

TOPPERS中級実装セミナー

(H8/3069F版:リアルタイムシステム構築)
アプリケーション実習編:2日目

TOPPERSプロジェクト
教育ワーキング・グループ

本ドキュメントに関して

■ 本ドキュメントの利用に関して以下の点にご留意ください

1. 著作権に関しての表記

＜TOPPERS中級実装セミナー アプリケーション実習編 (H8-3069F版：リアルタイムシステム構築) 2 日目＞

Copyright (C) 2004-2006 by 竹内良輔 (株)リコー GJ事業部
 Copyright (C) 2004-2006 by 森本亮太 (株)リコー MFP事業本部
 Copyright (C) 2004-2006 by 中嶋 哲 東芝情報システム(株)

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

- (1) 本資料を利用する場合には、上記の著作権表示およびこの利用条件が、そのままの形で資料中に含まれていること。
- (2) 本資料を改変する場合には、資料を改変した旨の記述を、資料中に含めること。ただし、TOPPERSプロジェクトの活動の一環として本資料を改変する場合には、資料を改変した旨の記述を含める必要はない。
- (3) 本資料の利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

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

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

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

TRONは“The Real-time Operating system Nucleus”の略称です。ITRONは“Industrial TRON”の略称です。 μ ITRONは“Micro Industrial TRON”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. RTOSを用いたリアルタイムシステム構築 1. 5時間
2. 実習課題の説明 0. 5時間
3. 開発環境の確認 0. 5時間
4. 話題沸騰ポット作成1 3. 5時間
 - 4-1. 作成のヒント1 (0. 5時間)
 - 4-2. 作成のヒント2 (0. 5時間)

■ 2日目

1. 話題沸騰ポット作成2 1. 5時間
2. レビュー 0. 5時間
 - 設計、実装の評価
3. リアルタイム性向上の手法 1. 0時間
4. タスク負荷の検証と対応 0. 5時間
5. 話題沸騰ポットの割込みの対応 2. 0時間
 - まとめ 0. 5時間

話題沸騰ポット作成2

1. 話題沸騰ポット作成2
2. レビュー
3. リアルタイム性向上の手法
4. タスク負荷の検証と対応
5. 割込みの対応
6. まとめ

続けて話題沸騰ポットシステムの作成を行います。

話題沸騰ポット作成

継続して作成を行ってください

- 疑問がある場合は、講師に質問しましょう
- あと、1.5時間です

レビュー

1. 話題沸騰ポット作成2
2. レビュー
3. リアルタイム性向上の手法
4. タスク負荷の検証と対応
5. 割込みの対応
6. まとめ

レビュー

作成したプログラムに自信のある方は発表を

- プログラムには、正解はありません
- 種々の要件で、実装は変わります
- 設計品質自体は吟味ができます

モジュール分割

サイズの最適化、システムの明解さ

重複の極小化、情報隠蔽

モジュールの凝集度

凝集度は高いほど保守性が良い

モジュールの結合度

結合度は弱いほど保守性が良い

リアルタイム性向上の手法

1. 話題沸騰ポット作成2
2. レビュー
3. リアルタイム性向上の手法
4. タスク負荷の検証と対応
5. 割込みの対応
6. まとめ

もう、話題沸騰ポットを作ってしまった後ですが、RTOSを使ったリアルシステムのリアルタイム性の向上のために手法について説明を行います。

タスクの優先度の決定

μ ITRONでの優先度の決め方

- 時間制約を満たせる範囲では
急ぐタスク(デッドラインの短いタスク)を優先
 - 一時的な負荷時に時間制約を満たせない場合は
重要なタスク(時間制約を満たせなかった場合
被害が大きいタスク)を優先する
- 時間制約を満たせるかをどう判断するか？
リアルタイムスケジューリング理論が適応可能(後述)
- 一時的な過負荷にはどう対応するか？
重要度の低いタスクをさぼる／精度を落とす
もっと高性能なハードウェア、匠の技術

2006/11/24

TOPPERSプロジェクト認定

9



μ ITRONを含め、多くの組込み用RTOSでは、静的優先度ベースのプリエンプティブスケジューリング方式を採用しています。この方式はシングルプロセッサではリアルタイムシステムを構築する上でもっとも単純でオーバーヘッドの少ない方式といえます。また、タスクの優先度の決め方の要点は単純であり、以下の2点です。

- 時間制約を満たせる範囲では
急ぐタスク(デッドライン:タスク周期)を優先します。
- 一時的な負荷時に時間制約を満たせない場合は
重要なタスク(時間制約を満たせなかった場合、被害が大きいタスク)を優先します。

説明では単純ですが、実際、時間制約を満たせるかどうかの判定は難しい。プロダクトの終了時点ではテストやツール等で、計測可能になっていると思われますが、設計初期では難しい判定となります。後述にリアルタイムスケジューリング理論の前提となるRate Monotonic analysisについて説明を行います、前提条件が多く、また、特異なドメインの特異な機能、性能要求がある開発では、単純に判定はできません。

もう1つ一時的な過負荷があり時間制約を満たせないタスクが発生するケースでは、最初から予期して対応することは難しく、対応策として、ハードウェアでの対応や、精度を落とす等の対応も実際のプロダクトでは難しいようです。一般には、十分にプロトタイプ設定を行ったあとプロダクト化を行うか、それもできない場合、ソフトウェアのチューニングやタスク数を減らす、割込みプログラム化や周期ハンドラ等の利用等、匠技術者による対応が必要となるのではないかと思います。この章の後半では周期ハンドラや割込みの採用についても記述します。

Rate Monotonic Analysis (RMA) 1

理論の前提条件

- リアルタイムスケジューリング理論
 - ▼ 時間制約を持った処理をスケジューリングする方法と、各処理が時間制約を満たすことができるかどうかを数学的に証明するための理論



- ▼ 体系的なリアルタイムシステムの設計手法
- リアルタイムスケジューリング理論の歴史
 - ▼ 古典的な研究は1970年代にさかのぼる
 - ▼ 1980年代後半から米国中心に研究が進む



軍事システムの複雑化により、体系的なリアルタイムシステム設計手法が求められた

- ▼ 代表的な成果: Rate Monotonic Analysis (RMA)



2006/11/24

TOPPERSプロジェクト認定

10

TOPPERS

優先度ベーススケジューリングの理論的なより所となっているのが、Rate Monotonic Analysis (RMA) 理論です。この起源は1973年に発表されたLiu & Laylandの論文にさかのぼります。起動頻度のより高いタスク(周期タスクでは周期のより短いタスク)により高い優先度を与えるスケジューリング手法をRate Monotonic Scheduling (RMS)と呼びます。この手法は固定優先度スケジューリングでは最適(すなわち、ある固定優先度スケジューリング方式でスケジュール可能ならば、必ずRMSでもスケジューリング可能)であることが知られています。また、スケジュール可能性がCPUの負荷率に基づいた計算により判定できるという特性を持ちます。

1980年の半ば以降、RMA理論は多くの研究者によって研究・拡張されてきました。例えば、優先度逆転の問題を解決する手法である優先度継承手法や、非周期タスクの応答性を向上させる遅延可能サーバ、あるいはボラティックサーバなどの手法が生み出されています。RMAはリアルタイムシステムのシマンティックな設定に数々の貢献を行った基礎理論といえます。

Rate: 周期、**Monotonic:** 単調、**Analysis:** 分析

参考文献

1. 白川洋充、竹垣盛一:リアルタイムシステムとその応用、システム制御情報ライブラリ
朝倉出版(2001年)
リアルタイムスケジューリング理論に対する唯一の日本語の書物
2. Jane W. S. Liu: Real-Time Systems, Prentice Hall
リアルタイムスケジューリング理論を幅広くかつ深く解説した書物、610ページもある
3. Giorgio C. Buttazzo: Hard Real-Time Computing Systems,
Kluwer Academic Publishers(1997年)
リアルタイムスケジューリング理論を中心に関連技術も紹介。Jane Liuよりも気軽に読める。

システムに対する負荷

最大負荷の決定／想定

- すべての処理が厳密に時間制約を満たしていることを保証するためには、システムに対する最大負荷が決まっていなければならない
 - システムに対する最大負荷は決定できないか、最大負荷を考えてシステムを設計するのが現実的でない場合、現実的な最大負荷を想定する
- 
- 最大負荷を決定／想定してシステムを設計
 - 最大負荷の記述方法

$$\text{最大負荷} = \sum_{\text{各タスク}} 1\text{回の最大実行時間} \times \text{最大実行頻度}$$

2006/11/24

TOPPERSプロジェクト認定

11



リアルタイムシステムの厳密なリアルタイム性を保証するには、システムに対する最大負荷が分かっている必要があります。しかし、設計の当初から十分な最大負荷算出した上で設計することはRTOSを用いる規模のシステムでは無理があります。それは以下の事情によります。

1. 設計と実装は異なる。設計者がすべてのプログラムを吟味して作成はできない
2. 要求機能は、時間経過とともに変化します
3. ミドルウェアやハードウェアなどの条件により、最大負荷は異なり、たぶん、それらを吟味してプロダクトを始めるより、ある程度、開発後に検証した方が効率がよい

そのため、実際のリアルタイムシステムの構築では、最大負荷を想定しシステムを設計します。

Rate Monotonic Analysis (RMA)2

理論の前提条件

- いくつかの大前提(以下は暗黙の前提とする)
 - ▼静的優先度割付け下でプリエンプティブな優先度ベーススケジューリング ➡ μ ITRONのスケジューリング方式
 - ▼システムの最大負荷が見積もれることが「保証」の基本的な前提
 - ▼周期タスク(または最小起動間隔がわかっている非周期タスク)を基本とする
 - ▼シングルプロセッサを基本とする
- 各種の資源に適用可能
 - ▼プロセッサ ➡ 以下ではプロセッサを例に議論を進める
 - ▼ネットワーク ... 完全なプリエンプティブではない点異なる

2006/11/24

TOPPERSプロジェクト認定

12



RMA理論には、いくつかの暗黙の前提が必要となります。

- ▼静的優先度割付け下でプリエンプティブな優先度スケジューリングを行っていること
現在の組み込み用のRTOSは、このスケジューリング方式を採用している
- ▼システムの最大負荷が見積もれることが「保証」の基本的な前提となる
プロダクトの終了時には可能であるが、実際のプロダクトでは要求機能や制約が変化する
加えて、プロダクトには競合相手があり、競合機種の仕様にも左右される
プロダクトによっては難しい前提となる
- ▼周期タスク(または最小起動間隔がわかっている非周期タスク)を基本とする
- ▼シングルプロセッサを基本とする

本講義では、プロセッサ資源に着目して進めます。もちろん、実際のハードウェアや外部のデバイス、通信によっても大きく影響を受けます。

Rate Monotonic Analysis (RMA)3

Critical Instant定理のタスクセットに対する仮定

1. 周期タスク(または最小起動間隔がわかっている非周期タスク)のみ考える
2. 各タスクの最大実行時間はあらかじめわかっている
- 3-4. 各タスクの相対デッドラインが周期(または最小起動間隔)以下
5. タスク間に同期通信がない
6. タスクは自ら実行を中断しない
7. タスク切替え等のオーバーヘッドは考えない
8. プロセッサが1つのケースのみ考える

