第 8 回 RTOS に求めるもの

こんにちは。 まず前々回の第 6 回「セマフォによる排他制御」の課題であった、バイナリセマフォしか提供していない RTOS のもとでタスクレベルでカウンティングセマフォを実現するプログラムの例です。

```
SEMAPHORE mutex = 1; /* control access to 'wait_c' */
SEMAPHORE wait_sem = 0; /*controls access to the counting sem */
volatile int wait_c = initial_sem_val; /* initial value */

cs_p() {
for(;;) {
    p(mutex);
    wait_c--;
    if (wait_c < 0) {
      v(mutex);
      p(wait_sem);
    }
    else {
      v(mutex);
      break;
    }
  }
}

cs_v() {
  p(mutex);
  wai_c++;
  if (wait_c <= 0) {
    v(wait_sem);
  }
  v(mutex);
}
```

どうでしょうか？
今回この解説をするスペースがないため、みなさん考えてみてください。

cs_p が semTake 相当の操作、cs_v が semGive 相当、p がバイナリセマフォしかサポートしてない環境での saemTake 相当、v が同じく semGive 相当です。バイナセマフォであっても複数のタスクが待てることが前提です。
実は私自身もまだ厳密には検証していません。もしもこのコードを使われる場合は自己責任でお願いします。
それでは、本来の話題にもどります。

かつてはアプリケーションの種類だけ RTOS の種類があると言われた時代もあります。現在はそれよりは統合されてきていますが、それでも多くの RTOS があり、RTOS によってさまざまな特徴があります。
VxWorks の特徴を改めていくつか説明します。
VxWorks はタスクやセマフォ、メモリプール、メッセージバッファなど操作の対象は動的に生成します。
VxWorks は最初に 1 つのユーザータスクを呼び出してくれます。通常はそのタスクでタスクやセマフォなど実行に必要なものを生成しますが、そうしなければならないわけではなく、実行中にタスクがほかのタスクや必要なセマフォ、リングバッファなどを必要に応じて生成していくという構成も可能です。

VxWorks の提供するシステムコールは、よく使うものは単純で使いやすくなっています。たとえばイベントフラグの機能は持っていません。その代りセマフォはバイナリセマフォ、カウンティングセマフォ、ミューテ

ックスの 3 種類のセマフォを提供しています。前にも書いたように、バイナリセマフォというものが存在しない RTOS もあります。そのような RTOS においてバイナリセマフォはあくまでもセマフォの値の最大値が 1 のカウンティングセマフォという扱いです。しかし VxWorks ではバイナリセマフォは排他制御にもっともよく使われるということで、アセンブラで書かれているなどの手段により速度が最適化されています。
カウンティングセマフォに関しては C 言語で作られていることにより、移植性が重視されています。
どちらも C 言語で作られていたとしても、多くの CPU においては他の演算の副作用で z(ゼロ)フラグがセット/リセットされることを利用した機械語の命令が生成されるため、0 か 0 以外かの判定処理のほうが、0 以外の数との比較よりも命令数も少なく、実行も高速になります。両者を区別したほうがより速度メモリとも最適化できるはずです。
それ以外に VxWorks は優先度継承などの機能を備えた排他制御専用のセマフォであるミューテックスも最初から標準で提供しています。

会社に勤めていた当時、開発していた製品の RTOS を mtos から VxWorks に置き換えるかどうか悩んでいた時、とりあえず VxWorks が機能的に使い物になるかどうかを確認するため、製品のソフトウェアの主なものだけでも VxWorks 対応に修正する必要がありました。しかしまだ VxWorks に移行すると決定したわけではないので、当時 mtos 上で動いているアプリケーションのソースコードを修正しなくとも VxWorks 上で動作させるための mtos ラッパー関数を作成して VxWorks の評価を行なったことがあります。アプリケーションは mtos の SVC を呼び出しているつもりでも、それは単なる変換関数になっていて、実は VxWorks のシステムコールを呼び出す関数です。mtos と VxWorks の提供する機能がほとんど同じであれば、何の苦労もありませんが、AND 待ちや OR 待ちを行うイベントフラグと非同期 I/O そして引数付きの起動要求のキューイングの実現は少し苦労しました。しかし RTOS が直接サポートするのに比べれば実行効率は劣りますが、イベントフラグはセマフォを使用して、非同期 I/O はタスクとドライバの間に完了を待つためのタスクをもう一つはさむことにより、引数のキューイングはリストを使うことにより実現しました。

VxWorks では最もよく使うのは単純な待ちなので、基本的な同期排他制御の手段としてはセマフォしか提供していません。そして AND や OR などの複雑な待ちを実現したい場合はセマフォとセレクトのシステムコールを組み合わせて使います。ここにも VxWorks のよく使う機能をシンプルに提供するという思想が見えます。
割り込みに対しては、割り込み処理 ISR を登録するのに intVecSet()というシステムコールを使用すると、メモリプールからメモリブロックを持ってきてそこに登録する ISR に対するラッパー（登録する ISR を呼び出す前に実行する前処理と、登録した ISR 実行後の後処理、そしてスケジューラの呼び出しなど）を生成し、その前処理のアドレスを割り込みベクターテーブルに登録します。この仕組みによって割り込みを抜ける時にスケジューラに戻るか、割り込まれた場所に戻るかの処理もユーザーが作らなくても実現してくれますが、intVecSet()というシステムコールを用いるとスケジューラを経由せず直接割り込まれた場所に戻ります。
なお、厳密には CPU のアーキテクチャによっては上記のようにならない場合もありますので注意してください。
他の RTOS には割り込み処理から呼び出せるサービスコールと呼び出せないサービスコールを区別しているものもあります。しかし VxWoks ではそのような区別はありません。待ちに入る可能性のあるシステムコールを割り込み処理から呼び出した場合は、システムコール内で自分がタスクから呼び出されたのかどうかを判定して、タスク環境以外から呼び出された場合は割り込みから呼び出されたシステムコールの実行を専門に行うための優先度を最高にした（システム）タスクに処理を引き継ぎそのタスクで処理するようになっています。

