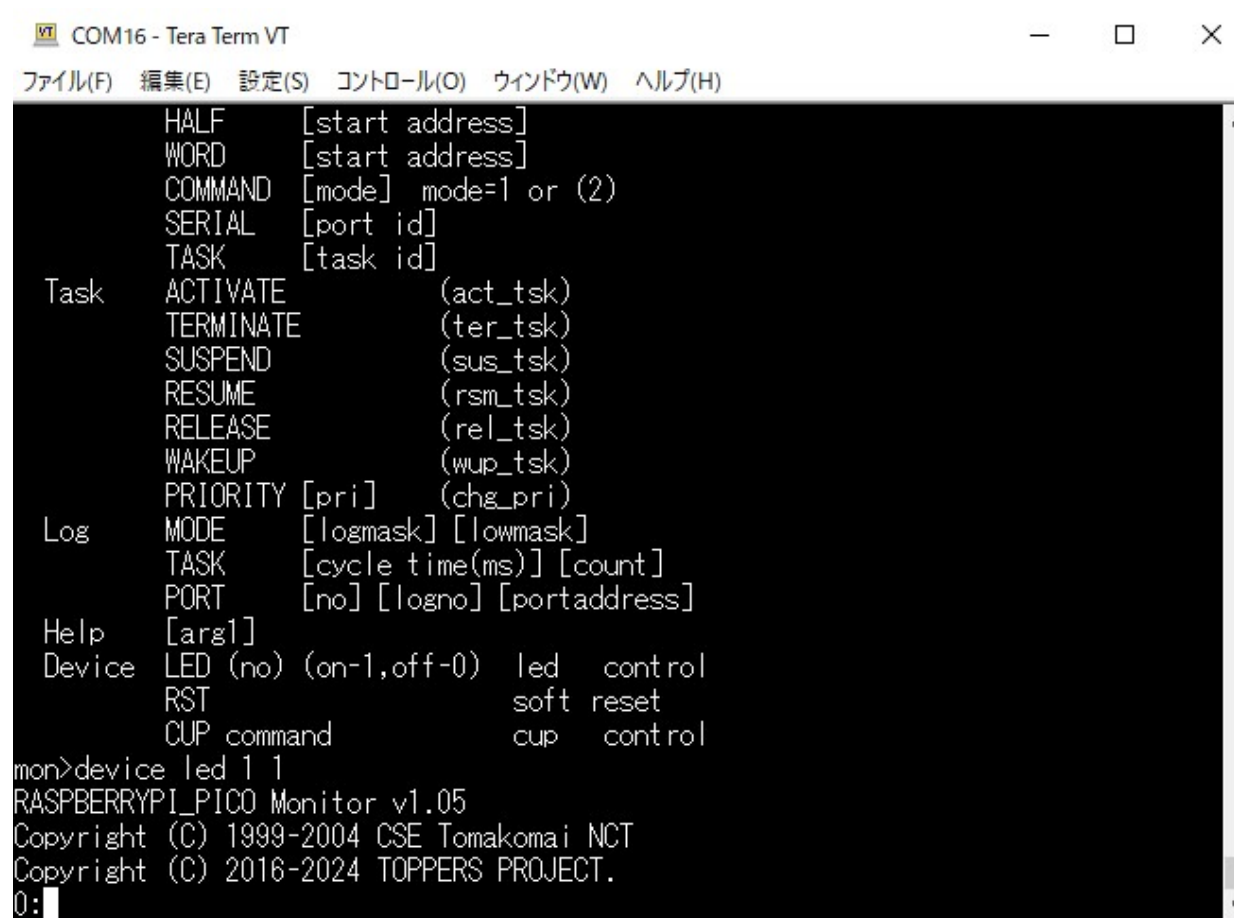
```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

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]
Device LED (no) (on-1,off-0) led control
      RST soft reset
      CUP command cup control

mon>device led 1 1
mon>
```

演習:リセット機能の確認

- Pico/Pico2はリセット・ボタンがない
- DEVICE RST(return)で、リセットされ、ROMモニタに戻る
 - 電源OFF/ONを行う必要はありません



```
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)

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]
Device LED (no) (on=1,off=0) led control
      RST soft reset
      CUP command cup control

mon>device led 1 1
RASPBERRYPI_PICO Monitor v1.05
Copyright (C) 1999-2004 CSE Tomakomai NCT
Copyright (C) 2016-2024 TOPPERS PROJECT.
0:
```

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

まとめ

リアルタイムOSを用いたシステム設計

- リアルタイムOSは、プログラム規模が大きい中規模組込みシステムで使用すると効果的です
- リアルタイムOSは、CPU処理の代行を行う機能が主で、システム設計を行う場合、機器の機能に対応したベース機能を含んだプラットフォームを構築すると、開発工数の削減や品質の向上につながります
- プラットフォームは、機器の機能に応じたメンテナンス機能を用意した方が、効率的な開発が行えます
 - メンテナンス機能の代表がタスクモニタです
- システム設計管理には、熟練したシステム管理者が必要です

μITRON仕様に関して

- μITRON仕様は、日本の組込みシステムでもっとも多く使われているリアルタイムOSの仕様です
- リアルタイムOSは、組込み機器が多くのバリエーションを持つためCPUの機能の代行する機能に限定した仕様となっています
- タスクを用いたシステムでは、同期や通信などの機能を用いて各機能が円滑に動作するようにシステム設計する必要があります
- TOPPERS/JSPカーネルはμITRON4.0仕様のスタンダードプロファイルに準じたリアルタイムOSで、TOPPERS/ASPカーネルはμITRON4.0仕様に新たな機能を付加し、オープンソースとして供給されています、itron.hをインクルードすることでμITRON4.0仕様で動作します

著者リスト

- リアルタイムOSの基礎
 - 高田 広章(名古屋大学), 本田 晋也(名古屋大学)
 - 竹内 良輔(教育WG)
- ITRON仕様の基礎知識
 - 高田 広章(名古屋大学), 本田 晋也(名古屋大学)
- TOPPERS/ASPの導入
 - 竹内 良輔(教育WG)
- システム検証モジュールの導入
 - 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。