パーソナルコンピュータ上における C言語処理系のコンカレント化

1G－6

石渡寛利　井上謙蔵
(東京理科大学)

１．はじめに

近年計算機の性能向上につれて、一台の計算機を複数のユーザーが同時に利用したり、一人のユーザーが複数のプログラム（プロセス）を同時に動かしたりといったことが、大型計算機のみならず小型の計算機でも可能になってきた。しかしそれらの制御を司る部分、つまりオペレーティングシステムの内部はユーザーから見ればブラックボックス化されており、それらの学習をする者にとっては不便きわまりない。

以前このような問題を解決するために、FORTRANに疑似並行処理の機能を持たせた SOS[1] 及びそれを改良した SOCPS[2] というものが開発されたが、プログラムの記述法が、非常に難しいといった欠点が多かった。そこで誰もが簡単に使うことの出来るパーソナルコンピュータ上のＣ言語に並行処理の機能を付加し、哲学者の食事問題や読み書き問題といった、ＯＳの古典的な諸問題を極簡単に記述、実行できる処理系 C-Concurrent Manager(以下 CCM)を開発したのでここに報告する。

２．動作原理

Ｃプログラムは、全て「関数」という単位で動作をする。関数はその中でのみ参照可能なローカルな変数（自動変数）を持つことが出来る。この変数は、関数が呼び出される度にスタック上にその領域が確保され、関数が終了するとその領域は解放される。つまり自動変数は、動的な領域に確保される変数であるといえる。図１にその様子を示す。

自動変数の他に「静的変数」及び「外部変数」というものがある。これらは、メモリ上の静的な領域に確保される。よって関数の呼び出しや終了によってその領域が変動することはない。

これより、関数が動作するときに必要なメモリ領域は、プログラムコード、スタック上の自動変数領域、及び静的変数・外部変数領域となる。ここで、プログラムコード及び静的変数・外部変数領域は、静的なため、関数によって変化するものではない。よって、各関数毎に固有のものは、スタック上の自動変数領域のみとなる。

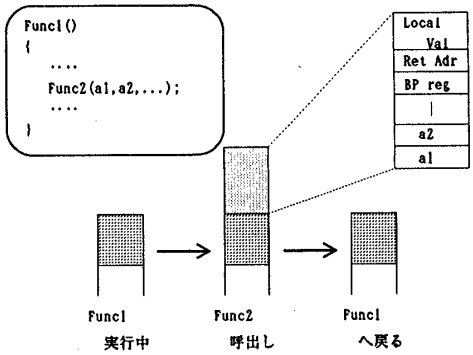

図１．　Ｃ言語の関数呼び出し

このことを逆に言えば、関数が動作するためには、その関数の自動変数領域をスタック上に用意してやれば良いと言うことになる。CCM では、この性質を用いて、実行すべき関数をダイナミックに切り替えることを可能にしている。具体的には、図２に示すとおりである。

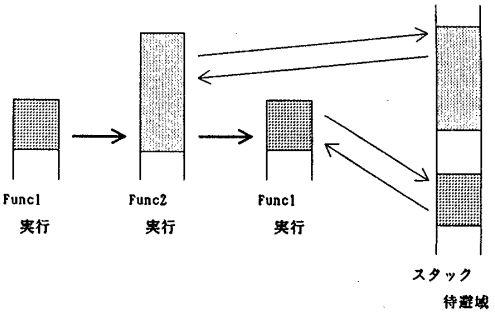

図２．　スタック領域の動的切替

CCM では、プロセスの切り替えにインターバルタイマ割り込みを用いている。インターバルタイマに対して、カウンタ値をセットすると、ある一定時間(0.004 ～ 26 ms)後に割り込みが発生する。これにより、プロセスに対してタイムスライスを与えることができる。その様子を図３に示す。

