待ち状態のない μITRON カーネル仕様案

RTOS 自動車応用技術委員会

編: 高田 広章 (豊橋技術科学大学 情報工学系／ ITRON 専門委員会)

31-Mar-1998

このドキュメントでは、RTOS 自動車応用技術委員会で検討を行った待ち状態のない μITRON カーネル仕様案と、その実装方法について述べる。本ドキュメントに記述する仕様を μITRON4.0 仕様の 1 つのプロファイルにすることを、RTOS 自動車応用技術委員会から提案する。本ドキュメントに紹介する実装方法は、提案した仕様が容易に実装できることを検証することを目的しており、あくまでも 1 つの実装方法の例にすぎない。

1 待ち状態をなくすメリット

待ち状態をなくす動機は、メモリ容量 (特に RAM 容量) を減らすことと、タスク切り替えの速度を向上させることである。

待ち状態 (強制待ち状態も含む) がなく、レディキューの回転 (`rot_rdq`) もない場合には、タスクが実行できなくなるのはプリエンプトされる場合のみで、すべてのタスクで 1 つのスタックエリアを共有できる。また、スタックを切り替える必要がないことから、タスク切り替え速度の向上も期待できる。

ただし、必要なスタックエリアのサイズは、同時に動作する可能性があるタスクの集合に含まれるタスクが使うスタックサイズの和の最大値となるため、1 つのスタックエリアを共有できることが、ただちに RAM 容量の削減につながるとは限らない。

このような待ち状態のないカーネルが、リアルタイムカーネルと呼べるかどうか (リアルタイムカーネルの要件を満たしているか) は議論の分かれるところであるが、これだけの機能でも十分なアプリケーションも多いと思われることから、本格的なリアルタイムカーネルへの入門的な位置付けで、このような仕様を μITRON 仕様のサブセットとして定義することは、用語やシステムコール名称の統一の点からは意味があるものと考えられる。

2 複数の仕様案が考えられるポイント

本ドキュメントで提案する仕様には、以下の点で大きな選択肢が残っている。細かな選択肢は他にも残っている。

初期優先度あたりのタスク数

同一初期優先度のタスクを 1 つに限るか複数許すかの選択肢がある。1 つに限る方法は、レディキューが単純になるという利点があるのに対して、複数許した場合には、同一優先度のタスクが同時に実行されることはないことから、さらなるスタックエリアの削減につながる。

そこで以下では、この両者の方法について検討する。2 つの方法で違ってくる箇所には、(a) 同一初期優先度のタスクを 1 つに限る場合、(b) 同一の初期優先度のタスクを複数許す場合、のいずれについての記述であるかを明記する。

ここで、実行時優先度の動的変更 (`chg_pri`) を許した場合、(a) の仕様でも、同一の実行時優先度のタスクは複数あっても良いものとする。これは、同一の実行時優先度のタスクを複数許しても実装がほとんど複雑にならないのに対して、同一の実行時優先度を許した方が必要な優先度段数を減らせるためである。

タスクの ID と初期優先度

(a) の場合には、タスクの ID をタスクの初期優先度と一致させることで、タスクの初期優先度を記憶する領域 (ROM) を節約する方法もある。

3 カーネル仕様案

ここでは、μITRON3.0 仕様のレベル S をベースに検討する。明示的に書かない限り、μITRON3.0 仕様のレベル E の機能はサポートしない。また、μITRON 仕様のサブセットとすることから、エラーコードについては、μITRON 仕様と同様とする。

サポートしない機能

待ち状態がないため、`rel_wai`, `slp_tsk`, `wup_tsk`, `can_wup`, `sus_tsk`, `rsm_tsk`, `dly_tsk` の各システムコールはサポートしない。また上に述べた通り、`rot_rdq` もサポートしない。

待ち状態があることが前提になっている同期・通信オブジェクトは使えない。具体的には、イベントフラグとセマフォは仕様から外す。メイルボックスは、待ちを使わない形で導入する。

カーネルオブジェクト（タスク，メイルボックスなど）の生成は、カーネル構成時に静的に行うものとし、そのためのシステムコール（cre_XXX）はサポートしない。また、割込みハンドラの定義（def_XXX）も同様とする。 μITRON4.0 仕様にあわせて、これらのカーネル構成の記述方法を標準化する。

タスクの状態

待ち状態（強制待ち状態も含む）がなく、タスクの動的な生成も無いことから、タスクの状態は、 RUN 状態、 READY 状態、 DORMANT 状態の 3 つのみである。