このように VxWorks は見えない部分でもさまざまな工夫がなされています。

RTOS に対する要求
ここで RTOS に関する要求などをまとめておきます。
RTOS に求める上位の抽象的な要求、RTOS を使用する目的には次のようなものがあります。

・リアルタイムシステムを構築するための OS
リアルタイム性を備えたシステムを構築しやすい
ただしリアルタイム OS を使えばリアルタイムシステムが構築できるわけではない
きちんとしたリアルタイム設計は必要

・システムの時間的複雑度を下げる
論理的な処理と時間的な流れ、制約を分離する事が出来る
それによりリアルタイム性に起因する不具合の発生を防止してシステムの品質を上げる
静的解析ツールが関数の複雑度を下げることによりソフトウェアの品質を上げるのに類似

・システムの論理的複雑度を下げる
まとまった処理をタスクとして他の処理と論理的に分離する事が可能

・組込みシステムにおけるフレームワークとしての役割
割り込み処理、ドライバ、タスク処理の組込み方が標準的に与えられていること
アプリケーションから利用可能なネットワーク、Web 機能や各種ライブラリが豊富

上記を実現するためにリアルタイム OS に必要な具体的機能は次のようになります。

- マルチタスク(スレッド）機能
- 優先順位に基づく、プリエンプティブ(preemptive)スケジューリング
- 割り込み、例外制御機能
- タスク間通信機能、タスク間の同期、排他制御機能
- 時間管理機能
- メモリ管理機能
- ドライバ作成のサポート
- 処理時間の上限が予測可能であれば理想的

最後にあげた処理時間の最悪値の保証は最近の CPU がキャッシュ、パイプラインを備えていることや DMAC（ダイレクトメモリコントローラ）などバスマスタになれるデバイスの存在などにより、実現は難しいですが、理想としてあげておきます。

上記の機能すべてを適切な時間以内で処理できる RTOS はおそらく存在しないでしょう。
しかし作成する組込みシステムを開発する場合、RTOS で提供していない機能が必要であれば自分で開発するという手段もあります。
組込みシステムの場合の場合、ハードウェアも含めて開発する場合が多く、その可能性は高くなります。

例えばどんなタイミングでいきなり電源を切られてもファイルが壊れないようなメモリファイルシステムも、ソフトウェアだけで実現するのは難しくとも、バックアップされた SRAM と組み合わせることができるのであれば、容易に実現できるかもしれません。

まとめ
この連載の最後になりましたが、汎用オペレーティングシステムの書籍として最も有名なものの一つである“Operating System Concept” 7th Edition, Abraham Silberschatz, Peter Bear Galvin, Greg Gagne （邦訳 オペレーティングシステムの概念 共立出版）においてもリアルタイムシステムにおけるリアルタイムオペレーティングシステムの代表として VxWorks が取り上げられています。リアルタイムシステムやリアルタイムオペレーティングシステムは同書においても最初の方の版では取り上げられていませんでした。
それだけリアルタイムシステムや組込みシステムが世の中に知られ始めて、避けて通れなくなってきたということです。
私も大学のオペレーティングシステムの授業においても、リアルタイム OS にかける時間をどんどん増やしている状況です。
今後も現実世界における VxWorks をはじめとするリアルタイム OS の役割は重要であり続けると思います。

私の「組込み講座・基礎編」は今回で終了になります。長い間お付き合いいただきありがとうございました。
また来月より「応用編」として連載を始めますので、引き続きよろしくお願いいたします。

第 8 回おわり

コラム

入出力処理 I/O ですが、VxWorks の場合はエラーなどの場合を除くと I/O からタスクに制御が戻ってくるのは I/O 処理が完了した場合だけですが、mtos では、I/O の種類には VxWorks のように処理が完了してから戻ってくるタイプーmtos ではこれを完了リターンと呼んでいましたー以外にタスクからはドライバに対して処理の依頼だけを行って、タスク自体はそのまま処理を続けるーmtos ではコーリングリターンと呼んでいましたーという種類がありました。
30 年以上前の mtos にあって VxWorks にその機能がないのかと思われるかも知れませんが、コーリングリターンを使いこなすためにはタスクの方で並行性を強く認識しないといけないため、アプリケーション作成が難しくなるという問題もあり、VxWorks では採用しなかったのでしょう。
どうしてもコーリングリターンを実現したければ、本文に書いたように VxWorks ではタスクからタスクの動的生成も自由にできるため、タスクを生成してそのタスクに I/O の完了を待たせればよいだけです。