ソフトウエア工学 17－10
（1981. 2. 12）

多重プロセス環境の支援システム

慶應義塾大学工学部数理工学科　大和喜一

1. はじめに

コンピュータシステムを用いて処理する作業の大半は，逐次的な処理と並列な処理の組合わせで構成されている。その中で，並列な部分の処理のためには実際に並列な処理を行なう事が理想的であるが，多くの場合それらの実行は逐次的な処理で代行できる。しかし相互に関係を持つ並列な部分の実行には，それらの実行順序を選んで正しい結果を得る事ができるようにしなければならない。このような並列に処理すべき各部分を効率良く別個の作業単位（以降，プロセスと書き表す）に割り当てて実行すると，それらの間でデータや制御信号を交わして同期をとるようにプログラムを書くことによって実行順序をある程度自由に制御する事ができる。また，別個のプロセスである事によって，他の作業からの不意の影響を受けずに済むと伴に，1つのプロセスにすると複雑なプログラムを単純にできる可能性がある。

これらの複数プロセスを使うことによる利点を得るためには，プロセス間の関係を保つためのいろいろな機能とそれを簡単に扱えるようにするためのシステムが必要となる。ここではUNIX†と言語Cを用いた複数プロセス取扱用プログラムについて紹介する。以降2.では並列プロセスの制御を記述する言語とその実行環境との関係から言語に要求される記述能力と生成コード，実行環境に対する要求などについて述べる。3.ではプロセスに関するUNIXの能力と不足している点を明らかにする。4.では本システムの要素とその概要を述べる。また，5.ではこのシステムのためにUNIXに追加した機能を列挙する。

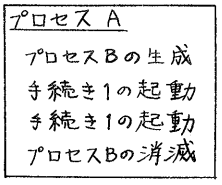
2. 記述用言語と実行環境

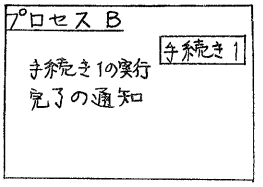
複数の並列に動作するプロセスを表現するために，Concurrent Pascal やModula-2など何種類かの言語が提案され，それらの中には実行環境が用意されてプログラムが現実に機動するものもいくつか存在する。複数のプロセスを駆使して記述されるプログラムに共通な事は，プログラム全体が論理的にはいくつかの独立した実行可能プログラムになる事である。各々のプログラムは順次処理だけで記述されていて，その中には他のプログラムの実行状態（実行の継続，停止等）を変更する命令も含まれている。主プログラムを実行するプロセスの制御に基いて，いくつかのプロセスが生成され実行される。生成されたプロセス上のプログラムが実行されて再び新しいプロセスが生成される。このような環境でのプロセスの実行には，1)そのプロセスで処理される手続きが必要とする親のプロセスが持つデータ，と2)そのプロセスで処理される手続きのコード全部が必要である。

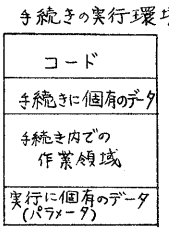
1)のデータは親のプロセスが生成したプロセスに対して必要な分だけ複製して与える，プロセスの実行に対する初期値である。これは通常の関数の呼び出しでは引数に相当するが，異なる点はそのプロセスのデータ空間にではなく新しく生成したプロセスの空間に複製する事である。親プロセスから与えられるデータに関しては複写で解決するが，1つのプロセスが動作し，停止した後，再度起動さ

† UNIX is a Trademark of bell lab.

