

TOPPERS基礎実装セミナー

(M16C版:基本)
基礎編:3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. 著作権に関しての表記

＜TOPPERS基礎実装セミナー(M16C版:基本)3日目＞

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 名古屋大学

Copyright (C) 2007 by 竹内良輔 (株)リコー

上記著作権者は、以下の (1)～(3) の条件を満たす場合に限り、本ドキュメント (本ドキュメントを改変したものを含む。以下同じ) を使用・複製・改変・再配布 (以下、利用と呼ぶ) することを無償で許諾する。

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

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

(3) 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

本ドキュメントは、無保証で提供されているものである。上記著作権者およびTOPPERSプロジェクトは、本ドキュメントに関して、特定の使用目的に対する適合性も含めて、いかなる保障もしない。また、本ドキュメントの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。
3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

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

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

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

スケジュール

■ 3日目

- | | |
|-------------------|-------|
| 1. リアルタイムOSの基礎 | 0.5時間 |
| 2. ITRONの仕様について学ぶ | 1.5時間 |
| 3. TOPPERS/JSPの導入 | 2.0時間 |
| 4. システム検証モジュールの導入 | 2.0時間 |
| 5. まとめ | 0.5時間 |

概要

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

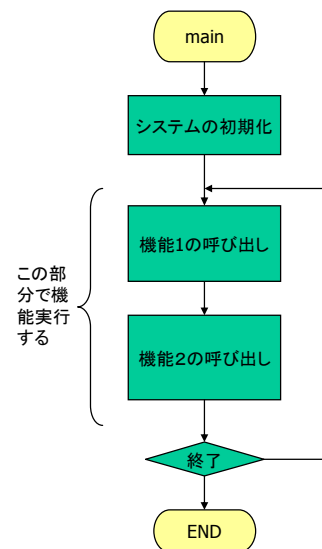
- リアルタイムOSの基礎
 - リアルタイムOSを用いた組込み開発
 - ITRON仕様
 - TOPPERS/JSPカーネル
- TOPPERS/JSPカーネルの導入
 - 開発環境の構築
 - サンプルプログラムのビルドと実行
- TOPPERS/JSPカーネルの拡張
 - システム検証モジュールの導入と改造

リアルタイムOSの基礎

1. [リアルタイムOSを用いた組み込み開発](#)
2. リアルタイムOSのメリットデメリット
3. タスク間の同期と通信

小規模組込み開発の開発手法

- 小規模組込み開発では、main関数中のループ部からシステムの制御に必要な関数を周期的に呼び出すことにより、制御を行うのが一般的な方法です
- 1, 2人の開発者でワンチップCPUに収まるソースコード一万行程度までのプログラムならば、十分な作りこみが可能です
- しかし、システムの規模が大きくなると種々の問題が発生します



2008/12/05

TOPPERSプロジェクト認定

7



小規模組込み開発は、一般的に以下の方式でプログラミングを行います。

(1) mainルーチン中に機器の機能を記述していく。

(2) 個々の機能は時分割で処理を行わなければならないため、メインルーチン中にループを作り周期的に機能チェックと実行を行うのが通常のプログラミング方式です。

例えばカップラメンタイマーでは、タイマ機能、ボタンの押下の判定、アラート処理等を周期ループの中で読み出しを行い、各機能はイベントと状態に従って、処理を行います。プログラミング規模が小さい場合はリアルタイム性を保ちながら、十分に品質の高いプログラムを開発することは可能です。

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

2008/12/05

TOPPERSプロジェクト認定

8



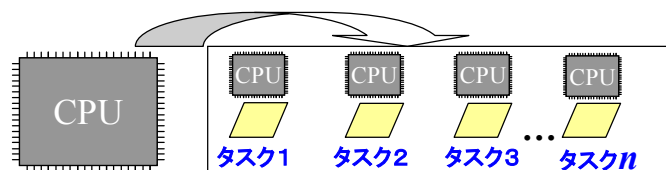
小規模組込みシステムでは、機器の機能も少なく、ワンチップCPUを用いて少人数で開発が可能です。小規模組込みシステムに複雑なUIや通信機能、ファイルシステム等々の機能アップを行うと、mainルーチンのループの形の方式では、機能を実装していくことが困難になっていきます。例えば、ネットワークにつながるUSBメモリを作ろうとすると、USBデバイスにあたる部分をETHER-NETにつながるデバイスに変更して、パソコン等と通信を行うプログラムを追加していく必要があります。USBデバイスでは割込み中でデータ通信を行い、USBの状態遷移等はデバイス側で行ってくれますのでプログラム量はたいして大きくはなりません。しかし、ETHER-NETデバイスの機能送受信の通信機能が主で、TCP/UDP/IP等のTCP/IP機能はプロトコルスタックはプログラムで搭載する必要があります。加えて、ストレージを管理するSMBやNETFS等ネットワークアプリケーションやDHCP、DNSクライアント等のIPを管理するネットワークアプリケーション等をどんどん追加していく必要があります。単純に考えて、USBメモリの開発スキルを持った技術者とネットワークの開発スキルを持った技術者が協力しあわないとプログラムの開発はできません。

それでは、ネットワークの通信機能を持つボードとUSBのストレージ管理をするボードを分けて、この間の通信方式を決めて機器化すれば、お互い小規模組込み開発手法で開発が可能かも知れませんが、コスト的には1ボードで作る2倍の部品コストがかかる可能性があります。

このような場合、リアルタイムOSを使用し、プロトコルスタックやネットワークアプリケーションをソフトウェア部品（汎用ミドルウェア）として取り込み、今までのUSBメモリのソフトウェアをうまく流用すれば、以外と簡単にネットワーク対応のUSBメモリを作ることには可能です。（商品価値の問題は別）このようにリアルタイムOSを使用することにより、新たな機能を取り込みが可能となります。

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



2008/12/05

TOPPERSプロジェクト認定

9



システム規模が比較的大きく、種々の機能を融合させた機器を管理するシステム設計を行うには、リアルタイムOSは有効な手段となります。(MDAと呼ばれる動的なモデル設計を用いたシステム開発も、比較的規模の大きな組込みシステムへの対応は可能ですが、開発ツールやランタイムと呼ばれる実行環境を熟知した技術者が必要であり、開発環境自体が高価であるという問題もあります)

リアルタイムOSの大きな特徴は、マルチタスク機能をもつことです。マルチタスク機能とは、各プログラムを別々のCPUで実行しているように見せる機能です。つまり、**main**ルーチンをひとつのCPU上に、いくつも書くことが可能となります。いくつもというのは生成したタスクの数分可能ということです。もちろん、ひとつのCPU上でいくつものプログラムを同時実行することはできません。これは、ある時間のみ参照すると、ひとつのタスクのひとつのプログラムしか実行していません。しかし、このタスクがすることがなくなった場合、別のタスクが実行を自動的に切り替えて行うことにより、長い時間をみれば、全てのタスクが実行されているように見せる機能です。もちろん、個々のタスクは周期的に暇な時間があり、その暇な時間で他のタスクが実行できなければなりません。

中規模組込み開発とシステム管理者

- 中規模組込みシステムではプラットフォーム部を先行開発しサブシステムを追加していく方式が一般的です、また、商品化後継続的に機能アップを行う必要もあります
- システム管理者はプラットフォーム部とサブシステム間のアーキテクチャを決め、サブシステム間のインターフェイスを明確化する必要があります
- 分業開発では、予測外の問題で開発遅延が発生することが多々あります、これらの問題解決もシステム管理者の仕事です
 - 開発時、遅れたサブシステムをスタブ化してシステム開発の遅れを回避したり
 - 問題発生時、原因がどの部署に起因するかを解析したり

2008/12/05

TOPPERSプロジェクト認定

10



中規模組込みシステムは、多人数でソフトウェアの開発を行います。多人数でプログラム開発する場合、誰かが取りまとめを行わないと、開発がうまくいかないことが多々あります。中規模組込みシステムの性格上、取りまとめを行う人はリアルタイムOS、ミドルウェアや機器の機能についても熟知した技術者でなければなりません。一般的にこのような技術者をシステム管理者やアーキテクトと呼びます。

一般的に、ソフトウェア規模の大きなシステムでは、各プログラムをサブシステムに分割して並行開発し、それを一回の結合でシステムが問題なく動作することは、まず、ありません。そればかりか、まったく動かない事態に陥るでしょう。また、逆に、小さなシステムからシステム設計し、最終的にサブシステム化も行わず、プログラム拡張で、中規模システム(大きなプログラムの塊)を構築することにも無理があります。これがもし可能だとしたら、1,2人の天才的なプログラマーによって構築されたシステムです。このようなシステムは、開発者がいなくなったり、いなくならなくても、機能追加を頻繁に行えば、最終的にメンテナンスのできないシステムになってしまいます。

まず、プロトタイプシステムという形で動作する基本システムを構築し、サブシステム化、モジュール化を行いながら、目標システムを構築していく方式をビルド式と呼びます。ある程度規模の大きいコンピュータシステムは、この方式で構築されています。オブジェクト思考的にはイテレーション開発とも言います。いずれにしても、しっかりしたシステム管理者が、システム管理を行わないとプロトタイプは、動作しないシステムに変わってしまいます。

リアルタイムOSの内容を理解する技術者の必要性

- 中規模組込みシステムでは、リアルタイムOSに熟達した技術者を社内または社外の協力会社に用意する必要があります
 システム開発中にアーキテクチャに関する問題解決が必要となるケースが多く発生する
- 中規模システムでは、汎用ミドルウェアの導入やデバイスドライバの導入が必要となります、この場合、リアルタイムOSに対して最適化を行う必要があります
- アプリケーションの障害でも、リアルタイムOS中でエグゼクションが発生することがあります
 – ソースコードのないリアルタイムOSでは解析が困難
- 省エネ対応のためにリアルタイムOSの改造やSoC化によって何度も、検証用のデバイスドライバを書き換える必要があります

2008/12/05

TOPPERSプロジェクト認定

11



中規模組込み開発では、しっかりしたシステム管理者が必要です。この技術者を社内または社外の協力会社から確保する必要があります。商品化後のメンテナンスを考えれば、最終的には、社内に育成を行うべきでしょう。当然、一般のプログラムが引き起こした問題も、システム上問題があればシステム管理者が解決しなければなりませんので、非常に重要できつい職種となります。

組込みシステムの特徴として、機能をソフトウェアで実装すればコストは削減できるが性能機能には制約ができ、ハードウェアで実装すれば、ある程度コストアップするが、性能機能は向上する傾向があります。（筆者の経験からくる見解です。低性能のCPUをハードウェアロジックサポートして低価格化するケースもあります）、同じような商品でも、バリエーションが必要であり、システムはハードウェアの改変に対応できるように設計を行う必要があります。加えて、ミドルウェアの導入やデバイスドライバなどのハードウェアに近いソフトウェアの対応など、種々の特殊技術が必要な開発行為があり、このような技術者を用意する必要があります。

大規模組み込みシステムとの比較

- UNIXやWindowsの汎用OSを組み込みシステムとして用いる場合とリアルタイムOSを用いる場合の比較を行いましょう
- 商品は品質、機能、コストを最適にすることにより商品性が向上します
- 汎用OSはシステムの安全性を向上させる点からは有用ですが、OSの実行管理のために多くのCPUパワーやメモリが必要となり性能やコストの面で問題があります
- 汎用OS自体の工数が多いため、それを最適化し、管理するためそれなりの技術者も必要となる(リアルタイムOSより複雑でノウハウは高い)

簡単な例として、パソコンの周辺機の管理に、パソコンと同等のシステムを用いて行うのは無理があります

2008/12/05

TOPPERSプロジェクト認定

12



大規模組み込みシステムとの比較を行います。より生産性、保守性、信頼性を向上させるために、汎用コンピュータ用のOSを組み込みシステムにしようすることは可能です。この場合、確実にコストアップになります。汎用コンピュータ用のOSはコンピュータを適切に動作させ、開発環境を提供することに最適となるようにプラットフォーム化(OS化)されています。これは組み込み機器の機器最適化を目標とすると、オーバーヘッドやメモリソースやCPUの性能の観点から高価なハードウェアを必要とします。また、汎用コンピュータ用のOSを本来の目的でなく、使用するとなると、汎用コンピュータ用のOS自体の改変も発生し、この運用、メンテナンスに新たな技術者が必要となります。

機器の非機能要件にあったシステムが必要なのであり、小規模、中規模、大規模のすべての組み込み開発は必要ということではないでしょうか。

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. リアルタイムOSのメリットデメリット
3. タスク間の同期と通信

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

2008/12/05

TOPPERSプロジェクト認定

14



リアルタイムOSを利用するメリットを列記します。

(1) マルチタスク機能を用いて、ソフトウェアの構造化が可能です。これにより生産性、保守性、信頼性が向上します。特に、システム規模が大きくなると、保守性や信頼性が極端に低下します。加えて、組込み機器は機種ごとの機能追加を行わないと(アジャイル・プロダクト)商品価値が低下します。このとき、タイムリーに商品化を行うために生産性、保守性、信頼性は重要なインパクトとなります。

(2) マルチタスク機能は、時間制約性を持った多重システムが容易になります。mainルーチンから関数コールで機能を実行する場合、個々の機能を性格に時間単位に実行することは、かなり、難しくなることを考えれば簡単に推論が可能です。このメリットにより、ソフトウェア部品をコンポーネントとして使用することが容易になります。

(3) ハードウェア、特にCPUの違いを隠蔽化できます。TOPPERS/JSP上で作成したsample1プログラムがいろいろなボード(CPU)上で無変更で実行が可能です。これはあくまでも、ハードウェアの知識のない技術者だけで、システム構築が可能という意味ではありません。アプリケーション部の開発をハードウェアから隠蔽化できるという意味です。

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

2008/12/05

TOPPERSプロジェクト認定

15



リアルタイムOSを用いるデメリットを列記します。

(1)アプリケーション以外に、リアルタイムOSもアプリケーション管理用に動作しますので、オーバーヘッドが発生するばかりではなく、リアルタイムOS実行用のワーク領域等メモリを使用します。また、タスクを切り替えるためのオーバーヘッドも発生します。

(2)いくつものタスクを実行するために、タスクが停止状態になった場合に実行データを保存するためのスタック領域がタスクごとに必要です。また、あるタスクを停止させても、別のタスクが実行していると、実行条件が変化し問題発生時、問題の特定が困難になります。行儀のよいタスク(サブシステム)間インターフェイス設計が必要です。

(3)アプリケーションの不具合であっても、リアルタイムOS内で問題が発生することが多々あります。一般的にリアルタイムOSはメモリ保護がなく、リアルタイムOS内のリソースも簡単に壊してしまえるからです。ツールによる開発環境の改善やリアルタイムOSの原理や内部構造を知る技術者は必要です。

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. リアルタイムOSのメリットデメリット
3. 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/VDK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

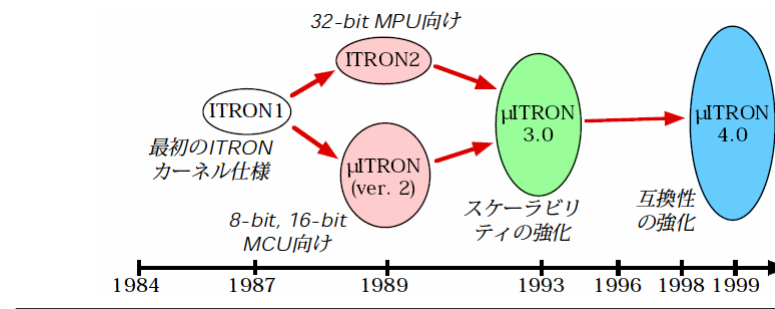
- 共有メモリ vs. メッセージ通信
 - 基本的には、一方で実現できることは、もう片方でも実現できる
 - 一般には共有メモリの方がオーバーヘッドが小さい
 - システム検証には、通信の粒度が大きくRTOSレベルで監視できるメッセージ通信の方が扱いやすい
 - 例えば、問題の切り分けが容易
 - 状況により、どちらかが有利/便利な状況がある
 - 情報の流れが 1:n の場合（ブロードキャストを含む）や、情報が必要なタイミングが不定の場合には、共有メモリが有利
 - 情報のキューイングが必要な時には、メッセージ通信が便利

