CSE 4502 – OS Lab Assignment

on

Process Scheduling

Name: Ratun Rahman

ID: 170042011

Date of Submission: 12 August 2020, Wednesday

Step 1: Get all the essential inputs.

```
float alpha=0.5;
int cpuBurst[5],arrivalTime[5],priority[5];
cout<<"ID: ";
cin>>n;
cout<<"Given CPU burst: ";
for(i=0; i<5; i++)
    cin>>cpuBurst[i];
cout<<"Arrival times: ";
for(i=0; i<5; i++)
    cin>>arrivalTime[i];
cout<<"Process priority: ";
for(i=0; i<5; i++)
    cin>>priority[i];
```

Given, alpha=0.5. Taking ID as n, CPU burst, arrival time and process priority.

Step 2: Find predicted burst and print them

```
//get predicted burst
int a[6];
a[0]=n;
for(i=1; i<6; i++)
{
    a[i]=ceil( a[i-1]*alpha+cpuBurst[i-1]*(1-alpha)); //predicted burst formula
}
cout<<"Predicted burst: [";
for(i=1; i<6; i++)
{
    cout<<a[i];
    if(i!=5)
        cout<<", ";
    else
        cout<<"]"<<endl;
}
cout<<endl;
```

Using the ceil value of the formula. And print them as it is showed in the output.

Sample Input:

```
ID: 92
Given CPU burst: 6 4 5 3 3
Arrival times: 0 1 4 6 7
Process priority: 3 1 4 5 2
Predicted burst: [49, 27, 16, 10, 7]
```

SJF

Step 1: Sequence

```
59      cout<<"a. SJF [All processes arrive at 0]:"<<endl;
60      vector<int>seq;
61      int temp,b[6]= {0},k=5;
62      while(k--)
63      {
64          temp=9999;
65          for(i=1; i<6; i++)
66          {
67              if(a[i]<temp&&b[i]!=1)
68              {
69                  temp=a[i];
70                  j=i;
71              }
72          }
73          //cout<<temp<<endl;
74          seq.push_back(j);//temp is the smallest burst and j it's index
75          b[j]=1;
76      }
77      cout<<"- Sequence: [";
78      for(i=0; i<5; i++)
79      {
80          cout<<seq[i];
81          if(i!=4)
82              cout<<", ";
83          else
84              cout<<"]"<<endl;
85      }
```

Declaring a temp value to get the minimum burst and save its index on j. push the j in seq vector and print it accordingly.

Step 2: Intervals

```
ll s=0;
float awt=0;
cout<<"- Intervals: [0, ";
for(i=0;i<5;i++){
    s=s+a[seq[i]];
    cout<<s; //the intervals value
    if(i!=4){
        awt=awt+s; // saving total time as well
        cout<<", ";
    }
    else
        cout<<"]";
}
cout<<endl;
```

As all processes arrive at 0, the shortest process will run first and everyone else will be at waiting time. Here, we are printing 0 for first process. And sum up all other process as intervals and print them.

Step 3: Average waiting time

```
cout<<"- Average Waiting Time: "<<awt/5<<"ms"<<endl;
cout<<endl;
```

We have already count the total waiting time while counting intervals time before. Divide it with process number to find average waiting time.

Sample Output:

```
a. SJF [All processes arrive at 0]:
- Sequence: [5, 4, 3, 2, 1]
- Intervals: [0, 7, 17, 33, 60, 109]
- Average Waiting Time: 23.4ms
```