れる時点での環境は，前に停止した状態を保っている必要がある事が多い。したがって，プロセスのデータ空間はそれが生成されてから消滅の指定がある迄保持しなければならない。手続きの実行が完了しても消滅せずに再び起動の指定が来るまで待機するように構成する。再度起動された場合には新しいパラメータが与えられるので，起動ごとに個有のデータをパラメータとする事になる。







各プロセスの作業を実行するために必要なコードは，同じプロセスから生成されたものでも異なるのが一般的である。これはプロセス上で実行される各手続き間の関係（厳密には動的な関係）に依存していて，別のプロセス上に存在するコードに環境を与えて実行する事は一つのプロセスを新たに生成する事に匹敵する。したがってプログラムの静的な関係から必要な手続きのコードを連結してプロセスの空間内に持つ方がのぞましい。プログラム中のすべての手続きを各プロセスが持つようにすると，不要なコードの複写による実行時のオーバヘッドの増大ばかりでなく作業領域の不足も考えられる。そこで，言語の設定に際しては各プロセスで必要とする手続きの弁別が可能であるようにする。

2つのプロセスが相互に起動を指令し合う状態について考えると，一般の手続きにおける入れ子の関係が想像されるが，プロセスで実行している手続きが個有のデータを持つ場合には新しいプロセスを生成する必要がある。しかし，個有データは現実の資源に結び付いている場合が多く入れ子関係に意味のない場合が多い。現実の資源を制御するプログラムではコルーチンの関係が一般的であると考えられるが，これはプロセスの切り換えによって実現できる。

3. プロセスの機能

複数のプロセスが利用できるオペレーティングシステムや言語の実行時ルーチンがいくつか存在している。UNIXにも複数のプロセスが自由に利用できる機能が備わっている。まず fork を実行して新しいプロセスを生成する。生成されたプロセスはもとのプロセスのメモリ空間の複製をメモリ空間に持つ。一旦生成されたプロセスは exec，break によってメモリ空間が変更されるが，exit を実行するまでは存在し続ける。

次に，プロセス外からの信号の受信及び例外の発生に対処するために signal がある。これを使ってプロセス間で信号の授受が可能であるが，ptrace というプロセストレースの機能を用いると複数のプロセスの実行を制御することができるようになる。

複数のプロセスが協力し合うプロセスであるときには，それらの間でデータを授受することができる。pipe とよばれるこの機能は論理的にはFIFOで，1対1の相互関係を保つには極めて便利であるが，3つ以上のプロセスの共通領域として用いるには特別な技巧を施さねばならない。また，プロセスの環境を複製したり大量のプロセス間のデータの移動には適しているが，使い方に難がある。

4. システムの構成

これまでに述べたような言語に関する制約や実行環境への要求に現実のシステムを整合させるために，いくつかの実行時ルーチンを作製した。

1) プロセス生成ルーチン

前にも述べた通り，各プログラムの流れを並列プロセス間の関係としてとらえると，プログラムの実行はプロセスの生成，待機，再開動，消滅で置き換えられる。そこで，各プログラムから得られるプロセスの関係を木として表わした表を使ってプロセスを生成し，各プロセス上で指定したプログラムを実行させるルーチンを作製した。このルーチンを用いると，いくつかの関連を持つ実行可能プログラムを簡単に動かす事ができる。また，各プロセスで実行するプログラムとして実行可能プログラムのロードを行なうプログラムを用いると，現実に複数プロセッサとメモリが存在する状態を模倣することもできる。

2) 共通メモリ参照用ルーチン

複数のプロセスが共用する広いデータ空間を確保するために，メモリ用のプロセスがプロセス生成ルーチンによって作られている。この内容を更新したり読み出したりするためのルーチンが用意されている。このメモリは複数のプロセスから自由に参照されるので，次のような機能を持たせて利用し易くしている。

