6M-7 可変構造型並列計算機のオペレーティング・システム －プロセス管理－

恒冨邦彦　福澤祐二　福田晃　村上和彰　富田眞治
(九州大学大学院総合理工学研究科)

1. はじめに

我々は，種々の並列処理に柔軟に対処できるマルチプロセッサ「可変構造型並列計算機」[1] の開発を進めている．本稿では，本計算機を共有メモリ型マルチプロセッサとみなして設計した並列オペレーティング・システム (OS)[2] のプロセス管理について述べる．

2. タスクとスレッド

従来のOSが実行の単位としているプロセスを，資源の割り当て対象であるタスクと制御の流れであるスレッドに分割し，個別に制御を行う．同一のタスク内で複数のスレッドが命令コード，静的データを共有することにより，同一仮想空間上で並列実行を行いながら情報のやり取りを迅速に行うことが可能となる．タスクおよびスレッドは動的に生成され，親子関係を持つ．

3. ユーザインターフェース

3.1 スレッド

ユーザがスレッドを自由に操作できる様に，以下の操作を提供する．

thread_fork は，ユーザの設定するプライオリティーを持つスレッドを1個生成する．生成されたスレッドは，引数によって指定された関数を処理する．また引数によりプロセッサ番号を指定すればそのプロセッサで実行される．従ってユーザ独自のプロセッサ割り当て方法が実現可能となり，その割り当てによる処理能力の比較が手軽に行えるようになる．

thread_exit は，自スレッドを終了する．

thread_wait は，引数で指定した子スレッド，または子スレッド全てが終了するまで実行を中断する．子スレッド終了後，再び実行を再開する．

この他に，スレッド同士が同期をとりたいときにはOSによって提供されたP／V命令を用いることができる．

3.2 タスク

基本的なシステムコールはUNIXと互換性を持たせている．従ってタスク操作のシステムコールはUNIXのプロセス操作のシステムコールとほぼ同じである．しかし，UNIXは，マルチスレッドの概念については定義されていないので，スレッドとプロセス関係のシステムコールとの関連を決定する必要がある．forkは，タスクを1つ生成する．このスレッド生成に関する問題は後述する．スレッドの生成と同じように，引数においてプロセッサ番号が指定されていれば，そのプロセッサを獲得して，その上で実行させることができる．execは，UNIXのものと同じであり，指定されたプログラムを獲得して現タスクの仮想空間上に重ね書きし，スレッドは新しいプログラムの先頭から実行を開始する．waitは子タスクの終了まで処理を中断する．exitは，自タスク内の全スレッドを強制終了する．

forkとwaitの実行時に複数のスレッドが存在する場合には，それぞれ2通りの選択が考えられる．forkの場合は

(1) forkを発するスレッドのみを子タスク内に生成する．

(2) タスク内の全てのスレッドを子タスク内に生成する．

という選択肢が考えられる．forkシステムコールを発したスレッドのみが，他のスレッドと異なる仮想空間上で実行することを望んでいると考えると (1) が適当である．また，forkを既存のプロセスの複製の生成だと考えると，(2) の方がよい．forkは新しく仮想空間を生成して，その上にexecを用い，新しく実行ファイルをロードし実行させるためにしばしば用いられる．この場合，方法 (2) は複数のスレッド生成がオーバヘッドとなる．また，方法 (2) はforkシステムコールを発するスレッド数が増えると，生成されるタスク数が膨大な数となり，この制御が困難となる．従って，我々は (1) を採用した．

waitの場合には

(1) waitを実行するスレッドのみを子タスクの終了まで中断する．

(2) タスク内の全てのスレッドを子タスクの終了まで中断する．

という選択肢が考えられる．(1) は，子タスクの終了を待つのは，そのスレッド自身のみであるという考えに基づく．また，waitをタスク同士の同期ルーチンと見れば，(2) が適

An Operating System for the Reconfigurable Parallel-Processor -Process Management-
Kunihiko TSUNEDOMI, Yuji FUKUZAWA, Akira FUKUDA, Kazuaki MURAKAMI and Shinji TOMITA
Kyushu University

当と思われる．我々は，waitを実行したスレッドが，特に子タスクと依存関係が深いと考えられることから（1）を採用した．

4. スケジューリング

4.1 プロセッサの割り当て対象

異なるタスク内の複数のスレッドを実行しようとする場合，そのスレッドの内どれから実行するか，どの様にプロセッサを割り当てるかを決定する必要がある．その際，以下の2通りの選択肢が考えられる．