ITRONの基礎知識

1. [ITRON仕様](#)
2. ITRONの同期・通信機能
3. TOPPERS/JSP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仕様に策定・公表
 - 現世代のリアルタイムカーネル技術の範囲では、完成度の高い仕様に
- ➡ 組込みシステム開発のオープン化に対する基盤



2008/12/05

TOPPERSプロジェクト認定

23



トロン (TRON: The Real-time Operating system Nucleus) は、理想的なコンピュータアーキテクチャの構築を目的として、1984年に東京大学の坂村健博士によって開始されたプロジェクトです。産業界と大学の協力のもとに、まったく新しいコンピュータの体系の実現を目指しています。トロンプロジェクト推進を行うに際して、実質的ないくつかのドメインを対象としてサブプロジェクト単位に、具体的な仕様の標準化を行ってきました。ITRON (組込みシステム用のリアルタイムOS仕様とその関連仕様)、BTRON (パソコンやワークステーション用のOS仕様とその関連仕様)、CTRON (通信制御や情報処理を目的としたOSインターフェース仕様)、トロン電子機器HMI (各種の電子機器のヒューマンマシンインターフェースの標準ガイドライン) 等のサブプロジェクトです。

最初のITRON仕様は、1987年にITRON1仕様という形でまとめられました。ITRON1仕様に従っていくつかのリアルタイムカーネルが開発・応用され、仕様の適応性の検証に重要な役割を果たしました。その後、8～16ビットのMCUに適応するための機能を絞り込んだ μ ITRON仕様 (Ver.2.0)、逆に大規模な32ビットのプロセッサに適応するためのITRON2.0仕様の検討を進め、共に1989年に仕様を公開しました。 μ ITRON仕様は限られたリソースのMCUでも実用的な性能を発揮できたことから多くのMCUに実装され、多くの組込み機器に応用されました。 μ ITRON仕様が広域な分野に応用されるにつれて要求が明確化されたことと、 μ ITRON仕様が32ビットMCUに応用され始め、8ビットから32ビットまでの各規模のMCUに適応できるスケーラビリティを持った仕様の要求が高まったことから1993年に μ ITRON3.0仕様として策定しなおし公開を行いました。

その後、ITRON TCP/IP API仕様やJTRON仕様の策定を経て、 μ ITRON3.0仕様の見直しが指摘され、1999年に μ ITRON4.0仕様が公開されました。 μ ITRON4.0仕様は組込みシステム開発のオープン化に対する基盤となるべく構築された仕様です。

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



2008/12/05

TOPPERSプロジェクト認定

24



ITRON仕様の特徴について列記します。

- 1) OSの小型軽量化が可能であり、ワンチップマイコンでも、実質的な性能を発揮することができます。
- 2) 仕様の理解が容易です。これは、技術者の教育のための標準化の側面を重視して仕様策定した結果です。
- 3) 完全にオープンな標準仕様であり、ロイヤリティなしで実装することができます。
- 4) 8ビットワンチップマイコンから32ビットのRISCプロセッサまで、多種多様なプロセッサに対して実装可能であり、すでに、実装されている。そのため、異なるプロセッサへの移行は容易である。
- 5) 多くの組み込み機器で使用実績がある。
- 6) 多くのメーカやベンダでサポートしている。

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

2008/12/05

TOPPERSプロジェクト認定

25



μ ITRON4.0仕様について説明を行います。カーネルの機能は以下のものがあります。

- 1) タスク管理機能: マルチタスク機能を管理する基本的な機能
- 2) タスク付属同期機能: タスク機能に用意された基本的な同期機能
- 3) タスク例外処理機能: タスクに発生した例外事象の処理をタスクコンテキストで実行するための機能
- 4) 同期・通信機能: タスクとは独立したオブジェクトにより、タスク間の同期・通信を行う機能
- 5) 拡張同期・通信機能: タスクとは独立したオブジェクトにより、高度なタスク間の同期・通信を行う機能
- 6) メモリプール機能: ソフトウェアによって動的なメモリ管理を行う機能
- 7) 時間管理機能: 時間に依存した処理を行う機能
- 8) システム状態管理機能: システムの状態を変更・参照する機能
- 9) 割り込み管理機能: 外部割り込みによって起動される割り込みハンドラおよび割り込みサービスルーチンを管理するための機能
- 10) サービス管理機能: 拡張サービスコールの定義と呼び出しを行うための機能

μ ITRON4.0仕様には、より良いスケーラビリティを実現するために、フルセットと最小セットの間に、スタンダードプロファイルと自動車制御用プロファイルを設けている。

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

- 割り込みハンドラ(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

- 割り込みハンドラから呼び出せないAPIがある

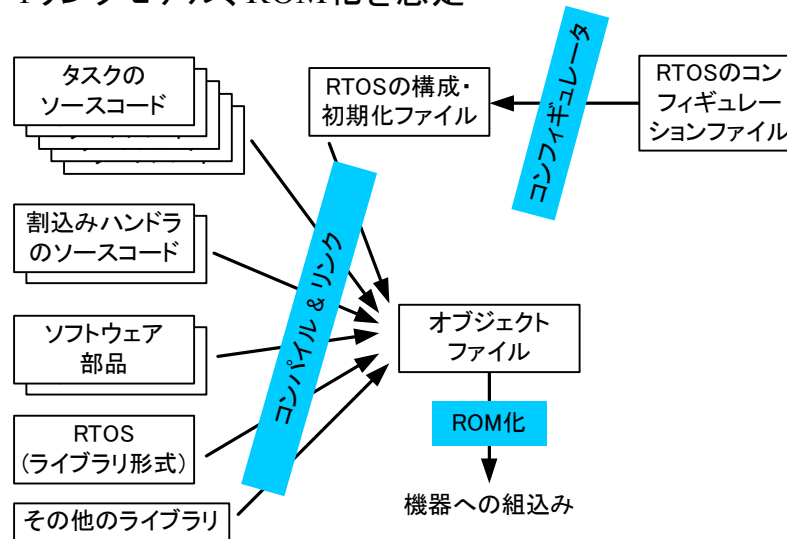
- 待ち状態に入るAPI

サービス・コールとは、タスクなどのアプリケーション・プログラムからリアルタイム・カーネルに要求を出すために使うインターフェースで、一般にはAPIと呼ばれます。 μ ITRON3.0仕様までは「システム・コール」と呼ばれていました。ITRON仕様のサービス・コールの名称にはガイドラインがあり、前半が操作方法を現し、後半は操作対象を表します。サービスコール「act_tsk」では、actは起動を表すactivateの省略、tskはタスクの省略です。サービス・コールが正常終了するとE_OKが戻り値として返されます。

割り込みハンドラ(非タスクコンテキスト)から呼び出されるサービス・コールは"i"で始まる名称として区別されます。非タスクコンテキストは、待ち状態がないため、待ち状態を要求するサービス・コールは呼び出せません。

RTOSを用いた組み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



2008/12/05

TOPPERSプロジェクト認定

27



RTOSを用いた組み込みソフトウェアの構築方法を説明します。基本的に1リンクモデルとし、ROM化を想定した方法について説明を行います。

ユーザープログラムは、タスクのソースコードとして作成します。割り込みに関する処理は割り込みハンドラのソースコードとして記述します。これらのプログラムに、ソフトウェア部品とライブラリ化されたRTOSとその他のライブラリを加え、RTOSのコンフィギュレーションファイルから生成されるRTOSの構成・初期化ファイルをコンパイル・リンクしてオブジェクトファイル化しROM化して機器に組み込みを行います。

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

2008/12/05

TOPPERSプロジェクト認定

28



RTOSの構成や初期状態を決める手順をRTOSのコンフィギュレーションと言います。どのような構成にするか、また、初期状態にするかはRTOS毎に異なります。例えば、RTOSとして用いる機能や用いない機能を変更したり、オブジェクトの最大数を設定したり、優先度の段階を設定したり、静的にオブジェクトを生成を行う場合は生成定義情報を設定することができます。

コンフィギュレーションによりRTOSのコードに変更を加え、コンフィギュレーション結果をひとつまたは複数のソースコードとして生成する機構をコンフィギュレーション機構と呼びます。

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

μITRON4.0仕様では、カーネルオブジェクトを静的に生成することが出来ます。コンフィギュレーションファイルを作成し、このファイル中にオブジェクトの生成情報を静的APIで記述しておくことにより、コンフィギュレータがソースコードを自動生成してくれます。

静的APIの例としては、CRE_TSKは静的に生成するタスクオブジェクトの属性を指定するものです。

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

2008/12/05

TOPPERSプロジェクト認定

30



μ ITRON4.0仕様では、システムを構成するオブジェクトを静的に生成するための方法(コンフィギュレータ)が標準化され、静的APIが規定されました。静的APIとは、コンフィギュレーション・ファイル内に記述する。オブジェクトの生成内容を示すAPIです。C言語APIのサービス・コールの名称は小文字で記述しますが、静的APIの名称は大文字で記述します。パラメータはC言語APIと基本的に同じですが、C言語APIで構造体として指定するパラメータは、静的APIではそのままの形で記述します。

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

2008/12/05

TOPPERSプロジェクト認定

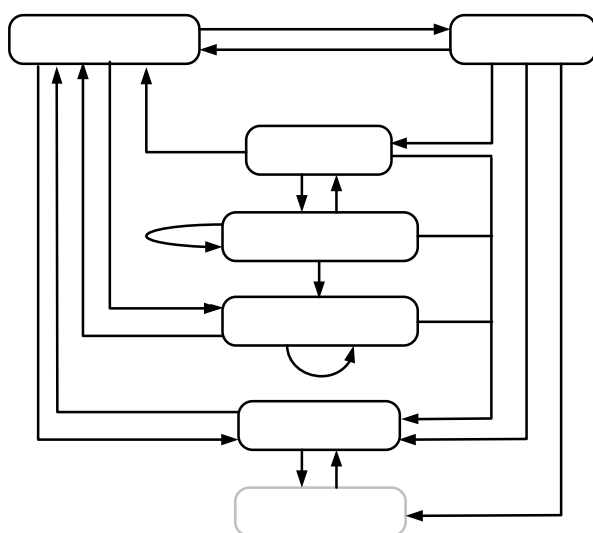
31



μ ITRON4.0仕様のマルチタスク機構について説明を行います。タスクのスケジューリング方式はプリエンプティブな優先度ベーススケジューリングです。同一優先度のタスクはキューイングされ、FCFS (First Come First Served) 方式で実行します。同一優先度でもキューイング順番を回転させる機能 (rot_rdq サービスコール) により実行順序の変更が可能です。優先度についても優先度変更機能 (chg_pri サービスコール) により動的に変更が可能です。

タスクの状態は、実行、実行可能、休止、待ち、強制待ち、二重待ち、未登録の7の状態をサポートしています。

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)

2008/12/05

TOPPERSプロジェクト認定

32



タスクの7つの状態

1) 実行状態(RUNNING)

現在そのタスクが実行中であるという状態。但し、非タスクコンテキストを実行している場合は、基本的に非タスクコンテキストの実行開始前に実行していたタスクを実行状態にあるものとする。

2) 実行可能状態(READY)

そのタスクは実行可能であるが、そのタスクより優先順位の高いタスクが実行中であるために、そのタスクが実行できない状態。

3) 待ち状態(WAITING)

何らかの条件が整うまで自タスクの実行を中断するサービスコールを呼び出されたことにより、実行が中断された状態

4) 強制待ち状態(SUSPENDED)

他のタスクによって、強制的に実行を中断させられた状態、但し μ ITRON4.0仕様では、自タスクを強制待ち状態にすることも出来る。

5) 二重待ち状態(WAITING-SUSPENDED)

待ち状態と強制待ち状態が重なった状態、待ち状態のタスクに対して、強制待ち状態への移行が行われると、二重待ち状態に移行する

6) 休止状態(DORMANT)

タスクがまだ起動されていないか、実行を終了した状態

7) 未登録状態(NON-EXISTENT)

タスクがまだ生成されていないか、削除された後のシステムに登録されていない仮想的に状態

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)

2008/12/05

TOPPERSプロジェクト認定

33



タスク管理機能は、タスクの状態を直接的に操作・参照するための機能です。タスクを生成・削除する機能、タスクを起動・終了する機能、タスクに対する起動要求をキャンセルする機能、タスクの優先度を変更する機能、タスクの状態を参照する機能が含まれます。タスクはID番号で識別されるオブジェクトです。タスクのID番号をタスクIDと呼びます。

タスクは、実行順序を制御するために、ベース優先度と現在優先度を持つ、単にタスクの優先度といった場合には、タスクの現在優先度を示します。タスクのベース優先度は、タスクの起動時にタスクの起動時優先度に初期化されます。タスクに対する起動要求はキューイングされます。つまり、すでに起動されたタスクを再起動しようとする、そのタスクを起動要求があったという記録が残り、後でそのタスクが終了した時に、タスクを自動的に再起動します。ただし、起動コードを指定してタスクを起動するサービスコール (sta_tsk) は、起動要求はキューイングされません。

リアルタイム・カーネルは、タスクの終了時に、ミューテックスのロック解除を除いては、タスクが取得した資源 (セマフォ資源、メモリブロックなど) を開放する処理を行いません。これらの資源の解放はアプリケーションで行わなければなりません。

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)

2008/12/05

TOPPERSプロジェクト認定

34



タスク付属同期機能は、タスクの状態を直接に操作することにより同期を行う為の機能です。タスクを起床待ちにする機能とそこから起床する機能、タスクの起床要求をキャンセルする機能、タスクの待ち状態を強制解除する機能、タスクを強制待ち状態へ移行する機能とそこから再開する機能、自タスクの実行を遅延する機能が含まれています。

タスクに対する起床機能は、キューイングされます。つまり、起床待ち状態でないタスクを起床しようとする、そのタスクを起床要求が記録として残り、後でそのタスクが起床待ちに移行しようとした時に、タスクを起床待ちにしません。

タスクに対する強制待ち要求は、ネストされます。つまり、すでに強制待ち状態(二重待ち状態を含む)になっているタスクを再度強制待ち状態に移行させようとする、そのタスクを強制待ち状態に移行させようとしたという記録が残り、後でそのタスクを強制待ち状態(二重待ち状態を含む)から再開させようとした時に、強制待ちからの再開を行いません。

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)
 - オーバーランハンドラ(オーバーラン時に呼ばれる)
- 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

2008/12/05

TOPPERSプロジェクト認定

35



時間管理機能は、時間に依存した処理を行うための機能です。システム時刻管理、周期ハンドラ、アラームハンドラ、オーバーランハンドラの各機能が含まれます。周期ハンドラ、アラームハンドラ、オーバーランハンドラを総称してタイムイベントハンドラと呼びます。

•システム時刻管理

システム時刻管理機能は、システム時刻を操作するための機能です。システム時刻を設定・参照する機能、タイムスティックを供給してシステム時刻を更新する機能が含まれます。

システム時刻は、システム初期化時に0に初期化します。以降、アプリケーションによってタイムスティックを供給するサービスコール(`isig_tim`)が呼び出される毎に更新します。`isig_tim`を呼びだれる度とにシステム時刻をどれだけ更新するかは、実装依存になります。但し、システム時刻を更新する機能をカーネル内部に持つことも可能で、その場合には`isig_tim`をサポートする必要はありません。

•タイムイベントハンドラ

周期ハンドラは、一定周期で起動されるタイムイベントハンドラです。周期ハンドラ機能には、周期ハンドラを生成・削除する機能、周期ハンドラの動作の開始・停止する機能、周期ハンドラの状態を参照する機能が含まれます。周期ハンドラはID番号で識別されるオブジェクトである。周期ハンドラのID番号を周期ハンドラIDと呼びます。周期ハンドラの起動周期と起動位相は、周期ハンドラの生成時に、周期ハンドラ毎に設定することができます。カーネルは、周期ハンドラの操作時に、設定された起動周期と起動位相から、周期ハンドラを次に起動すべき時刻を決定します。周期ハンドラの生成時には、周期ハンドラの生成時には、周期ハンドラを生成した時刻に起動位相を加えた時刻を、次に起動すべき時刻とします。周期ハンドラを起動すべき時刻になると、その周期ハンドラの拡張情報(`exinf`)をパラメータとして、周期ハンドラを起動します。また、このとき、周期ハンドラの起動すべき時刻に起動周期を加えた時刻を、次に起動すべき時刻を決定し直す場合があります。周期ハンドラの起動位相は、起動周期以下であることを原則とします。起動位相に起動周期よりも長い時間が指定された場合の動作は実装依存となります。周期ハンドラは、動作している状態か、動作していない状態かのいずれかの状態を取ります。周期ハンドラが動作していない状態の時は、周期ハンドラを起動すべき時刻になっても周期ハンドラを起動せず、次に起動すべき時刻の決定のみ行います。周期ハンドラの動作を開始するサービスコール(`sta_cyc`)が呼び出されると、周期ハンドラを動作している状態に移行させ、必要なら周期ハンドラを次に起動すべき時刻を決定しなおします。周期ハンドラの動作を停止させるサービスコール(`stp_cyc`)が呼び出されると、周期ハンドラを動作していない状態に移行させます。周期ハンドラを生成した後どちらの状態になるかは、周期ハンドラの属性によって決めることができます。

