**Practical No 6**

**Aim :** Develop, debug and Execute a C program to simulate the Round Robin CPU scheduling algorithms to find turnaround time and waiting time.

**Apparatus:** Computer system with windows installed in it.

Mingw compiler for C/C++, and a text editor for developing C code file (Dev C++).

**Theory :**

**What is RR scheduling?**

* RR is short for ‘Round Robin’ Scheduling algorithm.
* This algorithm is designed especially for time sharing systems.
* It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
* A small unit of time, called a time quantum, or time slice, is defined. A time quantum is generally from 10 to 100ms in length.
* To implement RR scheduling, we treat the ready queue as a FIFO queue of processes.
* The performance of the RR algorithm depends heavily on the size of the time quantum.
* Every process is allocated no more than 1 time quantum in a row, unless it is the only process in the ready queue.
* To implement RR scheduling, we treat the ready queue as a FIFO queue of processes.
* One of the two scenarios can arise when interrupt occurs:
  + The process may have a burst time less than that of 1 time quantum, in this case the process itself will release the CPU voluntarily.
  + If the CPU burst of the current running process is longer than 1 time quantum, the timer will go off and cause an interrupt to the operating system. A context switch will be executed, and the process will be pout at the tail of the ready queue.
  + The CPU scheduler will then select the next process in the ready queue.

**What is a Time Quantum?**

* A time quantum is defined as a small unit of time or a time slice.
* A time quantum is generally from 10 to 100ms in length.
* The performance of the RR algorithm depends heavily on the size of the time quantum.
  + If the quantum is extremely large, the RR policy is the same as the FCFS policy.
  + If the quantum is extremely small (e.g. 1ms), the RR approach can result in a large number of context switches.
* We want the time quantum to be large with respect to the context switch time.
* If the context-switch time is approximately 10 percent of the time quantum, then about 10% of the CPU time will be spent in context switching.
* Most computing systems have a time quantum ranging from 10ms to 100ms.
* The time required for a context switch is typically less than 10 microseconds.
* Thus, the context-switch time is a small fraction of the time quantum.

**Example :**

|  |  |
| --- | --- |
| **Process** | **Burst Time** |
| **P1** | **24** |
| **P2** | **3** |
| **P3** | **3** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P1 | P1 | P1 | P1 |

0 4 7 10 14 18 22 26 30

Waiting time for p1 = (10-4) = 6

Waiting time for p2 = 4

Waiting time for p3 = 7

Average waiting time =p1 + p2 + p3 / 3

= (6 + 4 + 7)/ 3

= 17/4

= 5.67ms

Turnaround time for p1 = 30ms

Turnaround time for p2 = 7ms

Turnaround time for p3 = 10ms

Average turnaround time = 30 + 7 + 10 / 3

= 47 / 3

= 15.67ms

**Code :**

#include<stdio.h>

int main()

{

int i,n,process[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,burstTime=0;

int bt[10],temp,j,at[10],wt[10],tt[10],ta=0,sum=0;

float wavg=0,tavg=0,tsum=0,wsum=0;

printf(" -------Shortest Job First Scheduling ( NonPreemptive )-------\n");

printf("\nEnter the No. of processes :");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\tEnter the burst time of process %d :",i+1);

scanf(" %d",&bt[i]);

printf("\tEnter the arrival time of process %d :",i+1);

scanf(" %d",&at[i]);

}

/\*Sorting the tasks according to Arrival Time\*/

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

if(at[i]<at[j])

{

temp=process[j];

process[j]=process[i];

process[i]=temp;

temp=at[j];

at[j]=at[i];

at[i]=temp;

temp=bt[j];

bt[j]=bt[i];

bt[i]=temp;

}

}

}

for(j=0;j<n;j++)

{

burstTime=burstTime+bt[j];

min=bt[k];

for(i=k;i<n;i++)

{

if (burstTime>=at[i] && bt[i]<min)

{

temp=process[k];

process[k]=process[i];

process[i]=temp;

temp=at[k];

at[k]=at[i];

at[i]=temp;

temp=bt[k];

bt[k]=bt[i];

bt[i]=temp;

}

}

k++;

}

wt[0]=0;

for(i=1;i<n;i++)

{

sum=sum+bt[i-1];

wt[i]=sum-at[i];

wsum=wsum+wt[i];

}

wavg=(wsum/n);

for(i=0;i<n;i++)

{

ta=ta+bt[i];

tt[i]=ta-at[i];

tsum=tsum+tt[i];

}

tavg=(tsum/n);

printf("\n");

printf("\n RESULT:-");

printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around" );

for(i=0;i<n;i++)

{

printf("\n process%d\t %d\t %d\t\t %d\t\t\t%d" ,process[i] ,bt[i] ,at[i] ,wt[i],tt[i]);

}

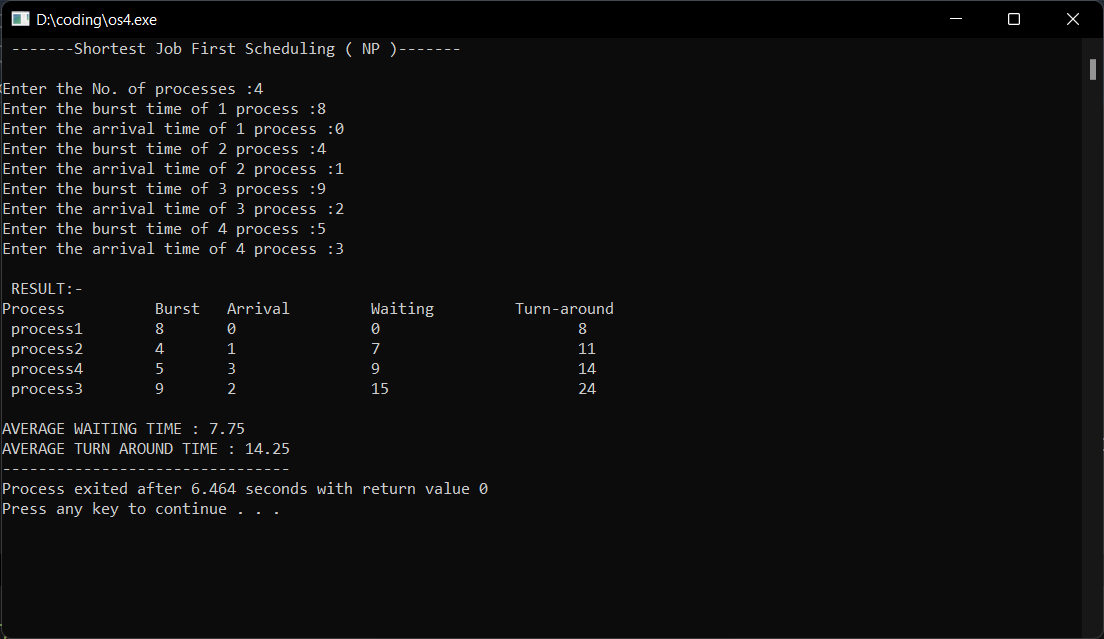
printf("\n\nAVERAGE WAITING TIME : %f",wavg);

printf("\nAVERAGE TURN AROUND TIME : %f",tavg);

return 0;

}

**Output**:



**Conclusion**:

Hence, by performing this practical I got to know about the concept of Shortest job first scheduler algorithm, its advantages, disadvantages, its use, and its implementation. I also developed, debugged and executed a c program to simulate the SJF (nonpreemprtive) CPU scheduling algorithms to find turnaround time and waiting time.