**Student Name :** Mochi Raviteja

**Student ID :** 11601732

**Email Address:** [mochiraviteja@gmail.com](mailto:mochiraviteja@gmail.com)

**Section :** K1616

**Roll No :** A08

Ques. 14. Write a program to implement priority scheduling algorithm with context switching time. Prompt to user to enter the number of processes and then enter their priority, burst time and arrival time also. Now whenever operating system preempts a process and shifts cpu’s control to some another process of higher priority assume that it takes 2 seconds for context switching(dispatcher latency).Form a scenario, where we can give the processes are assigned with priority where the lower integer number is higher priority and then context switch .. as the process waits the priority of the process increase at rate of one per 2 time units of wait. Calculate waiting time and turnaround time for each process.

# Explanation:

1. In the first we sort the processes according to the arrival time
2. Next,we calculated the over all burst time of the processes
3. Then,we take the time as initial arrival time and run loop upto total burst time
4. In the loop we applied another loop to find the highest prority of the processes that are in ready queue state
5. Every time in the loop we check whether the processes in the ready queue state has higher priority if it then we context switch
6. Atlast,we find the turnaround time and waiting time the loop
7. Finally,we print all the values

# ALGORITHM FOR QUESTION NO:14

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Call \_sortArrival()

Set time=process\_queue[0].arrivaltime

For i=0 to n,then

Burst+=process\_queue[i].burst\_time

For time to time<=Burst,then

Set largest = 9

For i=0 to n,then

If process\_queue[i].arrival\_time <= time and process\_queue[i].priority >

process\_queue[largest].priority and process\_queue[i].status != 1 ,then

set largest=i

if process\_queue[largest].burst\_time==0 && process\_queue[i].status != 1,then

Set process\_queue[i].status = 1;process\_queue[largest].priority=0;

Set process\_queue[largest].ct=time;

Set i=-1;

Set time++ and set process\_queue[largest].burst\_time

For i=0 to i<n,then

process\_queue[i].turnaround\_time=(process\_queue[i].ct)-

(process\_queue[i].arrival\_time);a=a+process\_queue[i].turnaround\_time;

process\_queue[i].waiting\_time=process\_queue[i].turnaround\_time

-x[i]);b=b+process\_queue[i].waiting\_time

Exit

# Complexity of the code:

1) For the sorting we have the complexity in worst case scenario is O(n2)

2)For the loop of for calculating the complete time is came to be as O(totalbursttime)

3)For calculating the turnaround time and waiting time we have O(n)

4)we have the complexity of the overall algorithm is O(totalbursttime\*n)

# Code for the above algorithm:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<stdio.h>

int n;

struct process

{

char process\_name;

int arrival\_time, burst\_time, ct, waiting\_time, turnaround\_time, priority;

int status;

}progress[10];

int main()

{

int i;

int time = 0, burst\_time = 0, largest,x[10];double a,b;

char c;

float wait\_time = 0, turnaround\_time = 0, average\_waiting\_time, average\_turnaround\_time;

printf("enter the no. of processes\n");

scanf("%d",&n);

for(i = 0, c = 'P'; i < n; i++, c++)

{

progress[i].process\_name = c;

printf("\nEnter Details For Process[%C]:\n", progress[i].process\_name);

printf("Enter Arrival Time:\t");

scanf("%d", &progress[i].arrival\_time );

printf("Enter Burst Time:\t");

scanf("%d", &progress[i].burst\_time);x[i]=progress[i].burst\_time;

printf("Enter Priority:\t");

scanf("%d", &progress[i].priority);

progress[i].status = 0;

burst\_time = burst\_time + progress[i].burst\_time;

}

struct process temp;

int i1, j;

for(i1 = 0; i1 < n - 1; i1++)

{

for(j = i1 + 1; j < n; j++)

{

if(progress[i1].arrival\_time > progress[j].arrival\_time)

{

temp = progress[i1];

progress[i1] = progress[j];

progress[j] = temp;

}

}

}

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

{

x[i]=progress[i].burst\_time;

}

progress[9].priority = -9999;

for(time = progress[0].arrival\_time; time <= burst\_time+1;)