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

2008/12/05

TOPPERSプロジェクト認定

36



アラームハンドラは、指定した時刻に起動されるタイムイベントハンドラです。アラームハンドラ機能には、アラームハンドラを生成・削除する機能、アラームハンドラの動作を開始・停止する機能、アラームハンドラの状態を参照する機能が含まれています。アラームハンドラはID番号で識別されるオブジェクトです。アラームハンドラのID番号をアラームハンドラIDと呼びます。

アラームハンドラを起動する時刻(これをアラームハンドラの起動時刻と呼びます)は、アラームハンドラ毎に設定することができます。アラームハンドラの起動時刻になると、そのアラームハンドラの拡張情報(exinf)をパラメータとして、アラームハンドラを起動します。アラームハンドラの生成直後には、アラームハンドラの起動時刻は設定されておらず、アラームハンドラの動作は停止状態です。アラームハンドラの動作を開始するサービスコール(sta_arm)が呼び出されると、アラームハンドラの起動時刻を、サービスコールが呼び出された時刻から指定された相対時刻後に設定します。アラームハンドラの動作を停止するサービスコール(stp_arm)が呼び出されると、アラームハンドラの起動時刻を解除しアラームハンドラの動作を停止します。

オーバランハンドラは、タスクが設定された時間を越えてプロセッサを使用した場合に起動されるタイムイベントハンドラです。オーバランハンドラ機能には、オーバランハンドラを定義する機能、オーバランハンドラの動作を開始・停止する機能、オーバランハンドラの状態を参照する機能が含まれます。

オーバランハンドラの起動条件として設定される時間(これをタスクの上限プロセッサ時間と呼びます)は、タスクごとに設定ができます。カーネルは上限プロセッサ時間が設定されたタスクに対して、上限プロセッサ時間が設定されて以降にそのタスクが使用した累積プロセッサ時間(これをタスクの使用プロセッサ時間と呼びます)を管理し、タスクの使用プロセッサ時間が上限プロセッサ時間を越えると、オーバランハンドラを起動します。システムに定義できるオーバランハンドラは1つだけであるため、オーバランハンドラには、起動の原因となったタスクのID番号(tsikid)と拡張情報(exinf)をパラメータとして渡します。タスクの生成直後には、タスクの上限プロセッサ時間は設定されていません。オーバランハンドラの動作を開始するサービスコール(sta_ovr)が呼び出されると、指定されたタスクの上限プロセッサ時間の設定を解除する。タスクの上限プロセッサ時間の設定は、そのタスクが原因となってオーバランハンドラを起動する時や、そのタスクが終了する時にも解除を行う。タスクが使用したプロセッサ時間には、少なくとも、そのタスク(タスク例外処理ルーチンを含む)とたそのタスクから呼び出したサービスコールの実行時間が含まれる。逆に、他のタスク(タスク例外処理ルーチンを含む)と他のタスクから呼び出したサービスコール実行時間は含まれません。

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?

システム状態管理機能は、システムの状態を変更・参照するための機能です。タスクの優先順位を回転する機能、実行状態のタスクIDを参照する機能、CPUロック状態への移行・解除を行う機能、タスクのディスパッチを禁止・解除する機能、コンテキストやシステム状態を参照する機能などがあります。

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

2008/12/05

TOPPERSプロジェクト認定

38



割込み管理機能は、外部割込みによって起動される割込みハンドラおよび割込みサービスルーチンを管理する為の機能です。割込みハンドラを定義する機能、割込みサービスルーチンを生成・削除する機能、割込みサービスルーチンの状態を参照する機能、割込みを禁止・許可する機能、割込みマスクを変更・参照する機能が含まれます。割込みサービスルーチンはID番号で識別されるオブジェクトである。割込みサービスルーチンのID番号は割込みサービスルーチンIDと呼びます。

割込み管理機能では、以下のデータ型を持ちます。

INHNO 割込みハンドラ番号
 INTNO 割込み番号
 IXXXX 割込みマスク

割込みマスクのデータ型の一部のXXXXは、実装定義でターゲットプロセッサのアーキテクチャに応じた適切な文字列に定めます。割込みハンドラの記述形式は実装依存です。割込みサービスルーチンを起動する際には、その割込みサービスルーチンの拡張情報(exinf)をパラメータとして渡します。割込みサービスルーチンのC言語による記述形式は以下の通りです。

```
void isr (VP_INT exinf)
{
    割込みサービスルーチン本体
}
```

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、タスクに対して例外を発生させることが可能
 - 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

<code>def_exc</code>	CPU例外ハンドラの定義
<code>ATT_INI</code>	初期化ルーチンの追加

2008/12/05

TOPPERSプロジェクト認定

39



CPU例外とは、プロセッサ(CPU)が発行する割込みに類似した機能で、プロセッサの実行の異常や特別な処理を要求する場合に発行されます。 μ ITRON4.0仕様ではCPU例外はCPU例外ハンドラによってアプリケーション(プラットフォーム)に取り込むことができます。カーネルはCPUハンドラの前後に行うべき処理(出入口処理)を用意し、CPU例外発生時、出入口処理を経由してCPU例外ハンドラを起動します。CPU例外ハンドラはCPU例外ハンドラ番号によって識別されます。例外処理では、以下のデータ型を持ちます。

EXCNO CPU例外ハンドラ番号

CPU例外ハンドラの記述形式は実装依存です。CPU例外の出入口処理はCPU例外ハンドラを起動する際には、CPU例外ハンドラに拡張情報(**exinf**)をパラメータとして渡します。CPU例外ハンドラのC言語による記述形式は以下の通りです。**exinf**の内容は実装依存でCPU例外発生時のCPUレジスタの内容やCPUの状態を表すステータスレジスタの内容を保存した領域を示します。通常のCPUではCPU例外は割込みのレベルに関係なく発生し、マスクの設定等はありません。

```
void exc(VP_INT exinf)
{
    CPU例外ハンドラルーチン本体
}
```

初期化ルーチンはカーネルの初期化時に実行するプログラムです。このルーチンは**ATT_INI**:初期化ルーチンの追加静的APIにより、定義することができます。

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?

2008/12/05

TOPPERSプロジェクト認定

40



タスク例外処理機能は、タスクを指定してタスク例外処理を要求するサービスコールを呼び出すことで、指定したタスクに実行中の処理を中断させ、タスク例外処理を実行させる機能です。CPU例外と異なる点はタスクと同じコンテキストで実行される点です(CPU例外は割込みと同じ非タスクコンテキストで実行されます)。タスク例外処理が終了すると、タスクは中断した時点から実行が継続されます。

μ ITRON4.0仕様ではタスク例外機能は静的APIとC言語APIの両方が定義されています。タスク例外処理ルーチンは起動時に保留例外要因(**texptn**)とタスクの拡張情報がパラメータして渡されます。C言語記述とアセンブラ記述が選択でき、C言語記述の場合以下の記述となります。

```
void texrtn (TEXPTN texptn, VP_INT exinf)
{
    タスク例外ルーチン本体
}
```

上記記載の**sns_tex**はタスク例外の禁止状態を参照するサービスルーチンで、自タスクのタスク例外禁止状態の場合にTRUE、タスク例外許可状態の場合にFALSEを返します。

ITRONの基礎知識

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

ITRONの同期・通信機能

μ ITRON4.0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

2008/12/05

TOPPERSプロジェクト認定

42



μ ITRONでは、タスク間同期・通信のために以下の機能を用意しています。

• 共有メモリによる通信、排他制御のための機能

- セマフォ CRE_SEM、cre_sem、acre_sem、del_sem、sig_sem、isem_sig、wai_sem
pol_sem、twai_sem、ref_sem
- イベントフラグ CRE_FLG、cre_flg、acre_flg、del_flg、set_flg、iset_flg、clr_flg、wai_flg
pol_flg、twai_flg、ref_flg
- ミューテックス CRE_MTX、cre_mtx、acre_mtx、del_mtx、loc_mtx、ploc_mtx、tloc_mtx
unl_mtx、ref_mtx

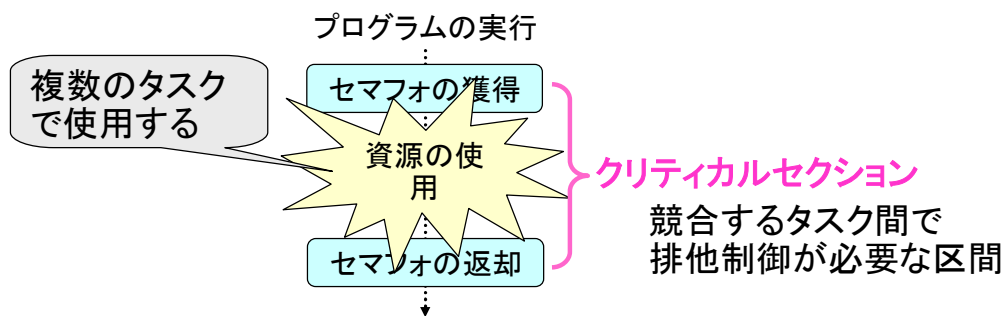
• メッセージ通信のための機能

- データキュー CRE_DTQ、cre_dtq、acre_dtq、del_dtq、snd_dtq、psnd_dtq、ipsnd_dtq
tsnd_dtq、fsnd_dtq、ifsnd_dtq、rcv_dtq、prcv_dtq、trcv_dtq、ref_dtq
- メールボックス CRE_MBX、cre_mbx、acre_mbx、del_mbx、snd_mbx、rcv_mbx、prcv_dtq
trcv_dtq、ref_mbx
- メッセージバッファ
CRE_MBF、cre_mbf、acre_mbf、del_mbf、snd_mbf、psnd_mbf、tsnd_mbf
rcv_mbf、prcv_mbf、trcv_mbf、ref_mbf
- ランデブ CRE_POR、cre_por、acre_por、del_por、cal_por、tcal_por、acp_por
pacp_por、tacp_por、fwd_por、rpl_rdv、ref_por、ref_rdv

セマフォ、イベントフラグ、データキュー、メールボックスはスタンダードプロファイルとして用意しています。

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。



2008/12/05

TOPPERSプロジェクト認定

43



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

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

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

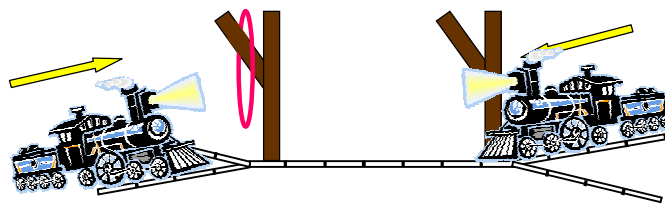
セマフォ(Semaphore) : サービスコール&語源

- 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

2008/12/05

TOPPERSプロジェクト認定

44



μ ITRON4.0仕様のセマフォのサービス・コールは以下の通りです。

1. セマフォの生成

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

ID semid セマフォID番号

ATR sematr セマフォ属性(TA_FIFO || TA_TPRI)

UINT isemcnt セマフォ資源数の初期値

UNIT maxsem セマフォの最大資源数

2. セマフォ資源の返却

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

ID semid 資源返却対象のセマフォ番号

3. セマフォ資源の取得

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

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

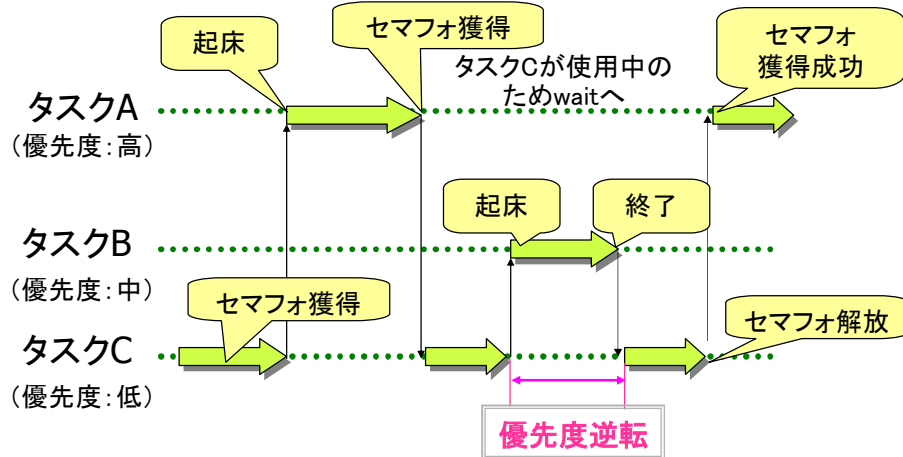
ID semid 資源獲得対象のセマフォ番号

TMO tmout タイムアウト指定(twai_semのみ)

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)



2008/12/05

TOPPERSプロジェクト認定

45

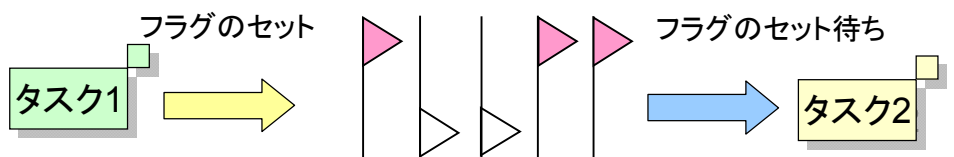


優先度逆転とは、排他制御の設定等により、本来優先度の高いタスクが優先度の低いタスクの実行により待たされる現象を指します。優先度逆転により、デッドラインミス(機器が一定時間以内に処理しなければならない処理を処理できない現象)が発生しやすくなります。上記の例では、優先度の低いタスクの排他制御中に、実行を待っているタスクより優先度が低いタスクが起床して、予定以上の実行遅延が発生しています。

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



2008/12/05

TOPPERSプロジェクト認定

46



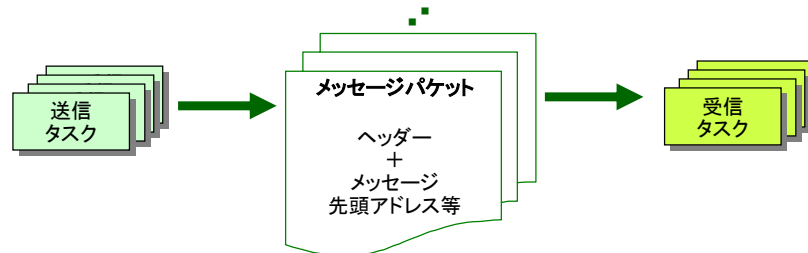
イベントフラグは、イベントの有無をビットごとのフラグで表すことにより、タスク間の同期を行うためのオブジェクトです。

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

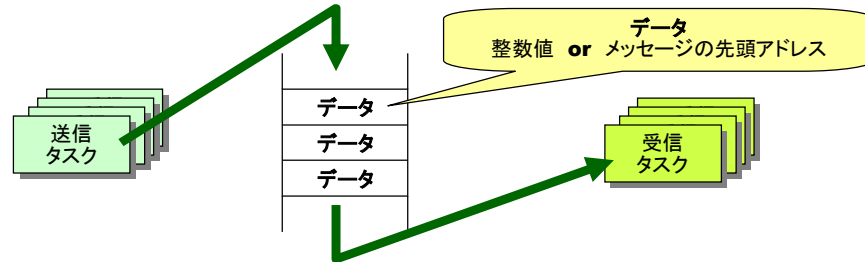
イベントフラグ機能には、イベントフラグを生成、削除する機能、イベントフラグをセット、リセットする機能、イベントフラグを設定を待つ機能、イベントフラグの状態を参照する機能で構成されています。

