ラウンドロビンアルゴリズムによる CPU スケジューリングのシミュレーションプログラム

1. CPU バースト時間の表

シミュレーションを行うには，それぞれのプロセスがどのような CPU バースト時間，CPU 遊休時間（入出力バースト時間等）になるかを想定します。本例では，cpuBurst という配列を用意し，量子時間単位毎に 1（CPU バースト時間を示す）か 0（CPU 遊休時間を示す）で示します。先頭の添字はジョブ番号（あるいはプロセス番号），後ろの添字は量子時間とします。以下の指定ではジョブ数 4 まで，量子時間は最大 16 まで指定できるものとしています。

```
#include "stdafx.h"
#define NumJob  4
#define MaxTime 16
static int cpuBurst[NumJob][MaxTime]=    //NumJob は省略してもよい
      {{1,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0},
       {1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0},
       {1,0,1,0,1,0,1,0,0,1,0,1,0,1,0,0},
       {1,0,1,1,1,1,0,1,0,1,0,1,0,1,0,0}};
```

2. CPU バースト時間の表で最大時間を求める

上記 1 の例では，cpuBurst という配列の後の方には 0 が並んでいます。表を表示するとき，最後の 0 が続いている部分を除外するため，後方からチェックして 0 でない箇所を探索します。

```
int effectiveNumTime(){
        for(int n= MaxTime-1;n>=0;n--)
                for(int i=0;i<NumJob;i++) if(cpuBurst[i][n]>0)return n;
        return 0;
}
```

メインプログラムを次のように記述して，実行してみましょう。

```
int _tmain(int argc, _TCHAR* argv[]){
        printf("%d", effectiveNumTime();
        getchar();
}
```

最後の getchar()は，実行後，何らかの入力がなされない限り MS-DOS 画面が消えないようにするための便宜的な呼び出しです。

3. CPU バースト時間の表を表示する

まず，見出し部分を表示します。たとえば，次のように関数 effetiveNumTime を呼び出して，表示する量子時間だけ数字を表示します。タイトルの次の行の空白は，表示ジョブ番号の分だけ入れます。この空白数は後で調整しても構いません。

```
int n=effectiveNumTime(); printf("¥n¥n ** CPU Burst¥n          ");
for(int j=1;j<=n;j++)printf("%2d",j %10);
```

CPU バースト時間をジョブ数だけ表示しますので,次のような繰り返しになることは明らかでしょう。for に続く{ と } の間に 1 行分の処理を書きます。

```
for(int i=0;i<NumJob;i++){
}
```

1 行の処理では，まず，ジョブ名を表示します。ここでは，i=0 のとき'A'，i=1 のとき'B'，i=2 のとき'C'等と表示することにします。このようなとき，C 言語では次のように書きます。

```
printf("¥n Job %C : ",'A'+ i);
```

1 ジョブの終了は CPU バーストで終わるはずですので，後ろから 0 でない値をチェックすることでジョブ完了の時間を求めることができます。

```
for(n=MaxTime-1;n>=0;n--)if(cpuBurst[i][n]>0) break;
```

ジョブ完了時刻が n の値として求まりますので，1 行の CPU 時間と遊休時間を表示します。CPU 時間（cpuBurst の値が 1）は“＝”，遊休時間（cpuBurst の値が 0）は“…”で表示します。すなわち，次のように記述します。

```
for(int j=0;j<=n;j++){
    if(cpuBurst[i][j]!=0) printf("＝"); else printf("…");
}
```

これまでの記述を，dspBurst という関数にまとめましょう。すなわち，

```
void dspBurst(){ // CPU バースト表を表示
  int n=effectiveNumTime(); printf("¥n¥n ** CPU Burst¥n          ");
   for(int j=1;j<=n;j++)printf("%2d",j %10);
   for(int i=0;i<NumJob;i++){
     printf("¥n Job %C : ",'A'+ i);
     for(n=MaxTime-1;n>=0;n--)if(cpuBurst[i][n]>0) break;
     for(int j=0;j<=n;j++)
     {if(cpuBurst[i][j]!=0) printf("＝"); else printf("…");}
   }
}
```

メインプログラムを次のように変更して，デバッグしてみましょう。

```
    int _tmain(int argc, _TCHAR* argv[]){
        dspBurst(); getchar();
 }
```

