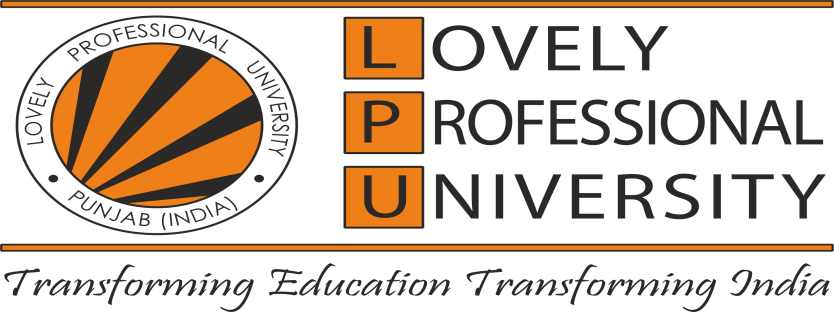
**An Online System on**

**OPERATING SYSTEM**

**Course code: CSE 316**

**Section: K18GE**

****

**Submitted by:-**

Name: Gourav Tanwar

Roll No.: 10

Registration No.: 11811502

**Submitted to:-**

Name of the faculty: Milanjit Kaur

Date of submission of report: 10-April-2020

**Department of Intelligent Systems**

**School of Computer Science Engineering**

**Lovely Professional University, Jalandhar**

APRIL-2020

**Student Name: GOURAV TANWAR**

**Student ID: 11811502**

**Email Address: gouravtanwar4237@gmail.com**

**GitHub Link: https://github.com/Gourav4237/Operating-System.git**

**Code: C language program**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//header files

#include<stdio.h>

typedef struct

{

int arrival\_time,

int burst\_time,queue;

int process\_id;

}process;

int main()

{

int size,quantum,done =0;//Number of processes

int count = 0,i,j;

int time,smallest,waiting\_time[50],turnaround\_time[50],completion[50],temp\_burst[50];

double average\_waiting,average\_turnaround;

printf("\nEnter the number of processes:");

scanf("%d",&size);

process no[size];

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

{

printf("\nProcess ID:\t");

scanf("%d", &no[i].process\_id);

printf("Arrival Time:\t");

scanf("%d", &no[i].arrival\_time);

printf("Burst Time:\t");

scanf("%d", &no[i].burst\_time);

printf("Queue1/Queue2(1/2):\t");

scanf("%d", &no[i].queue);

}

printf("\nEnter the time quantum for Round Robin:");

scanf("%d",&quantum);

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

{

temp\_burst[i] = no[i].burst\_time;//storing the burst time in a array for further use.

}

no[size+1].burst\_time = 999;// assign a max burst for comparison

for(time=0;count!=size;time++) // shortest remaining time first scheduling

{

smallest = size+1;

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

{

if(no[i].arrival\_time<=time && no[i].burst\_time<no[smallest].burst\_time && no[i].burst\_time>0 && no[i].queue == 1)

{

smallest=i;

}

}

if(count >= size/2) //round robin scheduling

{

while(1)

{

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

{

if(no[j].arrival\_time<=time && no[j].burst\_time > quantum && no[j].queue == 2 )

{

no[j].burst\_time -= quantum;

smallest = j;

}

else if(no[j].arrival\_time<=time && no[j].burst\_time < quantum && no[j].queue == 2)

{

no[j].burst\_time = 0;

smallest = j;

goto a;

}

}

}

}

no[smallest].burst\_time--; //decrementing the burst time

a: if(no[smallest].burst\_time == 0)

{

count++;

completion[smallest] = time+1;

turnaround\_time[smallest] = completion[smallest] - no[smallest].arrival\_time;

waiting\_time[smallest] = turnaround\_time[smallest] - temp\_burst[smallest];

}

}

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

printf("\n\nProcess Id\tArrival Time\t Burst Time\t Waiting Time\tTurnaround Time");

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

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

{

printf("\n P%d \t\t%d\t\t%d \t\t%d\t\t%d",no[i].process\_id,no[i].arrival\_time,temp\_burst[i],waiting\_time[i],turnaround\_time[i]);

average\_waiting += waiting\_time[i];

average\_turnaround += turnaround\_time[i];

}

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

printf("\n\nAverage waiting time = %lf\n",average\_waiting/size);

printf("Average Turnaround time = %lf",average\_turnaround/size);

}

**PROBLEM IN TERMS OF OPERATING SYSTEM**

Design a scheduler with multilevel queue having two queues which will schedule the processes on the basis of  pre-emptive shortest remaining processing time first algorithm (SROT) followed by a scheduling in which each process will get 2 units of time to execute. Also note that queue 1 has higher priority than queue 2.  Consider the following set of processes (for reference)with their arrival times and the CPU burst times in milliseconds.