SRTF

Step 1: Find Sequence and intervals and total waiting time

```
//srtf
cout<<"b. SRTF:"<<endl;
ll waiting[5]={0},sl=s,time,c=0,a2[6];
ll smallValue,smallIndex,previousIndex=9999;
vector<int>seq2,int2;
for(i=1;i<6;i++)
{
    a2[i]=a[i]; //a2=CPU burst
}
for(time=0; c!=5; time++) // calculating for every second
{
    smallValue=9999;
    for(i=1; i<6; i++)
    {                   //if it arrives and not finished yet, find smallValue
        if(arrivalTime[i-1]<=time && a2[i]<smallValue && a2[i]>0 ){
            smallIndex=i;
            smallValue=a2[i];
        }
    }
    if(smallIndex!=previousIndex){ //if the process was not running last sec
        seq2.push_back(smallIndex);
        int2.push_back(time);
    }
    a2[smallIndex]--;
    for(i=1; i<6; i++)
    {                   // if process arrived and its not the smallestIndex
        if(arrivalTime[i-1]<=time && i!=smallIndex && a2[i]>0 )
            waiting[i-1]++;
    }
    if(a2[smallIndex]==0)
        c++;
    previousIndex=smallIndex;
}
```

Here, we are calculating for every second until all process is finished. If process is arrived (arrivalTime[i-1]<time) and not finished yet (a2[i]>0), we are taking the smallest value's index as smallIndex. If smallIndex and previousIndex are not same we are adding smallIndex in **sequence** and time in **intervals**. Also while the process is running

all other arrived non-zero process are counted as **waiting** time. Continuing the loop until all processes are finished (c=5).

Step 2: Print sequence, intervals, average waiting time

```
cout<<"- Sequence: [";
for(i=0;i<seq2.size();i++){
    cout<<seq2[i];
    if(i!=seq2.size()-1)
        cout<<", ";
    else
        cout<<"]"<<endl;
}
cout<<"- Intervals: [";
for(i=0;i<int2.size();i++)
    cout<<int2[i]<<", ";
cout<<time<<"]"<<endl;
awt=0;
for(i=0;i<5;i++)
    awt=awt+waiting[i];
cout<<"- Average waiting time: "<<awt/5<<"ms"<<endl;
cout<<endl;
```

Print them according to the given example.

Sample Output:

```
b. SRTF:
- Sequence: [1, 2, 3, 4, 5, 4, 3, 2, 1]
- Intervals: [0, 1, 4, 6, 7, 14, 23, 37, 61, 109]
- Average waiting time: 23.4ms
```

PPS

Step 1: Find Sequence and intervals and total waiting time

```
//pps
cout<<"c. PPS:"<<endl;
ll a3[6],waiting2[5]={0};
vector<int>seq3,int3;
c=0,previousIndex=9999;
for(i=1;i<6;i++)
{
    a3[i]=a[i];
}
for(time=0; c!=5; time++) // checking every time
{
    p=9999;
    for(i=1; i<6; i++)
    {                       // if process arrives and its not finished yet
        if(arrivalTime[i-1]<=time && priority[i-1]<p && a3[i]>0 ){
            smallIndex=i;   //get lowest priority index
            p=priority[i-1];//get the priority
        }
    }
    if(smallIndex!=previousIndex){ // if current index and previous index does not match
        seq3.push_back(smallIndex); //smallIndex to sequence
        int3.push_back(time);      //time to interval
    }
    a3[smallIndex]--;
    for(i=1; i<6; i++)
    {
        if(arrivalTime[i-1]<=time && i!=smallIndex && a3[i]>0 )
            waiting2[i-1]++;  //if process is available and not finished, other than the current
    }                         //index, they are in waiting list
    if(a3[smallIndex]==0)
        c++;
    previousIndex=smallIndex;
}
```

Here, we are calculating for every second until all process is finished. If process is arrived (arrivalTime[i-1]<time) and not finished yet (a2[i]>0), we are taking the lowest priority index as smallIndex. If smallIndex and previousIndex are not same we are adding smallIndex

in **sequence** and time in **intervals**. Also while the process is running all other arrived non-zero process are counted as **waiting** time. Continuing the loop until all processes are finished (c=5).