定義されたデータと対比できる表が表示されたら完成です。違っていたら，どこか間違っているはずですので，チェックしてください。

4. 各プロセス(ジョブ)用経過時間の表

現在実行中のプロセスがどのプロセスかを識別するための変数 Exe と，各プロセスがバースト表のどの位置にあるかを識別するためのカウンタ配列 cpuCount を用意します。

```
static int cpuCount[NumJob], Exe=-1;;
```

変数 Exe はプロセス番号を示しますが，どのプロセスも実行中でないとき負の数（ここでは - 1）

を設定します。配列 cpuCount は，以下のように初期設定します。

```
void clearCount(){ for(int i=0;i<NumJob;i++) cpuCount[i]=0;}
```

配列 cpuCount は，CPU 待ち時間以外，すなわち添字で指定されるプロセスが CPU 実行状態に入っているとき，または遊休時間のときカウントします。なお，CPU バースト表の最大時刻に達しているときを除外します。

```
void countTime(){// 遊休状態のプロセスと実行プロセスの時間をカウント
      for(int i=0;i<NumJob;i++)
        if(cpuCount[i]<MaxTime && (cpuBurst[i][cpuCount[i]]==0|| i==Exe))
                                                    cpuCount[i]++;
}
```

5. シミュレーション終了の判定

まず，シミュレーションの終わりを考えてましょう。配列 cpuCount は，各プロセスのバースト表での位置を示していますので，すべてのプロセスのカウンタがバースト表の最後に来ていたら終わりとみなします。すなわち次のように記述することができます。

```
int notEnd(){ // シミュレーション終了でないかを判定
      for(int i=0;i<NumJob;i++)if(cpuCount[i]<MaxTime) return true;
      return false;
}
```

6. ラウンドロビンカウンタ

ラウンドロビンアルゴリズムでは，プロセスを順に処理しますが，最後のプロセスが実行されると，次は先頭のプロセスに実行を移します。すなわち循環するカウンタが必要です。これをラウンドロビンカウンタ iRound と呼ぶことにしましょう。プログラム中に直接書いても構いませんが，単純化のために次のような関数を用意します。

```
static int iRound = 0;
⋮
void countRound(){ iRound++;if(iRound>=NumJob) iRound=0;   }
```

7. 実行プロセスを求める(ラウンドロビン法)

カウンタ iRound で示されるプロセスが未終了かつ CPU 時間のとき，そのプロセスを実行プロセスとします。なお，すでに終わっているプロセスのとき，またはそのプロセスが遊休時間のときは，次のプロセスをチェックします。実行するプロセスがないとき - 1 を返却します。

```
int exeProcess(){//実行プロセスの決定
    for(int i=0;i<=NumJob;i++, countRound())
      if(cpuCount[iRound]<MaxTime && cpuBurst[iRound][cpuCount[iRound]]!=0)
                                                    return iRound;
   return -1;
}
```

メインプログラムから，この関数を次のように呼び出して変数 Exe に設定します。

```
Exe= exeProcess();
```

８．実行プロセス番号を保存

上記７の処理で求まった実行プロセス番号を，次のような配列とカウンタを用意して保存します。

```
#define MaxExecTime 300
static int cpuProc[MaxExecTime], tCount=0;
```

変数 Exe に実行プロセス番号が設定されているものとし，保存できたとき true，できなかったとき false を返却します。

```
int saveProcNo(){//実行プロセス番号の保存
  if(tCount>=MaxExecTime){
     printf("¥n  *** Error : Area Over Flow");
     return false;
  }
  cpuProc[tCount++]=Exe;
  return true;
}
```

９．タイムチャートを表示する

シミュレーションでは，実行プロセスの選択と保存を繰返し，配列 cpuProc にプロセス番号を格納します。cpuProc の数字列をそのまま表示してもよいのですが，わかりやすくするためにプロセス別に線表（タイムチャートと呼ぶことにします）の形で表示してみましょう。量子時間 j のときのプロセス番号は cpuProc[j]ですので，プロセス i，量子時間 j の箇所を次のように表示すれば良いでしょう。

```
if(cpuProc[j]==i)printf("＝");
else printf("…");
```