{

largest = 9;

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

{

if(progress[i].arrival\_time <= time && progress[i].priority > progress[largest].priority && progress[i].status != 1)

{

largest = i;

}

if(progress[largest].burst\_time==0 && progress[i].status != 1)

{

progress[i].status = 1;progress[largest].priority=0;

progress[largest].ct=time;

i=-1;

}

}

time++;(progress[largest].burst\_time)--;

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

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

{

progress[i].turnaround\_time=(progress[i].ct)-(progress[i].arrival\_time);a=a+progress[i].turnaround\_time;

progress[i].waiting\_time=(progress[i].turnaround\_time)-(x[i]);b=b+progress[i].waiting\_time;

} printf("PNO\tAT\tCT\tTA\t\WTt\n");

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

{

printf("%c\t%d\t%d\t%d\t%d\n",

progress[i].process\_name,

progress[i].arrival\_time,

progress[i].ct,

progress[i].turnaround\_time,

progress[i].waiting\_time);

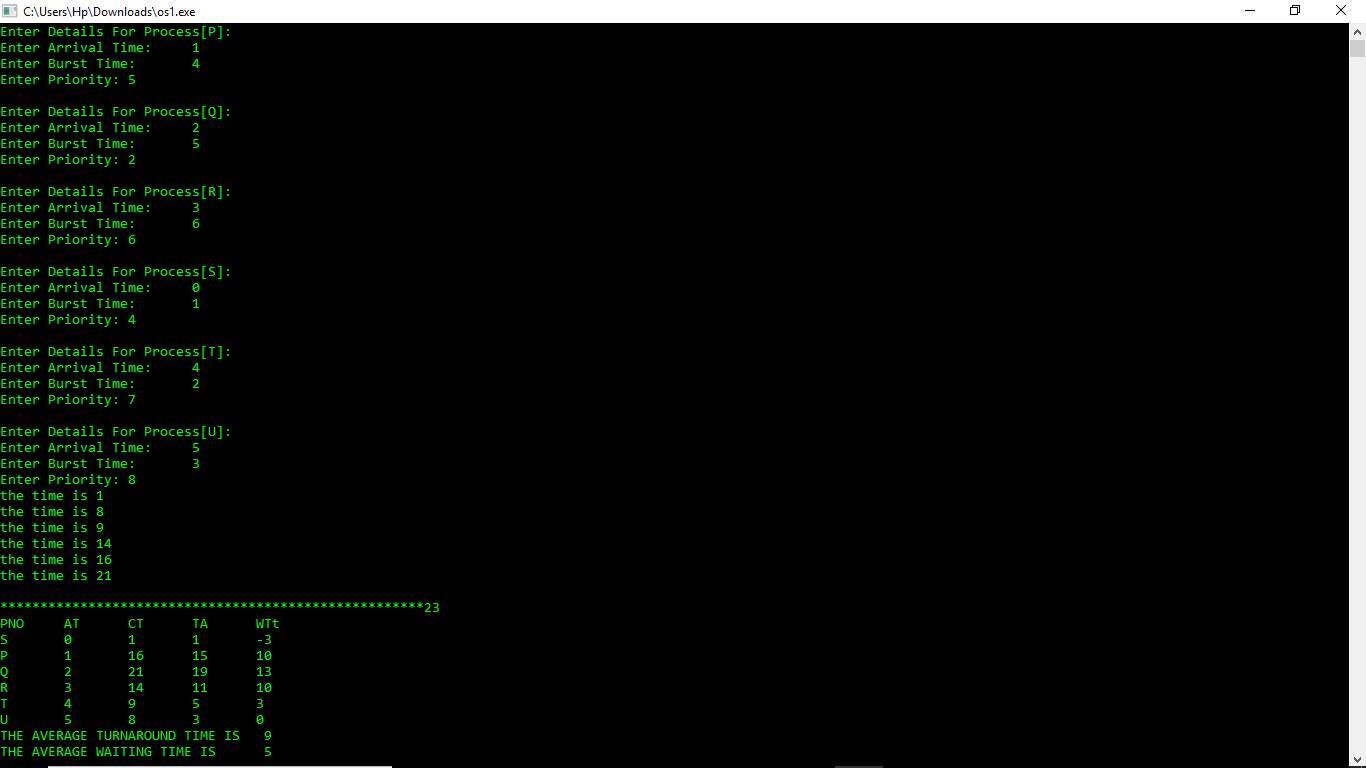
}

printf("THE AVERAGE TURNAROUND TIME IS \t %f\n\n\n",a/n);

printf("THE AVERAGE WAITING TIME IS \t %f\n",b/n);printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