カーネルの実装上は、 READY 状態を、 sta_tsk で起動されたがまだ実行は開始していない状態と、実行開始後により優先度の高いタスクにプリエンプトされた状態とに分けて考えた方がわかりやすい。

各システムコールの仕様

以下、それぞれのシステムコール（ないしは機能）について、必要性とその理由を述べる。

sta_tsk（タスクの起動）

必須の機能。すでに起動されているタスクを再度 sta_tsk した場合に、起動要求をキューイングする機能を持たせる。

プログラムの移植性を高めるためには、キューイングの最大回数を定めることが望ましい。キューイングの最大回数を増やすと、その分、 TCB（タスク制御ブロック）のサイズが大きくなる。最低限としては、 1 回のキューイングをサポートすれば大抵の応用には耐えられるものと思われる。

μITRON3.0 仕様にある起動されるタスクに起動コードを渡す機能も有益ではあるが、タスク起動のキューイング機能と起動コードを渡す機能の 2 つを同時に実現すると、メモリの使用量が増えるので望ましくない。起動コードは、 sta_tsk 発行前に共有変数に置いておく方法もあり、その機能を sta_tsk に持たせる必然性はないと考えられるので、サポートしない（この方針は μITRON4.0 仕様案と整合している）。

ext_tsk（タスクの終了）

サポートしない。トップレベルの関数から戻ることでタスクが終了する。

タスクを終了させる機能が必要なのは言うまでもないが、タスク内の任意の場所で ext_tsk を発行してタスクを終了できるようにすると、そのタスクが使っているスタックの底や、タスク起動前のレジスタを保存することが必要になる。それに対して、トップレベルの関数から戻ることでタスクが終了することとし、 ext_tsk は用意しないとすると、メモリ容量の節約が図れる。

ter_tsk（タスクの強制終了）

サポートしない。

アプリケーションサイドから見た場合に、タスクの強制終了は過渡的な状態で止まる可能性があるので危険である（多くの場合に、終了する必要があるかを時々チェックする方法で十分）。実装上も、 ter_tsk をサポートするためには、タスクが使っているスタックの底や、タスク起動前のレジスタを保存することが必要になる。

chg_pri（タスクの優先度の変更）

自タスクの優先度を変更するための chg_pri のみサポートする。ただし、初期優先度より低い優先度に変更することはできないという制限を設ける。

Priority ceiling protocol（厳密には highest locker protocol）を実現するためには、自タスクの優先度が変更できることが必要である。それに対して、他のタスクに対する chg_pri は、スタックが 1 本で済まなくなるのでサポートしない。そのため、タスク ID を渡す必要がなくなり、 chg_pri のパラメータが 1 つですむことになるが、上位互換性のために自タスクを明示的に指定させる方法が妥当と思われる。

自タスクを初期優先度より低い優先度に変更することを許すと、自タスクが実行開始される前に実行されていたタスクへ切り替える必要がある場合が発生するため、初期優先度より低い優先度への変更は許さない（変更しようとした場合にはエラーとする）。

dis_dsp，ena_dsp（タスクディスパッチの禁止 / 許可）

必要な機能。 dis_dsp は、自タスクを最高優先度に chg_pri することと等価であることから、実装も容易。

get_tid

サポートする。

必要性については疑問があるが、実装上の負担はなく、使わない時はリンクされないので、あえて除く理由がないものと思われる。

メイルボックス

メイルボックス機能はサポートするが、待ちを伴う rcv_msg はサポートしない。つまり、メッセージを送信する snd_msg（μITRON4.0 仕様では snd_mbx と改名される予定）と、受信する prcv_msg（同 prcv_mbx）だけをサポートする。

ただし、 μITRON4.0 仕様案では、メイルボックスの実装方法はリンクリストによるものに一本化され、リングバッファによる方法はスタンダードプロファイルではサポートされない。自動車制御目的には、リングバッファによる方法を採用する

方が適当と思われ、その場合、システムコールの名称は変更する必要があるだろう。

いずれにしても、実装上はメイルボックス機能は単なるライブラリ関数として実現できることから、両方提供するとしてもオーバヘッドはない。また、その他のバリエーションの提供も容易である。

メッセージ送信によりタスクを起動する機能が欲しいが、そのような新しい機能を導入すると μITRON のサブセットでなくなってしまうため、メッセージを送って、タスクを起動するという複合コールを、ユーザサイドで用意する方法が妥当と考えられる。

`loc_cpu`, `unl_cpu`, `ret_int` (割込み管理)

必要な機能。実装も容易。

`dis_int`, `ena_int`, `chg_iXX`, `ref_iXX` (割込み管理)

実装依存の機能として残す。

`set_tim`, `get_tim` (時間管理)