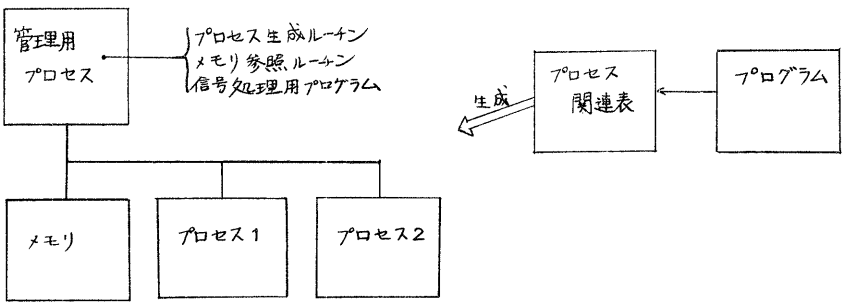
i) 排他的な参照を可能にする参照モード —— メモリの参照にはモードを与えているが，その1つにテスト・アンド・セットのモードを用意した。これによってメモリの参照にロックをかける機能を作っている。

ii) 並列プロセッサ用シミュレータのための，参照時刻順の参照 —— 並列プロセッサを模倣するために各プロセスに別個の時計を持たせるが，その時刻の順に共通メモリの参照改変を行なうと，時間の調整によって同期しているプログラムを運用する事ができる。この機能を与えるために，参照時刻に関する順序づけができるようになっている。したがって，実際に参照が行なわれるのは，メモリのプロセスの時刻が変更された時になる。

3) 信号処理用プログラム

プロセスの実行によって割出し命令が実行されたりプロセッサの割り込みが生じたりした場合の処理ルーチンは各プログラム内で処理されるが，各プログラムに共通の部分はひとまとめにして別のプロセスで処理した方が効率が良いばかりか，いろいろな情報を得ることができる(ptraceを利用する)。このための処理プログラムを管理するプログラムを用意した。

これらのプログラムの関係を以下に図示する。



5. 基本システムの拡張

複数プロセス間での作業を簡単かつ効率良くするために，基本システムであるUNIXに次のような変更を行なった。

1) プロセスの実行時間を測定するために，インターバルタイマの処理を行なう。時間の測定はプロセスの切換えが行なわれるごとに実行される。

```
settime (sys, user)  long sys, user;
```

settime によって sys. と user がそのプロセスのシステム時間と利用者時間にそれぞれ設定される。実行時間の測定には，

```
gettime ( time )   long time[2];
```

を用いて利用者プログラムから参照可能である。(time[0] ‥ sys , time[1] ‥ user)

2) プロセス間でのデータの相互参照を容易にするために，プロセス空間の一部を複数のプロセスの共通空間とする機能を与えた。これによって完全に分離してしまうメモリ空間の関係が少し緩和された。

```
ch = comexpd (size)
```

で size バイトだけプロセスの空間が拡張され，fork によってプロセスが複製されるとこの空間だけは両方のプロセスから共通に参照されるようになる。これまでは共通に参照可能なものはファイルとpipe だけであったので特別に相互排除制御のための機能を持たなかったが(pipe は1回の読み書きに対して排他的である)共通メモリの扱いには何らかの操作が必要なので，UNIX の資源に関する同期基本命令である sleep と wakeup を共通メモリに連結した操作

```
enq(ch), deq(ch)
```

を用意した。これらの機能を用いるプログラムの形態を以下に示す。

```
struct {
        int  a;
        int  b;
        char c[10];};
func( )
{
        int *p, x; extern *caddr;
        x = comexpd(100); p=caddr;
        enq(x); p->a = 1; deq(x);
        if( fork()==0){
            enq(x); printf("%d\n",p->a);
            p->a = 2; deq(x);}
        else {
loop:       enq(x);
            if(p->a == 1) {deq(x); goto loop;}
        ....
```

6. おわりに

現在このシステムはUNIXの変更と実行時ルーチンの大半が完了しているだけで，言語の仕様及び実行可能ファイル群を作製するためのユーティリティの変更が未完である。実際に何重かになったプロセス群が動くと速度は極度に低下するが，これはUNIXのスケジューリングが大きく効いているためと考えられる。

プロセス間の関係の抽出とそれから得られる順次処理プログラム群の処理について更に実例を集めてその効率を考えていく予定である。

参考文献

1) Christian Jacobi, "A User Guide to the Modula-2 System", Institut fur Informatik, ETH, July 1980