Step 2: Print sequence, intervals, average waiting time

```
cout<<"- Sequence: [";
for(i=0;i<seq3.size();i++){
    cout<<seq3[i];
    if(i!=seq3.size()-1)
        cout<<", ";
    else
        cout<<"]"<<endl;
}
cout<<"- Intervals: [";
for(i=0;i<int3.size();i++)
    cout<<int3[i]<<", ";
cout<<time<<"]"<<endl;
awt=0;
for(i=0;i<5;i++)
    awt=awt+waiting2[i];
cout<<"- Average waiting time: "<<awt/5<<"ms"<<endl;
cout<<endl;
```

Print them according to the given example.

Sample Output:

```
c. PPS:
- Sequence: [1, 2, 5, 1, 3, 4]
- Intervals: [0, 1, 28, 35, 83, 99, 109]
- Average waiting time: 45.4ms
```

RR

Step 1: Find Sequence and intervals and total waiting time

```
//rr
cout<<"d. RR [All process arrive at 0]:"<<endl;
vector<int>seq4,int4;
int q=ceil(a[1]*0.5+cpuBurst[0]*0.5),m,waiting4[5]={0}; //Value of Q
c=0,s=0;
awt=0;
int4.push_back(s); //inital value of interval
cout<<"- Quantum (Q): "<<q<<endl;
for(i=1;c!=5&&a[i]>0; i++){
    m=min(q,a[i]); //minimum of Q and CPU burst
    a[i]=a[i]-m;
    s=s+m;
    awt=awt+(5-c-1)*m;
    //cout<<awt<<" "<<c<<" "<<endl;
    if(a[i]==0)
        c++;
    seq4.push_back(i);
    int4.push_back(s);
    if(i==5)
        i=0;
}
```

Here we are calculating the value of Q using the given formula. Then we are counting the minimum of Q and CPU burst and using this as interval time. We are also calculating the sequence. Here in the loop we keep continuing check if all the processes are available or not. Then we also loop the value of i (1 to 5 if available) and keep checking until all the processes are done (c=5)

Step 2: Print sequence, intervals, average waiting time

```
cout<<"- Sequence: [";
for(i=0;i<seq4.size();i++){
    cout<<seq4[i];
    if(i!=seq4.size()-1)
        cout<<", ";
    else
        cout<<"]"<<endl;
}
cout<<"- Intervals: [";
for(i=0;i<int4.size();i++){
    cout<<int4[i];
    if(i!=int4.size()-1)
        cout<<", ";
    else
        cout<<"]"<<endl;
}
cout<<"- Average waiting time: "<<awt/5<<"ms"<<endl;
cout<<endl;
```

Print them according to the given example.

Sample Output:

```
d. RR [All process arrive at 0]:
- Quantum (Q): 28
- Sequence: [1, 2, 3, 4, 5, 1]
- Intervals: [0, 28, 55, 71, 81, 88, 109]
- Average waiting time: 59ms
```

For my ID, 11

```
ID: 11
Given CPU burst: 6 4 5 3 3
Arrival times: 0 1 4 6 7
Process priority: 3 1 4 5 2
Predicted burst: [9, 7, 6, 5, 4]

a. SJF [All processes arrive at 0]:
- Sequence: [5, 4, 3, 2, 1]
- Intervals: [0, 4, 9, 15, 22, 31]
- Average Waiting Time: 10ms

b. SRTF:
- Sequence: [1, 2, 5, 4, 3, 1]
- Intervals: [0, 1, 8, 12, 17, 23, 31]
- Average waiting time: 8.4ms

c. PPS:
- Sequence: [1, 2, 5, 1, 3, 4]
- Intervals: [0, 1, 8, 12, 20, 26, 31]
- Average waiting time: 9.6ms

d. RR [All process arrive at 0]:
- Quantum (Q): 8
- Sequence: [1, 2, 3, 4, 5, 1]
- Intervals: [0, 8, 15, 21, 26, 30, 31]
- Average waiting time: 18.4ms


Process returned 0 (0x0)   execution time : 22.152 s
Press any key to continue.
```