サポートする。ただし、システムクロックのビット数は実装依存とする。

必要性については疑問があるが、実装上の負担はなく、使わない時はリンクされないので、あえて除く理由がないものと思われる。

周期起動ハンドラ

サポートする。

μITRON3.0 仕様では、周期起動ハンドラはレベル E になっているが、μITRON4.0 仕様案ではスタンダードプロファイルに入っている。ただし、μITRON3.0 仕様の周期起動ハンドラの仕様にはやや問題があるので、μITRON4.0 仕様における仕様を採用する。

アラームハンドラ

サポートしない。

マイクロ秒単位の時間指定ができるなら有用と思われるが、そうでないなら必要性は低い。

仕様のまとめ

以上で、いくつかの選択肢を除いて、カーネルの仕様としてはほぼ固まっている。機能的には、OSEK/VDX OS 仕様の、BCC2 に相当するものとなる。

4 実装方法 (例)

ここでは、以上で定義した仕様の 1 つの実装方法を示す。これとは別の実装方法が考えられることは言うまでもない。

データ構造

システム全体で共有する情報として、次のようなものが必要と考えられる。

- レディーキュー
- 実行中のタスクの TCB へのポインタ (またはタスク ID)
- 実行中のタスクの優先度
- ディスパッチ禁止フラグ

レディキューの構造は、(a) の場合と (b) の場合で大きく異なる。(b) の場合は、通常のリアルタイムカーネルと同様の構造になるのに対して、(a) の場合は各初期優先度を持ったタスクが READY 状態かどうかをあらわすビットマップだけで十分である。実行中のタスクの優先度の最上位ビットをディスパッチ禁止フラグに使うことで、ディスパッチ禁止フラグをチェックするオーバヘッドをなくすことができる。

各タスクの TCB は、RAM 上に置く必要がある情報と、ROM 内に置けばよい情報に分けられる。RAM 上に置く必要がある情報としては、次のものが考えれる。

- タスクの状態 (1 bit; RUN/READY 状態か、DORMANT 状態か)
- `sta_tsk` のキューイングフラグ (1 bit)
- レディーキューにつなぐためのエリア ((b) の場合のみ)

また、ROM 内に必要な各タスク毎の情報として、次のものがある。

- タスクの開始番地
- タスクの初期優先度 ((a) で初期優先度をタスク ID と一致させた場合には不要)
- タスクの拡張情報 (下の説明参照)

タスクの拡張情報は、タスクの起動時にタスクのメインルーチンにパラメータとして渡される。同一のルーチンを使って複数のタスクを動かす場合には有用であるが、そうでない場合には必要性が低い。ROM 容量の制約が厳しい場合には、タスクのメインルーチンにタスク ID を渡す方法も考えられる。タスク ID は `get_tid` で取り出せるため、それで代用させる方法も考えられる。

その他に、メイルボックスの管理ブロック、周期起動ハンドラの管理ブロックなどが必要となる。

各システムコールの実装方法

`sta_tsk`

起動対象のタスクが RUN 状態または READY 状態であれば、TCB 中の `sta_tsk` のキューイングフラグをセットする。すでにキューイングフラグがセットされていた場合にはエラーとする。

以下、起動対象のタスクが DORMANT 状態であった場合を考える。まず、起動対象タスクを READY 状態に移行させる (TCB 中のタスク状態を更新し、レディーキューに入れる)。起動対象のタスクの初期優先度が、現在実行中のタスクの優先度以下であった場合、またはディスパッチ禁止フラグ

がセットされている場合はこれで完了。起動対象タスクの方が優先度が高い場合、実行中のタスクの TCB へのポインタ、優先度をスタックに保存し、それらの変数をそれぞれ、起動対象タスクの TCB へのポインタ、起動対象タスクの初期優先度に設定する。また、起動対象タスクで壊されるレジスタもスタックに保存する (スタックへの保存順序は要検討)。以上の処理の後、タスクのメインルーチンに渡すパラメータを設定し、起動対象タスクの開始番地をサブルーチンコールする。

タスクの終了

タスクのトップレベルの関数から戻った場合、終了したタスクが実行される前に実行されていたタスクの優先度 (スタックの先頭付近に記憶されている) と、レディーキュー中の最高優先度タスクの優先度を比較する。

前者の方が優先度が高いか同じ場合、終了したタスクが実行される前に実行されていたタスクの実行を再開することになる。まず、スタックに保存されているレジスタ、再開するタスクの TCB へのポインタ、再開するタスクの優先度を、実行中のタスクの TCB へのポインタ、優先度を示す変数に戻し、やはりスタック上に保存されているタスクの実行再開番地へと戻る。