# Test cases:



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Process no | | process name | | Arrival time | | Burst time | | | prority |
| 1 | | p | | 1 | | 4 | | | 5 |
| 2 | | q | | 2 | | 5 | | | 2 |
| 3 | | r | | 3 | | 6 | | | 6 |
| 4 | | s | | 0 | | 1 | | | 4 |
| 5 | | t | | 4 | | 2 | | | 7 |
| 6 | u | | 5 | | 3 | | | 8 | |
|  | | | | | | | | | | | |
|  | | | | | | |

Ques. 15. A uniprocessor system has n number of CPU intensive processes, each process has its own requirement of CPU burst. The process with lowest CPU burst is given the highest priority. A late-arriving higher priority process can pre-empt a currently running process with lower priority. Simulate a scheduler that is scheduling the processes in such a way that higher priority process is never starved due to the execution of lower priority process. What should be its average waiting time and average turnaround time if no two processes are arriving are arriving at same time.

1. **Description :**

It is a Priority Scheduling Algorithm, there are n number of processes each having different burst times, priorities and arrival times. Any late arriving higher priority process will pre-empt the current running process and keep its state waiting, so that the higher priority process will be running. It repeats until the n processes completes their execution.

1. **Algorithm :**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<malloc.h>

#include<time.h>

#include<pthread.h>

#include<semaphore.h>

struct Process {

int priority,Rtime,Atime,Btime,id;

clock\_t arrival;

int flag,completed,Arrival\_Flag;

sem\_t se;

struct Process \*next;

};

int i=0,z=1,l=1;

float WT=0,TAT=0;

clock\_t Start\_Time,count;

struct Process \*front=NULL,\*temp,\*Front\_P=NULL;

typedef struct Process node;

void \*processor(node \*S) {

int Preemtion\_Flag=0,Null\_Flag=0;

while(1) {

sem\_wait(&S->se);

if((S->Atime<=(clock()-Start\_Time)/CLOCKS\_PER\_SEC && S->Arrival\_Flag==1) || Preemtion\_Flag==1) {

S->Arrival\_Flag=0;

Preemtion\_Flag=0;

count=clock();

}

if(S->flag==1) {

printf("\nProcess-%d Running \t\t\t\t\tTimer :%d",S->id,(clock()-Start\_Time)/CLOCKS\_PER\_SEC);

S->flag=0;

S->arrival=clock();

}

if((clock()-count)/CLOCKS\_PER\_SEC==1) {

count=clock();

printf("\nProcess-%d Running\t\t\t\t\tTimer :%d",S->id,(clock()-Start\_Time)/CLOCKS\_PER\_SEC);

S->Rtime-=1;

if(S->Rtime==0) {

TAT+=(clock()-Start\_Time)/CLOCKS\_PER\_SEC-S->Atime;

WT+=((clock()-Start\_Time)/CLOCKS\_PER\_SEC)-S->Atime-S->Btime;

if(Front\_P->next!=NULL) {

Null\_Flag=1;

if(Front\_P->Atime==(clock()-Start\_Time)/CLOCKS\_PER\_SEC){

Front\_P->next=Front\_P->next->next;

}

else {

Front\_P=Front\_P->next;

}

sem\_post(&Front\_P->se);

}

S->completed=1;

printf("\nProcess-%d completed executing, ",S->id);

if(Front\_P->next==NULL && Null\_Flag==0){

Front\_P=NULL;

l=1;

}

else{

printf("next Process is: %d",Front\_P->id);