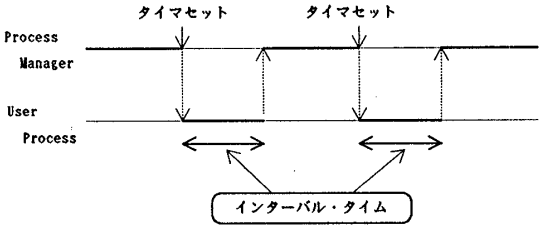

図３．　インターバルタイマによるタイムスライス

Concurrent-C Language on Personal Computer
Hirotoshi ISHIWATARI, Kenzo INOUE
Science University of Tokyo

CCM では、プログラムの動作開始後、全ての制御をプロセスマネージャーの管理下に置く。そしてプロセスマネージャーによって、前記のスタック領域のスワッピングやタイマ割り込みの制御を行なうことにより、並行処理を可能にしている。

3．同期制御

CCM では、次に示すような同期制御の関数を用意しており、プログラマによって簡単に使用することが出来る。

- セマフォ
  - P命令：P_op(semaphore *);
  - V命令：V_op(semaphore *);
- セマフォ・マルチプル
  - P命令：P_mlt(semaphore *,...);
  - V命令：V_mlt(semaphore *,...);
- イベント待ち行列
  - await(eventq *);
  - cause(eventq *);

4．プログラム記述例

CCM による並行処理プログラムの記述例をいくつか示す。なお、説明の便宜上一部言葉による表現を用いた。

①哲学者の食事問題

```
#define N 哲学者数
semaphore fork[N];  /*セマフォの宣言*/
PROCESS cmain()     /*エントリプロセス*/
{
  PROCESS philo();  /*子プロセスの宣言*/
  PR_ID   ph[N];    /*プロセスIDの宣言*/
  int     i;

  for(i=0;i<N;i++)  /*セマフォの初期化*/
    ini_sem(&fork[i],1);
  for(i=0;i<N;i++)/*哲学者プロセスの生成*/
    ph[i] = create(philo);
  for(i=0;i<N;i++)/*哲学者プロセスの起動*/
    execute(ph[i]);
  ....
  term();           /*プロセスの終了*/
}

PROCESS philo()   /*哲学者プロセス*/
{
  for(;;) {
    P_mlt(&fork[左],&fork[右]); /*P命令*/
        /*哲学者食事中*/
    V_mlt(&fork[左],&fork[右]); /*V命令*/
        /*哲学者思案中*/
    }
}
```

②生産者消費者問題

```
semaphore sem;      /*セマフォの宣言*/
PROCESS cmain()     /*エントリプロセス*/
{
  PROCESS creater();/*生産者プロセスの宣言*/
  PROCESS consumer();/*消費者プロセス 〃  */
  PR_ID   cre,con;  /*プロセスIDの宣言*/

  cre=create(creater); /*生産者生成*/
  con=create(consumer);/*消費者生成*/

  execute(cre);        /*生産者起動*/
  execute(con);        /*消費者起動*/
  ....
  term();          /*プロセスの終了*/
}

PROCESS creater() /*生産者プロセス*/
{
  for(;;) {
      /*資源の生産*/
    V_op(&sem); /*セマフォV命令*/
  }
}

PROCESS consumer()/*消費者プロセス*/
{
  for(;;) {
    P_op(&sem); /*セマフォP命令*/
      /*資源の消費*/
  }
}
```

4．おわりに

昨今広く用いられているC言語に並行処理の機能を付加したことにより、C言語について多少の知識がある人ならば簡単にプログラムを組むことが出来るシステムが完成した。本システムは、PC-9801上のLattice-C及びTurbo-Cにおいて実現されている。

【参考文献】

[1] 土居範久
ALGOL あるいは FORTRAN 内でコオペレーティングプロセスを実現するためのオペレーティングシステム
情報処理 Vol.16 No.1 p.7～14 (1975)

[2] 林田　博
オペレーティングシステムに関する研究
東京工業大学理学部情報科学科
昭和５０年度研究論文