-------------------------------------

Process  Arrival-Time   Burst-Time

-------------------------------------

P1             0              5

P2             1              3

P3             2              3

P4             4              1

-------------------------------------

Calculate the average turnaround time and average waiting time for each process. The input for number of processes  and their arrival time, burst time should be given by the user.

In the given problem we design a scheduler with Multilevel Queue having two queues SORT and Round Robin(RR). These queues schedule the process on the basis of pre-emptive shortest remaining process time first algorithm. It means that the process having the smallest execution time is chosen for the next execution. Although a process with short burst time begins, the current process is removed or pre-empted from execution, and the job which is sorter is executed first. By following the schedulling we have also given that each process will have 2sec time to execute the process. After 2sec CPU will switch to next process. It is also given that Q1 has higher priority then Q2 it means firstly execute Q1 then Q2.

This program contains the concept of MULTILEVEL QUEUE, SHORTEST JOB FIRST, ROUND ROBIN and PRIORITY algorithm.

**ALGORITHM:**

1. Take all the inputs from the user including quantum number.
2. Storing the burst time into an array.
3. Use SHORTEST JOB FIRST concept to solve the process according to their arrival time and smallest burst time.
4. Use ROUND ROBIN for process to come after quantum time complete.
5. After completion of process, select next process which has minimum arrival time and minimum burst time.
6. After completion of all processes, print AVERAGEWAIT TIME and TURNAROUND TIME.

**COMPLEXITY OF THE ALGORITHM:**

The complexity of the scheduling algorithm is O(n).

**CONSTRAINTS**

* Round robin CPU scheduling algorithm cannot be implemented in real time operating system due to high context switching, large waiting time, large response time, large turnaround time and less throughput.
* The proposed algorithm improves all the drawbacks of round robin CPU scheduling algorithm.
* It reduces the problem of starvation and also implement the concept of aging.

**CODE SNIPPET:**

for(time=0;count!=size;time++) // shortest remaining time first scheduling

{  smallest = size+1;  
         for(i=0;i<size;i++)

     {

if(no[i].arrival\_time<=time && no[i].burst\_time<no[smallest].burst\_time && no[i].burst\_time>0 && no[i].queue == 1)

{  
                smallest=i;  
            }  
         }  
  
    if(count >= size/2) //round robin scheduling  
      {  
          while(1)  
            {