（1）タスクを認識せず，スレッド単位にプロセッサを割り当てる．

（2）タスク対応にプロセッサを割り当て，そのタスク内のスレッドを実行する．タスク内のスレッド全てが終了または中断しない限りプロセッサはタスク内のスレッドの処理を行う．

我々は，タスクの切り替えを極力避けるために（2）を採用した．タスクの切り替えが頻繁に発生すると，仮想空間の切り替えを行う必要があるので，そのオーバヘッドが大きくなり，また，TLBのヒット率が低下するのに伴い処理速度も低下するからである．

4.2 プロセッサの割り当て手順

プロセッサの割り当て手順を図1，図2に示す．スレッドが生成される（図1）と，まず，プロセッサ指定のあるものとないものに区別して当該スレッドがレディキューに登録される．次にプロセッサ管理テーブルを検索し，そのスレッドを実行すべきプロセッサを捜す．指定されたプロセッサがidle状態ならば，そのプロセッサに割り込みをかけ，スレッドの処理を行わせる．busy状態であれば，プロセッサ割り当て待ちキューに要求内容を登録する．また，プロセッサを指定していなければ，idle状態のプロセッサを探し，以下同じ処理を行う．

スレッドが終了するとき（図2）には，上記の逆の操作を行う．スレッド終了時に当該プロセッサはレディキューを調べ，実行可能なスレッドが存在すれば実行する．存在しなければプロセッサ割り当て待ちキューを検索し，実行可能なタスクが存在すれば，そのタスクのスレッドを実行する．存在しなければ，プロセッサ管理テーブルに自プロセッサをidleとして登録する．

レディキューもプロセッサ割当て待ちキューも，基本的にはFIFO制御を行っているが，プロセッサ番号を指定してある場合には，キューの先頭にあるスレッドでなくても，それを検索しているプロセッサが該当するプロセッサであれば，そのスレッドを実行に移す．これにより，プロセッサの指定による実行も可能となる．

5. おわりに

以上，「可変構造型並列計算機」のOSのプロセス管理について述べた．今後，タスクとスレッドの操作に最低限必要と思われる関数を実現していく予定である．スレッドによる低粒度な並列処理環境を実現することにより，可変構造型並列計算機の性能を最大限引き出せると考える．

参考文献

〔1〕村上ほか：可変構造型並列計算機のシステム・アーキテクチャ，情報処理学会「コンピュータアーキテクチャ」シンポジウム論文集，Vol.88，No.3，pp.165－174（1988年）

〔2〕福田ほか：可変構造型並列計算機の並列／分散オペレーティング・システム，情報処理学会研究報告，89－OS－43（1989年）

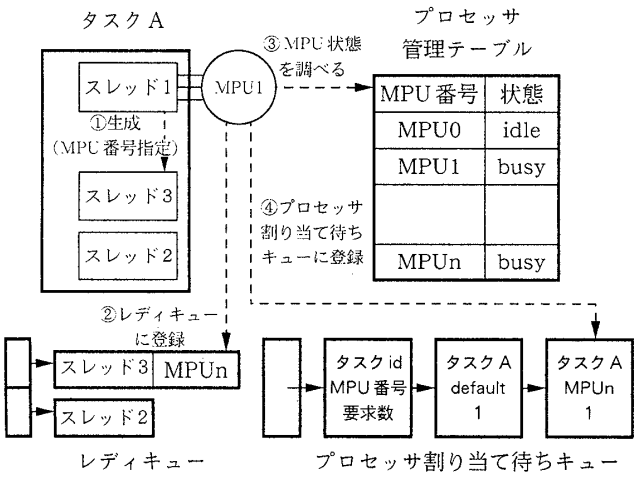

図1　スレッドの生成

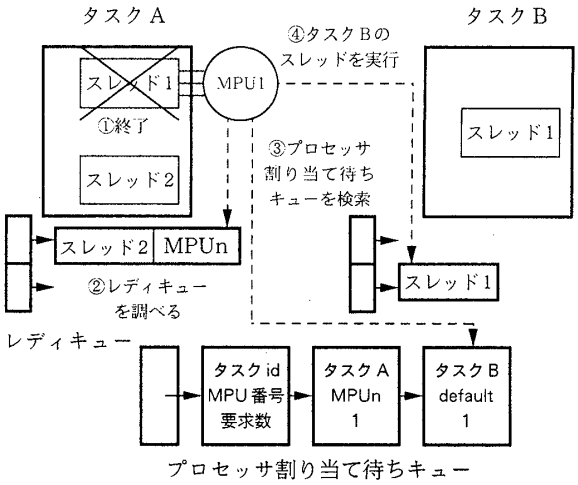

図2　スレッドの終了

MPU：MicroProcessor Unit