2006/11/24

TOPPERSプロジェクト認定

13



Critical Instant定理はRMAの出発点となる重要な定理です。この定理で扱うタスクセットには以下の仮定があるものとします。(通常のリアルタイムシステムではありえない仮定もあります)

1. 周期タスク(または最小周期起動間隔がわかっている非周期タスク)のみを考える
2. 各タスクの最大実行時間はあらかじめわかっている
- 3-4. 各タスクの相対デッドラインが周期(または最小起動間隔)以下
5. タスク間には同期通信がない
6. タスクは自ら実行を中断しない
7. タスク切替え等のオーバーヘッドは考えない
8. プロセッサが1つのケースのみ考える

注意:タスクは任意の位相で起動するものとする

3-4を共通にしているのは、後述の「プロセッサ使用率によるスケジューリングチェック」のタスクセットの仮定と比較が可能にしている。

Rate Monotonic Analysis (RMA)4

Critical Instant定理

- Critical instantとは何(定義)
 - ▼ あるタスクに対するcritical instantとは、そのタスクが起動してから終了までの時間が最も長くなる状況
- Critical instant定理
 - ▼ 静的な優先度割付け下でプリエンティブな優先度ベーススケジューリングを行った場合、あるタスクに対するcritical instantは、以下にあげる前提条件下でそのタスクが起動されると同時に、そのタスク以上の優先度をもつすべてのタスクが同時に起動された場合

前提条件

タスクがデッドラインをミスしない場合範囲のみ考える
→タスクが多重に起動されることはない

2006/11/24

TOPPERSプロジェクト認定

14

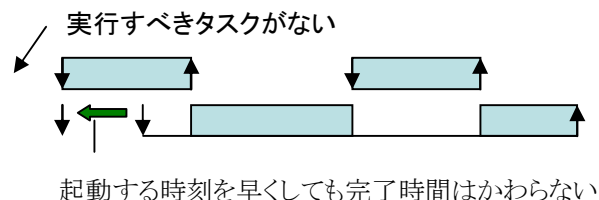


Criticalは「危険な」、instantは「瞬間」、Critical instant定理とは、「あるタスクの最も危険な状況は、そのタスクが起動されると同時に、そのタスク以上の優先度を持つすべてのタスクが同時に起動された場合である」という定理です。つまり、この場合にあるタスクのデッドラインミスが発生しやすいということになります。

図解による証明

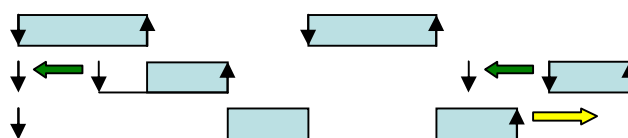
ステップ1

高優先度タスク
着目するタスク



ステップ2

最優先度タスクA
高優先度タスクB
着目するタスクC



起動時刻を早くすると着目するタスクの
完了時刻は遅くなる方向へ

Critical Instant定理からいえること スケジュール可能性の必要十分条件

- Critical Instantに起動された場合でもデッドライン内に実行が終わるなら、そのタスクはスケジューリング可能
- その逆も成立
- タスク i (タスクのインデックスは優先度の高い順につける)がスケジューリング可能な必要十分条件

$$\exists t, 0 < t \leq D_i, \sum C_j \left\lceil \frac{t}{T_j} \right\rceil \leq t$$

T_j : 優先度 j 番目となるタスクの周期

D_j : 優先度 j 番目となるタスクの相対デッドライン

C_j : 優先度 j 番目となるタスクの最大実行時間

2006/11/24

TOPPERSプロジェクト認定

15



Critical Instantは最も、危険な状況です。つまり、タスクの実行時間が最も長くなる状況です。この状況デッドラインを守るならば、そのタスクはスケジューリング可能といえます。理論的には、リアルタイムシステムのすべてのタスクが上記の必要十分条件を満たせば、リアルタイム性が保証できることになります。

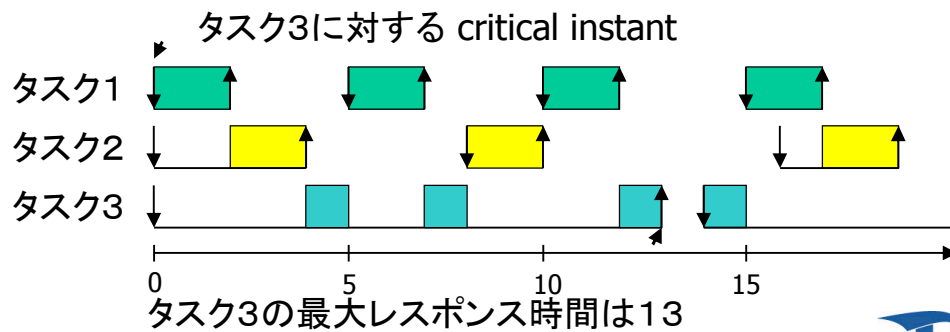
$\left\lceil \frac{t}{T_j} \right\rceil$ は最大頻度、切り上げ値です。

D_j は優先度 j 番目のタスクの相対デッドラインを示します。相対デッドラインとは、起動時間からの相対時間です。相対デッドラインに対して、絶対デッドラインがあり、絶対デッドラインは何年、何月、何日、何時、何分、何秒のように絶対的な時間でデッドラインを表します。

スケジュール可能性の検証1

検証のための簡単なタスクセットを用意します

j	T_j	D_j	C_j	使用率(負荷)
1	5	5	2	40%
2	8	8	2	25%
3	14	14	3	21.4%



2006/11/24

TOPPERSプロジェクト認定

16



必要十分条件の検証のために、簡単なタスクセットを用意します。

タスク1が一番優先度が高く、周期:5、相対デッドライン:5、最大実行時間:2

タスク2が二番目に優先度で、周期:8、相対デッドライン:8、最大実行時間:2

タスク3が最低の優先度、周期:14、相対デッドライン:14、最大実行時間:3

使用率はタスク3を例にとると、14の周期で3タスクを実行するつまり、 $\frac{3}{14} = 0.214285\dots$

これらの負荷をすべて足しても、86.4%であり、使用率からはスケジューリングは可能となります。

着目のタスク3はCritical Instantでも、 $t=13$ に実行終了し、デッドラインは守れます。

スケジュール可能性の検証2

必要十分条件式による評価

- 右式に当てはめると $\exists t, 0 < t \leq D_i, \sum C_j \left\lceil \frac{t}{T_j} \right\rceil \leq t$

$$t = 13 \leq 14 \text{ で } 2 \left\lceil \frac{13}{5} \right\rceil + 2 \left\lceil \frac{13}{8} \right\rceil + 3 \left\lceil \frac{13}{14} \right\rceil = 13 \leq 13$$

- 最大レスポンス時間の算出方法

response time analysis

$$r^{(0)} = C_i$$

$$r^{(i+1)} = \sum C_j \left\lceil \frac{r^{(i)}}{T_j} \right\rceil$$

$$r^{(0)} = 3$$

$$r^{(1)} = 7$$

$$r^{(2)} = 9$$

$$r^{(3)} = 11$$

$$r^{(4)} = 13$$

$$r^{(5)} = 13$$

一致したところが
レスポンス時間

- このアプローチの弱点

各タスクの正確な最大実行時間がわかっていることが必要

2006/11/24

TOPPERSプロジェクト認定

17



右式に当てはめ、タスク3の相対デッドロック14以下の $t = 13$ で $13 \leq 13$ が成り立ちます。

$$t = 12 \text{ とすると } 2 \times 2 + 2 \times 3 + 3 \times 1 = 13 \leq 12$$

$$t = 14 \text{ とすると } 2 \times 2 + 2 \times 3 + 3 \times 1 = 13 \leq 14$$

$t = 13$ でスケジューリング可能と判定できます。

最大レスポンス時間の算出は計算機を用いて、最大レスポンスを計算する方法を示します。

$$r^{(0)} = 3$$

$$r^{(1)} = 2 \times \left\lceil \frac{3}{5} \right\rceil + 2 \times \left\lceil \frac{3}{8} \right\rceil + 3 \times \left\lceil \frac{3}{14} \right\rceil = 2 + 2 + 3 = 7$$

$$r^{(2)} = 2 \times \left\lceil \frac{7}{5} \right\rceil + 2 \times \left\lceil \frac{7}{8} \right\rceil + 3 \times \left\lceil \frac{7}{14} \right\rceil = 4 + 2 + 3 = 9$$

$$r^{(3)} = 2 \times \left\lceil \frac{11}{5} \right\rceil + 2 \times \left\lceil \frac{11}{8} \right\rceil + 3 \times \left\lceil \frac{11}{14} \right\rceil = 6 + 2 + 3 = 11$$

$$r^{(4)} = 2 \times \left\lceil \frac{13}{5} \right\rceil + 2 \times \left\lceil \frac{13}{8} \right\rceil + 3 \times \left\lceil \frac{13}{14} \right\rceil = 6 + 4 + 3 = 13$$

最大実行時間はプログラム記述終了後に計測が可能となります。この時間をシステム設計時に確定することは難しく、システムすべての時間の計測を行う為に、多くの工数を必要となります。

スケジュール可能性の検証3

最適な優先度割付け

Critical instant定理と同じ仮定下で、タスクにどのように優先度を割り当てるのが最適か？

- Deadline monotonic scheduling

相対デッドラインが短いタスクほど高い優先度を割り付ける

 **急ぐタスクほど高い優先度**

歴史的には「相対デッドライン＝周期」の前提で議論が始まった。周期が短いタスクほど高い優先度を割り付ける方式をrate monotonic scheduling (RMS)と呼ぶ

2006/11/24

TOPPERSプロジェクト認定

18



Monotonicは「単調」という意味で、Deadline monotonic schedulingは、相対デッドラインが短いタスクほど、高い優先度を割付ける方式です。最初の研究では、「相対デッドライン＝周期」の前提で議論が始まった関係で、deadlineをrateに置き換えて方式を命名されるケースが多々あり、一般化されました。

プロセッサ使用率とスケジュール可能性1 タスクセットに対する仮定

1. 周期タスク(または最小起動間隔がわかっている非周期タスク)のみ考える
2. 各タスクの最大実行時間はあらかじめわかっている
3. 各タスクは周期の始めに実行可能になる
4. 各タスクのデッドラインは周期の終わり
「相対デッドライン=周期」
5. タスク間に同期通信がない
6. タスクは自ら実行を中断しない
7. タスク切替え等のオーバヘッドは考えない
8. プロセッサが1つのケースのみ考える

➡ プロセッサに対する負荷が変わらないタスクセット

2006/11/24

TOPPERSプロジェクト認定

19



プロセッサの使用率によるスケジュール可能性チェックの検証を行う前に、タスクセットの仮定を行います。この仮定は前述のcritical instant定理のタスクセットの仮定とほぼ同じですが、3と4に関して若干の差異があります。

critical instant定理の3, 4の条件は各タスクの相対デッドラインが周期以下という仮定を行っているのに対して、この仮定では周期の始めにタスクは実行可能になり、相対デッドラインは周期の終わりとしています。この仮定はプロセッサに対する負荷が変動しないタスクセットを仮定しています。