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

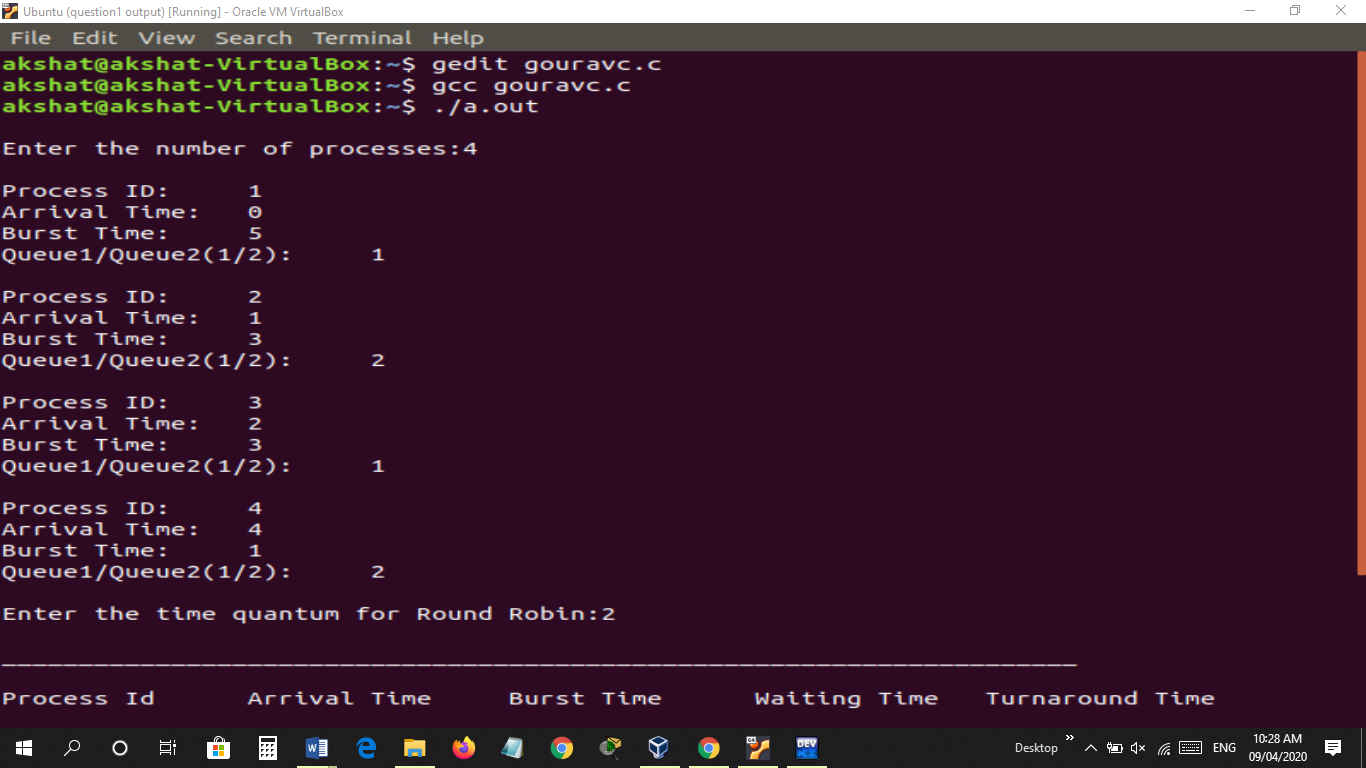
  {  
          if(no[j].arrival\_time<=time && no[j].burst\_time > quantum && no[j].queue == 2 )