ただし，実行時間でない箇所の区切りを付けるため次のように工夫します。

```
if(cpuProc[j]==i)printf("＝");
else {if((j%10)==9)printf("･+");else printf("…");}
```

また，プロセスは，CPU バーストで終了しますので，最後の該当プロセス番号をそのプロセスの終了とみなします。終了している場合，空白の方が分かりやすいので，プロセス i のとき，

```
int mm;for(mm=n;mm>=0;mm--) if(cpuProc[mm]==i)break;
```

とし，量子時間 j のとき，以下のように表示します。

```
if(j>mm){if((j%10)==9) printf(" +");else printf("  ");}
```

線表の見出しでは，どのプロセスも動いていない後方の値を無視する処理を入れます。完成したタイムチャート表示関数は，以下のとおりです。

```
void dspTimeChart(){//タイムチャート表示
      int n; for(n=tCount-1;n>=0;n--)if(cpuProc[n]>0) break;
      printf("¥n¥n ** Time Chart¥n          ");
      for(int j=1;j<=n+1;j++)printf("%2d",j%10);
      for(int i=0;i<NumJob;i++){
         printf("¥n Job %C : ",'A'+ i);
         int mm;for(mm=n;mm>=0;mm--) if(cpuProc[mm]==i)break;
```

```
            for(int j=0;j<=n;j++)
               if(j>mm){if((j%10)==9) printf(" +");else printf("  ");}
               else if(cpuProc[j]==i)printf("＝");
               else {if((j%10)==9)printf("･+");else printf("…");}
         }
   }
```

10. ターンアラウンド計算表表示

ターンアラウンドを計算するには，まずプロセス終了時点を求めます。

```
int mm; for(mm=tCount-1; mm>=0; mm--) if(cpuProc[mm]==i)break;
```

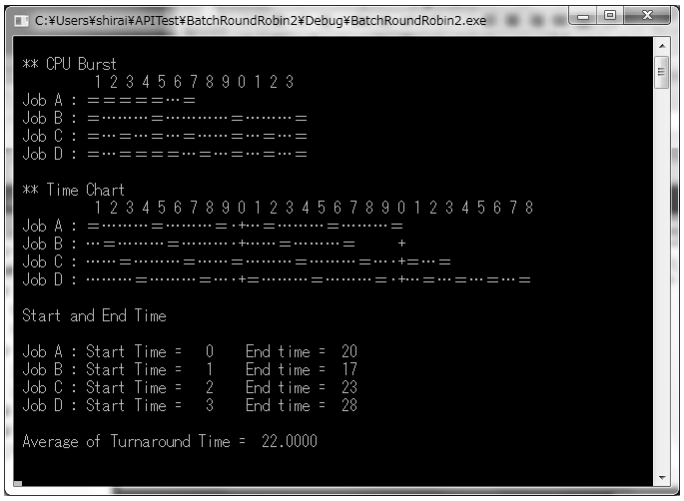
この mm を合計して，プロセス数で割れば平均ターンアラウンドになります。ついでにプロセス開始時間も求める処理を追加して，次のような関数にしました。

```
void dspTurnaround(){//ターンアラウンド計算表表示
   int T=0; printf("¥n¥n Start and End Time¥n");
   for(int i=0;i<NumJob;i++){
         int ms; for(ms=0; ms<=tCount-1; ms++) if(cpuProc[ms]==i)break;
         int mm; for(mm=tCount-1; mm>=0; mm--) if(cpuProc[mm]==i)break;
         mm++; T += mm;
         printf("¥n Job %C : Start Time = %3d    End time = %3d ",'A'+ i,ms,mm);
   }
   printf("¥n¥n Average of Turnaround Time = %8.4lf¥n¥n",
                                    (double)T / (double)NumJob);
}
```

11. メインプログラム

以上を呼び出すメインプログラムです。

```
int _tmain(int argc, _TCHAR* argv[]){
  clearCount();
  for(iRound=0;notEnd();countRound()){
        Exe= exeProcess();
        countTime();
        if(!saveProcNo()) break;
  }
  dspBurst(); dspTimeChart();
  dspTurnaround();
  getchar(); return 0;
}
```