}

}

else if(Front\_P!=S) {

printf("\nProcess-%d Context Switched to Process-%d",S->id,Front\_P->id);

Preemtion\_Flag=1;

sem\_post(&Front\_P->se);

sem\_wait(&S->se);

}

}

if(S->completed==1) {

break;

}

sem\_post(&S->se);

}

}

void Ppush(node \*temp) {

node \*Start=Front\_P;

if(Front\_P==NULL) {

Front\_P=temp;

Front\_P->next=NULL;

}

else{

int p=temp->priority;

if (Start->priority > p) {

temp->next = Front\_P;

Front\_P=temp;

}

else {

while (Start->next != NULL && Start->next->priority< p) {

Start = Start->next;

}

temp->next = Start->next;

Start->next = temp;

}

}

}

void push() {

temp=(node \*)malloc(sizeof(node));

printf("\nEnter Priority of %d Process :",(i+1));

scanf("%d",&temp->priority);

printf("Enter Arrival Time :");

scanf("%d",&temp->Atime);

int p=temp->priority;

temp->id=i+1;

printf("Enter Burst Time :");

scanf("%d",&temp->Rtime);

temp->Btime=temp->Rtime;

sem\_init(&temp->se,0,0);

int t=temp->Atime;

temp->flag=1;

temp->Arrival\_Flag=1;

temp->completed=0;

node \*Start=front;

if ((Start->Atime > t) || (Start->Atime==t && Start->priority > temp->priority)){

temp->next = front;

front=temp;

}

else {

while (Start->next != NULL && Start->next->Atime <= t) {

if(Start->next->Atime==t && temp->priority<Start->next->priority) {

break;

}

else

Start = Start->next;

}

temp->next = Start->next;

Start->next = temp;

}

}

void main() {

int n,m=1;

pthread\_t p[10];

printf("Enter No.of Processes :");

scanf("%d",&n);

while(i<n) {

if(front==NULL) {

front=(node \*)malloc(sizeof(node));

printf("Enter Priority of %d Process :",(i+1));

scanf("%d",&front->priority);

printf("Enter Arrival Time :");

scanf("%d",&front->Atime);

printf("Enter Burst Time :");

scanf("%d",&front->Rtime);

front->flag=1;

front->id=i+1;

front->Arrival\_Flag=1;

front->completed=0;

front->Btime=front->Rtime;

sem\_init(&front->se,0,0);

front->next=NULL;

}

else {

push();

}

i++;

}

Start\_Time=clock();

count=clock();

i=0;

printf("\n\t\t\t\t\t\t\tTimer :0");

while(front!=NULL) {

temp=front;

if(temp->Atime<0) {

printf("\nInvalid Arrival Time for Process-%d",temp->id);

i++;

front=front->next;

}

if((clock()-Start\_Time)/CLOCKS\_PER\_SEC==temp->Atime) {

if(l==1) {

l=0;

sem\_post(&temp->se);

}

pthread\_create(&p[i],NULL,processor,temp);

front=front->next;

Ppush(temp);

i++;

}

if(((clock()-count)/CLOCKS\_PER\_SEC==1 && Front\_P==NULL)) { //TIMER PRINTS

count=clock();

m=0;

printf("\nNo Process is Running\t\t\t\t\tTimer :%d",(clock()-Start\_Time)/CLOCKS\_PER\_SEC);

}

}

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

pthread\_join(p[i],NULL);

}

printf("\nAverage Waiting Time :%f\nAverage Turn Around Time :%f",(float)WT/n,(float)TAT/n);

}

1. **Description (Purpose Of Use) :**

Complexity when taking input : O(n)

Complexity When Creating Processes : O(n)

Complexity When Running the Processes :O(t)

Where ‘n’ is number of Processes

And ‘t’ is Burst time of the processes

TOTAL TIME COMPLEXITY : = O(n) + O(n) + n \* O(t)

= 2 \* O(n) + n \* O(t)

= O(n) + O(t)

= O(n+t)

