TOP－1オペレーティングシステム
(4) スレッド機構

3P-7

吉永秀志，山内長承，穂積元一
日本アイ・ビー・エム株式会社　東京基礎研究所

1．はじめに

カーネルにより制御されるスレッド機構をAIX*を基としたTOP-1 OSのプロセス機構上に実現し、それによってひとつのプロセス内に複数のスレッドを作ることを可能にした。スレッドはプロセスの構成要素の内でプセッサ資源を利用する単位と考えられている。しかしその実態はそれを実現しているOSにより異なる。本稿ではTOP-1 OS上で実現したスレッドの目的、機構を説明する。

2．スレッド機構導入の目的

スレッド機構は、(1)TOP-1共有メモリ型マルチプロセッサアーキテクチャを有効に利用するプロセスモデルを実現するため、(2)現在のUNIX**プロセスに比べて、プロセス資源の共有をさらにすすめ、協調動作するプログラムを作りやすいプロセスモデルを実現するために、導入された。

TOP-1 OSにおいては、マルチプロセッサ上で並列に動作し、共有メモリ上に高速のIPC機構をもつスレッドを提供することことをめざした。さらにスレッドがアドレス空間、ファイルなどを共有することにより、ユーザに、通信しながら並列に動作するプログラムや、大量のデータを共有して並列処理するプログラムなどを作りやすい環境を提供することをめざした。

3．スレッド機構

3．1．プロセスモデル

スレッドはプロセスを構成する要素の中で、プロセッサ資源を利用する単位であることを述べた。TOP-1 OSにおけるプロセスの特徴は、ひとつのプロセス内にスレッドを複数持てることである。それらのスレッドはプロセッサ以外の資源、つまり、アドレス空間、ファイル、プロセス間通信チャネルを共有する。さらにTOP-1 OSではそれぞれのスレッドはシステム内で固有の識別子を与えられ、ディスパッチの対象であり、シグナル送受信の対象であり、システムコールを利用することができる。プロセス間には従来どおり親子関係が存在するが、スレッド間には親子関係は存在しない。

アドレス空間はプロセスに対してひとつしか存在しないので、スレッドのスタックおよび固有のデータは、それぞれのスレッドで異なったアドレスにとらなくてはならない。

3．2．初期化スレッド

スレッドは原則としてすべて対等な存在であるが、プロセスの初期化、終了、プロセスに対するシグナル処理に関しては初期化スレッドと呼ばれる特定のスレッドが処理することにしている。初期化スレッドはプロセス生成時に作られ、プロセスの終了時まで動作する。言い換えると初期化スレッドの終了時点がプロセスの終了の時点となる。初期化スレッドのみを持つプロセス、言い換えるとスレッドが一つしか存在しないプロセスは従来のUNIXプロセスと変りない。

3．3．シグナル処理

スレッドを導入したことにより、スレッドごとにシグナル処理ができるようにシグナル機構を変更した。

まずスレッドをシグナル受信の単位とした。それぞれのスレッドは従来どおり、到着したシグナル処理に関して、(1)何もしない、(2)システム定義のハンドラの起動、(3)ユーザ定義のハンドラ起動のいずれかを選択できる。

また従来のシグナルをプロセスに対するものとスレッドに対するものに分類した。従来のUNIXではシグナルはプロセスにハードウェア例外、システムや他プロセスからの制御、キーボードからのジョブ制御などの信号の通知に使われてきた。マルチスレッドの環境では、たとえば演算オーバーフローなどのハードウェア例外は、その原因となったスレッドのみに通知すれば十分である。またキーボードからジョブ制御のキーやコマンドが入力された時にはプロセス内のすべてのスレッドが応じる必要がある。このようにスレッド機構の下では、シグナルごとに何に作用すべきかを検討する必要がある。

3．4．ユーザーインターフェース

マルチスレッドのプロセスをユーザーが実現するためにスレッドを制御するシステムコールを用意した。表1にこれらを示す。またスレッドの導入により従来のプロセスを制御するためのシステムコール、fork, exec, exit, wait, kill, signalを再定義した。表2にこれらを示す。

th_createはスレッドを生成し実行させる。&fnはス

TOP-1 Operating System -- (4) Thread Mechanism
Hideshi YOSHINAGA, Nagatsugu YAMANOUCHI, and Motokazu HOZUMI
IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.