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



2008/12/05

TOPPERSプロジェクト認定

47



メールボックスやデータキューは二つのタスクの間でデータをやり取るための機構です。セマフォやイベントフラグと異なり、受信側のタスクはデータは蓄積しておき送信側の状態と関係なく、自分の好きなタイミングでデータに応じた処理を行うことができます。これを非同期通信といいます。データキューは1ワードデータ通信を行うことができます。メールボックスはヘッダー部以外の構造を定義していませんが、をメモリプールと併用して通信を行うとメッセージデータの生成・削除機能を持つメッセージ通信が可能となります。

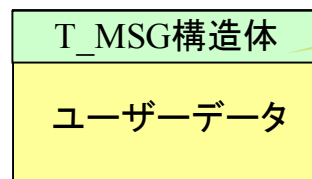
データキューやメールボックスでもシステムの構成により同期通信は可能です。例えば、データキューはデータ数を1にすると同期した通信も可能となります。また、メールボックスを双方向に設定して、コマンドとアンサー形式で通信を行うと、同期形式の通信を行いことができます。

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

2008/12/05

TOPPERSプロジェクト認定

48



μ ITRON4.0のメールボックス機能のサービス・コールは以下の通りです。

1. メールボックスの生成

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

ID mbxid 生成対象のメールボックスのID番号

ATR mbxatr メールボックス属性((TA_TFIFO||TA_PRI) | TA_MFIFO||TA_MPRI)

PRI mprihd 優先度別のメッセージキューヘッダ領域の先頭番地

2. メールボックスへの送信

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

3. メールボックスからの受信

ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg);

ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg);

ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);

T_MSG構造体の型定義は以下の通りです。

```
typedef struct t_msg {
    struct t_msg *next;
} T_MSG;
```


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

2008/12/05

TOPPERSプロジェクト認定

49



μ ITRON4.0のデータキュー機能のサービス・コールは以下の通りです。

1. データキューの生成

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

ID dtqid 生成対象のデータキューのID番号

ATR dtqatr データキュー属性(TA_TFIFO||TA_PRI)

UINT dtqcnt データキュー領域の容量(データの個数)

VP dtq データキュー領域の先頭番地

2. データキューへの送信

ER ercd = snd_dtq(ID dtqid, VP_INT data);

ER ercd = psnd_dtq(ID dtqid, VP_INT data);

ER ercd = ipsnd_dtq(ID dtqid, VP_INT data);

ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);

3. データキューへの強制送信

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

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

4. データキューからの受信

ER ercd = rcv_dtq(ID dtqid, T_INT *p_data);

ER ercd = prcv_dtq(ID dtqid, T_INT *p_data);

ER ercd = trcv_dtq(ID dtqid, T_INT *p_data, TMO tmout);

まとめ

- 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.

2008/12/05

TOPPERSプロジェクト認定

50



μITRONの代表的な同期・通信機能は以下の4つの機能です。

- 1) セマフォ
- 2) イベントフラグ
- 3) メールボックス
- 4) データキュー

システム設計で同期・通信機能を実装する場合、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です。特に、メールボックスやデータキューのような通信機能で非同期通信を実装する場合、多用しすぎると不具合発生時に問題解析が難しくなるケースが多く、注意が必要です。また、各機能の細かいパラメータの設定で、タスクの振る舞いが変わるので、プログラミングの際には注意して実装を行ってください。

ITRONの基礎知識

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

TOPPERS/JSPカーネルについて

- JSPとは
 - JSP = **J**ust **S**tandard **P**rofile
 - TOPPERSプロジェクトで開発されている μ ITRON4.0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
- 開発の目的
 - μ ITRON4.0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/JSP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

ITRON導入実習：実習内容

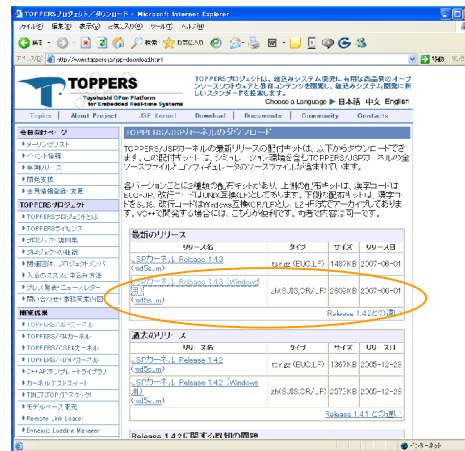
- TOPPERS/JSPカーネルの導入実習を行う
- TOPPERS/JSPカーネルについての説明
- カーネルのダウンロードと環境づくり
 - ダウンロードとツールを作成します
- 開発・実行環境の使用方法についての説明
 - 開発環境と2種類の実行環境の使用方法について解説
- カーネルのビルドと実行(デバグガ使用版)
 - M16C用カーネルをビルドし、サンプルプログラムを実行します
- サンプルプログラムの説明
 - サンプルプログラムの説明と実行確認

TOPPERS/JSP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/JSPのダウンロード

- TOPPERSプロジェクトのWebから最新版のjspカーネル(Windows版)をダウンロードします
- TOPPERSのディレクトリで解凍してください
- 解凍後、バージョンが判るように、ディレクトリ名をjspからjsp-1.4.3に変更してください



2008/12/05

TOPPERSプロジェクト認定

57



まず、TOPPERSプロジェクトのWebからJSPカーネルの最新版をダウンロードします。

<http://www.toppers.jp/jsp-download.html>

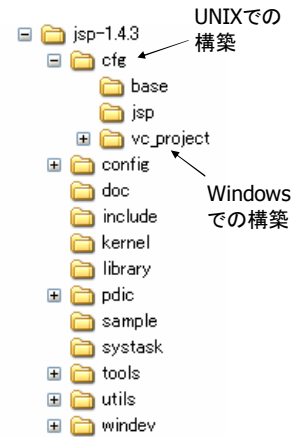
最新のリリースからRelease 1.4.3をダウンロードします。

LINUXやCygwin上で開発を行う場合は、tar.gz(EUC,LF)タイプのJSPカーネル Release 1.4.3を今回はM16C用なので、lzh(SJIS,CR/LF)タイプのJSPカーネル Release 1.4.3 (Windows用)をダウンロードします。LINUXやCygwin上で開発を行うにはGNU等のクロス開発環境を構築する必要があります。M16C用のクロス開発環境は本セミナーで構築した開発環境がそのまま使用できます。

ダウンロードしたjsp-1.4.3.lzhを適当なディレクトリで解凍を行ってください。jspというディレクトリができます。他のJSPのバージョンと区別するために、ディレクトリ名をjsp-1.4.3に書き換えてください。

コンフィギュレータの構築

- JSPは μ ITRON4.0のスタンダードプロファイルに準拠しているため、静的APIをプログラミング言語に変換するコンフィギュレータが用意されている
- LINUXやCygwinでは、cfgの下でMakefileを用いてコンフィギュレータのビルドを行う
- Windowsではcfg/vc_projectの下でVC++用のワークスペースを用いてビルドを行う(VC++が必要)



2008/12/05

TOPPERSプロジェクト認定

58



JSP用の開発環境の構築1(コンフィギュレータの構築)

TOPPERS/JSPは μ ITRON4.0仕様スタンダードプロファイルに準拠しているため、静的APIで記述したコンフィギュレーションファイルをC言語のインクルードまたはソースファイルに変換を行う必要があります。この機能を行うのがコンフィギュレーションツールです。TOPPERS/JSPのコンフィギュレーションツールは変換を行うコンフィギュレータ(cfg)とパラメータの検証を行うパラメータチェックプログラム(chk)で構成されます。これらのツールはC++言語で記述されていますので、実行できるようにビルドを行わなければなりません。

•LINUX・Cygwin上でのビルド

ビルドには、セルフコンパイラを使用します。cfgディレクトリに移動して、make dependで依存関係ファイル(Makefile.depend)を生成した後、makeコマンドでビルドを行います。

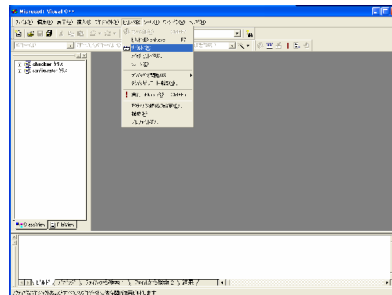
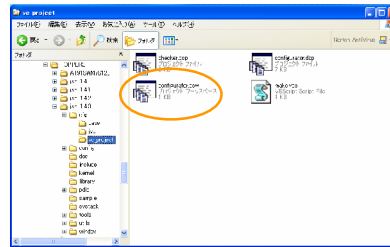
```
% cd cfg
% make depend
% make
```

•Windows上でのビルド

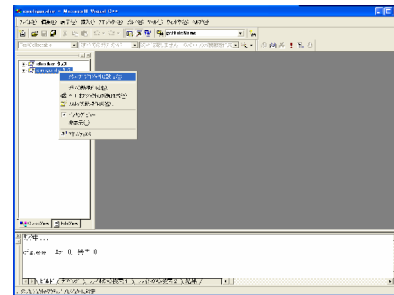
Windows上でビルドを行うためには、マイクロソフト社のC++言語開発環境(VC++)が必要です。この環境のワークスペースがcfg/vc_projectディレクトリに用意されています。

VC++を用いたコンフィギュレータのビルド

- configurater.dswをダブルクリックするとVC++が起動します
- checkerクラスとconfiguratorクラスをプロジェクトに選んで、ビルドを行ってください



chkのビルド



Configuratorプロジェクトの選択

2008/12/05

TOPPERSプロジェクト認定

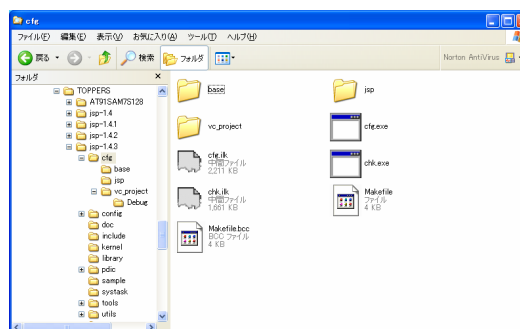
59



Windows上でのコンフィギュレーションツールのビルド方法について説明を行います。
configurater.dswをダブルクリックするとVC++が起動します。**configurator**クラスをプロジェクトに指定してビルドを行ってください。**cfg.exe**が生成されます。**checker**クラスをプロジェクトに指定してビルドを行ってください。**chk.exe**が生成されます。

コンフィギュレータの確認

- cfgディレクトリの下にcfg.exeとchk.exeが生成されます
- 本実習ではVC++を用意できないため、あらかじめcfg.exeとchk.exeを用意しました
- これを、皆さんのcfgディレクトリの下にコピーしてください



本セミナーでは、VC++をパソコンにセットアップしてありません。開発環境のjsp-1.4.3/cfgディレクトリの下に、**cfg.exe**と**chk.exe**を用意しましたので、皆さんのパソコンの先ほど解凍したディレクトリjsp-1.4.3/cfgの下にコピーしてください。

ユーティリティの構築

- JSPのプログラム構築では、TCBのビット位置の生成や割込みベクトル・テーブルを自動生成するためのユーティリティを使用します
- LINUXやCygwinではユーティリティはPERLスクリプトで記述されていますので、ビルド等は必要ありません
- Windows上ではPERLスクリプトの実行は難しいので、M16Cのユーティリティは、C++言語で用意しました
 - M16CのGNU版コンパイラはないので、NC30WAを使ったWindows上の開発環境に合わせる必要がある
- M16C用の2つのユーティリティもVC++を用いて構築する必要があります

2008/12/05

TOPPERSプロジェクト認定

61



TOPPERS/JSPではC言語で記述したタスク管理情報(TCB)をアセンブラにうまく引き渡すためのインクルードファイルを生成したり、実装依存で割込みベクトル・テーブルを自動的に生成するための専用ユーティリティを使用します。LINUXやCygwin環境では、これらのユーティリティはPERLスクリプトで記述されていますので、実行形式に変換を行う必要はありません。Windows環境では、PERLスクリプトの実行は難しいので、TOPPERS/JSPのM16Cのユーティリティはコンフィギュレーションツールと同様にC++言語で用意しました。特にNC30WAアセンブラはGNUアセンブラとは定数の記述に差異があり、NC30WAアセンブラ専用のアセンブル記述データに変換する必要がありました。

M16C用のユーティリティは以下の2つがあります。

(1) m16coffset.exe

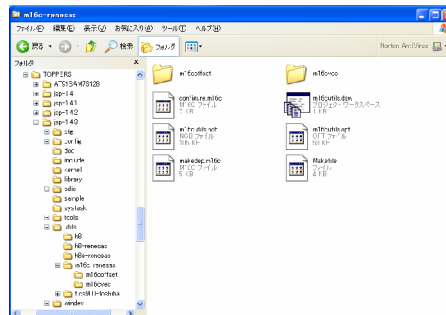
makeoffset.cからTCB構造体の項目のオフセットを取り出しoffset.incに変換する。

(2) m16cvec.exe

kernel_cfg.cから割込みベクタ情報を取り出し割込みベクター定義ファイルm16cvec.incを生成する。

M16C専用ユーティリティのビルド

- utils/m16c-renesasの下でm16cutils.dswワークスペースからVC++を起動してm16coffset.exeとm16cvec.exeをビルドしてください
- 本セミナーではVC++が用意できないため、2つのプログラムを用意しましたのでutils/m16c-renesasの下にコピーしてください



2008/12/05

TOPPERSプロジェクト認定

62



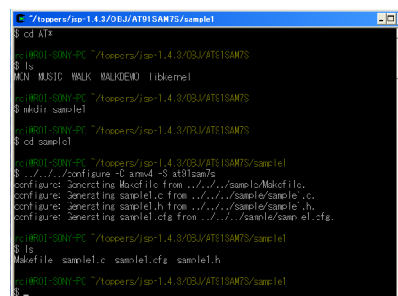
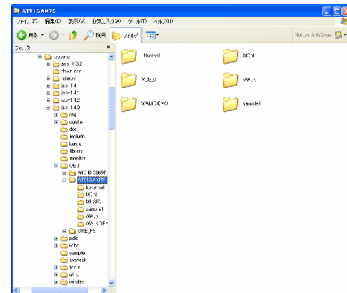
ユーティリティプログラムをビルドするVC++用ワークスペースm16cutils.dswをjsp-1.4.3/utils/m16c-renesasディレクトリの下にありますので、それを使ってビルドを行ってください。

ワークスペースをうまく開けない場合は、m16cutils.dsw等が文字化けしている可能性がありますので、jsp-1.4.2等過去のdswと入れ替えてください。

本セミナーでは、VC++をパソコンにセットアップしてありません。開発環境のjsp-1.4.3/utils/m16c-renesasディレクトリの下に、m16coffset.exeとm16cvec.exeを用意しましたので、皆さんのパソコンの先ほど解凍したディレクトリjsp-1.4.3/utils/m16c-renesasの下にコピーしてください。

TOPPERS/JSP用ワークスペースの作成

- 開発用のワークスペースを作成します
- LINUXやCygwin環境ではGNUの構築環境と同じようにconfigureを用いてワークスペースを作成します
- OBJ/AT91SAM7S/sample1をサンプルプログラム構築用のワークスペースしたい場合、ベースラインのconfigureコマンドを呼び出して環境構築します



2008/12/05

TOPPERSプロジェクト認定

63



TOPPERS/JSP用のワークスペースを作成します。教育WGでは以下の約束事でワークスペースを構築することになります。

jsp-1.4.3/OBJ/[大文字のシステム名]/[開発プログラム名]

この約束事項にのっとり、ATMEL社のarm7であるat91sam7s用のTOPPERS/JSPのサンプルプログラムであるsample1を作成すると以下のワークスペースディレクトリを作成します。

jsp-1.4.3/OBJ/AT91SAM7S/sample1

このワークスペースで、LINUXやCygwinの場合、configureを用いてsample1用の開発環境をこうちくするには以下のコマンドを実行します。

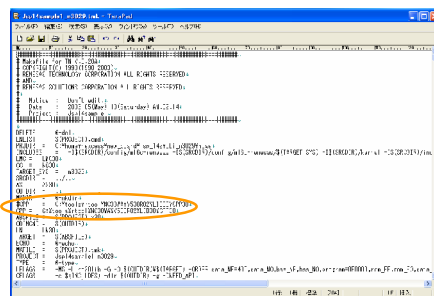
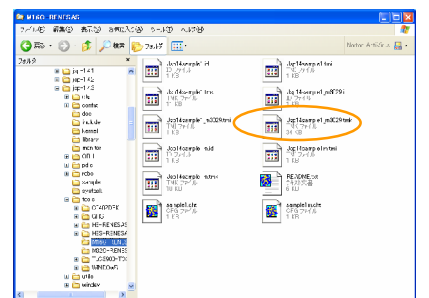
```
% mkdir OBJ/AT91SAM7S/sample1
% cd OBJ/AT91SAM7S/sample1
% ../../configure -C arm7 -S at91sam7s
```