プロセッサ使用率とスケジュール可能性2

タスクセットがスケジュール可能になる十分条件

■ プロセッサ使用率からの十分条件

$$\sum_{j=1}^n \frac{C_j}{T_j} \leq n(2^{1/n} - 1)$$

n : タスク数

T_j : 優先度が j 番目のタスクの周期

C_j : 優先度が j 番目のタスクの最大実行時間

左辺: プロセッサ使用率

n	右辺
2	0. 83
3	0. 78
∞	0. 69

➡ 悲観的な上限値(あくまで十分条件)
平均0. 88まで大丈夫という結果も

! 失った31%は、優先度を静的に割り付けていることから
くる本質的な限界

2006/11/24

TOPPERSプロジェクト認定

20



プロセッサ使用率に対する、プロセッサに対する負荷が変動しないタスクセットがスケジュール可能になる十分条件は以下の通りです。

$$\sum_{j=1}^n \frac{C_j}{T_j} \leq n(2^{1/n} - 1)$$

n : タスク数

T_j : 優先度が j 番目のタスク周期

C_j : 優先度が j 番目のタスク最大実行時間

左辺: プロセッサ使用率(言い換えるとCPUの実際の負荷)

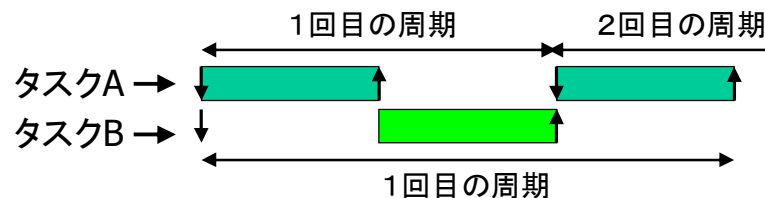
特に問題と思われるのは、プロセッサの使用率がかなり低い点です。これはあくまで十分条件であり、例えば、 n が ∞ の場合、実際にプロセッサ69%しか使用できないとすると、リアルタイム性や性能上大きな問題となります。残りの31%は、保証ができないだけで、実際にはスケジューリングは可能のケースもあると考えてください。それは静的な優先順位設定に依存してるのですが、この検証と回避方法は以下のページで行います。

この十分条件は悲観的な上限値です。ある科学者が乱数を用いて、平均的な上限値を算出した結果、0.88までは大丈夫という報告もあります。

プロセッサ使用率とスケジュール可能性3

失った31%は何か？(証明の入口)

- 周期比率が1対1.5の2つのタスクを考える



タスクAのプロセッサ使用率は半分程度にもかかわらず、
タスクBの1回目の周期で使えるプロセッサ時間は3分の
1程度しかない

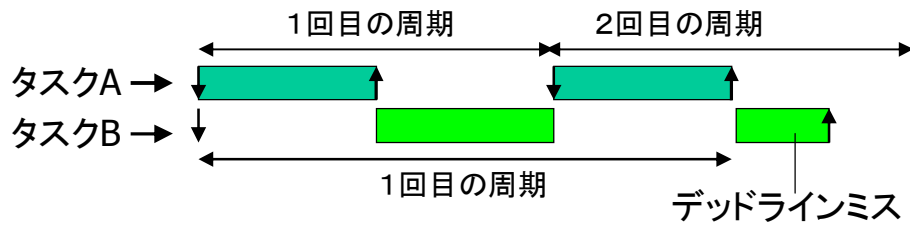


残りの6分の1のプロセッサの使用率は利用できない
実は周期の比が $1:\sqrt{2}$ の時に最悪になる

周期比率が1:1.5の例では、タスクBの最大実行時間がこれ以上長いとデッドラインミスが発生します。しかし、デッドラインまでは3分の1の時間が残っています。つまり、周期の比率によっては、プロセッサの使用率にあまりがあっても、スケジューリングできないケースがあるということです。

プロセッサ使用率とスケジュール可能性4

静的優先度割付けの本質的な限界



- タスクBの1回目の周期の絶対デッドラインは、タスクAの2回目の周期のデッドラインよりも早いですが、相対デッドラインはタスクAの方が短いため、タスクAに高い優先度が割付けられている



- 優先度を固定していることから生じる本質的な問題

2006/11/24

TOPPERSプロジェクト認定

22



静的なタスク優先度割付では、優先度の低いタスクがデッドラインミスを発生しそうな状態でも、デッドラインまで余裕のある優先度の高いタスクが優先して実行されます。これは静的タスク優先度方式の本質的な問題です。実際のシステム設計では、**harmonic task set**になるように設計を行ったり、タスクの実行時間を減らすようにプログラミングを行ったり、割込み等をうまく使うことにより、ある程度、プロセッサの負荷を減らしリアルタイム性を確保するような設計を行うことが可能です。

プロセッサ使用率とスケジュール可能性5 上限値が悲観的な理由

- プロセッサ使用率が69%は極めて特殊なタスクセットで生じる

n 個のタスクの周期の比が以下の場合

$$1 : 2^{\frac{1}{n}} : 2^{\frac{2}{n}} : \dots : 2^{\frac{n-1}{n}}$$



すべてのタスクの比率が1:2以内

→ 周期の比が大きくなると上限値は大きくなる

- Harmonic task setの場合

それぞれのタスクの周期の比が倍率関係

プロセッサの使用率100%までスケジュール可能

Harmonic task subset を含む場合も有利に



プロセッサの使用率が最悪(69%)となるのは、タスクの周期の比が特別の場合です。

すべてのタスクの比率が1:2以内の場合です。システムを設計する上でこの周期比となる設計は避けるべきです。最もプロセッサの使用率が高くできる周期比のタスクセットはharmonic task set呼ばれ、タスクの周期の比が整数倍率関係となっています。

プロセッサ使用率とスケジュール可能性6

動的優先度割付けの場合(参考)

優先度割付けを動的に変化させることで解決

■ Earliest deadline first scheduling (EDFS)

- ▼ デッドラインが近いタスクから順に高い優先度を割り付ける。最適な動的優先度割付けアルゴリズム
- ▼ タスクセットがスケジュール可能になる必要十分条件

$$\sum_{j=1}^n \frac{C_j}{T_j} \leq 1 \quad \longrightarrow \quad \text{プロセッサの能力を100\%使える}$$

n : タスク数

T_j : 優先度が j 番目のタスクの周期

C_j : 優先度が j 番目のタスクの最大実行時間



参考として、動的なタスク優先度スケジューリングを行うEarliest deadline first scheduling (EDFS)について、説明を行います。このスケジューリング方式はデッドラインが近いタスクの優先度を動的に高くすることによりデッドラインミスを防ぐ方式です。

RTOSには、EDFSもサポートしているものがあります。それでは、多くのRTOSはEDFSを採用していないのでしょうか？実はEDFSにも、欠点があるのです。

優先度ベーススケジューリングでは、タスクの実行時間が長くなると、優先度の低いタスクが割を食うことになりますが、EDFSでは、タスクの実行時間が長くなると、どのタスクが割を食うかがわからないのです。そうすると、必ずデッドラインミスを起こしたくないタスクがある場合、どのような対策で回避するか対策を立てられなくなるのです。

タスクセットに対する仮定を緩める1

- 「1. 周期タスクのみ考える」と「2. 最大実行時間がわかっている」は、最大負荷を見積もれるという大前提から本質的
- 最大実行時間の分かっている周期タスクのデッドラインを守りつつ、そうでないタスクを最も早く実行する方法は研究されている
- 「3. 各タスクは周期の始めに実行可能に」は、実行可能になるのが周期の始めより遅れる効果を考慮することは可能
- 「3－4. 相対デッドラインは周期以下」と「4. 各タスクのデッドラインは周期の終わり」は、タスクが多重起動される場合を考えなければならないが条件が複雑にはなるが、仮定をはずすことは可能

RMAでのタスクセットに対する仮定を緩める可能性について検討します。

タスクセットに対する仮定を緩める2

- 「5. タスク間に同期通信がない」は、低い優先度のタスクに待たされる最大時間(最大ブロック時間: B_i)がわかれば、その効果を式に入れることが可能

$$\forall i, \exists t, 0 < t \leq D_i, \sum C_j \left\lceil \frac{t}{T_j} \right\rceil + B_i \leq t$$

- 「6. タスクは自ら実行を中断しない」は、十分に研究されていない課題
- 「7. タスク切替え等のオーバーヘッドは考えない」は、単純なタスク切替えのオーバーヘッドは考慮可能
 → タスク切替え2回分の実行時間を、各タスクの最大実行時間に加える
- 「8. プロセッサが1つ」については、タスクを実行するプロセッサが固定しない場合への拡張は困難



2006/11/24

TOPPERSプロジェクト認定

26

TOPPERS

- $\lceil x \rceil$ は $x \leq n$ を満たす、最小の整数、つまり、切り上げ値
- $\lfloor x \rfloor$ は $n \leq x$ を満たす、最大の整数、つまり、切り捨て値 (ガウスの記号 $[]$ におなじ)

非周期タスクの扱い もうひとつのタスクの振舞い

■ 目標

- ▼ 周期タスクが時間制約を満たすことを保証しつつ、できる限り早く非同期タスクを実行する

■ 最も簡単な方法 — バックグラウンド方式

- ▼ 非周期タスクを最低優先度で実行する
 - ▼ 実現が最も簡単
 - ▼ 周期タスクの時間制約は通常の方法で扱える
 - ▼ 非同期タスクをなるべく早く実行したとは言えない

■ 非周期タスクサーバ方式

- ▼ 非周期タスクを実行するサーバタスクを用意する
- ▼ サーバタスクの実行制御方法にいくつかのアルゴリズムが提案されている

2006/11/24

TOPPERSプロジェクト認定

27



組込みシステムが巨大化すると、リアルタイム性を必要とするタスクの比率が全体のタスクに比べ低くなります。実際に巨大なリアルタイムシステムの構築は難しいものです。このようなシステムでは非周期タスクの有効的な実行が大きな課題となります。もっとも簡単な実現方式は最低優先度で実行するタスクの実行方式をうまく選定することです。タスクの数が多い場合は、`rot_rdq`と`irotd_rdq`サービスコールを用いて、ラウンドロビンの実行を行えます。複雑となりますが、非同期タスクサーバを用いる方式もあります。非同期タスクサーバには以下のアルゴリズムが研究されています。

• ボーリング方式

サーバの周期と最大実行時間を決め、通常の周期タイマと同様にスケジューリングします。

実行すべきタスクがない場合、サーバはすぐに実行終了し、次の周期まで実行されません。

• 優先度交換サーバ (priority-exchange server)

• 遅延可能サーバ (deferrable server)

• スポラディックサーバ (sporadic server)