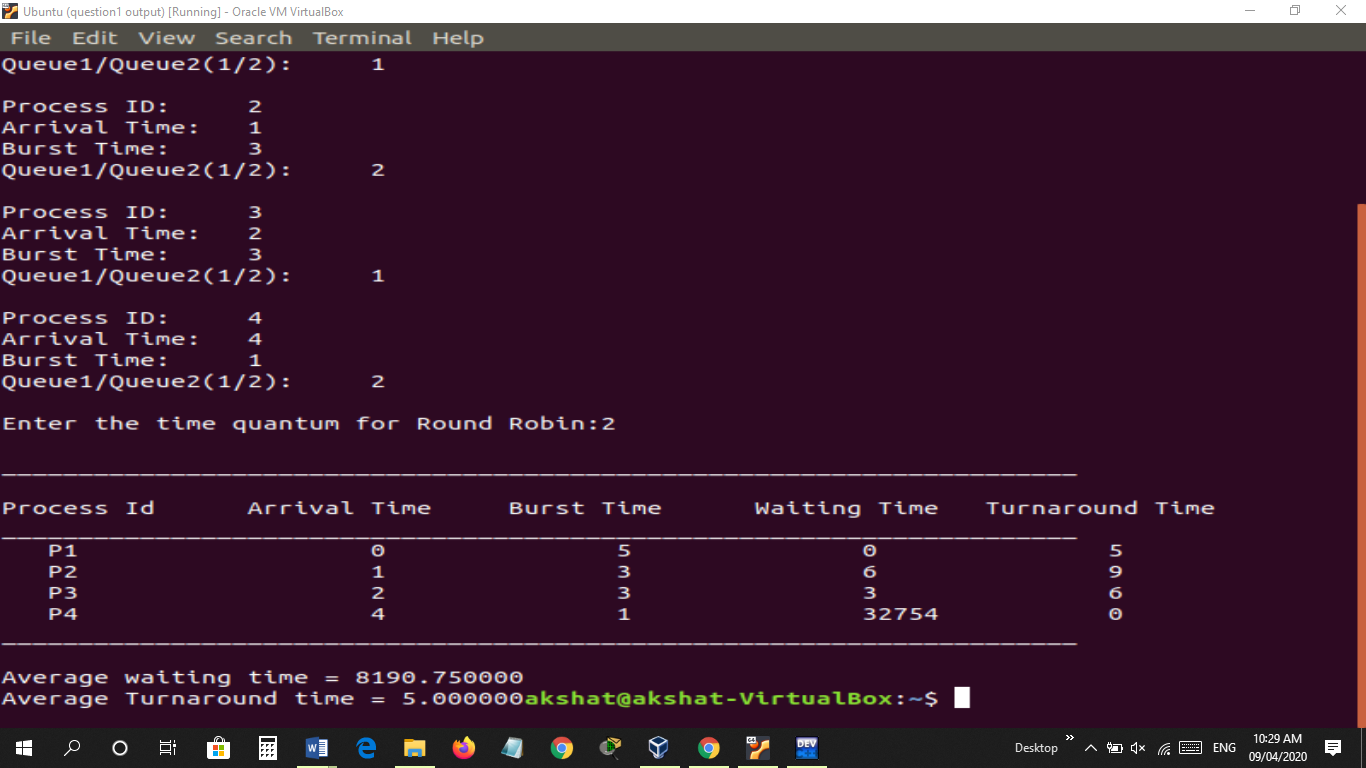
  {  
                      no[j].burst\_time -= quantum;  
                      smallest = j;  
                  }  
                  else if(no[j].arrival\_time<=time && no[j].burst\_time < quantum && no[j].queue == 2)  
                  {  
                      no[j].burst\_time = 0;  
                      smallest = j;  
                      goto a;  
                  }  
              }  
            }  
          }

**BOUNDARY CONDITION:**

* Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
* Process cannot be interrupted until it terminates itself or its time is up.
* If a process with long burst time is running CPU, then later coming process with less CPU burst time may starve.
* It does not have overheads.
* The process is rigid.
* No cost associated.

**TEST CASE**





**EXPLANATION**

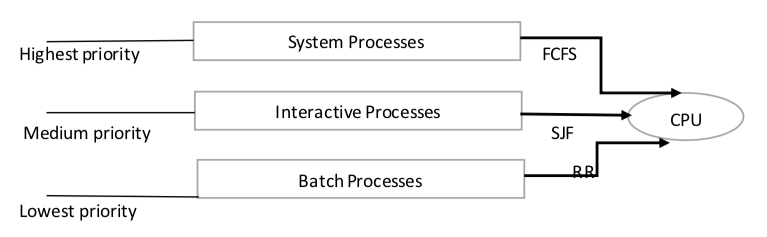
Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

* Multiple queues are maintained for processes with common characteristics.
* Each queue can have its own scheduling algorithms.
* Priorities are assigned to each queue.

#### Example of MultiLevel Queue Scheduling

Here, we will take 3 different types of processes called system processes, interactive processes and Batch processes. All the three processes have their own queue. Kindly look at below figure.



* System process has the highest priority. If an interrupt is generated in the system, then the Operating system stops all the processes and handle the interrupt. According to different response time requirements, process needs different scheduling algorithm.
* Interactive process has medium priority. If we are using VLC player, we directly interact with the application. So all these processes will come in an interactive process. These queues will use scheduling algorithm according to requirement.
* Batch processes has the lowest priority. The processes which run automatically in the background comes under the batch processes. It is also known as background processes.

Therefore, according to process priority and type, the processes are scheduled with different scheduling algorithm.

– Multilevel queue scheduling must be different in scheduling among the processes and generally are implemented as fixed preemptive priority scheduling.

– Another case is if there are no processes in system and interactive queues. Suppose we are executing batch processes in the queue and all were empty initially. Assume that a new process enters either the system or interactive queue. Then, we have to preempt batch process queue scheduling wherein we have to give high priority to the queue which are above the batch queue.