TOPPERS/JSP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

M16C-M3029用のサンプルワークスペース

- M16C用のサンプル作成用ワークスペースはtools/M16C-RENASASに用意されています
- TMからJsp14sample1_m3029.tmkをプロジェクトとして開けば開発環境を構築できます
- Jsp14sample1_m3029.tmkのコンパイラ指定(CPPの設定)が実際の環境とあっていませんのでエディタで修正しておいてください



CPP = %MTOOLENT%LIB30%CPP30

2008/12/05

TOPPERSプロジェクト認定

65



M16Cは特殊な開発環境を用いるためtools/M16C-RENASASディレクトリの下にsample1のビルド環境を用意しました。TMを使ってビルドが行えるように3つのtmkファイルがあります。

- (1) Jsp14sample1m.tmk OASKS16miniボード用
- (2) Jsp14sample1.tmk OASKS16ボード用
- (3) Jsp14sample1_m3029.tmk HSB16C29S64NE用

今回はJspsample1_m3029.tmkを使用します。ダウンロードしたJspsample1_m3029.tmkのコンパイラのパス設定がTOPPER/JSP開発グループのパス設定となっており、皆さんの開発環境とはことなります。このままではコンパイラを起動できないため、直接Jspsample1_m3029.tmkの書換えを行ってください。(TM) 起動までに書換えをおこなうこと。

エディタで、25行目の以下の記述を

CPP = C:%tools%rtool%NC30WA%V530R02%LIB30%CPP30

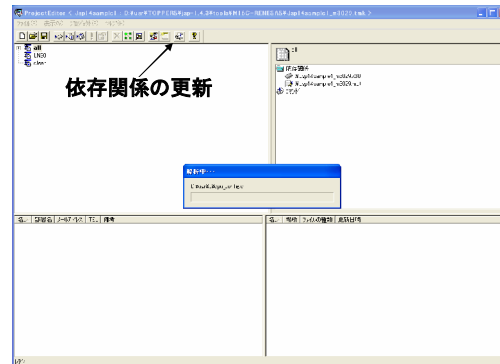
以下の記述に修正してください。

CPP = %MTOOLENT%LIB30%CPP30

プロジェクトの設定と依存関係の更新

- TMでtools/M16C-RENESAS/ Jsp14sample1_m3029.tmkをプロジェクトとして開き、プロジェクトエディタを呼び出してください
- プロジェクトエディタ中の「依存関係の更新」ボタンを押し、依存関係を再構築してください

TMK中の各ファイルの依存関係が、皆さんの環境と異なりますので、ビルド時コンパイルエラーとなります



2008/12/05

TOPPERSプロジェクト認定

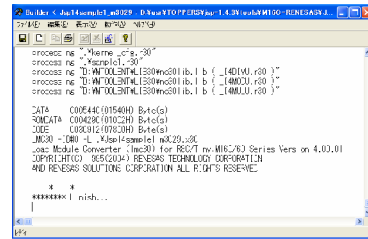
66



コンパイラのパス以外にも、設定が異なるものが多々ありますが、TMのプロジェクトエディタ中の[依存関係の更新]を行うと、これらを修復してくれます。[依存関係の修復]をせず、コンパイルを行うと、参照のファイルの場所が異なるためコンパイルエラーが発生します。

サンプルプログラムのビルド

- 「リビルド」からサンプルプログラムをビルドする
 - プログラムと同時にTOPPERS/JSPカーネルのソースコードも同時にコンパイルする
 - ワーニングが幾つか表示されるが無視すること
- TOPPERS/JSPカーネルを使ったプロジェクトは一部の依存関係が不十分なため(kernel.cfgの生成), 常に「リビルド」によりビルドを行ってください

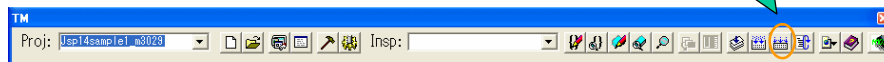


正常にビルドできると以下の
ファイルを作成する

Jsp14sample1_m3069.mot

Jsp14sample1_m3069.x30

リビルド



2008/12/05

TOPPERSプロジェクト認定

67



TMを使って、Jsp14sample1_m3029.tmkをプロジェクトファイルとして開き、[依存関係の修復]の後、[リビルド]ボタンを押して、リビルドを行ってください。コンパイル、リンクが正常に行われると以下のファイルが生成されます。

Jsp14sample1_m3069.mot

Jsp14sample1_m3069.x30

コンソールの出力先の設定

- TOPPERS/JSPカーネルは、コンソールの出力先としてUARTを使用する
- 出力先はカーネルのソースコード中にあるマクロで変更可能(初期設定はUART0)
 - config/m16c-renesas/m3029/sys_config.h
 - 1とするとUART0, 2とするとUART1が出力先となる

```
#define LOGTASK_PORTID 1
#define CONSOLE_PORTID 1
```

- 直接実行版では、フラッシュメモリ書き込みソフトがUART1を使用するため、UART1に変更した方が都合がいい
- デバグ使用版では、デバグがUART1を使用するため、カーネルでは、常にUART0を使用する必要がある

2008/12/05

TOPPERSプロジェクト認定

68



TOPPERS/JSPはバナー表示やログの表示先としてコンソールを使用します。コンソールはUARTに割り当てられており、HSB16C29S64NEでは2つのUARTが使用できます。コンソール使用するUARTの設定は、config/m16c-renesas/m3029ディレクトリ中のsys_config.hの修正で変更が可能です。現状は以下の設定となっています。

```
#define LOGTASK_PORTID 1  ## UART0を指定
```

```
#define CONSOLE_PORTID 1  ## UART0を指定
```

この設定を2に変更するとUART1にコンソールを変更できます。

```
#define LOGTASK_PORTID 2  ## UART1を指定
```

```
#define CONSOLE_PORTID 2  ## UART1を指定
```

デバッグ等を行う場合、フラッシュROMへの書き込みソフトやデバグがUART1を使用する。デバグを使用する場合、コンソールをUART1にするとデバグの通信データに、コンソールへの出力データが混じり正常に動作しなくなるため注意が必要です。

コンソールの設定(デバッグ使用と直接実行)

- M16C用JSPはデバッグ使用と直接実行の2つの実行環境を選べます
- デバッグ使用版(デバッグ時の実行環境)
 - デバッグ(KD30)を使用してプログラムをダウンロードして実行
 - ソースコードレベルデバッグが可能
 - ホストPCにシリアルポートが1ポート必要
 - UART0と接続する
- 直接実行版(商品化時の実行環境)
 - プログラムをフラッシュメモリに書き込み実行
 - ソースコードレベルデバッグは不可能
 - タスクモニタを用いたデバッグは可能

UART0



本セミナーではデバッグを使い開発を行います。コンソールをUART0に設定しますので、UART0用のコネクタとパソコン本体のシリアルポートを接続して使用してください。

2つの環境での実行方法

- デバッガ使用版
 - TMのツールバーからKD30を実行してダウンロードして、実行する
- 直接実行版
 - FlashStaを使って、Jsp14sample1_m3029.motをフラッシュメモリに書き込みます
 - 書き込みが終了したら **FlashStaを終了させた後**、SW1をオフ(右側)にしてリセットを押す



2008/12/05

TOPPERSプロジェクト認定

70

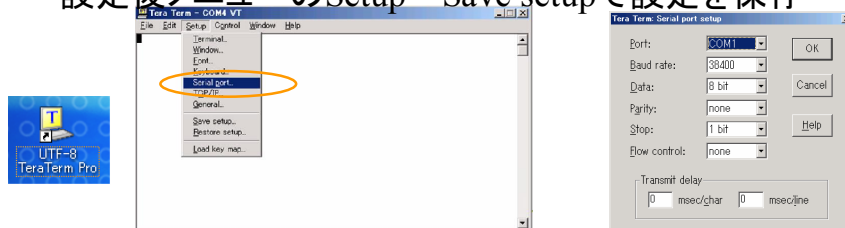


デバッガを用いたデバッグ方法はJsp14sample1_m3029.x86データをデバッガを用いて、ダウンロードし実行します。

フラッシュROMに書き込んでデバッグを行う場合、SW1をオン(左側)に設定してボードの電源をいれ、FlashStaを用いてJsp14sample1_m3029.motをフラッシュROMに書き込みます。書き込みが終わったら、FlashStaを終了させてから、SW1をオフ(右側)にしてリセットを押して、プログラムを実行させていただきます。

TeraTermの実行(コンソール表示・入力用)

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



2008/12/05

TOPPERSプロジェクト認定

71



コンソールからの表示、データの入力用にTeraTermを使用します。UARTでのデータ通信は、ポートとパソコンの双方で通信スピード(BPS)やデータフォーマットをあわせる必要があります。

TOPPERS/JSPでの設定は、`config/m16c-renesas/m3029/sfruart.c`にて設定を行っています。このソースを修正することにより、設定変更が可能です。現状の設定値は以下の通りです。

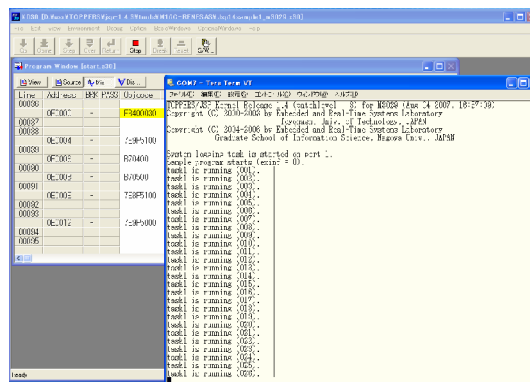
Baudrate:	38400bps
Data:	8bit
Parity:	none
Stop bit	1bit
Flow control:	なし

TeraTermの[Setup]→[Serial Port]設定にて、設定をあわせてください。

設定変更後[Setup]→[Save Setup]設定にて、設定を保存してください。

KD30デバッガを用いての実行確認

- KD30デバッガを起動して、TOPPERS/JSPサンプルプログラムを起動してください
- TeraTermに表示が行われることを確認してください



KD30デバッガを起動しビルドしたプログラムをダウンロードし、その後実行してください。TeraTerm上にsample1プログラムを実行していることを確認してください。

TOPPERS/JSP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/JSP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

2008/12/05

TOPPERSプロジェクト認定

74



sample1プログラムはJSPカーネルの実行テスト用のプログラムです。sample1プログラムの機能を用いて演習を行います。まず、その前にsample1プログラムの内容について説明を行います。

sample1プログラムは4つのタスクを実行します。そのうち3つは同じプログラムで並列に実行します。このタスクを並列実行タスクと呼びます。もうひとつのタスクは3つの並列実行タスクを管理するタスクで、コンソールからの入力コマンドを並列実行タスクに伝えて実行指示を行います。このタスクをメインタスクと呼びます。並列実行タスクはタスクの起動時、引数として管理番号(1-3)を与えられ、自分の管理番号に従った動作を行います。並列実行タスクは起動時は同一の優先度で、通常は待ち状態がなく、実行状態または、実行可能状態となっています。

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

2008/12/05

TOPPERSプロジェクト認定

75



sample1のコマンド中、プログラムの管理用のコマンドを以下に列記します。

- 1 以後のコマンドの対象を並列実行タスク1とする
- 2 以後のコマンドの対象を並列実行タスク2とする
- 3 以後のコマンドの対象を並列実行タスク3とする
- v 発行したコマンドを表示する(デフォルト)
- q 発行したコマンドを表示しない
- r 三つの優先度(9, 10, 11)をrot_rdqサービスコールを用いて回転させる
- c 周期ハンドラを動作させる。周期ハンドラでは三つ優先度のレディキューの回転を行う
- C 周期ハンドラを停止させる。
- z CPU例外を発生させる
- Z CPUロック状態でCPU例外を発生させる
- V vxget_timで性能測定用システム時刻を2回読む
- v 発行したサービスコールを表示する。(デフォルト)
- q 発行したサービスコールを表示しない
- Q プログラム(カーネル)を終了する
- Ctrl-C プログラム(カーネル)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

2008/12/05

TOPPERSプロジェクト認定

76

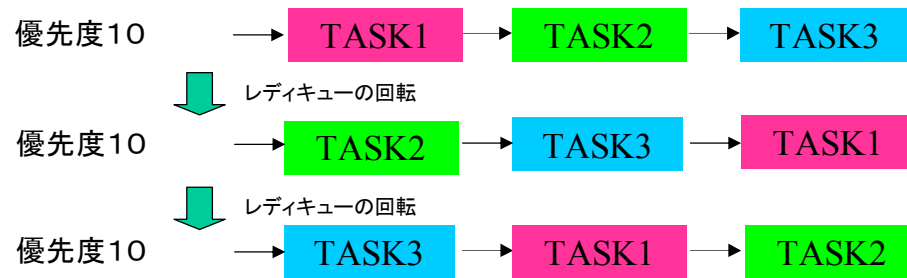


並列実行タスクへのコマンドを列記する。

a	並列実行タスクをact_tskにより起床する
A	並列実行タスクに対する起床要求をcan_actをによりキャンセルする
e	並列実行タスクにext_tskを呼び出させ終了させる
t	並列実行タスクをter_tskにより強制終了する
>	並列実行タスクの優先度を9にする
=	並列実行タスクの優先度を10にする
<	並列実行タスクの優先度を11にする
G	並列実行タスクの優先度をget_priで読み出す
s	並列実行タスクにslp_tskを呼び出させ、起床待ちにさせる
S	並列実行タスクにtslp_tsk(10秒)を呼び出させ、起床待ちにさせる
w	並列実行タスクをwup_tskにより起床する
W	並列実行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	並列実行タスクをrel_tskにより強制待ち解除にする
u	並列実行タスクをsus_tskにより強制待ち状態にする
m	並列実行タスクの強制待ち状態をrsm_tskにより解除する
M	並列実行タスクの強制待ち状態をfrsm_tskにより強制解除する
d	並列実行タスクにdly_tsk(10秒)を呼び出させ、時間待ちをさせる
x	並列実行タスクにパターン0x0001の例外処理を要求する
X	並列実行タスクにパターン0x0002の例外処理を要求する
y	並列実行タスクにdis_texを呼び出させ、タスク例外を禁止する。
Y	並列実行タスクにena_texを呼び出させ、タスク例外を許可する。

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



2008/12/05

TOPPERSプロジェクト認定

77

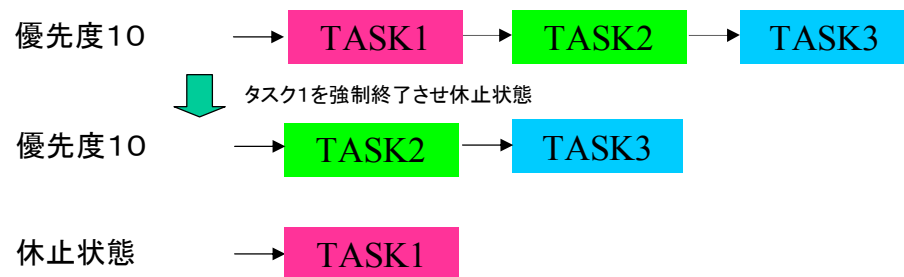


sample1を用いた演習を行います。sample1の起動時の並列実行タスクは優先度10で並列実行タスク1から順番に起動します。並列実行タスクには待ち状態がないため、起床後レディキューの先頭のTASK1が常に実行状態となります。rコマンドを用いて、レディキューを回転させ、ほかの並列実行タスクが実行状態になることを確認しましょう。