レッドに実行させる同じプログラム内の関数へのポインタをとる。つまりスレッドのインストラクションポインタ値となる。stkはスタックセグメント中でこのスレッドが使用するスタックエリアへのポインタをとり、スタックポインタ値となる。&rtはユーザ空間中のリターンバリューエリアへのポインタをとる。これはスレッドが何らかの理由で終了した場合に、終了状態をシステムにより通知してもらう時に使用する。

th_yieldはスレッド切り替えをシステムに要求する。これは共有しているデータセグメント上での高速なスレッド間通信を実現するために用意された。データセグメント上に共有変数をつくり、ユーザモードのビジーウエイトロック機構とこのシステムコールを組合せることにより、ロックに成功するときは高速に完了し、失敗してもCPUを放棄することによりCPUを無駄に使用しないプロセス間通信が実現できる。

th_pinvalidはプロテクションされていないアドレス空間内でアクセス禁止エリアを設定する。スレッドはスタックセグメント中にそれぞれのスタックエリアをとるので、あるスレッドのスタックが他のスレッドのスタックエリアを破壊する危険がある。このためそれぞれのエリアの間にアクセス禁止エリアを設定することによりこれを防ぐ。laddrはあるアドレスをさし、このアドレスを含むページがアクセス禁止になる。

4．スレッド機構をもつオペレーティングシステム

スレッドまたはライトウエイト・プロセスを備えたシステムとして、Mach[1], ULTRIX[2], Dynix, Sprite, Silicon Graphics System V, SunOS, OS/2*[3]などが報告されている。

Dynix及びSpriteではアドレス空間の一部を自動的に共有するプロセスモデルを提供している。Silicon Graphics System Vはプロセスがどの資源を共有するかを指定する機能をもっている。これらのシステムでは資源共有に関しては改良されているものの、それぞれがプロセス資源をもつために、システム資源消費については従来と変わらない。

SunOSにおけるスレッドは一つのプロセスの中にコルーチンで実現されている。このためスレッド切り替えは高速であるが、マルチプロセッサ上で並列実行させるためにはマルチプロセス上でのコルーチンにしなくてはならない。しかしその時はプロセス間の資源共有が問題になる。

Mach, ULTRIX, OS/2 およびTOP-1 OSにおけるスレッドはひとつのプロセス内にマルチスレッドを実現しているためプロセス資源を共有しながら並列実行が可能になっている。そしてコルーチンの機構もそのうえに構築できる。TOP-1 OSではMachと違って新しいカーネルのうえにUNIXカーネルを載せるという方式はとらず、前述の目的を満たすためにAIXカーネル内にスレッド機能のみを実現した。

5．まとめ

実現したスレッド機構のうえではひとつのプロセスの中に複数のスレッドを作ることができ、マルチプロセッサ上で並列実行ができる。さらにこのようなスレッド機構をユーザが利用するためのシステムコールを用意した。これらを用いて協調動作するプログラムが効率よく実現できる。

スレッドは資源を共有しているので、生成や終了がプロセスに比べてより速く完了する。スレッドの生成はプロセスの場合と比べて約1/9で完了し、スレッドの終了はプロセスに比べて約1/3で完了することが確かめられた。

【参考文献】

[1] M.Accetta et al., "Mach: A New Kernel Foundation For UNIX Development," Summer USENIX Conference, 1986

[2] D.S.Conde et al., "ULTRIX Threads," Summer USENIX Conference, 1989

[3] Ed Iacobucci, "OS/2 Programmer's Guide," Osborne McGraw-Hill

System call	Function
tid = th_create (&fn,stk,&rt)	Forks a thread
th_gettid()	Gets its thread id
th_kill(tid,sig)	Sends a signal to a thread
th_check(tid)	Checks if the thread exits
th_yield()	Requests a thread switch
th_pinvalid(laddr)	Invalidates a page
th_pvalid(laddr)	Validates back a page

表1．スレッド制御のためのシステムコール

System call	Function
fork()	Forks a single-threaded process
execl()	Loads a program in a single-threaded process
exit(st)	Exits a process (first thread) Exits a thread (other threads)
wait(&st)	Waits for termination of a child process
kill(pid,sig)	Sends a signal to a process
signal (sig,&fn)	Defines a signal handler for each signal in every thread

表2．再定義したシステムコール

* AIX and OS/2 are trademarks of IBM Corporation.

** Developed and Licensed by AT&T.