Slack Stealing方式

原理的には、上記の方式よりさらに非周期タスクを早くスケジューリングできます。但し、アルゴリズムが複雑でオーバーヘッドが大きくなります。

理論から実システムへ 種々の課題

- 現状のRTOSはRMAと最も相性が良い
 - ▼プリエンプティブな優先度ベーススケジューリングは、ほとんどのリアルタイムOSがサポートしている
- リアルタイムOSのオーバーヘッドを考える
 - ▼システムコールの処理時間は、それを呼び出したタスクの最大実行時間に含めて考える
 - ▼タスク切替え2回分の処理時間を各タスクの最大時間に加える(タスク間に同期・通信がない場合)
- タスクと割込みハンドラ、タイムイベントハンドラを使い分ける
 - 以降の章で説明
- タスクの最大実行時間の見積もり
 - ▼タスクの最大実行時間の見積もりの難しさは、理論の実システムへの応用を阻んでいる最大の要因

2006/11/24

TOPPERSプロジェクト認定

28



ここまでの内容で、 μ ITRONでのタスクに関する適切なスケジューリング方式が明らかになってきたはずですが。具体的には

1. Deadline monotonic scheduling

相対デッドラインが短いタスクほど高い優先度を割り付ける

2. Harmonic task set

タスクの周期の比を倍率関係とすることにより、プロセッサ使用率を向上させる

μ ITRONではタスク以外にも機能の記述が可能です。タスク以外は非タスクコンテキストと呼ばれており、割込みハンドラやタイムイベントハンドラがこれに対応します。これらのハンドラを用いれば、タスクのルール以外でのプログラム記述が可能になります。

RMAではタスクの最大実行時間の見積もりが大きな課題となります。先にも記述しましたが、実際のリアルタイムシステムは種々の要求により、プロダクト終了までプログラム自体が確定できなかったり、組込みシステムの肥大化により見積もることが困難となっています。加えて、リアルタイムOSの利用の利点である外部リソースのミドルウェアに関してはブラックボックスの状態である。実際のリアルタイムシステム開発では、プロトタイプ設計をきちんと行い、実行時間の測定や予測を行ったり、プロダクト時も、スパイラルな開発手法を用いて、大きなタスク負荷が発生しないような開発方式を採用することにより安全性を高める等の工夫が必要となります。また、多くのシステムを開発してきたソフトウェアエンジニアの経験や感等も重要な要素となります。

非タスクコンテキストとタスクコンテキスト CPUの実行は2つのコンテキストに分類

- μ ITRON4.0仕様ではCPU実行を以下の処理単位で実行管理している

1	割込みハンドラ、サービスルーチン	非タスクコンテキスト
2	タイムイベントハンドラ	非タスクコンテキスト
3	CPU例外ハンドラ	呼び出しコンテキストによる
4	拡張サービスコールルーチン	呼び出しコンテキストによる
5	タスクとタスク例外	タスクコンテキスト

- タスクとタスクの一部とみなされるコンテキストをタスクコンテキスト、それ以外を非タスクコンテキストと呼ぶ
- 非タスクコンテキストはタスクコンテキストより優先順位が高い

2006/11/24

TOPPERSプロジェクト認定

29



タスクの処理の一部とみなすことのできるコンテキストを総称してタスクコンテキスト、そうでないコンテキストを総称して非タスクコンテキストと呼びます。 μ ITRON4.0仕様では、以下の5つの処理単位の実行制御を行います。これらの実行状態もどちらかのコンテキストに分類されます。

a) 割込みハンドラ

a.1) 割込みサービスルーチン

b) タイムイベントハンドラ

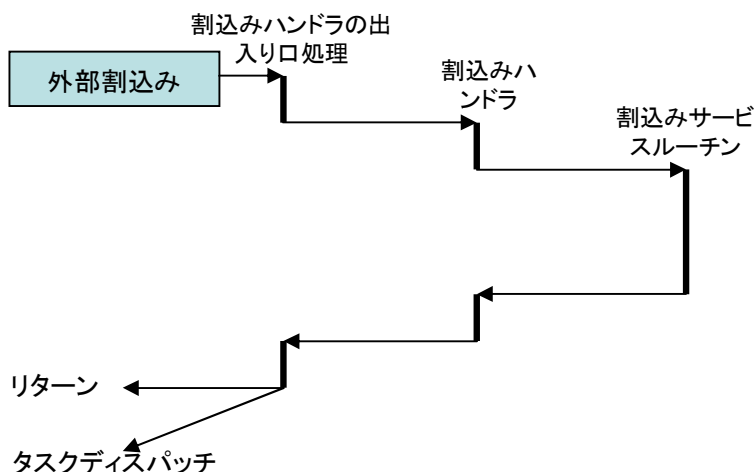
c) CPU例外ハンドラ

d) 拡張サービスコールルーチン

e) タスク

e.1) タスク例外処理ルーチン

対応は上の表の通りである。非タスクコンテキストはタスク優先度が高く、非タスクコンテキストを用いれば、より高いリアルタイム性をオーバーヘッドを少なく実現することが可能です。以下の割込みモデルはタスクの実行中にタスク処理を遅延させ割込みサービスルーチンを実行させます。

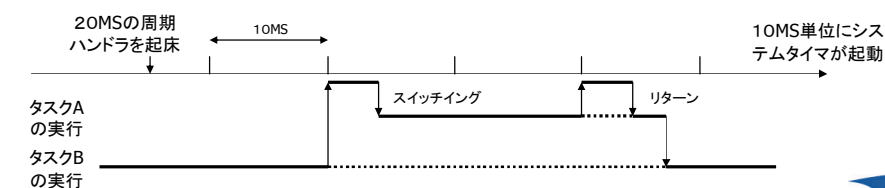


より効果的な周期ハンドラ タスクの切替えなしにプログラムを実行

- μ ITRON4.0では以下の3つのタイムイベントハンドラが定義されている

周期ハンドラ	スタンダードプロファイル
アラームハンドラ	フルセット
オーバーランハンドラ	フルセット

- 周期ハンドラは一定時間で起動されるタイムイベントハンドラで、タスクに優先して実行される



2006/11/24

TOPPERSプロジェクト認定

30



μ ITRON4.0ではタイムイベントハンドラは以下の3つのハンドラが定義されています。周期ハンドラはスタンダードプロファイルで定義されたハンドラでTOPPERS/JSPでも使用することができます。

- 周期ハンドラ(一定周期で起動する)
- アラームハンドラ(指定した時刻に起動する)
- オーバーランハンドラ(タスクが指定した時間を越えてプロセッサを使用したときに起動する)

タスクを周期的に起動する場合、前の章で学んだ通り、より高い優先度のタスクの実行状態によって実行精度のくまりが発生します。周期ハンドラはタイマ割込みにより起動し、非タスクコンテキストで実行されるため、高い精度で実行されます。また、ハンドラ中のプログラムの結果により、実行後タスクスイッチングが行われるケースを除いてタスクスケジューラを介さず実行が可能な為、実行処理効率が高くなります。

非タスクコンテキストの実行

非タスクコンテキスト中は待ち状態をつくれない

- 非タスクコンテキストではタスクのような待ち状態はないため、サービスコールは限定される
- 待ち状態となるサービスコールは使用できず、タスクにイベントを伝えるサービスコールのみ使用できます

機能	TOPPERS/JSPで使えるサービスコール
タスク管理機能	iact_tsk
タスク付属同期機能	iwup_tsk, irel_wai, iras_tex
同期通信機能	isig_sem, iset_flg, ipsnd_dtq, ifsnd_dtq
システム状態管理機能	irotd_rdq, iget_tid, iloc_cpu, iunl_cpu

2006/11/24

TOPPERSプロジェクト認定

31



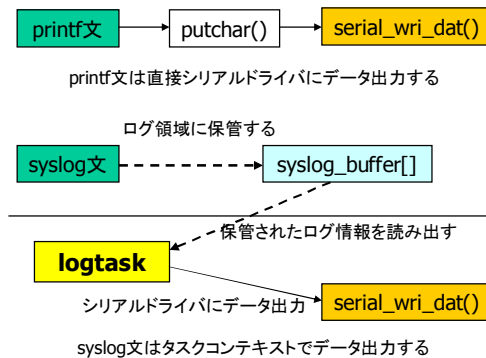
非タスクコンテキスト実行はCPUの例外処理中に実行します。そのため、待ち状態を作れないばかりか、ソフトデレイで待ち状態を作ってしまうと、他の非タスクコンテキストやタスクコンテキストの実行を妨げてしまいます。そのため、使用できるサービスコールはタスクにイベントを伝達するものに限られます。同様に非タスクコンテキストからドライバなどのデバイスを制御する関数の呼び出しにも制限があります。H8/3069Fで使用しているシリアルやネットワークドライバ内部ではハードウェアからの割り込み関数を使って動作しています。周期ハンドラや割り込みルーチンはタイムクリティカルな実行は可能ですが、その反面、多用しすぎたり、複雑な実装を行うと、全体としてのリアルタイム性を損なう場合があり、システム設計上十分な配慮が必要です。

非タスクコンテキストで使えるサービスコールの説明を行います。

iact_tsk	タスクを起動する
iwup_tsk	タスクを起床する
irel_wai	待ち状態の強制解除を行う
iras_tex	タスク例外処理の要求
isig_sem	セマフォ資源の返却を行う
iset_flg	イベントフラグのセットを行う
ipsnd_dtq	ポーリングモードでデータキューへの返却を行う
ifsnd_dtq	データキューへの強制返却を行う
irotd_rdq	タスクの優先順位の回転を行う
iget_tid	実行状態のタスクIDの参照を行う
iloc_cpu	CPUロック状態への移行
iunl_cpu	CPUロック状態の解除

syslogとprintf文 2つのログ表示手段

- 非タスクコンテキスト実行プログラムのデバッグは基本的に待ち状態を作れない為に、ICE等がない場合はログデバッグが主体となる
- printf文は、デバイスドライバによるデータ出力を行う為、使用できない
- syslog文またはsyslog_n文によるログ出力が使用可能



2006/11/24

TOPPERSプロジェクト認定

32



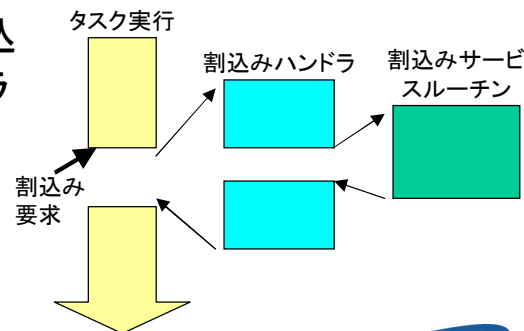
printf文はUNIXの標準入出力機能を用いてメッセージ出力するC言語関数です。printf文を含む標準入出力の機能はstdio.hインクルードファイル(newlib中に定義)にて宣言されています。標準入出力機能は入力先、出力先をファイルとして定義します。UNIXのファイルにはキャラクタデバイスとストレージデバイスの宣言上の差異がないため、シェルの機能と組み合わせることで自由に入出力先の設定が可能です。TOPPERS/JSPでは標準ではシリアルデバイスしかサポートしておらず、UNIXのシェルのような強力なユーザインターフェイスを持たないため、stdio.hの機能をサポートすることはできません。しかし、開発環境がタイムリーに準備できない場合、LINUX等のUNIX環境でアプリケーションプログラムをシミュレーション開発を行うのは非常に有効な手段であり、UNIX上の開発ではprintf()、sprintf()、putchar()を多く用いて開発を行います。そこで、タスクモニタを使用すると、シリアルデバイスに対するprintf()、putchar()が使用できるようにしてあります。この場合、monitorディレクトリ中のstdio.hインクルードファイルを使用するようにしてください。

