## Operating System Assignment

*Mini Project submitted in partial fulfilment of the requirements for the Degree of*

**BACHELORS OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

By

**ALOK Tiwari**

**11703642**

**K17FB | 09**

Under the Guidance of

**Ms. Milanjit Kaur**

**ASSISSTANT PROFESSOR**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

**Email:**

**aloktiwari1404@gmail.com GitHub Link:**

<https://github.com/aloktiwari1404/operating-sys>

1. **Description of the program:**

When more than one process is runnable, the operating system must decide which

one is first. The part of the operating system concerned with this decision is called

the schedular, and algorithm it uses is called scheduling algorithm, by scheduling

the tasks.

1. **Explain the problem in terms of Operating System Concepts.**

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time

slot in a cyclic way. It is pre-emptive as process are assigned CPU only for a fixed slice of

time at most. This time is given by the user at the run time or is fixed and is known as

**Time Quantum.** This assignment consists of a round robin algorithm with non-preemption.

It asks for number of processes and their **Arrival and Burst time** respectively and

after applying the algorithm, it gives out the **Turn Around Time and Waiting time** for each

process.

1. **Write the algorithm for the proposed solution of the assigned problem.**

**Steps to find Waiting Time:**

1- Create an array rem\_bt[] to keep track of remaining

burst time of processes. This array is initially a

copy of bt[] (burst times array)

2- Create another array wt[] to store waiting times

of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is

not done yet.

a- If rem\_bt[i] > quantum

(i) t = t + quantum

(ii) bt\_rem[i] -= quantum;

c- Else // Last cycle for this process

(i) t = t + bt\_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt\_rem[i] = 0; // This process is over

**Function for finding Average Time :**

**void findavgTime(int processes[], int n, int bt[],**

**int quantum)**

**{**

**int wt[n], tat[n], total\_wt = 0, total\_tat = 0;**

**// Function to find waiting time of all processes**

**findWaitingTime(processes, n, bt, wt, quantum);**

**// Function to find turn around time for all processes**

**findTurnAroundTime(processes, n, bt, wt, tat);**

**// Display processes along with all details**

**cout << "Processes "<< " Burst time "**

**<< " Waiting time " << " Turn around time\n";**

**// Calculate total waiting time and total turn**

**// around time**

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

**{**

**total\_wt = total\_wt + wt[i];**

**total\_tat = total\_tat + tat[i];**

**cout << " " << i+1 << "\t\t" << bt[i] <<"\t "**

**<< wt[i] <<"\t\t " << tat[i] <<endl;**

**}**

**cout << "Average waiting time = "**

**<< (float)total\_wt / (float)n;**

**cout << "\nAverage turn around time = "**

**<< (float)total\_tat / (float)n;**

**}**

1. **Calculate complexity of implemented algorithm.**

The basic complexity of Round Robin Algorithm is O(1) when it is run using dynamic programming but in our programme it is O(n^2) where n is the number of processes entered by the user.

1. **Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.**

#include<stdio.h>

int main()

{

int i, limit, sum = 0, x, count = 0, slice,j;

int wait = 0, tat = 0,pos,z,p[10],prio[10], a\_time[10],

b\_time[10], temp[10],b;

float avgWait, average\_tat;

printf("\nEnter Total Number of Processes:");

scanf("%d", &limit);

x = limit;

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

{

p[i]=i+1;

prio[i]=0;

printf("\nEnter sum Details of Process[%d]\n", i + 1);

printf("Arrival Time:\t");

scanf("%d", &a\_time[i]);

printf("Burst Time:\t");

scanf("%d", &b\_time[i]);

temp[i] = b\_time[i];

}

printf("\nEnter the Time Quantum:");

scanf("%d", &slice);

printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\t Priority\n");

for(sum = 0, i = 0; x != 0;)

{

for(z=0;z<limit;z++)

{

int temp1;

pos=z;

for(j=z+1;j<limit;j++)

