5N-4 UNIXネットワークにおける並列実行環境の構築

大島龍博, 徳崎宣久, Bernady Apduhan, 有田五次郎

九州工業大学

1. はじめに

我々の研究開発システムであるHYPHEN B-16は, ＦＩＦＯキューを同期手段とする待ちなし並列プログラム(Self synchronizing Parallel Program,以後ＳＰＰと記述する)[1]をソフトウェアモデルとするメモリ共有型並列処理システムである. 研究室では, 実際のHYPHEN B-16システムの設計製作[6]と並行して, 関数型言語ＰＩＬＡＦ[3][4]や単一UNIX上の疑似並列実行開発環境を構築してきた. しかし単一UNIX上での疑似並列環境では, 並列タスクの粒度や数, あるいは速度といった点で制限が多く, クリティカルなタイミングのシミュレーションといった点でも問題が残っていた.

ところが, 近年複数のUNIXマシンをネットワークで結合したシステムが利用できるようになり, 疎結合並列計算機の環境を構築できる環境になった.

本稿では, このUNIXネットワーク上にＦＩＦＯキューを同期手段とするメモリ共有型並列処理環境を構築することを考える.

2. ＵＮＩＸにおける並列実行単位

一般的なUNIXマシンにおいて, マルチプロセスを実現するのは, 言語Ｃにおいて fork & exec システムコールを発行し子プロセスを生成すれば良く, 非常に容易である. しかしこの方法だけでは, 異なるノード上にプロセスを生成できないため, 通信を利用した並列実行単位の生成／消去メカニズムが必用である.

また fork システムコールによって, 子プロセスに引き継がれる変数領域は, 親プロセスの変数のコピーであり, その実体ではないため, メモリ共有の環境を得るためには, マルチノード間アクセスメカニズムが必要になる.

そのうえ並列実行を効率よく行うためには, 高速な同期手段を用意しなければならない.

並列実行単位に関しては, Mackオペレーティングシステム上のスレッド[7]が有名であるが, HYPHEN B-16では, ＳＰＰセグメントという並列実行単位を提案している[6].

ＳＰＰセグメントは, スレッドにＰＢ(Parallel･Branch), ＥＸＴ(EXchange Task)による同期メカニズムを組み込んだものと考えると, ほぼ１対１に対応する.

このＰＢ, ＥＸＴによって記述されたプログラムはＳＰＰと呼ばれ, 同期のための合流点を持っていない.

図１において, 親プロセスから生成された子プロセスは, プロセスの環境を引継ぎ, 次のexecシステムコールにそなえる. 通常の利用では, 親プロセスの全てをコピーする必要はないためvforkシステムコールが用いられる. HYPHENではタイムスライスをしないため新しく生成された子プロセスは, 親プロセスがEXTシステムコールをするまで実行が遅延される.

この時点では, 並列に実行されるＳＰＰプロセスも, その並列度もいっさいわからない. これがわかるのは, 次のexecシステムコールによって, 実際にプロセスをオーバーレイロードしたときである.

したがってひとつのプロセスを並列に分散させるためには, プロセス自身で分散ロードを行うことが必要である.

HYPHENでは, この分散ロードの問題に対しては, 他のプロセッサに対する最初のPB操作をきっかけにしてこれを行う. すなわち, PBによってＦＩＦＯにキューイングされたプロセスＩＤをEXTによって取り出したとき, 新しいプロセスＩＤであれば, 分散ＯＳがＳＰＰセグメントをロードする. またすでに存在するなら制御を移すだけである.

プロセスの消去に関しては, 比較的簡単で, 親プロセスのwaitシステムコールの完了時にゾンビ状態の子プロセスを消去すればよい.

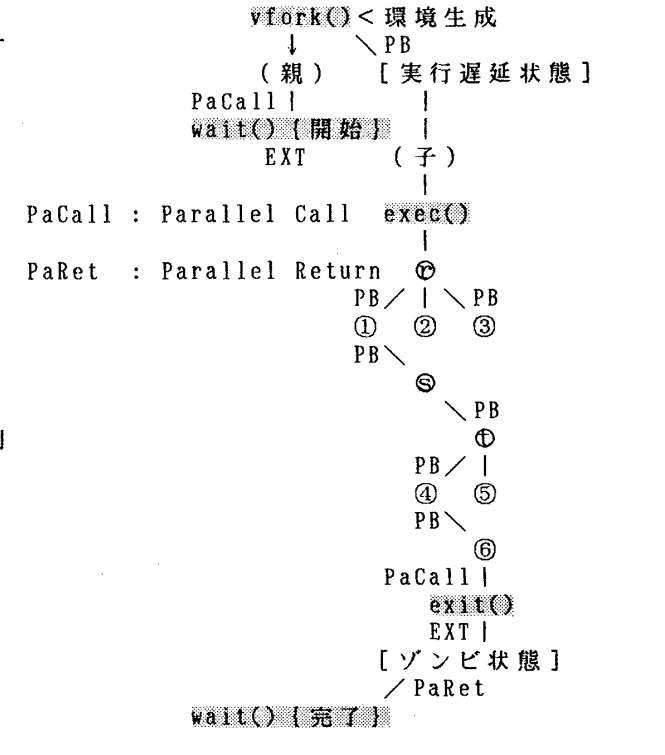

図１ UNIXとSPPにおけるプロセス生成／消去

The Development of a Parallel Processing Environment in a UNIX Network

Tatsuhiro OHSHIMA, Nobuhisa TOKUZAKI, Bernady Apduhan,Itsujiro ARITA

Department of Computer Science, Faculty of Engineering,Kyushu Institute of Technology