しかし、シリアルドライバを直接使用するprintf()は非タスクコンテキスト中では使用できません。この場合、TOPPERS/JSPでサポートしているsyslog()またはsyslog_n()を使用してください。syslog文は実行時、SYSLOG構造体のデータ領域syslog_buffer[]のログ情報を蓄え、出力はlogtaskでシリアルデバイスに出力しますので、コンテキスト状態に関係なく使用できます。

さらに効果的な割込み

ハードウェアイベントが必要な時だけプログラム実行

- 割込みはハードウェアからの実行要求があった場合、プロセッサに要求を伝えるための機構です
- プロセッサはハードウェアを監視する必要がなくなる
- μ ITRON4.0では割込み時、割込みハンドラと割込みルーチンが実行され、ユーザは割込みサービスルーチンを記述できます



2006/11/24

TOPPERSプロジェクト認定

33

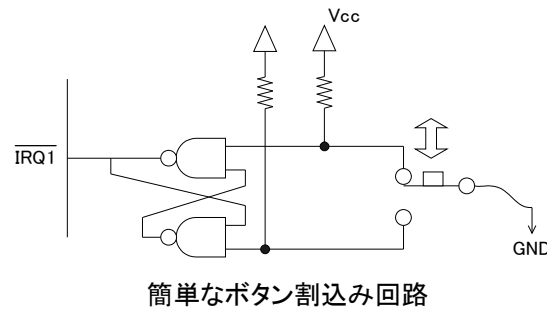


プロセッサが外部のハードウェアに機能を依頼する場合、ハードウェアからの要求を効率よく取り込むための機構が必要です。これが割込み機能です。 μ ITRON4.0が対象とする汎用のマイクロプロセッサはこの機構が組み込まれており、 μ ITRON4.0でも円滑に割込みを管理する機能が組み込まれています。周期ハンドラと比べて、周期的なハードウェアの監視機構が必要なくなるため、より効率よくプロセッサを使用できます。

μ ITRON4.0での割込みプログラムは割込みハンドラと割込みサービスルーチンに分けて実行されます。割込みハンドラは μ ITRON4.0の実装者が提供しますが、割込みサービスルーチンはミドルウェアの提供者を含めてユーザが作成すべきプログラムです。割込みを用いればより効率よく、デッドラインの制御が可能です。

割込みにはハードウェアの対応が必要 割込みを円滑に動作させる回路が必要

- 割込みの対応には、ハードウェアの対応が必要であり、システムの構築時プロダクトでの取り決めが必要
- 割込み回路はコストアップの可能性がある、システム全体での吟味が必要（話題沸騰ポットのボタン、蓋割込みはオーバースペック？）

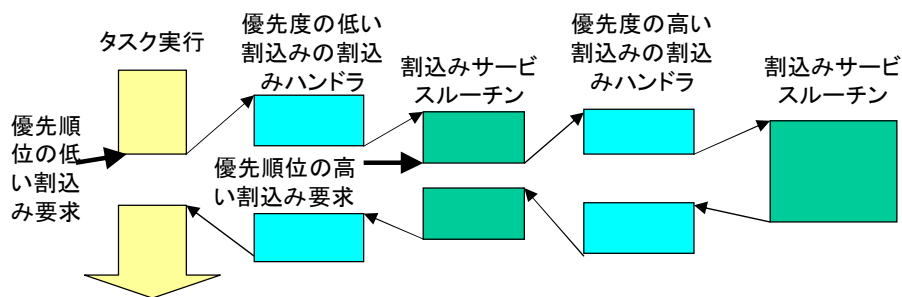


エッジトリガのボタン割込み回路の例をのせましたが、話題沸騰ポットのように、ボタンが5つもあるとこれらをまとめたり、オン、オフで割込みを発生させたり等、かなり、高価な回路になります。製品コストを考えれば、余計な回路がなく、ワンチップCPUのポートを使って直にボタンを監視する方がはるかにコストダウンできます。プロダクトとしては、割込み等を用いることによって、プロセッサの負荷を軽減し、トータルコストとしてコストダウンできることを示す必要があります。

ここまでやるか、多重割込み

他の割込み中に割込みを実行

- プロセッサが多くの機能を外部のハードウェアに依存する場合、多くの割込みを管理しなければならない
- デッドラインの短いハードウェアが他の割込みによって阻害されることを避けるための機構が多重割込みです



2006/11/24

TOPPERSプロジェクト認定

35



プロセッサが多く割込みを管理し、割込みサービスルーチンの実行時間が、最もデッドラインの短い割込みよりも長い場合、デッドラインの長い割込み中に、優先度の高い(デッドラインの短い)割込みを割り込ませて、この問題を解決する必要があります。この機構が多重割込みです。

多くのプロセッサでは割込みに優先順位が設定されており、優先順位の低い割込みが実行中で割込み禁止状態でない場合、その優先順位より高い割込みの要求が発生した場合、その割込みが実行可能です。

タスク負荷の検証と対応

1. 話題沸騰ポット作成2
2. レビュー
3. リアルタイム性向上の手法
4. タスク負荷の検証と対応
5. 割込みの対応
6. まとめ

2006/11/24

TOPPERSプロジェクト認定

36



ポーリング版の話題沸騰ポットのタスク実行状態を、タスクモニタを使って調べてみます。
ここでは、TOPPERS/JSP H8版で割込みに対応させるための手順について説明を行います。

ポットポーリング版でのタスク検証 タスクモニタで実行状態の確認

- ポットシミュレーション実行時、タスクモニタのLOG TASKコマンドにてタスクの実行状態を確認し、タイマを起動してください
- タイマを起動すると、ほとんどタスクの空き時間がないことがわかります

タイマを起動すると
タスク未実行時間が
なくなる

2006/11/24

TOPPERSプロジェクト認定

37



話題沸騰ポットシステムを起動させ(完成できなかった人はPOTTN中のjsp.motをダウンロードして実行させてください)、タスクモニタからLOG TASK コマンドを1000msで起動させてください。

mon>LOG TASK 1000←

Vacacyの項目にタスクの1000ms実行中の空き時間が表示されます。

この時間は割込み時間を含むため、正確なタスク未実行時間ではありません。また、各タスクの実行時間もnetDeviceのアクセス時間を含まない為、正確な時間とはいえません。

このあと、ポットシミュレータで、キッチンタイマーを起動するとid=0001のtimer_task実行し始め、0から実行時間になります。

このとき、vacacyを見ると0msに近い表示に変わります。キッチンタイマーが動作すると、優先順位の低いタイマータスクはデッドラインに近い動作をしていることが分かります。

ポーリング版のタスク設定

リアルタイム性の検証

- event_task以外のタスクは周期的に起動して、話題沸騰ポットの内部の監視を行う
- timer_taskは優先度が低いため、全てのタスクが動いていると実行が妨げられる

ID	タスク関数名	優先度	機能
1	timer_task	12	各タスクの起動プログラム、タイマが起動すると200msごとに時間を監視する、ブザー時間の制御を行う
2	button_task	8	50msごとにボタンと蓋を監視する
3	heater_task	11	100msごとにヒータを制御する
4	error_task	10	100msごとにヒータエラーを監視する
5	event_task	4	button_taskからのイベントを解析し各タスクに動作を指示する

2006/11/24

TOPPERSプロジェクト認定

38



POTTN中の話題沸騰ポットのプログラム内容に即した説明を行っています。

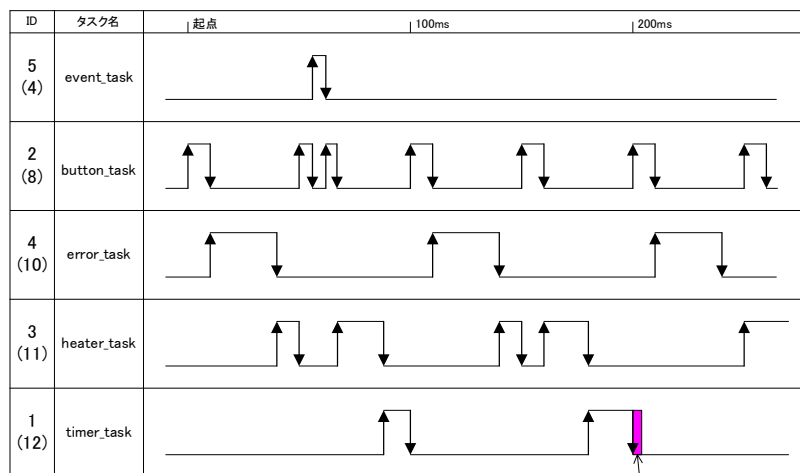
このタスクの周期はharmonic task setとなっています。話題沸騰ポットの本体がパソコン上にありポートアクセスをLANを通していているため、かなりのオーバーヘッドがあります。また、表示タイミングが1秒おきにも関わらずtimer_taskを200ms間隔で動作させているのは、ブザーのオンオフ時間もtimer_taskで同時に制御を行っているためです。実機ならば問題なくH8/3069Fで話題沸騰ポットは制御ができます。

button_taskは50msごとにボタンと蓋のポートを監視し、状態が変化するとevent_taskにイベントフラグを使ってイベントを送ります。button_taskとevent_taskをひとつにすれば、タスクのオーバーヘッドは軽減できますが、POTTNでは割込み化しやすいように、この構成をとっています。heater_taskはモードに合わせてヒータの制御を専門に行うタスクで、error_taskは温度を監視しエラーの検出を専門に行うタスクです。

話題沸騰ポットタスクの実行状態

netDeviceのアクセス時間をタスクに加えた図

■ 各タスクの状態を仮のタイムチャートで表したもの



timer_taskのデッド
ラインミス

2006/11/24

TOPPERSプロジェクト認定

39



キッチンタイマーを起動した場合のタスクのタイミングチャートを示します。このタイミングチャートはnetDeviceのアクセス時間を各タスクに含み、ログタスクやタスクモニタは無いものとし、割込みやタスクスウィッチングのオーバーヘッドを無視したものです。従って、実際の動作とは異なります。仮にタスクの実行周期を以下のように仮定しました。