{

if(prio[j]<prio[pos])

pos=j;

}

temp1=prio[z];

prio[z]=prio[pos];

prio[pos]=temp1;

temp1=b\_time[z];

b\_time[z]=b\_time[pos];b\_time[pos]=temp1;

temp1=a\_time[z];

a\_time[z]=a\_time[pos];

a\_time[pos]=temp1;

temp1=p[z];

p[z]=p[pos];

p[pos]=temp1;

temp1=temp[z];

temp[z]=temp[pos];

temp[pos]=temp1;

}

{

}

if(temp[i] <= slice && temp[i] > 0)

{

sum = sum + temp[i];

temp[i] = 0;

count = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - slice;

sum = sum + slice;

}

for(b=0;b<limit;b++)

{

if(b==i)

prio[b]+=1;

else

prio[b]+=2;

}

if(temp[i] == 0 && count == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t\t %d\t\t %d\t\t%d", p[i],b\_time[i], sum-a\_time[i], sum-a\_time[i]-b\_time[i],prio[i]);

wait = wait+sum-a\_time[i]-b\_time[i];

tat = tat+sum-a\_time[i];

count = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(a\_time[i + 1] <= sum)

{

i++;}

else

{

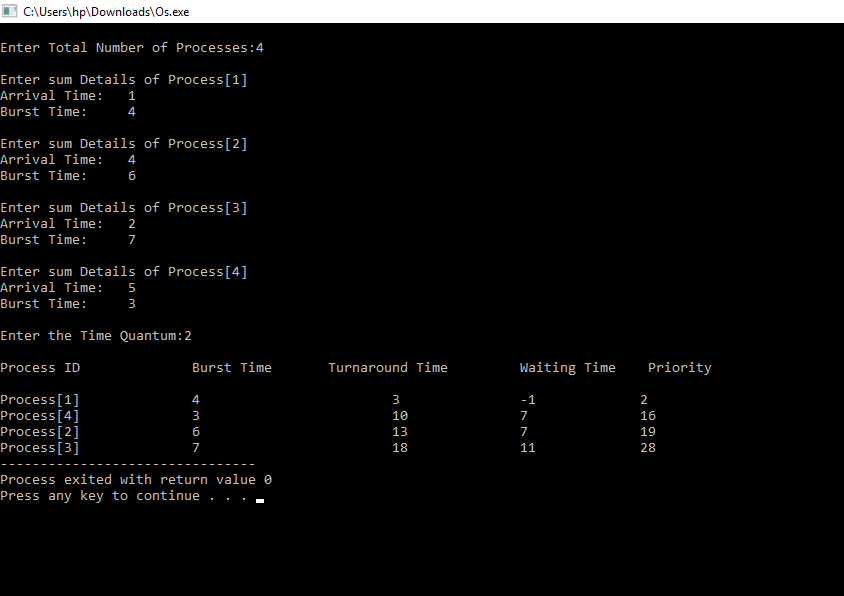
i = 0;

}

}

return 0;

}



1. **Boundary conditions:**
2. If the burst time and arrival time of different processes collides unrationally then negative values/garbage values may occur.
3. If the size of the array index goes out of scope.
4. If the process is getting terminated very early.
5. The boundary conditions for the Round Robin algorithm is to find the Average and Waiting time for processes.
6. On every request, the system determines whether the process entered into the system will get executed or not.
7. The Turn Around Time cannot exceed the Completion Time.

**7. Test cases:**

Case1:

output:enter the number of processes/process

input:User entered the number (integer value)

Case 2:

Output:enter the arrival and burst time of the process \*4 times

Input:user has to enter the value of arrival and burst time for for processes

Case 3:

output:enter the time quantum.

Input:use

**8. Have you made minimum 5 revisions of solution on GitHub?**

Yes I have made minimum 5 revisions of solution on Github.

**GitHub Link:** <https://github.com/aloktiwari1404/operating-sys>