

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



Non-Preemptive SJF Scheduling

Lab 10



اللهم علمنا ما ينفعنا،، وانفعنا بما علمتنا،، وزدنا علماً



Lab Objective

- To practice the Non-Preemptive SJF scheduling.



SJF Scheduling

- Associate with each process the length of its next CPU burst. Use these length to schedule the process with the shortest time
- Two schemes:
 - *Non-preemptive*: once CPU given to the process it cannot be preempted until completes its CPU burst.
 - *preemptive*: if a new process arrives with CPU burst length less than remaining time of current executing process, preempt.
- SJF is optimal – gives minimum average waiting time for a given set of processes.



Procedure

- Write a C++ program that simulate the **Non-Preemptive SJF** CPU scheduling policy.
- Assume that you have only three processes.
- The inputs to the program are the arrival time and burst time of each process.
- The output of the program are the response time, waiting time, and turnaround time for each of the three process.



Procedure (Cont.)

- The following is a sample run of the program (the underlined numbers are entered by the user who runs the program):

```
what is p1 arrival time 0
what is p1 burst time 7
what is p2 arrival time 2
what is p2 burst time 4
what is p3 arrival time 4
what is p3 burst time 1
process Number1
arrive at 0
waiting Time = 0
response Time= 0
Turnaround Time = 7
process Number3
arrive at 4
waiting Time = 3
response Time= 3
Turnaround Time = 4
process Number2
arrive at 2
waiting Time = 6
response Time= 6
Turnaround Time = 10
total waiting time = 9
average waiting time = 3
```

-----*****SJF *****-----



```
#include <iostream>
using namespace std;

float n, tempb, tempa, tempp, tw, average, gap, arrive[3], burst[3],
process[3], start[3], finish[3], waiting[3], response[3], turnaround[3];

//////////start finish function//////////
void start_finish()
{
    start[0]=arrive[0];
    finish[0]=arrive[0]+burst[0];
    for(int i=1;i<3;i++)
    {
        gap=0;
        if(arrive[i]>finish[i-1])
        {
            gap=arrive[i]-finish[i-1];
            start[i]=finish[i-1]+gap;
        }//end if
        else
            start[i]=finish[i-1];

        finish[i]=start[i]+burst[i];
    }//end for
} //end start_finish function
```



```
int main()
{
    int i,j;

    ////////////////////////////////// Get values from User////////////////////////////////
    for(i=0;i<3;i++)
    { n=i+1;
      process[i]=n;
      cout<<"what is p"<<n<<" arrival time\t";
      cin>>arrive[i];
      cout<<" what is p"<<n<<" burst time\t";
      cin>>burst[i];
    }//end for
```




```
//////////Sort process based on arrival time//////////
```

```
for(i=0;i<2;i++)
  for( j=i+1;j<3;j++)
  {
    if(arrive[j]<arrive[i])
    {
      tempa=arrive[i];
      arrive[i]=arrive[j];
      arrive[j]=tempa;

      tempb=burst[i];
      burst[i]=burst[j];
      burst[j]=tempb;

      tempp=process[i];
      process[i]=process[j];
      process[j]=tempp;
    }//end if
  }//end for
```

```
//////////calculate start and finish times//////////
```

```
start_finish();
```



```
//////////Resort//////////
```

```
for(i=0;i<2;i++)
  for( j=i+1;j<3;j++)
  {
    if(arrive[j]<=start[i] && burst[j]<burst[i])
    {
      tempa=arrive[i];
      arrive[i]=arrive[j];
      arrive[j]=tempa;

      tempb=burst[i];
      burst[i]=burst[j];
      burst[j]=tempb;

      tempc=process[i];
      process[i]=process[j];
      process[j]=tempc;
    }//end if
  }//end for
```

```
//////////calculate start and finish times //////////
```

```
start_finish();
```



```
///calculate response, waiting, turnaround times for each process///
```

```
tw=0;
```

```
for(i=0;i<3;i++)
```

```
{ response[i]= start[i]-arrive[i];
```

```
waiting[i]= start[i]-arrive[i];
```

```
turnaround[i]= finish[i]-arrive[i];
```

```
tw+=waiting[i];
```

```
}//end for
```

```
///calculate average waiting time///
```

```
average= tw/3;
```

```
//////////Display results//////////
```

```
for(i=0;i<3;i++)
```

```
{
```

```
cout<<"process Number"<<process[i]<<' \n'<<"arrive at
```

```
"<<arrive[i]<<' \n'<<"waiting Time = "<<waiting[i]<<' \n'<<"response
```

```
Time= "<<response[i]<<' \n'<<"Turnaround Time =
```

```
"<<turnaround[i]<<' \n' ;
```

```
}
```

```
cout<<"Total waiting time = "<<tw;
```

```
cout<<"\n \n Average waiting time = "<<average;
```

```
cout<<"\n\n\t\t\t\t\t-----*****SJF *****-----\n";
```

```
return(0);
```

```
}//end main
```



Practice

- Write, compile and run a C++ program that simulates the non-preemptive SJF
- The main process creates two threads to perform the task of the function `start_finish`



Check Cases

	Input		Output		
	Arrival	Burst	Response	Waiting	Turnaround
P1	4	2	2	2	4
P2	1	5	0	0	5
P3	3	6	5	5	11
P1	2	4	6	6	10
P2	0	5	0	0	5
P3	5	3	0	0	3
P1	1	3	3	3	6
P2	1	2	0	0	2
P3	2	1	1	1	2
P1	6	3	3	3	6
P2	0	2	0	0	2
P3	3	6	0	0	6



Question?

