**Question Number 3:**

Consider a scheduler which schedules the job by considering the arrival time of the processes where arrival time if given as 0 is discarded or displayed as error. The scheduler implements the shortest job first scheduling policy, but checks the queue of the processes after the every process terminates and time taken for checking and arranging the process according to the shortest job is 2 time unit. Compute the waiting time, turnaround time and average waiting time and turnaround time of the processes. Also compute the total time taken by the processor to compute all the jobs.

The inputs for the number of requirements, arrival time and burst time should be provided by the user.

Consider the following units for reference.

Process Arrival time Burst Time

1 0 6

2 3 2

3 5 1

4 9 7

5 10 5

6 12 3

7 14 4

8 16 5

9 17 7

10 19 2

Develop a scheduler which submits the processes to the processor in the defined scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time.

**Algorithm for Shortest Job First:**

1. Create array process[] to have total number of processes, bursttime[] to have burst times of the processes, arrivaltime[] to have arrival times of the processes.

2. Arrays process[], bursttime[], arrivaltime[] to be initialized with the total number of processes as specified by the user.

3. Iterate unless arrays process[], bursttime[], arrivaltime[] have values entered by the user.

4. Iterate until the processes gets sorted.

5. Print the remaining process and calculate the waiting and turn around time.

6. Print the calculated average waiting and turnaround time.

Code Snippet:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n,m,i,j=0,p=0;

float avg=0.0,avg1=0.0,t;

printf("Enter the total number of process that is to be executed:");

scanf("%d",&n);

m=n;

struct process

{

int process[n], bursttime[n], arrivaltime[n], waittime, turntime;

} pp;

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

{

printf("\nPlease enter the Arrival time of the process %d :",i);

scanf("%d",&pp.arrivaltime[i]);

if(pp.arrivaltime[i]==0){

printf("\*Wrong arrival time8\*\*\n");

printf("\nEnter again : ");

scanf("%d",&pp.arrivaltime[i]);

}

printf("\nPlease enter the burst time of the process %d :",i);

scanf("%d",&pp.bursttime[i]);

printf("\n\n");

pp.process[i]=i+1;

}

//Sorting of the Given process

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

{

int temp=0,temp1=0,temp2=0;

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

{

if(pp.bursttime[i]>pp.bursttime[j])

{

temp=pp.bursttime[i];

temp1=pp.arrivaltime[i];

temp2=pp.process[i];

pp.bursttime[i]=pp.bursttime[j];

pp.arrivaltime[i]=pp.arrivaltime[j];

pp.process[i]=pp.process[j];

pp.bursttime[j]=temp;

pp.arrivaltime[j]=temp1;

pp.process[j]=temp2;

}

else

{ continue;

}

}

}

pp.turntime=0;

pp.waittime=0;

//Printing the remaining process and calculating the waiting time and turn around time

do{

printf("\n\n");

printf("\n\t========================================================================================================\n");

printf("\n\n\t\t\t\*\*\*\*\*\*\*The Processes Are\*\*\*\*\*\*\*\*\* \n\n");

printf("\n\t\t\tProcess \tArrival time \t Burst time \n");

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

printf("\t\t\tP%d\t\t %d \t\t\t %d\n",pp.process[i],pp.arrivaltime[i],pp.bursttime[i]);\

}

printf("\n\n\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\n\t\t\tProcess p%d is done!!!!\n\n",pp.process[0]);

printf("\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

if(p==0){

pp.turntime=pp.bursttime[0];

pp.waittime=0;

}

else{

t=pp.turntime-pp.arrivaltime[0];

if(t<0.0){

pp.waittime=0.0;

pp.turntime=pp.turntime+pp.bursttime[0];

}

else{

pp.waittime=pp.turntime-pp.arrivaltime[0];

pp.turntime=pp.turntime+pp.bursttime[0];

}

}

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

pp.process[j]=pp.process[j+1];

pp.bursttime[j]=pp.bursttime[j+1];

pp.arrivaltime[j]=pp.arrivaltime[j+1];

}

avg=avg+pp.waittime;

avg1=pp.turntime;

printf("\t\t\tTurn Around time : %f",pp.turntime);

printf("\t\t\t\tWaiting time : %f\n",pp.waittime);

sleep(2);

n=n-1;

p++;

}while(n!=0);

//Showing the average waiting time and Turnaround time

printf("\n\n\n\n");

printf("\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Final Result ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n");

printf("\t===========================================================================================");

avg=avg/m;

printf("\n\n\t\tAverage Waiting time : %f",avg);

avg1=avg1/m;

printf("\t\t\tAverage Turnaround time : %f",avg1);

if(n==0){

printf("\n\n\n\t\t\t\t\t\t\tThank You");

}

return 0;

}

Complexity: O(n2)

Boundary Conditions:

* Job completion time must be known earlier, but it is hard to predict.
* SJF can't be implemented for CPU scheduling for the short term.
* This algorithm may cause very long turnaround times or starvation.
* It leads to the starvation that does not reduce average turnaround time.

Test Cases:

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| 1 | 0 | 6 |
| 2 | 3 | 2 |
| 3 | 5 | 1 |
| 4 | 9 | 7 |
| 5 | 10 | 5 |
| 6 | 12 | 3 |
| 7 | 14 | 4 |
| 8 | 16 | 5 |
| 9 | 17 | 7 |
| 10 | 19 | 2 |

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| 1 | 2 | 2 |
| 2 | 5 | 4 |
| 3 | 9 | 9 |
| 4 | 14 | 4 |
| 5 | 16 | 7 |
| 6 | 19 | 5 |
| 7 | 21 | 7 |
| 8 | 22 | 1 |
| 9 | 25 | 6 |
| 10 | 27 | 3 |