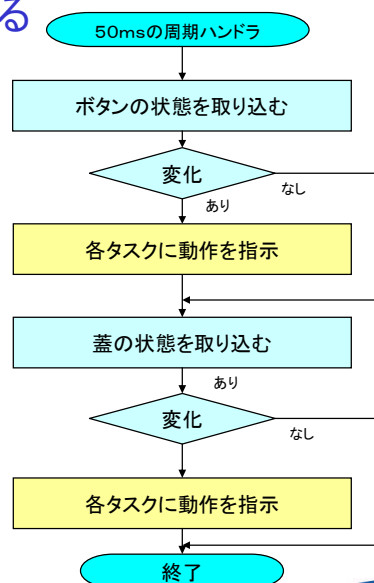
- 1) event_task 5ms
- 2) button_task 10ms
- 3) error_task 30ms
- 4) heater_task 30ms
- 5) timer_task 35ms

実際はLCDの表示の為に、heater_taskとtimer_taskはセマフォを使って排他制御しているため、より複雑な状態遷移が発生していることが予想されます。

理想的な改善

コストとリアルタイム性を考える

- `button_task`と
`event_task`を組み合わせ
て、割込みや周期
ハンドラに置き換
えることにより、実行
効率を改善する
- コストを考えると周期
ハンドラの方が有用



2006/11/24

TOPPERSプロジェクト認定

40



前述の通り、POTTNの話題沸騰ポットは、割込み化しやすい構成となっています。基本的には `button_task` をボタンと蓋の割込みサービスルーチンに書き換えて `event_task` にイベントフラグを用いて状態遷移を伝える形態すれば簡単な対応が可能です。 `button_task` と `event_task` をいっしょにして割込みサービスルーチンにすれば、タスクを2つも削減できるので、もっとプロセッサ負荷を軽減できますが、`netDevice` の制約上この開発環境では対応できません。同様に、 `button_task` と `event_task` をひとつにして、周期ハンドラにすればコストアップせずにプロセッサ負荷の軽減が可能です。

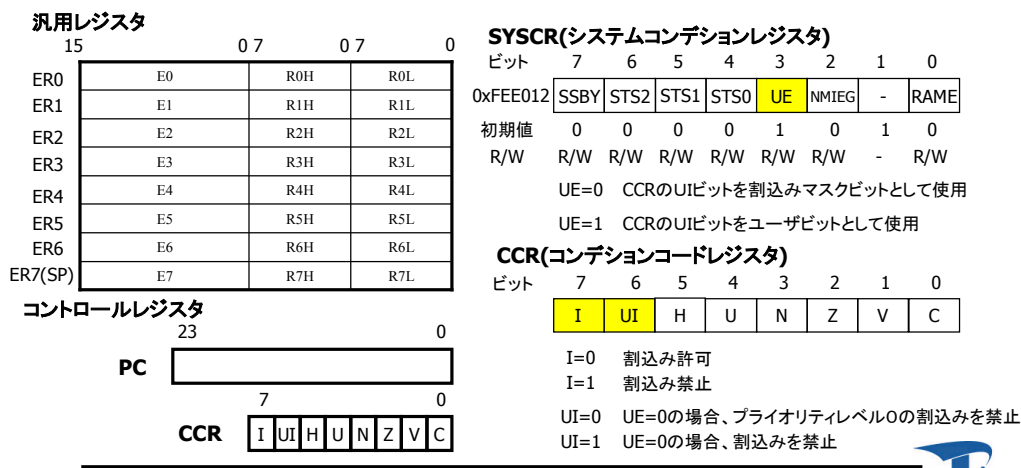
[`netDevice`での制約]

周期ハンドラから `netDevice` 上のポートアクセスは制約上、この開発環境では対応できません。TOPPERS/JSPのWindows版では、割込みや周期ハンドラでもポートのアクセスは可能です。これはWindows版では、本当の割込みや周期ハンドラ内でポートアクセスしているのではなく、どちらもWindowsのthreadを擬似的に割込みや周期ハンドラに見立てて処理を行っている為、実際はWindowsのthreadより、COMを通して、話題沸騰ポットシミュレータにアクセスしています。そのためポートのアクセスが可能なのです。しかし、`netDevice` ではH8の実際の割込みサービスルーチンからTINETの各タスクへのアクセスと、その実行待ちをおこなわなければなりません。これは、非タスクコンテキストの属性上不可能です。この点をご容赦ください。

JSP:H8/3069F版の割込み制御1

H8/3069Fのレジスタ

- H8/300H CPUで割込みの許可、禁止を設定するレジスタはSYSCRとCCRです



2006/11/24

TOPPERSプロジェクト認定

41



TOPPERS/JSPで割込み制御を行う為に、H8/300Hに対してどのような設定を行えば、割込み制御が可能か考えましょう。H8/300Hでは、一般的なCPUの制御に関するレジスタは「汎用レジスタ」と「コントロールレジスタ」の2種類があります。コントロールレジスタ中のIとUIビットが割込みに関連します。Iビットが0のとき割込み許可、1の時割込み禁止となります。UIビットはシステムコンデションレジスタのUEビットの機能設定を行います。UIビットが0の時、UE=0プライオリティレベル0の割込みは禁止となります。UIビットが1の時、UE=0プライオリティレベル0と1の割込みが禁止となります。どちらの場合も、NMIは禁止できません。

JSP:H8/3069F版の割込み制御2

H8-3069Fの優先度設定

- JSP:H8-3069F版の実装では、IPRA,BとCCRのビットとUIビットを使用して割込みと優先度設定を行っている

	ビット	7	6	5	4	3	2	1	0
IPRA	0xFEE018	IREQ0	IREQ1	IREQ2	IREQ4,5	WDT RFSH	ITU0	ITU1	ITU2
	初期値	0	0	0	0	0	0	0	0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	ビット	7	6	5	4	3	2	1	0
	0xFEE019	ITU3	ITU4	DMAC0,1	-	SCI0	SCI1	A/D	-
IPRB	初期値	0	0	0	0	0	0	0	0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

プライオリティレベルは、0の時、該当の割込み要求はプライオリティレベル0(低い)、1の時、該当のプライオリティレベル1(高い)に設定される

起動時にIPRA,BはJSPの初期化でオール0に設定される

SYSCR	CCR		実行可能な割込み
UEビット	Iビット	UIビット	
1	0	—	すべての割込みを許可
	1	—	NMIのみ許可
0	0	—	すべての割込みを許可
	1	0	NMIとプライオリティレベル1の割込みを許可
	1	1	NMIのみ許可

2006/11/24

TOPPERSプロジェクト認定

42



IPRAとIPRBの2つのレジスタはプライオリティレベルの設定を行います。上記のビットが0のときはプライオリティレベル0、ビットが1のとき、プライオリティレベル1です。ビットは割込み要因別に設定ができます。TOPPERS/JSPのH8版の1. 4. 1と1. 4. 2では実装が変更となりました。1. 4. 2版では電源投入後、IPRAとIPRBの各ビットは0に設定されます。UEビットはブート時以外は0に固定されています。CCRレジスタのIとUIビットを使用して割込みの制御と割込み禁止許可の制御を行っています。そのため、割込みは高いと低い2つのプライオリティレベルに割付られます。

JSP:H8/3069F版の割込み制御3

H8-3069Fの優先度定義

- CCRに設定できる値に対応して3つの割込みレベルが設定できる。JSP1.4.2では、このレベルをh8.hインクルードファイルに定義している

IPM_LEVEL0 全ての割込みを許可する

IPM_LEVEL1 NMIとプライオリティレベル1の
割込みを許可する

IPM_LEVEL2 NMIのみ許可する

TOPPERS/JSP1.4.2では3つの割込みレベルを設定している。この設定はh8.hインクルードファイルに定義している。このレベルはCCRレジスタの設定によって制御を行うことができる。

- IPM_LEVEL0 全ての割込みを許可する。
CCRレジスタのIビット=0、UIビット=0の設定
- IPM_LEVEL1 NMIとプライオリティレベル1の割込みを許可する。
CCRレジスタのIビット=1、UIビット=0の設定
- IPM_LEVEL2 NMIのみを許可する。
CCRレジスタのIビット=1、UIビット=1の設定

JSP:H8/3069F版の割込み制御4

割込みベクトル

- H8/300Hは0番地から256バイトの割込みベクタ領域が設定されている
- 割込み要因に対応したベクタアドレスから割込みがスタートする

番号	要因	ベクタアドレス	IPR	備考
0	リセット	0x000000		リセット起動アドレス
7	NMI	0x00001C		
12	IRQ0	0x000030	IPRA7	
13	IRQ1	0x000034	IPRA6	ボタンの割込みとして使用
14	IRQ2	0x000038	IPRA5	蓋の割込みとして使用
15	IRQ3	0x00003C		
16	IRQ4	0x000040	IPRA4	
17	IRQ5	0x000044		IF_ED用割込み
24	IMIA0	0x000060	IPRA2	システムタイマ割込み
25	IMIB0	0x000064		
26	OVI0	0x000068		
56	ERI1	0x0000E0	IPRB2	シリアルエラー割込み
57	RXI1	0x0000E4		シリアル受信割込み
58	TXI1	0x0000E8		シリアル送信割込み
59	TEI1	0x0000EC		

JSPで使用している割込みベクタ

2006/11/24

TOPPERSプロジェクト認定

44

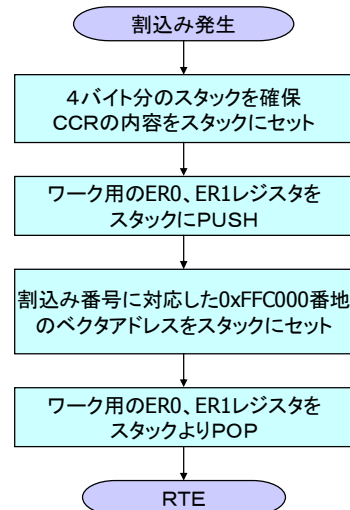


H8/300Hはアドレス0番地から256バイトに割込みベクタが割当てられています。ひとつのベクタが4バイトなので64のベクタを定義できます。64のベクタ中には未使用のものもあります。上記の表はTOPPERS/JSPのH8/3069F版実装で使用している割込みとその他の重要な割込みを抜き出しました。IRQ1とIRQ2の割込みはnetDeviceを用いた仮想的な設定です。割込みを起動させる場合は、ベクタに割込みハンドラの起動アドレスを設定しなければなりません。また、割込みハンドラはアセンブラのRTE命令で終了しなければなりません。

JSP:H8/3069F版の割込み制御5

ROMモニタがベクトルを擬似実行

- ダウンロードプログラムの割込みベクタ領域は0xFFC000番地から設定されている
- ROMモニタは割込み発生時、0xFFC000番地からのベクタに置き換えて割込みハンドラを実行し直している



