第2章 セマフォ，モニタ，ランデブ，チャネル通信…

並列プログラミングに必須の排他制御の書き方

竹岡 尚三

CPUコアが複数あり，複数の処理が同時に実行されている場合，それらがメモリなどの資源にアクセスするときに，競合しないようにしなければならないことは一般に知られています．

このような排他制御は，ソフトウェア層でもハードウェア層でも必要となります．ハードウェア層の場合，正しく排他制御や調停機構を作らなければハードウェアが壊れてしまいます．調停機構はアービタ（Arbiter）とも言います．

本稿では，ハードウェア層での調停はできているとして，ソフトウェア層での制御についてさまざまな方法を説明します．資源へのアクセスを制御するという目的は同じですが，実現するアプリケーションや，使用する言語や処理系との兼ね合い，ターゲットとするハードウェアなどによって，適切な実装方法は異なります．

排他制御の必要性を考える

マルチコアやマルチタスク（マルチスレッド）の並列/並行システムで，正しく意味のある計算を行うには，排他制御か同期制御が必要です．排他と同期は裏表の関係で，その本質は同じものと考えることができます．

● 排他制御しないと正しい計算ができない

▶サンプル・プログラムと処理内容

ここでは排他制御を考えます．排他制御は相互排他（mutual exclusion）とも言います．

処理の具体的な例としてマルチスレッドで，1つの変数に総和を集めるサンプル・プログラムを基に考えます（**リスト1**）．

リスト1をprog1.cというファイル名で保存し，次のように実行します．

```
$ cc -pthread -o prog1 prog1.c↵
```

リスト1　複数のスレッドから同じ変数にアクセスする（prog1.c）

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t sum_mutex;
long long s=0;

void *
part_sum(void *argp)
{
  int i, *a=argp, start, end;
  start= a[0];  end= a[1];
  printf("thread start=%d end=%d¥n",start,end);

  for(i=start;i<=end;i++){
    //pthread_mutex_lock(&sum_mutex);
    s = s + i;
    //pthread_mutex_unlock(&sum_mutex);
  }
}

int a1[]={ 0,499999 };
int a2[]={ 500000,999999 };

int main(void)
{
  int i,x;
  pthread_t thread1, thread2;

  if(pthread_create(&thread1, NULL, part_sum, a1) !=  0) {
    printf("thread1 create failed\n");
    exit(EXIT_FAILURE);
  }

  if(pthread_create(&thread2, NULL, part_sum, a2) !=  0) {
    printf("thread2 create failed¥n");
    exit(EXIT_FAILURE);
  }

  if(!((pthread_join(thread1, NULL) == 0) &&
       (pthread_join(thread2, NULL) == 0))){
    exit(EXIT_FAILURE);
  }

  printf("s=%lld¥n", s);

  exit(EXIT_SUCCESS);
}
```