3. 並列（分散）プログラムの構築

図２に並列プログラムの構築法を示す．ここで重要なのは，異なるシステムに分散させる単位であるＳＰＰセグメントの記述である．
（ここで並列プログラムは，ＳＰＰモデルによるものを考えるが，より一般的なＰＰモデルによる記述をするにはカウンタセマフォを用いればよい）

通常のプログラムは，ただひとつのメインルーチンを持つが，分散させて実行させるために，ＳＰＰセグメント単位でメインルーチンを持たなくてはならない．

また，ＳＰＰセグメントの構成要素であるＳＰＰタスクは，任意の順序で動的に実行されるため，任意の関数を動的に実行できなくてはならない．

さらには，ＳＰＰセグメントを異なるプロセスとして実行するため，ＳＰＰプロセスＩＤとノード上でのunixプロセスＩＤの対応がつかなくてはならない．

任意の関数に制御を移すメカニズムは，下記に示すようにsetjmp()とlongjmp()を使ったタスクゲートと呼ぶ制御構造で実現できる．

これはタスクゲートで登録した関数番号を引き数としてlongjmp()を呼ぶと，任意の関数を実行できるため，main()中のタスクゲートで登録し，EXTのなかで，longjmp()を実行してやればよいことになる．

```
タスクゲート
        main()
          {
  登録    → if ( setjmp(task[task_0]))
                 {
          実行 →    タスク０本体
                 };
  登録    → if ( setjmp(task[task_1]))
                 {
          実行 →   タスク１本体
                 };

              ext();
            };

        EXT()
          {
            while(ＦＩＦＯ読出不可）；
            ＦＩＦＯ読み出し；
            longjmp(task_No);
          }
```

メインプログラムは，上記で示した通りタスク登録のためのローカルメインになるが，このプログラム群には２章で述べた，プロセスディスクリプタが必要である．このプロセスディスクリプタには，分散ＯＳが，ＳＰＰプロセスを管理するのに必要な情報が記述されている．

またＰＢ，ＥＸＴ等の手続きは，ＦＩＦＯに対する通信プログラムが中心である．共有変数にたいするアクセスプログラムもリモートプロシージャコールを用いたものである．

このような，タスクゲートを含むローカルメインや，ＰＢ，ＥＸＴ等の通信モジュール，共有変数アクセスのための手続きをすべてユーザが記述するのはたいへんな労力を要する．

したがって，ユーザの記述ためにＰＣ（Parallel flow Control language)というものを考え，ユーザにはこのレベルでのプログラムをしてもらう，これは一種のメタ言語であり，このＰＣの存在によりunix上での言語を使って並列プログラムを構築することが可能になる．

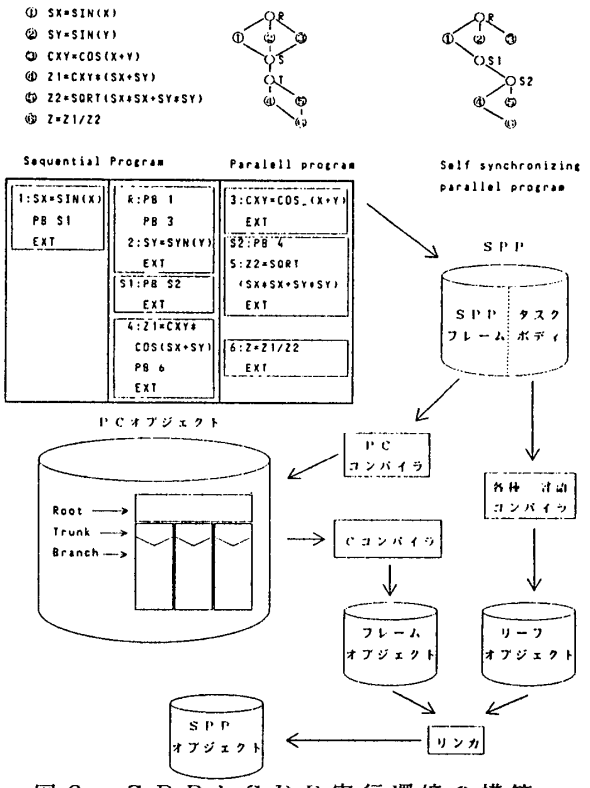

図２　ＳＰＰとＳＰＰ実行環境の構築

4. おわりに

unix上に並列実行環境を構築したことによって，分散ＯＳや並列コンパイラ等の研究に柔軟に対応できるようになった．

5. 参考文献

1)有田五次郎:”ＦＩＦＯキューを同期手段とする並列プログラムについて(Ⅰ),(Ⅱ)”，情報処理学会論文誌 VOL.24,NO.2,PP.221-237
2)有田五次郎:”ＦＩＦＯキューを同期手段とする並列プログラムについて(Ⅲ)”，情報処理学会論文誌 VOL.24,NO.6,PP.838-846
3)荒巻 他:関数型言語PILAF1.2並列処理系の開発,第40回電気関係学会九州支部連合大会論文集
4)徳崎 他:関数型言語PILAF1.2並列コンパイラの開発,第41回電気関係学会九州支部連合大会論文集
5)B.O.Apduhan:THE IMPLEMENTATION OF SPP(Self Synchronizing Parallel Program) KERNEL FUNCTIONS IN UNIX ENVIRONMENT,第40回電気関係学会九州支部連合大会論文集
6)大島 他:”８０２８６をプロセッサエレメントとするメモリ共有型並列処理システムの開発”，情報処理学会マイクロコンピュータ研究会 VOL.87,NO.86
7)Eric.Cooper,Richard P.Draves,"C Threads" Department of Computer Science Carnegie Mellon University Pittsburgh,Pennsylvania 15213 Draft of 20 July 1987