2006/11/24

TOPPERSプロジェクト認定

45

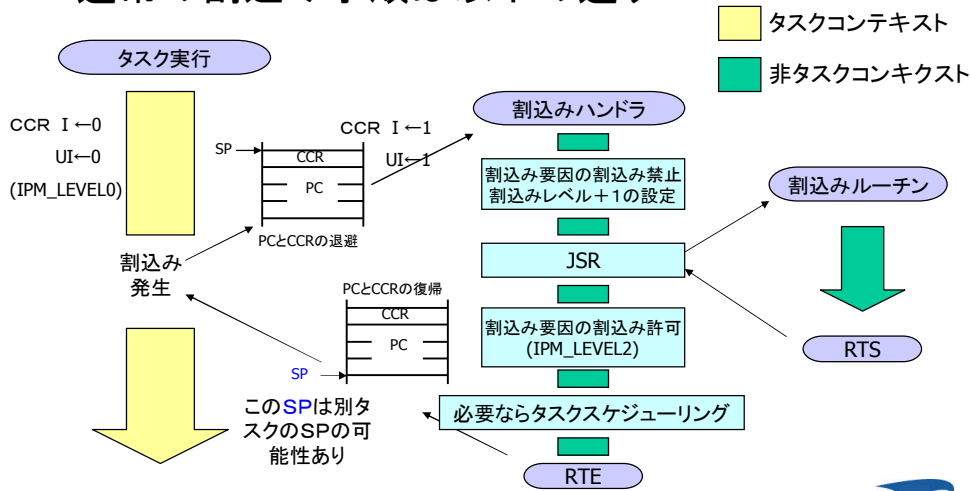


0番地から384KBはフラッシュROMの領域であり、割込みベクタを書き換えるにはフラッシュROMの書き換えが必要になります。ROMモニタでプログラムをダウンロードして実行する場合、割込みベクタは内蔵RAMの0xFFC000番地から256バイトの領域に設定されます。ROMモニタがROMに書かれた状態で、割込みが発生すると発生した割込みベクタに対応した0xFFC000番地からのベクタアドレスを読み出して、その番地に切替えて割込みを実行する機能が組み込まれていますので、ダウンロードした割込みベクタで割込みの検証を行うことができます。

JSP:H8/3069F版の割り込み制御6

割り込みの手順

■ 通常の割り込み手順は以下の通り



2006/11/24

TOPPERSプロジェクト認定

46

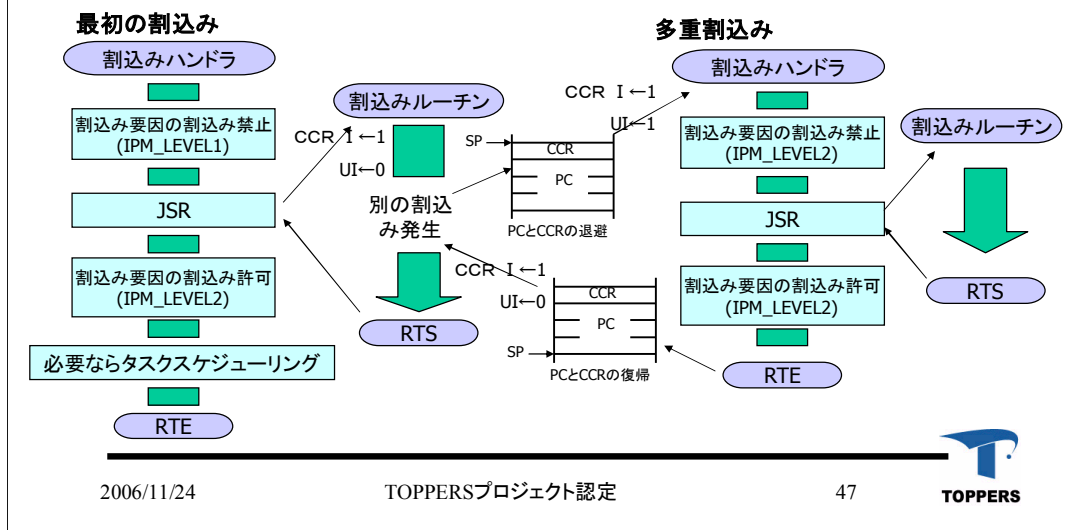


タスクコンテキストでは、CCRのI=0、UI=0 (IPM_LEVEL0) の状態で実行されています。IPRAとIPRBは割り込みのプライオリティレベルに従った設定を行う必要があります。割り込みレベルをIPM_LEVEL0に設定した場合は、IPRA、IPRBの対応ビットを0、IPM_LEVEL1に設定した場合はビット1が設定します。割り込み要因が発生するとSPに現状のCCRレジスタの値と割り込み終了後復帰するPCのアドレスを退避します。その後、Iビットに1、UIビットに1をセットし、割り込みベクタのアドレスから実行します。割り込みベクタには対応の割り込みハンドラルーチンのスタートアドレスがセットされていますので、割り込みハンドラが起動します。その後、発生した割り込みに対応したプライオリティレベル+1の割り込み設定を行います。これにより自分自身の割り込みが多重に発生することがなくなります。この状態で割り込みサービスルーチンを実行します。実行後、IビットとUIビットを1にして割り込み禁止にした後、必要ならタスクスケジューリングを行います。タスクスケジューリングの必要がない場合、RTE命令を実行すると、SPからCCRレジスタとPCを復帰し、タスク実行に戻ります。

JSP:H8/3069F版の多重割込み

最初の割込みがIPM_LEVEL0の場合

- IPM_LEVEL0割込みレベルが通常レベルで発生するとその割込みルーチンはIPM_LEVEL1で実行する



最初に発生した割込みがIPM_LEVEL0の場合、割込みルーチンは割込みレベル+1つまりIPM_LEVEL1で実行されます。このレベルはプライオリティレベル1の割込みを許可しますので、割込みルーチン実行中に、プライオリティレベル1の割込み要因が発生した場合には、この処理中に割込みが多重に発生します。処理については、タスクのスケジューリングを行わない以外は最初の割込みと同等です。

JSP:H8/3069F版の割込みハンドラ

割込みハンドラと割込み許可禁止関数の命名規則

- 割込みハンドラはDEF_INHサービスコールからの命名規則に従って自動生成されます
- 割込み中のレベル設定値は自分で定義しなければなりません
- レベル設定値は割込みレベル+1とする

割込みサービスルーチン名	割込みハンドラ名	(割込みサービスルーチン名)_entry ← ベクタにセット
	割込み設定値	(割込みサービスルーチン名)_intmask

2006/11/24

TOPPERSプロジェクト認定

48



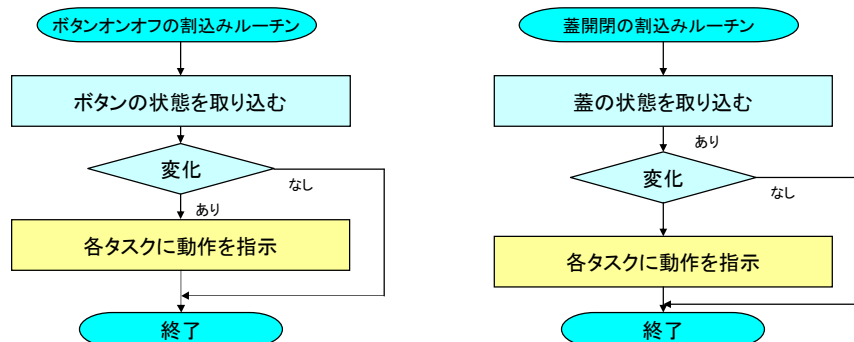
TOPPERS/JSP1.4.2 H8版ではコンフィギュレーションファイルにDEF_INHサービスコールを登録することにより、割込みハンドラを自動生成します。このとき、ハンドラ名は「割込みハンドラの起動番地名=割込みサービスルーチン名」命名規則に従って決められます。例えば、割込みサービスルーチン名がbutton_handlerならば、button_handler_entryとなります。この関数はベクタに自動設定されます。(JSP1.4.1 H8版では、この機能はなく、ベクタの設定を自分で書き換えなければならなかった。)

次に、H8の実装特有ですが、対応の割込みが発生した場合、自分より上位の割込みのみを許可するために、レベル設定値を定義しなければなりません。この設定値は割込みハンドラ内で自動的に設定されます。

割込みを使った負荷改善

2つのタスクを割込みで対応

- ボタンと蓋のオンオフで割込みを発生するようにハードウェアを改良した場合、button_taskを2つの割込みルーチンに置き換えます



2006/11/24

TOPPERSプロジェクト認定

49

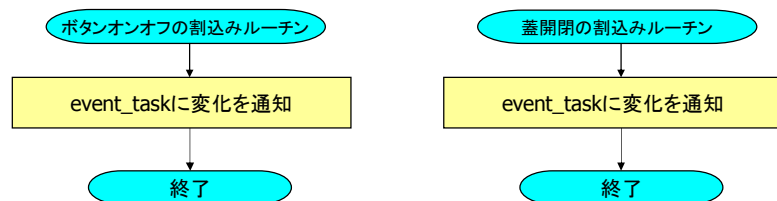


ボタンと蓋のオンオフ割込みをサポートした場合の割込みサービスルーチンの実装例です。割込みの最初でボタンまたは蓋の状態を取り込み、状態に応じてtimer_taskやheater_taskにイベントを発行します。本教材のnetDeviceでは状態の取り込みの部分でポートへのREADが発生しますので適応できません。

netDeviceを用いた改善

非タスクコンテキストでのタスク待ち？

- 実習の話題沸騰ポットのボタンや蓋はnetDevice上に割り当てられています
- netDeviceでは、割り込みルーチンからnetDevice上のポートの読み出しができません
- そのため、button_taskのみ割り込み化してボタンと蓋の状態遷移のみをevent_taskの伝達するように改良します



2006/11/24

TOPPERSプロジェクト認定

50



netDevice上で実装するには、割り込み中でポートのREADができない為、イベントだけをevent_taskに伝達する形が良いでしょう。50ms周期のbutton_taskがなくなりますので、プロセッサ負荷は改善されます。

話題沸騰ポットの 割込みの対応

1. 話題沸騰ポット作成2
2. レビュー
3. リアルタイム性向上の手法
4. タスク負荷の検証と対応
5. 割込みの対応
6. まとめ

2006/11/24

TOPPERSプロジェクト認定

51



教材を用いて、割込みの実装を行きましょう。

ボタンと蓋の割込み

IRQ1(ボタン)、IRQ2(蓋)に設定

- 話題沸騰ポットシミュレータH8-3069F対応版はボタンのオンオフ割込みをIRQ1に、蓋の開閉割込みをIRQ2に発生できるように設定してあります
- これらの割込みをCPUで取り込むには以下の改造が必要となります
 - 1) DEF_INHをコンフィギュレーションファイルに登録
 - 2) 割込みサービスルーチンの作成
 - 3) 割込みレベルとレベル設定値の設定
 - 4) 割込み設定レジスタの初期化を初期化ルーチンに登録

2006/11/24

TOPPERSプロジェクト認定

52



話題沸騰ポットシミュレータで対応しているボタンのオンオフ割込み (IRQ1で対応しています)と蓋の開閉割込み (IRQ2)をシステム対応します。以下の手順で対応してください。

- 1) ボタン割込みと蓋割込みの割込みサービスルーチンを作成します。
- 2) コンフィギュレーションファイル中に、2つの割込みのDEF_INHサービスコールを追加します。
- 3) 割込みベクタとレベル設定値を定義します。
- 4) 割込み設定レジスタの初期化を初期化ルーチン`akih8pot_device.c`の`init_pot()`に登録します。