プログラムを再起動するには、KD30で実行をストップし、RESET後、再実行で何度でも、再実行できます。

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



2008/12/05

TOPPERSプロジェクト認定

78

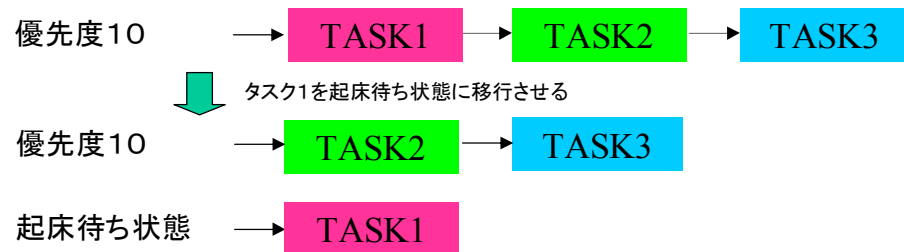


次の演習です。tコマンドを使ってTASK1を終了させてください。その後、rコマンドを用いてレディキューを回転させ、終了状態のTASK1が動作しないことを確認してください。

aコマンドを使ってTASK1を再起動させ、rコマンドにて、TASK1が実行してい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



2008/12/05

TOPPERSプロジェクト認定

79



次の演習です。sコマンドを用いてTASK1を起床待ち状態に移行させ、rコマンドを用いてレディキューを回転させ、起床待ち状態のTASK1が実行しないことを確認してください。次に、wコマンドを用いて、TASK1を起床させ、rコマンドにて、TASK1が実行していることを確認してください。最後に、dコマンドを用いてTASK1を10秒間の実行待ち状態に移行させ、rコマンドにて、TASK1が10秒間は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



2008/12/05

TOPPERSプロジェクト認定

80



次の演習です。>コマンド用いて、TASK1の優先度を高くして、rコマンドにて、TASK1しか実行しないことを確認してください。今度は、<コマンドを用いて、TASK1の優先度を低くして、rコマンドにて、TASK2、TASK3しか実行し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. 起動スイッチの設定

2-2. タスク実行時間の測定

2-3. LED用拡張コマンド

タスクモニタの重要性1

- 中規模システムでは、複数のサブシステム分離開発する。サブシステムの結合を行うと種々の原因でいろいろな問題が発生する。ひとつのサブシステムの不具合により、別のサブシステムが誤動作する可能性もある
- 評価専門の部署で商品評価を行う。デバッグ環境の整っていない環境で、不具合の一時調査を行う必要がある
- いろいろな開発、評価の部署で問題が発生した場合、商品に近い開発形態で問題解決する手段が必要となる



商品形態でタスクモニタを実装することにより、問題調査を行うことができる

2008/12/05

TOPPERSプロジェクト認定

82



中規模組込みシステムでは、開発規模が大きくなるため少人数ですべてを開発できなくなります。そこで全体のプログラムをサブシステムに分割して分業開発を行う必要が発生します。少人数では個々インターフェイス部を同一の開発が設計を行い、最適な整合性を構築可能でしたが、多人数の開発では、個々のサブシステム間のインターフェイスは、仕様書を取り交わして行き違いがないように調整を行う必要があります。また、サブシステムの開発を行う過程で、インターフェイスを最適化する必要が発生したりします。中規模の組込みシステムではアプリケーションの不具合にてリアルタイムOS内でCPU例外が発生したり、原因と発生問題の差異があり、サブシステム間をまたがって、問題解決を行う必要性があります。

また、開発規模が大きいと、品質評価専門の部署でテストを行い、そこで問題が発生するケースが多々あります。この場合、ICEやソフトデバッガ等、整ったデバッグ環境で解析をおこなうこと自体が困難になるケースが多々あります。

このような場合、プラットフォーム部に評価検証ができる機能を、実装しておくことが問題解決や品質向上の手助けになります。このような機能としてタスクモニタが良く使われます。

タスクモニタの重要性2

- 中期規模システムでは、結合環境でサブシステムや専用ミドルウェアの状態表示やモード設定のためのインターフェイスがあったほうが、問題解決を行いやすい
- 開発メーカ供給のデバッガやICEでは、ユーザーシステムに特化した機能を組み込めない
 - このような開発環境は基本的に改造ができない



ソース公開されているタスクモニタに、ユーザーシステム専用の機能を組み込むみ改造することにより、開発効率を上げることができる

中規模組み込みシステムでは、機器の機能に依存したハードウェアやミドルウェアを使用するケースが多々あります。このようなハードウェアやミドルウェアの状態を監視したり、テストを行ったり、アプリケーションの依存せず設定を変えたりする機能を用意すると、非常に便利がケースがたつたあります。開発メーカが供給したデバッガやICEでは、特殊に用途の拡張は難しく、タスクモニタ等を改良してユーザーシステム専用の機能を追加することにより、開発効率や品質の向上を図ることが可能となります。

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をサンプルプログラムに取り込みを行う
- monitorディレクトリ以下のファイルをjsp-1.4.3ディレクトリの下にコピーする
- タスクモニタを取り込みには以下の変更が必要です
 - タスク化するため、コンフィギュレーションファイルの変更を行います
 - インクルードパスの追加と部品となるファイルの追加を行います
 - KD30に対応する修正を行います

2008/12/05

TOPPERSプロジェクト認定

84



今度の演習ではTOPPERSプロジェクト教育WGが教育用に開発したタスクモニタをsample1プログラムに実装してみましょう。タスクモニタを構成する部品は配布プログラムのjsp-1.4.3/monitorディレクトリの下にあります。タスクモニタは、タスク機能を用いて1タスクとして動作するソフトウェア部品です。そのため、基本的に現状のプログラムの修正は必要ありません。monitorディレクトリを解凍したjsp-1.4.3ディレクトリの下にコピーしてください。タスクモニタを取り込むために以下の作業が必要となります。

- (1)タスクとして追加するために、コンフィギュレーションファイルに設定を追加する
- (2)部品となるプログラムをコンパイル、リンクするための設定をMakefileに追加する
- (3)KD30上でTOPPERS/JSPを動作させるためにプログラムの変更を行う

この問題は、全てのタスクが非実行状態になった場合に発生する。前回の演習では、sample1のプログラムの内容上、全てのタスクが非実行状態なることはないので、問題が発生せず、修正の必要はなかった。

タスクモニタの取り込み2 (CFGファイルの修正)

- sample1.cfgにmonitor.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を削除

```

INCLUDE("¥sample1.h¥");
CRE_TSK(TASK1, { TA_HLNG, (VP_INT) 1, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_HLNG, (VP_INT) 2, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK3, { TA_HLNG, (VP_INT) 3, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(MAIN_TASK, { TA_HLNG, 0, main_task, MAIN_PRIORITY,
                     STACK_SIZE, NULL });
DEF_TEX(TASK1, { TA_HLNG, tex_routine });
DEF_TEX(TASK2, { TA_HLNG, tex_routine });
DEF_TEX(TASK3, { TA_HLNG, tex_routine });
CRE_CYC(CYCHDR1, { TA_HLNG, 0, cyclic_handler, 2000, 0 });
#ifdef CPUEXC1
DEF_EXC(CPUEXC1, { TA_HLNG, cpuexc_handler });
#endif /* CPUEXC1 */

#include "../monitor/monitor.cfg"
#include "../systask/timer.cfg"
#include "../systask/serial.cfg"
#include "../systask/logtask.cfg"

```

修正

追加

2008/12/05

TOPPERSプロジェクト認定

85



タスクモニタの取り込み (開発環境に依存せず)

まず、タスクモニタ用のカーネルオブジェクトを、sample1のプログラム内に追加を行う必要があります。タスクモニタで使用するカーネルオブジェクトはjsp-1.4.3/monitor/monitor.cfg中に記載されています。

Sample1のコンフィギュレーションファイルであるsample1.cfgにmonitor.cfgをインクルードすれば、設定が追加されます。monitor.cfgの記述は以下の通りでモニタ用のタスクを追加しているだけです。

```

/*
 * @(#) $Id: monitor.cfg,v 1.1 2007/05/12 14:25:25 roi Exp $
 */
/*
 * モニタタスクのコンフィギュレーション
 */
INCLUDE("¥monitor.h¥");
CRE_TSK(MONTASK, { TA_HLNG|TA_ACT, (VP_INT) MONITOR_PORTID, monitor,
                  MONITOR_PRIORITY, MONITOR_STACK_SIZE, NULL });

```

sample1のメインタスクが自動実行しないようにCRE_TSK(MAIN_TASKのタスク属性からTA_ACTを削除します。

タスクモニタの取り込み3(モニタ部品の追加)

- LINUXやCygwinの環境では、Makefileにmonitorディレクトリ中のMakefile.configをインクルードすることによりパスや部品の設定ができます
- M16Cでは以下の2つの作業を行います
 - TMを終了し、J14sample1_m3029.tmkを改造
 - プロジェクトエディタを使って、タスクモニタのソースファイルを直接追加する

```
#
# ミドルウェアの Makefile のインクルード
#
include $(SRCDIR)/monitor/Makefile.config
```

2008/12/05

TOPPERSプロジェクト認定

86



タスクモニタの追加 (モニタ用プログラムの追加)

追加となるプログラムの設定はj_{sp}-1.4.3/monitor/Makefile.configに記載されています。LINUXやCygwin環境では、sample1のMakefileに、Makefile.configをインクルードさせればシステム依存部以外は設定が追加されます。システム依存部は別途設定を追加する必要があります。Makefile.configの内容は以下の通りです。

```
#
# コンパイルオプション
#
INCLUDES      := $(INCLUDES) -I$(SRCDIR)/monitor -I$(SRCDIR)/kernel
#
# モニターに関する定義
#
STASK_DIR := $(STASK_DIR):$(SRCDIR)/monitor:$(SRCDIR)/monitor/arch:$(SRCDIR)/monitor/stdio
STASK_COBJS := $(STASK_COBJS) task_expansion.o monitor.o stddev.o ¥
               printf.o sprintf.o scanf.o sscanf.o
ifeq ($(NETDEVICE),true)
ifeq ($(SUPPORT_TCP),true)
STASK_COBJS := $(STASK_COBJS) mon_tinet.o netdevice.o netdevnet.o
STASK_CFLAGS := $(STASK_CFLAGS) -DSUPPORT_NETDEV
endif
endif
```

システム依存部のソースはj_{sp}-1.4.3/monitor/archの下にソースファイルがあります。

M16Cの開発環境ではtmkファイルがMakefileの変わりにないので、このファイルを変更する必要があります。

タスクモニタの取り込み4(インクルードパスの設定)

- タスクモニタ用のインクルードパスを追加します
- TMを終了させてください
- エディタでJsp14sample1_m3029.tmk中のINCLUDES変数に-I\$(SRCDIR)/monitorを追加します

```
DELETE =      @-del
LNLIST =      $(PROJECT).cmd
PRJDIR =      D:¥usr¥TOPPERS¥jsp-1.4.3¥tools¥M16C-RENEAS
INCLUDES      = -I$(SRCDIR)/config/m16c-renesas -I$(SRCDIR)/config/
               m16c-renesas/$(TARGET_SYS) -I$(SRCDIR)/kernel -I$(SRCDIR)/
               include -I$(SRCDIR)/monitor -I../..sample -I.
LMC          =      LMC30
CC            =      NC30
TARGET_SYS    =      m3029
```

2008/12/05

TOPPERSプロジェクト認定

87



M16Cの開発環境では、Jsp14sample1_m3029.tmkを修正してタスクモニタ用の部品を追加する必要があります。TMは常にJsp14sample1_m3029.tmkファイルを管理し、設定に変更があると、このファイルの更新を行います。そのため、TMを終了させ自由にJsp14sample1_m3029.tmkの書換えが行えるようにする必要があります。タスクモニタ用のインクルード設定をJsp14sample1_m3029.tmkに追加してください。以下のようにINCLUDE環境変数にタスクモニタのディレクトリ設定、-I\$(SRCDIR)/monitorを追加します。ソースファイルの追加はTMを使って行いますので、エディタでの修正はこれだけです。

タスクモニタの取り込み5(ソースの追加)

- TMを起動し、プロジェクトエディタから以下のファイルを追加します
- 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	M16C依存部の追加
stdio/stddev.c	monitor.c	arch/m16c.c
stdio/printf.c	task_expansion.c	
stdio/sprintf.c		
stdio/scanf.c		
stdio/sscanf.c		

2008/12/05

TOPPERSプロジェクト認定

88



TMを起動してタスクモニタ用のソースファイルを追加します。モニタのソースは以下の3つに分かれています。この中のM16C依存部は**Makefile.config**に含まれていないソースであり、LinuxやCygwin環境でも、**Makefile**に直接記述して追加する必要のあるソースです。

- (1) モニタ基本部
- (2) 標準入出力部
- (3) M16C依存部

標準入出力の機能はタスクモニタの機能として取り込んでいます。NEWLIBで供給される標準入出力機能はLinux用であり、また、バージョンによってインターフェイス仕様がことなったりしますので、安定して標準入出力の機能を使用するためにタスクモニタの機能として取り込みを行いました。Linux上のプログラムのシミュレーション開発を行う場合、**syslog**も使用かと思いますが、一般的なプログラムは標準入出力を用いてログ出力させ、プログラム開発するケースが多く、これらのプログラムをできるだけ変更なく取り込むために標準入出力機能は必要となります。

M16C依存部は、メモリの配置、メモリのアクセス手順、レジスタの表示、エクセプションの隠蔽機能を持ちます。これらの機能はCPUやボードの設計によって実装がことなりますので依存部としています。

これらのソースファイルをTMを用いて、追加してください。

タスクモニタの取り込み6(ソースの修正)

- JSPはどのタスクも実行していない場合、WAIT命令を実行し省エネ状態に移行します
- KD30はWAIT命令の実行で処理停止しますので、KD30でデバッグする場合、アイドル状態でもWAIT命令を実行しないように修正を行う必要があります
- config/m16c-renesas/cpu_support.a30中にWAIT命令がありますので、これをコメントアウトします

157行目をコメントアウト

```

schdedtskが NULL の場合は、これより下には行かない
dispatcher_1:
    ldc      #RAMEND, isp          ; 割り込み用のスタックへ切替え
    inc.b    __kernel_intnest     ; 非タスクコンテキスト
dispatcher_2:
    fset     i                    ; 割り込み待ち
    wait     ; wait命令を使用すると
    nop      ; 消費電力を抑えることができる。
    nop      ; コメントをつけたまま(nopのみ)にすると
    nop      ; アイドルループと同じような動作になる
    ;;

```

2008/12/05

TOPPERSプロジェクト認定

89



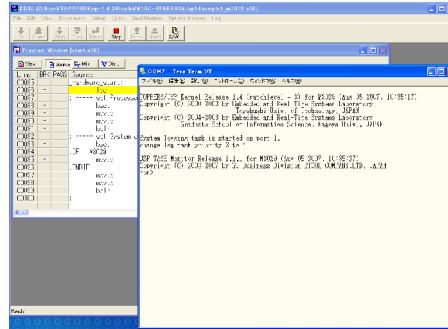
KD30上実行させるための修正

TOPPERS/JSPは、どのタスクも実行していない状態(アイドル状態)ではCPUの省電力化を行うためにWAIT命令を発行します。KD30はWAIT命令が発行されると実行ストップと判断して停止状態になります。従って、アイドル状態でもデバッグを停止させないようにWAIT命令を実行しないようにプログラムを修正します。もちろん、フラッシュROMに書き込んで実行させる場合、省電力を行いたいなら、WAIT命令を発行する必要があります。

アイドル中のWAIT命令の発行はjsp-1.4.3/config/m16c-renesas/cpu_support.a30中の157行目に記述されていますので、この命令をコメントアウトしてください。

タスクモニタのビルドと実行

- TMから「リビルド」を行います
- ビルド終了後、KD30を起動し、「GO」コマンドにてTeraTermにバナー表示と、タスクモニタのプロンプトが出ることを確認してください



TMから[リビルド]を行い、KD30を用いてダウンロード実行を行い、TeraTermにモニタプロンプトが表示されることを確認してください。

タスクモニタの使用方法

- 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

```

COM7 - Tera Term V1
mon/help
Display BYTE [start address] [end address]
          HALF [start address] [end address]
          WORD [start address] [end address]
          TASK
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)
          PRIORITY [pri] [chg_pri]
Log        WRITE [logmask] [lowmask]
          TASK [cycle time(ms)]
          PORT [no] [lsmol] [portaddress]
Help
mon>
  