後者の方が優先度が高い場合、レディーキュー中の最高優先度タスクの実行を開始する。具体的には、実行中のタスクの TCB へのポインタ、優先度を、それぞれ、実行を開始するタスクの TCB へのポインタ、そのタスクの初期優先度に設定し、タスクのメインルーチンに渡すパラメータを設定した後、そのタスクの開始番地をサブルーチンコールする。

chg_pri

実行中のタスク優先度を更新する。その結果、レディーキュー中の最高優先度タスクの優先度が、実行中のタスク優先度よりも大きくなる場合、レディーキュー中の最高優先度タスクの実行を開始する。実行を開始する方法については sta_tsk と同様。そうでない場合には、実行中のタスクの実行をそのまま継続する。

dis_dsp, ena_dsp

dip_dsp はディスパッチ禁止フラグをセットし、ena_dsp はそれをリセットする。ディスパッチ禁止フラグをリセットした場合には、実行中のタスクの優先度と、レディーキュー中の最高優先度タスクの優先度を比較する。後者の方が優先度が高い場合、レディーキュー中の最高優先度タスクの実行を開始する。実行を開始する方法については、 sta_tsk と同様。そうでない場合には、実行中のタスクの実行をそのまま継続する。

get_tid

実行中のタスクの TCB へのポインタからタスク ID を求めて返す (タスク ID が記憶されている場合には、そのまま返せばよい)。

メイルボックス

タスク切り替えを伴わないので、単なるライブラリ関数である。

loc_cpu, unl_cpu

loc_cpu は割込み禁止状態に設定し、 unl_cpu はそれを解除する。 μITRON3.0 仕様では、ディスパッチ禁止状態を unl_cpu で解除することも許されているので、 unl_cpu でディスパッチ禁止フラグをリセットする必要がある。ディスパッチ禁止フラグをリセットした場合の処理については ena_dsp と同様。

割込みハンドラの起動と ret_int

割込みハンドラの起動は、プロセッサの機能に任せる。割込みハンドラとタスクが同一のスタックを共有することが可能。

ret_int 発行時に、タスクの終了時と同様のタスク切り替え処理が必要になる。すなわち、実行中のタスクの優先度と、レディーキュー中の最高優先度タスクの優先度を比較し、後者が優先度が高い場合には、レディーキュー中の最高優先度タスクの実行を開始する。実行を開始する方法については、 sta_tsk と同様だが、割込みハンドラの中での処理なので、処理内容は少し変わるものと思われる。

dis_int, ena_int, chg_iXX, ref_iXX

実装依存であるが、素直に実装可能と思われる。

set_tim, get_tim

実装依存であるが、素直に実装可能と思われる。

周期起動ハンドラ

タイマに対する割込みハンドラを作り込むだけなので、素直に実装可能。

周期起動ハンドラの生成は静的に行うことから、生成される周期起動ハンドラの周期から、割込みハンドラ (の周期起動ハンドラ呼出し部) を静的に生成して効率を上げる方法も考えられる。

5 自動車制御用リアルタイム OS 仕様 (案) との関連

「自動車制御用リアルタイム OS 仕様 (案)」では、スタックエリアを節約するための機構として、同一優先度タスクで 1 つのスタックエリアを共有し、そのようなタスクは待ち状態に入れなくする方法を提案している。待ち状態に入る必要があるタスクには、他のタスクと異なる優先度を与えばよい。

本ドキュメントで提案する仕様では、タスクが同一優先度でなくても、待ち状態に入らなくすることでス

タックエリアが 1つですむ。このアイデアを適用すると、待ち状態に入れないすべてのタスクは 1つのスタックエリアを共有し、待ち状態に入れるタスクには個々にスタックエリアを割り付ける方法が考えられる。

待ち状態に入れるタスクに他のタスクと同一の優先度を割り付けられることから、この仕様の方が、原案よりも柔軟と考えられる。逆に、各タスクの属性として、待ちに入れるか否かを指定する必要がある。

「待ち状態のないカーネル仕様案」と「自動車制御用リアルタイム OS 仕様(案)」を 1つのカーネル仕様の中で定義する場合には、この考え方の方が整合性がよいものと考えられる。

なおこの方法は、OSEK/VDX OS 仕様の basic task と extended task への分類に対応し、奇抜な方法ではない。

6 まとめ

本ドキュメントに定義した仕様を、μITRON4.0 仕様の 1つのプロファイルとすることを、ITRON プロジェクトに対して、RTOS 自動車応用技術委員会から提案する。

以上