JSPへの割込み登録

コンフィギュレーションファイルへの割込み登録

- コンフィギュレーションファイル(*.cfg)にDEF_INH サービスコールで割込みハンドラの定義を追加します
- ボタンと蓋のパラメータは以下の通りです

割込み	割込みハンドラ番号	ハンドラ属性	ハンドラ起動番地
ボタン	IRQ_EXT1 (13)	TA_HLNG	割込みルーチン名
蓋	IRQ_EXT2 (14)	TA_HLNG	割込みルーチン名

TA_HLNGは高級言語用のインターフェイスで処理単位を起動することを示すオブジェクト属性

割込ルーチンの作成

JSP:DEF_INHからの呼び出しプログラム

- DEF_INHで定義した割込みルーチンはC言語関数でそのまま記述することができます
- 割込ルーチンのプロトタイプ宣言をpot1.hまたはpot2.hに追加します
- pot1.cまたはpot2.cの直接、割込みルーチンを追加します。

```
void button_int_handler(void)
{
    iset_flg(EVT_FLGID, EVT_BUTTON);
}
```

2006/11/24

TOPPERSプロジェクト認定

54



割込みサービスルーチンをpot1.cまたはpot2.cに直接記述します。

```
/*
 * ボタン割込みハンドラー
 */
void
button_int_handler(void)
{
    iset_flg(EVT_FLGID, EVT_BUTTON);
}
/*
 * カバー割込みハンドラー
 */
void
cover_int_handler(void)
{
    iset_flg(EVT_FLGID, EVT_COVER);
}
```

Event_taskは、EVT_BUTTONやEVT_COVERを取り込み、get_swch()関数でボタンやカバーの状態にて処理を行うように修正を行います。button_taskは不要となります。

[netDeviceでの制約]

C言語で割込み禁止許可関数を記述する場合、akih8port_device.cファイル中に記述をしてください。実際のシステムでは./config/h8中のファイルに記述します。

割込みレベルとレベル設定値の設定 IPM_LEVEL0に設定

- BUTTONとCOVERの割込みをIPM_LEVEL0に設定します
- 従って、レベル設定値はIPM_LEVEL1となります
- akih8pot_device.hに設定を追加します
- 割込み設定はakih8pot_device.cに記述します

```
#define (BUTTON割込みサービスルーチン名)_intmask IPM_LEVEL1
#define (COVER割込みサービスルーチン名)_intmask IPM_LEVEL1
```

2006/11/24

TOPPERSプロジェクト認定

55



akih8pot_device.hに割込みレベルとレベル設定値を定義します。IPRAレジスタは初期化でゼロに設定されていますので設定を変更する必要はないので、定義を行う必要ありません。

akih8pot_device.cのinitial_potに追加します。上記のとおり、デフォルトがレベル0なので、この記述を必ずしも、追加する必要はありません。

```
#define POT_DEF_BITS ((1<<H8IPR_IRQ1_BIT) | (1<<H8IPR_IRQ2_BIT))
```

```
void
```

```
Initial_pot()
```

```
{
```

```
/* 割込み要求時用プライオリティ・レベルを設定する。*/
```

```
sil_wrb_mem((VP)H8IPRA, (sil_reb_mem((VP)H8IPRA) & ~POT_DEF_BITS));
```

```
}
```

割り込み制御レジスタの設定1

IERとISCRの設定

■ H8/300HのIRQ割り込み制御レジスタは以下の3つがあります

ISCR(IRQセンスコントロールレジスタ)

ビット	7	6	5	4	3	2	1	0
0xFEE014	-	-	IRQ5SC	IRQ4SC	IRQ3SC	IRQ2SC	IRQ1SC	IRQ0SC
初期値	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

IRQnSC

- 0: 入力Lowレベル割り込みを発生
1: 入力立下がりエッジで割り込みを発生

IER(IRQイネーブルレジスタ)

ビット	7	6	5	4	3	2	1	0
0xFEE015	-	-	IRQ5E	IRQ4E	IRQ3E	IRQ2E	IRQ1E	IRQ0E
初期値	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

IRQnE

- 0: 割り込み禁止
1: 割り込み許可

ISR(IRQステータスレジスタ)

ビット	7	6	5	4	3	2	1	0
0xFEE016	-	-	IRQ5F	IRQ4F	IRQ3F	IRQ2F	IRQ1F	IRQ0F
初期値	0	0	0	0	0	0	0	0
R/W	-	-	R/W	R/W	R/W	R/W	R/W	R/W

IRQnF

- 割り込み発生で1にセットされる
リード後0を書き込むことによりクリアされる
参照しない場合はクリアを行う必要はない

2006/11/24

TOPPERSプロジェクト認定

56



H8/300Hで割り込みの設定を行うレジスタは以下の3つです。

1) ISCR(IRQセンスコントロールレジスタ)

割り込みの要因の入力をLowレベル(=0)か立下りエッジ(=1)を選択する。

ここでは、立下りエッジを選択してください。

2) IER(IRQイネーブルレジスタ)

割り込み要因ごとの割り込み禁止(=0)、割り込み許可(=1)を設定します。

パワーオンデフォルトは禁止(=0)なので、許可(=1)を設定してください。

3) ISR(IRQステータスレジスタ)

割り込み状態を参照します。割り込みが発生するとリセットを行うまで、その状態が残ります。

参照する必要はないので、このレジスタは設定の必要はありません。

(話題沸騰ポットシミュレータでは、対応していないので、常にゼロが返ります)

割込み制御レジスタの設定2

IERとISCRの設定の注意事項

- IREQ1とIREQ2はnetDeviceの仮想の割込みです、実際のハードウェア上のIERに割込み設定を行うと、IREQ1、2信号の状態で誤動作が発生します
- IERとISCRの設定は、`akih8pot_device.c`中にSILの記述を使って設定してください

SILの記述例

```
sil_wrb_mem((VP)H8IER, sil_reb_mem((VP) H8IER)|(1<<H8IER_IRQ1E_BIT));
```

↑
config/h8/akih8_3069f/h8_3069f.hに記述されている

注意:これらの宣言は自作せず、システムで設定されているものを使用するようにしましょう。(3048,3069でアドレスは異なるが名称は同じ)

2006/11/24

TOPPERSプロジェクト認定

57



本実習ではIERとISCRポートの設定は`akih8pot_device.c`のソースを書き換えて行ってください。実際のシステムでは./config/h8以下のソースファイルで設定を行っています。

```
#define POT_INT_BITS ((1<<H8IER_IRQ1E_BIT) | (1<<H8IER_IRQ2E_BIT))
void
Initial_pot()
{

    sil_wrb_mem((VP)H8ISCR, sil_reb_mem((VP)H8ISCR) | POT_INT_BITS);
    sil_wrb_mem((VP)H8IER, sil_reb_mem((VP)H8IER) | POT_INT_BITS);

}
```

[netDeviceの制約]

本セミナーの実装は実機とnetDeviceとを組み合わせる形で行っています。デバイスドライバのソース中は以下の対応があります。

- 1) `akih8pot_device.c`: netDevice用のドライバを集めたもの
- 2) `akih8lcd_device.c`: 実機の拡張ボード用のドライバを集めたもの

この2つのソースファイルの違いは、`akih8pot_device.c`は`<monsil.h>`をインクルードし、`akih8lcd_device.c`は`<sil.h>`をインクルードしている点です。`sil.h`はTOPPERS/JSP中のシリアルドライバでも使われていますが、SILインターフェイスでハードウェアにアクセスする為の設定を記述したインクルードファイルです。これに対し、`<monsil.h>`はSILインターフェイスでタスクモニタを介して、ハードウェアにアクセスする為の記述が書かれています。タスクモニタはnetDeviceがサーバと接続した状態ならば、ハードウェアにアクセスせず、サーバに対してアクセスを行います。そのため、IERとISCRのアクセスの記述自体を、`<monsil.h>`インクルードファイル下の`akih8pot_device.c`に記述する必要があります。

SILはどのような、利点があるのでしょうか？もし、話題沸騰ポットの試作機をハードウェアで誰かが作成した場合、`sil.h`の内容を`monsil.h`に移し変えれば、ソースを修正せず、実機でみなさんが作った話題沸騰ポット試作機のプログラムを動作させることができます。

SILについては初級実装セミナーの最後に説明を行っていますのでそちらを参照ください。

最後の注意

割込み中に、netDeviceのポートアクセスは不可

- 作業に入る前に、netDevice上のボタンと蓋のポートは割込みルーチンからは読出しができません
- 読み出した場合は、netDeviceを参照せず、直接H8/3069F拡張ボード上のポートデータ(ほとんどが1)を読み出しますので、正常に参照できません
- **ご注意ください**

まとめ

1. 話題沸騰ポット作成2
2. レビュー
3. リアルタイム性向上の手法
4. タスク負荷の検証と対応
5. 割込みの対応
6. まとめ

組込み開発環境について

デバッグ方法としっかりした単体テスト

- デバッグ方法
 - ▼syslog文やprintf文でログを表示させる
LinuxやWindows上のクロス環境で、シミュレーション開発や単体テストのprintf文をうまく利用する
 - ▼remoteデバッガによるデバッグ
 - ▼JTAG-ICEを使ったデバッグ
- デバイスドライバのデバッグ
 - ▼volatile宣言の使用
 - ▼SIL等の標準インターフェイスの使用
シミュレーションやテストにも有効

リアルタイム・スケジューリング理論

RMSとharmonic task set

- DMS/RMS (Deadline Monotonic Scheduling)
周期が短いタスクほど、高い優先度をつける
- Harmonic task set
それぞれのタスクの周期の比率が倍数関係
プロセッサの使用率を有効に引き出せる
- 周期ハンドラや割込みを有効利用
タスク以上の優先度、リアルタイムの向上
- 理論をリアルタイムシステム設計に有効活用する
には大きなハードルがある
 開発を通しての経験が大きなノウハウとなる

TOPPERS/JSPでの標準入出力

newlibの使用

- 本教材では、printf文等の標準入出力機能をタスクモニタにてサポートしています(タスクモニタの入出力はstdio.hに従っている)
- このプログラムはmonitorディレクトリの以下のソースが実行しています

stdio.h printf.c sprintf.c scan.c sscanf.c

- TOPPERS/JSPが使用しているnewlibの標準入出力用の関数が、newlibのソースを一部書き換えればそのまま使用が可能です、monitor/README.txtにその対応方法を追加しましたので興味のある方はnewlib版教材にもチャレンジしてみてください

教材の作成

参考資料

- 教材の作成に協力してくださった方にお礼を申し上げます

(株)アドバンスド・データ・コントロールズ 森田 浩

- 参考資料

「豊橋市中小企業技術者研修

リアルタイムOSを活用した組込みソフトウェア設計」

名古屋大学

高田広章

「4th Open SESSAME テキスト」

組込みソフトウェア管理者・技術者育成研究会

「H8/300Hシリーズ プログラミングマニュアル」

ルネサステクノロジ

「H8/3069R F-ZTATハードウェアマニュアル」

ルネサステクノロジ