可変構造型並列計算機のオペレーティング・システム －シグナル－

6M-8

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

現在我々は，128台のプロセッシング・エレメント（PE）を相互結合網で接続した可変構造型並列計算機[1]を開発中である．本稿では，本オペレーティング・システム[2]のシグナルの処理について述べる．

2. オペレーティング・システム（OS）概要

UNIX†などの従来のOS[3]では，シングル・プロセッサを対象にした設計であった．このためプロセス内部の並列性に対して，有効な並列処理のための手段，方法を持っていない．本計算機上のOSは，ソフトウェア資産の観点からUNIXとシステムコール・レベルで互換性を維持しており，128台のPEを有効に活用するために，プロセス内部の並列性に着目し，1つのプロセス内部を複数のスレッドとして並列に実行できる機能をシステムコール（またはライブラリ関数）として備えている．このOS上で，様々な並列プログラムを実行し並列処理に関するハードウェア，ソフトウェア更には，並列処理システム全般に渡る評価を行なうことを目的としている．

3. シグナル

UNIXにおけるシグナルは，ハードウェアにおける割込みの概念をソフトウェアに持ち込んだもので，システムコールのレベルで割込みを取り扱えるようにしたものである．割込みハンドラに相当するルーチンがシグナル・ハンドラと呼ばれる関数である．シグナルは発生原因によっていくつか種類（4.3BSDでは32種類）があり，これらのシグナルの発生源は主にシステムであるが，ユーザ・プロセスによって同期のために用いられることもある．シグナルはセットジャンプ，ロングジャンプと組み合わせて用いられることが多く，複数のスレッドを持つプロセスの実行ではインプリメンテーションによって4.に示す問題点が生じる．

4. マルチ・スレッドとシグナルに関する問題点

シグナルに関するシステムコールが従来のままで，プロセス内部を複数のスレッドで実行する際，どのスレッドがシグナルを受け付けるのか明確に定義されていない場合には，シグナル・ハンドラの処理内容によってプロセス内部のスレッドが次に述べるような非決定的な振る舞いをする事がある．

図1のようにセットジャンプ，ロングジャンプをシグナルと共に使用すると次の2つの場合が生じる．

```
#include    <signal.h>
#include    <setjmp.h>
      jmp_buf    env ;
main ( )
       {
       int    sig_handler ( ) ;
       if ( setjmp ( env )  ==  0 )
              {
              ...
              signal ( SIGINT , sig_handler ) ;
              ...
              }
       else
              {
              /*    シグナルの後処理    */
              }
       ...
       /*    複数のスレッドを用いたプログラム    */
       }

sig_handler ( sig , code , scp )
       int    sig , code ;
       struct    sigcontext    *scp ;
       {
       ...
       longjmp ( env , 1 ) ;
       }
```

図1 シグナルとセットジャンプを用いた例

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

† UNIXはAT&Tベル研の登録商標です．

1）セットジャンプを行ってスタックを保存したスレッドが，シグナルを受け付けてロングジャンプを行いスタックを回復する場合．

2）セットジャンプを行ったスレッドと無関係のスレッドが，シグナルを受け付けてロングジャンプを行う場合．

このようにスレッドとシグナルの関係が明確でない場合には，シグナルを受け取るプロセス側からは，どちらの状況が発生するかを予測することができない．

実際に問題となるのは2）の場合で，シグナル・ハンドラを実行したスレッドはシグナルを受け付ける以前の処理を再開することが出来なくなる．もし，実行されない部分が，プログラムの処理上実行される必要がある場合，処理が行き詰まったり，正しい結果が得られない状況となる．

また，シグナル・ハンドラの実行をスレッドに固定的に割り当てた場合には，シグナル・ハンドラ内ではスレッドIDに関連する処理に制限が加わり，プログラミングの自由度を低下させる．

以上のような問題点はそのインプリメンテーションに起因するものであり，OSの設計時に解決しなければならない．我々は，シグナルに関して5．に示すように仕様の拡張を行い，以上のような問題点を解決した．

5．本OSにおけるシグナルの拡張仕様

UNIXのシグナルの内で，一般にユーザがシグナル・ハンドラを設定できないのはSIGKILL（9）のみである．SIGKILLは受け付けたプロセスを必ず終了させるため，マルチ・スレッドで処理を行っている場合，受け付けたスレッドに関係なくプロセスは終了する．この他のシグナルに関しては，シグナル・ハンドラが定義されていなければSIGKILLと同様にプロセスを終了するが，シグナル・ハンドラを定義する場合は，どのスレッドがそのハンドラを実行するかを設定する．関数new_signalは図2のように定義する．

従来のUNIXとの互換性のため，スレッドに関する指定がない場合はTHREAD_SELFが指定されたものとする．

シグナル関連のシステムコールに関してはBSD系のUNIXを基本にしており，sigblockなどのシグナルの受信側のシステムコールは、前述のnew_signal関数と同様にスレッドID，THREAD_SELF，THREAD_ALL，THREAD_ANY，THREAD_EXTのどれかをパラメータに加える．

また，killなどのシグナル送信のためのシステム・コールに関しては，スレッドID，THREAD_ALL，THREAD_ANY，THREAD_EXTのどれかをパラメータに加える．また，シグナルの処理に関してもBSD系のUNIXと互換性が取れるように実現する．

6．おわりに

以上のようなシグナルの仕様は，スケジューリングやOSのリアル・タイム性などとの関係において，必ずしも最適であるとは断言できない．しかし，並列処理の面では必要不可欠な機能を導入しており，十分使用に耐え得ると考えている．

今後，現在構築中のOSにこれらの機能を組み込み，並列処理OSとして実用化する予定である．

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）．

［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）．

［3］Sun Microsystems，Inc．：UNIX Interface Reference Manual，Sun Microsystems（1986）．

```
new_signal (sig, sig_handler, thread_id) ;
        ／＊thread_idで指定されたスレッドがシグナル・ハンドラを実行する＊／

new_signal (sig, sig_handler, THREAD_SELF) ;
        ／＊この関数を実行したスレッドがシグナル・ハンドラを実行する＊／

new_signal (sig, sig_handler, THREAD_ALL) ;
        ／＊当該プロセスに属するスレッド全てがシグナル・ハンドラを実行する＊／

new_signal (sig, sig_handler, THREAD_ANY) ;
        ／＊当該プロセスに属するスレッドの内でシグナルを受け取った全てのスレッドがシグナル・ハンドラを実行する＊／

new_signal (sig, sig_handler, THREAD_EXT) ;
        ／＊新たに当該プロセスに属するスレッドを生成し，そのスレッドがシグナル・ハンドラを実行する＊／
```

図2 シグナルの拡張仕様