```

2008/12/05

TOPPERSプロジェクト認定

91



タスクモニタで使用可能なコマンドは`help(return)`で表示されます。教育WGのタスクモニタは2コマンド設定となっています。各コマンドは最初の一文字のみ入力で後のストリングは省略が可能です。例えば、ゼロ番地をバイト単位でダンプするコマンド: `display byte 0`は`d b 0`に省略できます。

標準的なタスクモニタのコマンドは以下の通りです。

(1) 表示コマンド

`Display BYTE {start address} {end address}` start addressからend addressまでをバイト単位で表示
`Display HALF {start address} {end address}` start addressからend addressまでを2バイト単位で表示
`Display WORD {start address} {end address}` start addressからend addressまでを4バイト単位で表示
`Display TASK` 各タスクの状態を表示する
`Display REGISTER` レジスタを保持してタスクが待ち状態の場合レジスタを表示

(2) 設定コマンド

`Set BYTE {start address}` スタートアドレスからバイトデータをセット、停止は(.)
`Set HALF {start address}` スタートアドレスから2バイトデータをセット、停止は(.)
`Set WORD {start address}` スタートアドレスから4バイトデータをセット、停止は(.)
`Set COMMAND {mode}` modeは1か2、1コマンドか2コマンドかの設定

(デフォルトは2コマンド)

`Set Serial {port id}` 標準出力先のポート番号の設定
`Set Task {task id}` コマンド設定対象のタスクの指定

(3) タスクコマンド

`Task ACTIVATE` 対象のタスクにack_tskを発行
`Task TERMINATE` 対象のタスクにter_tskを発行
`Task SUSPEND` 対象のタスクにsus_tskを発行
`Task RESUME` 対象のタスクにrsm_tskを発行
`Task PRIORITY {pri}` 対象のタスクに優先度priでchg_priを発行

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_id.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]
```

2008/12/05

TOPPERSプロジェクト認定

92



(4) LOGコマンド

Log MODE {logmask} {lowmask} syslogの表示モードを変更する
 Log TASK {cycle time[ms]} タスクの実行時間を表示する(注1)
 Log PORT {no} {logno} {port address} ポートへのアクセス履歴を表示する(注1)

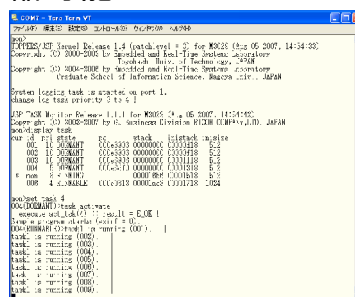
(5) HELPコマンド

Help

注1) 実装されないケースがあります。

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
- ・ 入力に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 4 : ID番号4(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



2008/12/05

TOPPERSプロジェクト認定

93



自動実行をやめた**sample1**プログラムを実行させます。**sample1**タスクを実行させるにはメインタスクを起動すれば、他のタスクは自動的に起動します。メインタスクのタスクIDは4なので、タスク4を**act_tsk**により起動させます。

```
mon> display task
```

タスクの状態を表示する

```
mon> set task 4
```

コマンド対象のタスクを4番にする

```
mon> task activate
```

act tskを発行する

sample1が実行することを確認してください。

演習: サンプルプログラムの停止

- サンプルプログラムは1～4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(4)を終了
- set task とtask terminateで、1～3のタスクを終了させる

リセットは直接実行版では、
リセットボタン押下。

デバッガ版では、KD30の
(Stop)→(Reset)→(Go)で簡単に行える

```

COM7 - Term VI
task1 is running (127). |
4: task1 is running (128). |
4: task1 is running (128). |
000: [omni=LOG WARNING] [omni=LOG EMERG]
004(WAITING)>display task
  id  pri  state  pc          stack  inistack  inisize
* 001  10  RUNNING  030a0387  00010000  00000118  512
  002  10  RUNNING  030a0327  00010104  00000118  512
  003  10  RUNNING  030a0327  00010204  00000118  512
  004  5  WAITING  030a0313  0001040f  00000118  512
  005  5  RUNNING  03010b08  00000118  512
  006  1  RUNNING  030a0313  00010ac3  00000118  1024
004(WAITING)>task terminate
execute pc=task(4) :: result = E_OK !
004(DORMANT)>set task 3
003(DORMANT)>task terminate
execute pc=task(3) :: result = E_OK !
003(DORMANT)>set task 2
002(DORMANT)>task terminate
execute pc=task(2) :: result = E_OK !
002(DORMANT)>set task 1
001(DORMANT)>task terminate
execute pc=task(1) :: result = E_OK !
001(DORMANT)>display task
  id  pri  state  pc          stack  inistack  inisize
* 001  10  DORMANT  030a0305  00010000  00000118  512
  002  10  DORMANT  030a0305  00010104  00000118  512
  003  10  DORMANT  030a0305  00010204  00000118  512
  004  5  DORMANT  030a0313  0001040f  00000118  512
  005  5  RUNNING  03010b08  00000118  512
  006  1  RUNNING  030a0313  00010ac3  00000118  1024
001(DORMANT)>

```

2008/12/05

TOPPERSプロジェクト認定

95



タスクモニタを使用してsample1を未実行の状態に戻してください。これを行うには4つのタスクを終了させる必要があります。タスクの終了はset task {id}にてタスクを指定し、task terminateコマンドを発行すれば終了状態になります。これをタスクID1から4に対して実行すればOKです。

KD30を使っている場合は、(Stop)→(Reset)→(Go)で再実行可能です。

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、LEDを操作します
- set byteコマンドでLEDが接続されているポート0のポートレジスタ(0x3e0)に書き込み動作を確認する

```

COM7 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(C) ウィンドウ(W) ヘルプ(H)
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>set byte 3e0
000003e0 0f =00 00
000003e1 00 =.
001(DORMANT)>set byte 3e0
000003e0 00 =0f 0f
000003e1 00 =.
001(DORMANT)>
001(DORMANT)>
001(DORMANT)>

```

0x0fを書き込み : 全LED 点灯

エコーバック

". "で終了

0x00を書き込み : 全LED 消灯

2008/12/05

TOPPERSプロジェクト認定

96

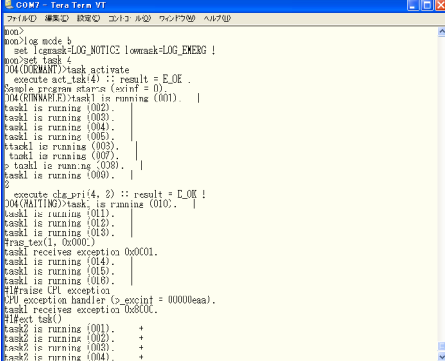


タスクモニタのメモリ設定機能を用いて、LEDの点灯、消灯を行います。LEDのポートアドレスは0x3e0番地です。この番地のビット0から3が各LEDの設定に対応しています。set byteコマンドを使って、0x3e0番地の0x00から0x0fまでの値を設定し、LEDの点灯、消灯の確認を行ってください。

演習: 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 `lp 2 (return)`を発行して優先度2に変更



```

MON>
MON>log mode b
set logmask=LOG_NOTICE logmask=LOG_ERROR !
MON>set task 4
MON(DORMANT)>task activate
execute act.task(4) :: result = E_OK .
sample program starts (count = 0).
MON(RUNNING)>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).
execute clg_pri(4, 2) :: result = E_OK !
MON(RUNNING)>task1 is running (010).
task1 is running (011).
task1 is running (012).
task1 is running (013).
task1 is running (014).
task1 receives exception 0x0001.
task1 is running (015).
task1 is running (016).
task1 receives CPU exception
CPU exception handler (c_excint = 00000000).
task1 receives exception 0x0001.
task1 is running (017).
task1 is running (018).
task1 is running (019).
task1 is running (020).
task1 is running (021).
task1 is running (022).
task1 is running (023).
task1 is running (024).

```

printf文によるデバッグ

- オースドックスなデバッグ手法
- LINUXやWindowsでシミュレーションしたソースをそのまま組み込みシステムに取り込み使用できる
- 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
 - PC上のプログラムの場合は、画面に文字を表示する
 - M16C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

C言語のデータ表示指定が**printf**であるため、**printf**文によりログ表示をさせつつデバッグを行うデバッグ方法はオースドックスなデバッグ方法といえます。特に、LINUXやWindowsで開発したプログラムは**printf**を使った開発が多く、LINUXやWindows上でシミュレーションを行ったプログラムを最小の修正で組み込み環境で動作させることができます。中規模組み込み開発では分業開発が主体であり、アプリケーションはプラットフォーム非依存に開発を行くことが可能です。LINUXやWindowsでのシミュレーションはセルフ開発環境であり、開発が容易です。この環境で十分テストを行い、品質を向上させれば生産性の向上につながります。

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. 起動スイッチの設定](#)
 - 2-2. タスク実行時間の測定
 - 2-3. LED用拡張コマンド

モニタタスクの終了判定

- タスクモニタのメインプログラムのmonitor関数中にコンパイルスイッチ
NEED_MONITORがあり、スイッチが設定されれば、
need_monitor()関数が呼ばれ、戻り値がFALSEならばモニタタスクを終了します

```

/*
 * モニタタスク
 */
#ifdef DEVSERV
void monitor(VP_INT exinf)
#else
void monitor_sub(VP_INT exinf)
#endif
{
    INT no, point;
    B c;

    /* モニタで使用するデータの初期化 */

    mon_portid = mon_default_portid = (ID)exinf;
    mon_datatype = DATA_BYTE;
    mon_logmask = LOG_NOTICE;
    mon_lowmask = LOG_EMERG;
    current_tskid = MONTASK;

#ifdef DEVSERV
    if(mon_portid != CONSOLE_PORTID)
        syscall(serial_opn_por(mon_portid));
#endif
#ifdef NEED_MONITOR
    if(!need_monitor())
        ext_tsk();
#endif
    /* NEED_MONITOR */
    _setup_stdio(&mon_portid);
}

```

タスクモニタのメインプログラムmonitor.c中のメイン関数monitorにはコンパイルスイッチNEED_MONITORの設定により、起動時need_monitor()関数を呼び出し、戻り値がFALSEならば、ext_tskサービスコールを実行する機能があります。NEED_MONITORコンパイルスイッチを設定してsample1をビルドし、need_monitor関数を自作し、DIPSWの4番の参照し、オンならばTRUEを、オフならばFALSEを返すようにプログラムを改造します。これにより、モニタタスクの起動設定ができるようになります。

この改造により、DIPSWの4番をオフにすると、タスクモニタが動作しないため、sample1を正常に動作させることができます。

演習:タスクモニタ実行判定関数の追加

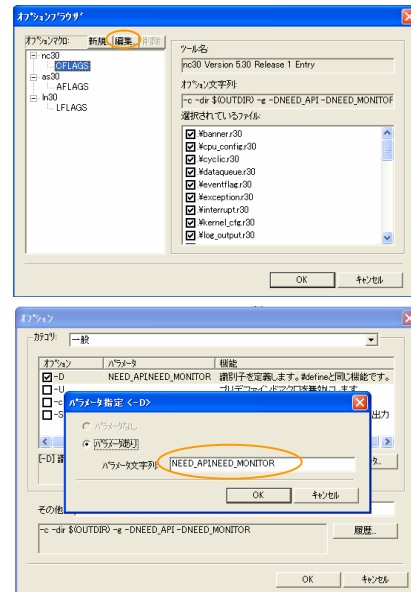
- Jsp14sample1_m3029への改造を行ってください
- コンパイルスイッチNEED_MONITORを設定する
- ソースファイルを追加し、need_monitor関数を追加する
- need_monitor関数中で、ディップスイッチの初期化を行う
- need_monitor関数中で、ディップスイッチの判定を行い、SW4がオンならばTRUEを返す、オフならばFALSEを返し、メインタスクを起動するようにプログラムする
- ディップスイッチの制御は2日目の「switch_dip」を参考にしてください
- SW4オフで起動し、サンプル1プログラムが正常に動作することを確認してください

演習を行います。DIPSWの4番の設定でタスクモニタの起動を設定するようにプログラムを改造してください。この演習は以下の手順で行います。

- 1) TMを用いて、コンパイルスイッチNEED_MONITORを追加する。
- 2) ソースファイルを追加し、ソースファイル中にneed_monitor関数を作成する。

演習:コンパイルスイッチの追加

- TMのプロジェクトエディター→オプションブラウザを開く
- CFLAGを選択し、編集ボタンを押す
- -Dオプションを選択後、パラメータボタンを押すと-Dオプションに対するダイアログが表示される
- パラメータ文字列に「,NEED_MONITOR」を追加しオプションブラウザを閉じる
- リビルドでneed_monitor関数がundefinedになる



2008/12/05

TOPPERSプロジェクト認定

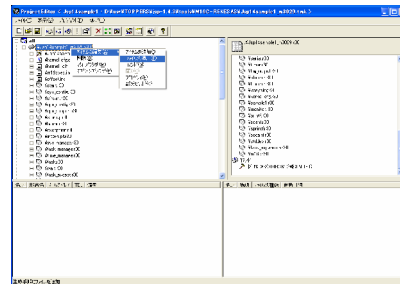
103



まず、コンパイルスイッチNEED_MONITORが有効になるように設定します。TMからJsp14sample1_m3029を開き、プロジェクトエディタを開きます。オプションブラウザからCFLAGを開き[編集]を押して-Dオプションのパラメータに「,NEED_MONITOR」を追加します。この状態で「リビルド」すると、need_monitor関数がundefinedになり、ビルドが完了しません。

演習:ソースファイルの追加

- need_monitor関数を追加するソースファイルcommand.cをtools/M16C-RENESASに追加する
- 作成したプログラムをプロジェクトエディタから追加してリビルドを行い
- KD30を起動して、SW4をオン、オフに切り替えて動作の確認を行います



2008/12/05

TOPPERSプロジェクト認定

104



次に、need_monitor関数を記載するソースファイルをtools/M16C-RENESASディレクトリ上に追加します。名称は先の実習内容も考慮してcommand.cとします。

演習: プログラムサンプル

- need_monitor関数のコーディング例を左に載せます

```

/*
 * タスクモニタ起動判定
 */
BOOL need_monitor(void)
{
    unsigned char reg;

    /* 入力ポートに設定 */
    *BREG_SFR_PD3=*BREG_SFR_PD3 & ~DSW_MASK;

    /* DIPスイッチ状態の読み込み */
    reg = *BREG_SFR_P3;
    reg = reg & DSW_MASK;
    if(reg & DSW4)
        return TRUE;
    else{
        syscall(act_tsk(MAIN_TASK));
        return FALSE;
    }
}

```

2008/12/05

TOPPERSプロジェクト認定

105



command.cに上記の内容でneed_monitor関数を追加します。これに加えてDIPSWの定義を記載しないと、コンパイルは通りません。switch_dip.cの内容を参考にして定義を追加します。command.cの作成後プロジェクトエディタのファイルの追加でcommon.cを追加します。

```

/*
 * プログラマブル入出力ポート3関連レジスタ
 */
#define BREG_SFR_P0    ((volatile unsigned char *)0x03E0) /* ポートP0レジスタ */
#define BREG_SFR_PD0   ((volatile unsigned char *)0x03E2) /* ポートP0方向レジスタ */
#define BREG_SFR_P3    ((volatile unsigned char *)0x03E5) /* ポートP3レジスタ */
#define BREG_SFR_PD3   ((volatile unsigned char *)0x03E7) /* ポートP3方向レジスタ */

/*
 * ポート3のDIPスイッチ接続ビット
 */
#define DSW1           0x01
#define DSW2           0x02
#define DSW3           0x04
#define DSW4           0x08
#define DSW_MASK       (DSW1|DSW2|DSW3|DSW4)

```

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. 起動スイッチの設定
 - [2-2. タスク実行時間の測定](#)
 - 2-3. 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が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**

タスクモニタにタスクの実行時間を測定する機能を追加します。中規模組込みシステムでは多くのタスクが並行実行します。ハードリアルタイム性を必要とするタスクが周期中に、どれくらいの時間CPUを占有するかを測定できれば、その他のタスクがどれくらいシステムを占有するか等の内容を把握できます。これにより、もっと性能を上げたい場合、どのタスクをどのように改造していくべきか、また、ハードウェアのランクアップが必要か等の判定に利用ができます。

このような機能の実装は、タスクスイッチング時間のオーバーヘッドの増加をまねきますので、管理して使用するようにしましょう。

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cpu_support.a30のMONコンパイルスイッチが1のとき有効となります