12. 全体プログラムリスト

```
#include "stdafx.h"
#define NumJob  4
#define MaxTime 16
#define MaxExecTime 300
static int iRound=0, cpuProc[MaxExecTime], cpuCount[NumJob], tCount=0, Exe=-1;
static int cpuBurst[][MaxTime]={{1,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0},
                                {1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0},
                                {1,0,1,0,1,0,1,0,0,1,0,1,0,1,0,0},
                                {1,0,1,1,1,1,0,1,0,1,0,1,0,1,0,0}};
// cpuBurst には，CPU バースト時間のとき 1，CPU 遊休時間のとき 0 を入れる。
int notEnd(){ // シミュレーション終了の判定
      for(int i=0;i<NumJob;i++)if(cpuCount[i]<MaxTime) return true;
      return false;
}
```

```
int effectiveNumTime(){// 各配列中ですべてが 0 でない箇所を見つける
      for(int n= MaxTime-1;n>=0;n--)
          for(int i=0;i<NumJob;i++) if(cpuBurst[i][n]>0)return n;
      return 0;
}
void countRound(){ iRound++;if(iRound>=NumJob) iRound=0;   }
void clearCount(){ for(int i=0;i<NumJob;i++) cpuCount[i]=0;}
void countTime(){// 遊休状態のプロセスと実行プロセスの時間をカウント
   for(int i=0;i<NumJob;i++)
      if(cpuCount[i]<MaxTime && (cpuBurst[i][cpuCount[i]]==0|| i==Exe)) cpuCount[i]++;
}
int exeProcess(){//実行プロセスの決定
  for(int i=0;i<=NumJob;i++, countRound())
     if(cpuCount[iRound]<MaxTime && cpuBurst[iRound][cpuCount[iRound]]!=0)return iRound;
  return -1;
}
void dspBurst(){ // CPU バースト表を表示
  int n=effectiveNumTime(); printf("¥n¥n ** CPU Burst¥n         ");
  for(int j=1;j<=n;j++)printf("%2d",j %10);
  for(int i=0;i<NumJob;i++){
    printf("¥n Job %C : ",'A'+ i); for(n=MaxTime-1;n>=0;n--)if(cpuBurst[i][n]>0) break;
    for(int j=0;j<=n;j++){ if(cpuBurst[i][j]!=0) printf("═"); else printf("…");}
  }
}
void dspTimeChart(){//タイムチャート表示
      int n; for(n=tCount-1;n>=0;n--)if(cpuProc[n]>0) break;
      printf("¥n¥n ** Time Chart¥n          ");for(int j=1;j<=n+1;j++)printf("%2d",j%10);
      for(int i=0;i<NumJob;i++){
         printf("¥n Job %C : ",'A'+ i);
         int mm;for(mm=n;mm>=0;mm--) if(cpuProc[mm]==i)break;
         for(int j=0;j<=n;j++)
                  if(j>mm){if((j%10)==9) printf(" +");else printf("  ");}
                  else if(cpuProc[j]==i)printf("═");
                  else {if((j%10)==9)printf("･+");else printf("…");}
      }
}
void dspTurnaround(){//ターンアラウンド計算表表示
      int T=0; printf("¥n¥n Start and End Time¥n");
      for(int i=0;i<NumJob;i++){
         int ms; for(ms=0; ms<=tCount-1; ms++) if(cpuProc[ms]==i)break;
         int mm; for(mm=tCount-1; mm>=0; mm--) if(cpuProc[mm]==i)break;
         mm++; T += mm;
         printf("¥n Job %C : Start Time = %3d    End time = %3d ",'A'+ i,ms,mm);
      }
      printf("¥n¥n Average of Turnaround Time = %8.4lf¥n¥n", (double)T / (double)NumJob);
}
int saveProcNo(){//実行プロセス番号の保存。
  if(tCount>=MaxExecTime){ printf("¥n  *** Error : Area Over Flow");return false;}
  cpuProc[tCount++]=Exe; return true;
}
int _tmain(int argc, _TCHAR* argv[]){ //シミュレーション実行
      clearCount();
      for(iRound=0;notEnd();countRound()){
         Exe= exeProcess();
         countTime();
         if(!saveProcNo()) break;
      }
      dspBurst(); dspTimeChart(); dspTurnaround();//結果表示
      getchar(); return 0;
}
```