```

__kernel_exit_and_dispatch:
    fclr    i                ; 割り込み禁止
    mov.b   #0, __kernel_intnest ; ネストカウンタクリア
dispatcher:
    mov.w   __kernel_schedtsk, a0 ;
    mov.w   a0, __kernel_runtsk ; schedtsk を runtsk に
    .IF MON == 1
; タスクモニター機能用分岐
; 不要の場合は (MON = 0)
    jsr.a   _iana_tsk        ; タスク実行情報の設定
    mov.w   __kernel_runtsk, a0 ; a0を復帰
    .ENDIF
    jz      dispatcher_1     ; schedtsk がなければ割り込み待ち
    ldc     TCB_sp[a0], isp   ; タスクスタックポインタを復帰
    jmp.i.a TCB_pc[a0]       ; 実行再開番地へジャンプ

```

2008/12/05

TOPPERSプロジェクト認定

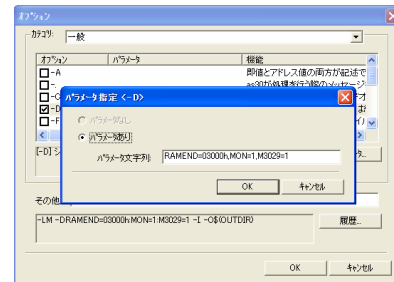
108



タスク時間測定機能は、M16Cのタスクスケジューラからiana_tsk関数を呼び出すことにより測定を行います。タスクスケジューラはconfig/m16c-renesasディレクトリ中のcpu_support.a30中にアセンブラで記述されています。この処理はアセンブラのコンパイルスイッチMONが1のとき有効となります。

演習:タスク実行時間ログ機能の追加

- Jsp14sample1_m3029にタスク実行時間測定機能を追加する
- プロジェクトブラウザ→オプションブラウザを開く
- AFLAGSを選択し、編集ボタンを押す
- -Dオプションを選択して、パラメータボタンを押す
- パラメータ「MON=0」を「MON=1」に変更する
- オプションブラウザを閉じ、リビルドを行う
- KD30を起動して、実行確認を行う



2008/12/05

TOPPERSプロジェクト認定

110



演習を行います。タスク時間を表示するLOG TASKコマンドを有効にしてください。まず、TMからJsp14sample1_m3029を開き、プロジェクトブラウザからオプションブラウザを開き、AFLAGSを選択し、[編集]ボタンを押します。-Dオプションを選択し[パラメータ]ボタンを押す。パラメータの「MON=0」を「MON=1」に変更し、オプションブラウザを閉じて、リビルドを行います。KD30を起動して、LOG TASKコマンドを試してください。

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. 起動スイッチの設定
 - 2-2. タスク実行時間の測定
 - [2-3. LED用拡張コマンド](#)

演習:タスクモニタの拡張コマンドでLEDを制御する

- タスクモニタはPIPE以下に新規のコマンドを追加できる
- PIPEコマンドでLEDコマンドを追加して、LEDの点灯、消灯を行う
- PIPEコマンドのサブコマンドとして、LEDコマンドを設ける
- 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

PIPE LED (LED番号1-4) (消灯-0、点灯-1)

演習を行います。タスクモニタは1コマンド目PIPEで2コマンド目を拡張できます。この機能を用いてLEDを点灯、消灯を行うコマンドを追加します。このコマンドをLEDコマンドします。

PIPE LED [1-4] [0-1] (return)

例えば、PIPE LED 1 1で、LED1を点灯

PIPE LED 4 0でLED4を消灯とします。

演習: PIPEコマンドの追加

- モニタタスクのコンパイルスイッチSUPPORT_PIPEを定義すると、PIPEコマンドにて、pipe_command関数を呼び出すように設定できる

以下、monitor.cのソースの記述

```
/*
 * メインコマンドテーブル
 */
static struct COMMAND_TABLE const mon_dispatch[] = {
    {"DISPLAY", display_command }, /* 表示 */
    {"SET", set_command }, /* 設定 */
    {"TASK", task_command }, /* itron TASK */
    {"LOG", log_command }, /* ロギング */
#ifdef SUPPORT_NETDEV
    {"NET", net_command }, /* ネットワーク */
#endif
#ifdef SUPPORT_PIPE
    {"PIPE", pipe_command }, /* 拡張コマンド */
#endif
    {"HELP", help_command } /* ヘルプ */
};
```

2008/12/05

TOPPERSプロジェクト認定

113

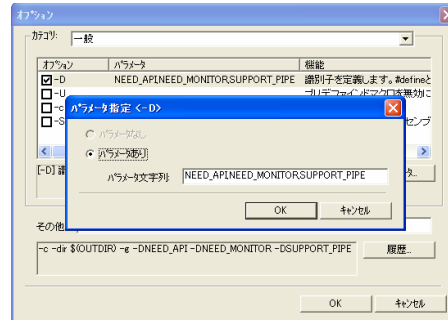


タスクモニタにPIPEコマンドを追加するには、タスクモニタのビルド時にSUPPORT_PIPEコンパイルスイッチを設定します。monitor.cファイルにはデスパッチテーブルがあり、このスイッチによりpipe_command関数を呼び出すようになります。pipe_command関数の引数にはPIPE以下のコマンド列が渡されます。このコマンド列を解析してLEDの制御を行います。

コマンドとして、「PIPE LED 1 0」入力されると、「LED 1 0」が引数として渡されます。

演習:SUPPORT_PIPEコンパイルスイッチ

- プロジェクトブラウザ→オプションブラウザを選択する
- CFLAGSの選び、編集ボタンを押す
- -Dオプションを選択し、パラメータボタンを押し、パラメータに「,SUPPORT_PIPE」を追加する
- オプションブラウザを閉じ、リビルドするとpipe_command関数がundefinedとなる



2008/12/05

TOPPERSプロジェクト認定

114



まず、コンパイルスイッチSUPPORT_PIPEが有効になるように設定します。TMからJsp14sample1_m3029を開き、プロジェクトエディタを開きます。オプションブラウザからCFLAGを開き[編集]を押し-Dオプションのパラメータに「,SUPPORT_PIPE」を追加します。この状態で「リビルド」すると、pipe_command関数がundefinedになり、ビルドが完了しません。

演習: pipe_commandの追加

- 起動スイッチの実習で追加したcommand.cファイルを拡張して、LEDコマンドを実装する
- インクルードファイルとしてstdio.hとmonitor.hを追加する
- need_monitor関数にLEDの初期化を追加する
- pipe_command関数を追加し、以下の機能を実装する
 - LEDコマンドの実装
 - LED以外のコマンドでは、ヘルプメニューを表示する
- LEDコマンドの判定は、タスクモニタ内の関数を流用する
- LEDコマンドからのパラメータの取り出しはsscanf関数を用いる

2008/12/05

TOPPERSプロジェクト認定

115



起動スイッチの実習で追加したcommand.cファイルにpipe_command関数を追加します。以下の順序で行ってください。

- 1) タスクモニタは標準入出力を用いて入力、表示を行いますので、インクルードファイルとしてstdio.hとmonitor.hを追加してください。
- 2) 電源投入時LEDの設定の初期化を行う必要がありますので、need_command関数にLEDの初期化を追加してください。
- 3) pipe_command関数を追加します。このとき、LEDコマンドの判定はタスクモニタ内の関数を流用してください。また、HELPまたはLED以外の入力があった場合、PIPEコマンドのHELP表示を行ってください。LED以下の数値の入力はsscanf関数を使ってください。

演習: command.cの作成(インクルードファイル)

- LEDコマンドのパラメータ取り込み用にsscanf関数を使用するため、プロトタイプ宣言のあるstdio.hをインクルードする
- LEDコマンドの解析用にタスクモニタで使用している構造体や関数を使用するため、monitor.hをインクルードする

```
/*  
 * モニタコマンド用ソースファイル  
 */  
#include <t_services.h>  
#include <stdio.h>  
#include "kernel_id.h"  
#include "monitor.h"
```

command.cへのインクルードファイルの追加は上記の通りです。

演習: command.cの作成(LEDの初期化)

- need_monitor関数に、LED用ポートの出力設定と、消灯設定を追加します
- もちろん、LEDの制御に必要な定義はswitch_dip.c等のソースファイルから取り込みを行う必要があります

```
/*
 * タスクモニタ起動判定
 */
BOOL need_monitor(void)
{
    UB reg;

    /* 方向レジスタ出力設定 */
    *BREG_SFR_PD0 = *BREG_SFR_PD0 | LED_MASK;

    /* 初期状態は消灯とする */
    *BREG_SFR_P0 = (*BREG_SFR_P0 & ~LED_MASK);

    /* 入力ポートに設定 */
    *BREG_SFR_PD3=*BREG_SFR_PD3 & ~DSW_MASK;

    /* DIPスイッチ状態の読み込み */
    reg = *BREG_SFR_P3;
    reg = reg & DSW_MASK;
    if(reg & DSW4)
        return TRUE;
    else{
        syscall(act_tsk(MAIN_TASK));
        return FALSE;
    }
}
```

2008/12/05

TOPPERSプロジェクト認定

117



need_monitor関数中のLEDの初期化関数は以下の二行です。LEDの方向レジスタを出力に設定します。

```
*BREG_SFR_PD0 = *BREG_SFR_PD0 | LED_MASK;
```

初期設定として消灯にします。

```
*BREG_SFR_P0 = (*BREG_SFR_P0 & ~LED_MASK);
```

これらの変数の設定をled_shift.cを参考にして追加してください。

```
/*
```

```
 * プログラマブル入出力ポート3関連レジスタ
```

```
*/
```

```
#define BREG_SFR_P0    ((volatile unsigned char *)0x03E0) /* ポートP0レジスタ */
```

```
#define BREG_SFR_PD0   ((volatile unsigned char *)0x03E2) /* ポートP0方向レジスタ */
```

```
/*
```

```
 * LED接続ビット
```

```
*/
```

```
#define LED1          0x08
```

```
#define LED2          0x04
```

```
#define LED3          0x02
```

```
#define LED4          0x01
```

```
#define LED_MASK      (LED1 | LED2 | LED3 | LED4)
```

演習: コマンド定義テーブルの取り込み

- タスクモニタ内のSUBCOMMAND_TABLEに定義したコマンドをコマンド番号に変換する機能を用いてLEDコマンドを簡略化します
- compare_word関数は、ここで定義したmon_pipeとコマンド文字列からコマンド番号に変換を行います

```
/*
 * パイプコマンド番号
 */
#define PIPE_LED      0
#define PIPE_HELP     1
/*
 * パイプコマンドテーブル
 */
static struct SUBCOMMAND_TABLE const mon_pipe[] = {
    {"LED",          PIPE_LED }, /* ロギングモード設定 */
    {"HELP",         PIPE_HELP }
};
```

2008/12/05

TOPPERSプロジェクト認定

118



コマンドデスパッチを簡単に行うために、タスクモニタ内の関数を流用します。タスクモニタではサブコマンドの解析にSUBCOMMAND_TABLE構造体を使っています。使用するサブコマンドの文字列をsubcommandに、取り出されるコマンド番号をtypeに設定します。SUBCOMMAND_TABLE構造体は以下の通りです。

```
struct SUBCOMMAND_TABLE {
    const char *subcommand; /* サブコマンド文 */
    B const type;          /* 実行タイプ */
};
```

演習: pipe_command関数の作成

- pipe_command関数は、引数としてPIPE以下の文字列が渡されます
- skip_command文字列“LED”を見つけるとTRUEを返します、skip_wordは“LED”を読み飛ばした文字番号をpointにセットします

```

UW pipe_command(B *command)
{
    INT  point=0;
    B    cmd=-1;
    INT  no, count;
    INT  arg1, arg2;
    UB   reg;
    count = sizeof(mon_pipe) / sizeof(struct SUBCOMMAND_TABLE);
    if(command[point]){
        for(no = 0 ; no < count ; no++){
            if(compare_word(mon_pipe[no].subcommand, command, 0)){
                skip_word(command, &point);
                cmd = mon_pipe[no].type;
                break;
            }
        }
    }
}

```

2008/12/05

TOPPERSプロジェクト認定

119



コマンドの解析で使用する2つの関数は以下のプロタイプ宣言です。compare_wordはmodeの設定がゼロの場合、sの文字列とdの文字列をsのサイズ分比較し等しければTRUE、等しければFALSEを返します。modeが1の場合、最初の1文字だけ比較します。skip_wordはスペース、タブ等のスキップを行います。変数cmdの中にコマンド番号がセットされます。

```
extern BOOL compare_word(const char *s, B *d, INT mode);
```

```
/*
```

```
 * 文字列をスキップする
```

```
*/
```

```
Inline void
```

```
skip_word(B *command, INT *point)
```

```
{
```

```
    while(command[*point] != ' ' && command[*point] != '\t'
```

```
        && command[*point] != ',' && command[*point] != 0)
```

```
        (*point)++;
```

```
    if(command[*point] != 0)
```

```
        (*point)++;
```

```
    skip_space(command, point);
```

```
}
```

演習: パラメータの解析とコマンド実行

- cmdにセットされたコマンド番号から処理を判別してください
- パラメータ1はarg1、パラメータ2はarg2に設定されますので、LEDの制御は自作してください
- あとは、KD30を使ってデバッグしてください

```
switch(cmd){
case PIPE_LED:      /* LEDの制御 */
    sscanf((char *)&command[point], "%d %d", &arg1, &arg2);
    if(arg1 >= 1 && arg1 <= 4){
        reg = *BREG_SFR_P0;
        :
        :
    }
    break;
default:
    printf(" LED (no) (on-1,off-0) led control\n");
    break;
}
return 0;
}
```

2008/12/05

TOPPERSプロジェクト認定

120



cmdにセットされたコマンド番号をswitch文で判定します。case PIPE_LED中にLEDコマンド記載します。sscanf関数でarg1にLED番号、arg2にオン、オフ設定番号がセットされます。arg1とarg2の変数を使用したプログラムは自作してください。defaultにはHELP処理を記載します。

TOPPERS/JSP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/JSP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

2008/12/05

TOPPERSプロジェクト認定

121



実習のまとめ

•コンパイラやリンカーなどの環境設定とターゲットボードがあれば、比較的簡単にTOPPERS/JSPの動作確認ができます。

•TOPPERS/JSPはカーネルのソースが供給されているので、リアルタイムOSのしくみを理解しやすく、システムに組込み時、カーネルに絡む問題が発生しても、カーネルを含めたデバッグができ、問題解決がしやすくなります。

•リアルタイムOS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で、多くのプログラム開発やミドルウェアやデバイスドライバの導入を行うことが多く、タスクモニタなど開発ツールを使用して、システムに密着したデバッグやログ機能を追加していくことにより、開発効率や品質の向上を図ることができます。

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

2008/12/05

TOPPERSプロジェクト認定

123



リアルタイムOSを用いたシステム設計のまとめ

- リアルタイムOSは、プログラム規模が比較的大きい中規模組込みシステムで使用すると効果的です。これはマルチタスク機構により、分割したプログラム開発が可能となるからです。
- リアルタイムOSはCPU処理を代行する機能が主です。システム設計を行う場合、組込み機器に依存した基本機能と手順(アーキテクチャ)をまとめてプラットフォームを作成し、その上位にアプリケーションを構築する形でシステム設計すると、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、機器の機能に対応したメンテナンス機能やログ機能を用意した方が、効率的な開発が行えます。
- システム設計やプラットフォーム開発には、熟練したシステム開発者が必要となります。

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はμITRON4.0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、オープンソースとして供給されています

2008/12/05

TOPPERSプロジェクト認定

124



μITRON仕様に関するまとめ

- 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で、デバイスドライバや汎用ミドルウェアが充実しています。
- リアルタイムOSはCPU処理を代行する機能が主です。機器に特化した形でプラットフォームをくみ上げて開発した方が有用な開発が可能です。
- 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。メッセージ通信は多用しすぎるとシステムが複雑化します。
- TOPPERS/JSPはμITRON仕様4.0のスタンダードプロファイルに準じたリアルタイムOSです。仕様策定時、検証用に使用しました。また、オープンソースとして供給されています。

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((株)リコー)
- 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/JSPの導入
 - 竹内 良輔((株)リコー)
- システム検証モジュールの導入
 - 竹内 良輔((株)リコー)

謝辞

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

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

両団体および上記教材の作者の皆様へ感謝します。

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