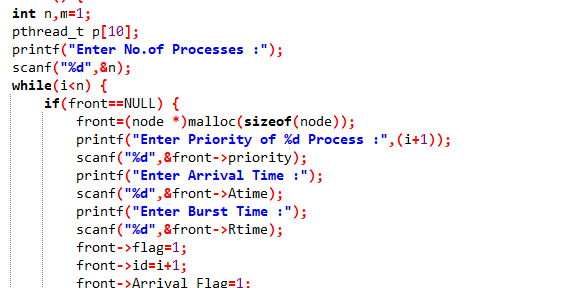
1. **Code Snippet :**

0 < N < 10^6 Where N is Number of Processes

0 < Atime < 10^6 Where ATime is Arrival Time of any Process

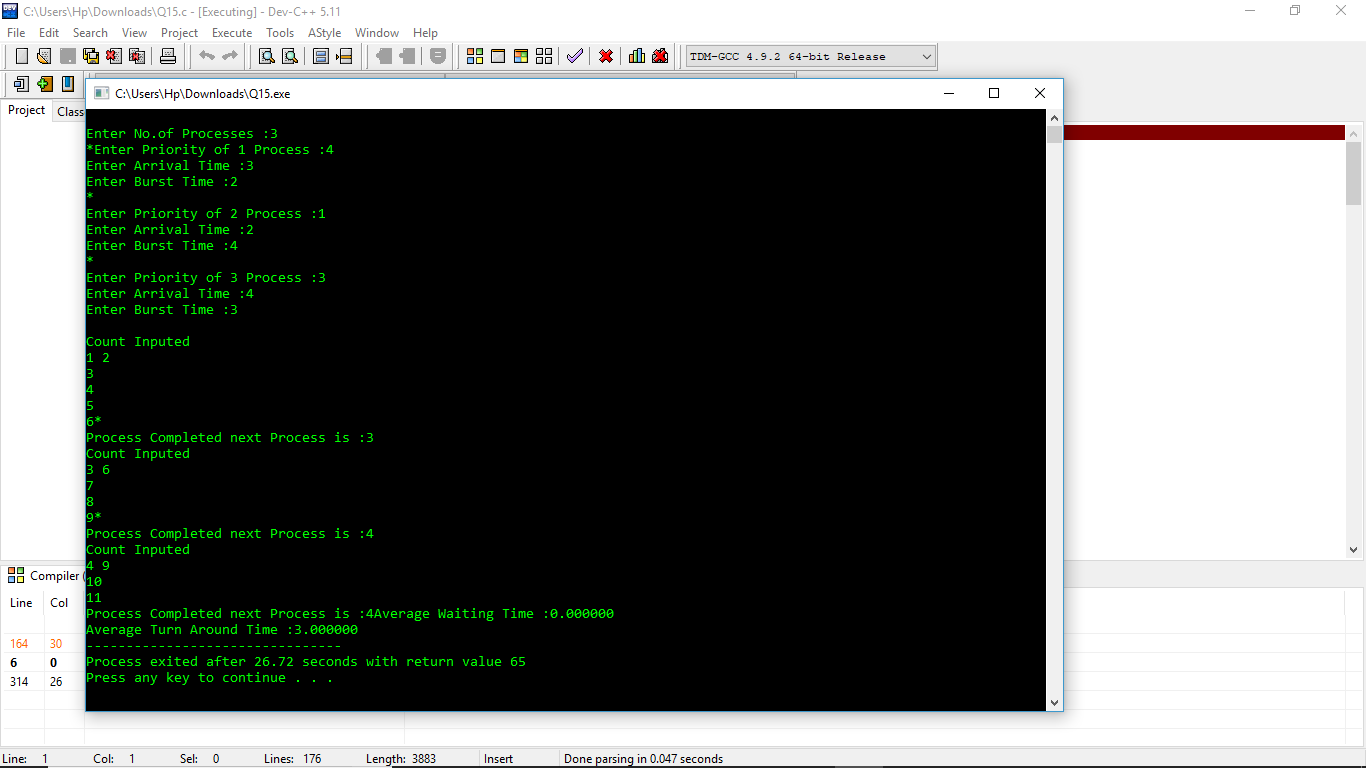
-10^6 < Priority < 10^6

0 < Burst Time < 10^6

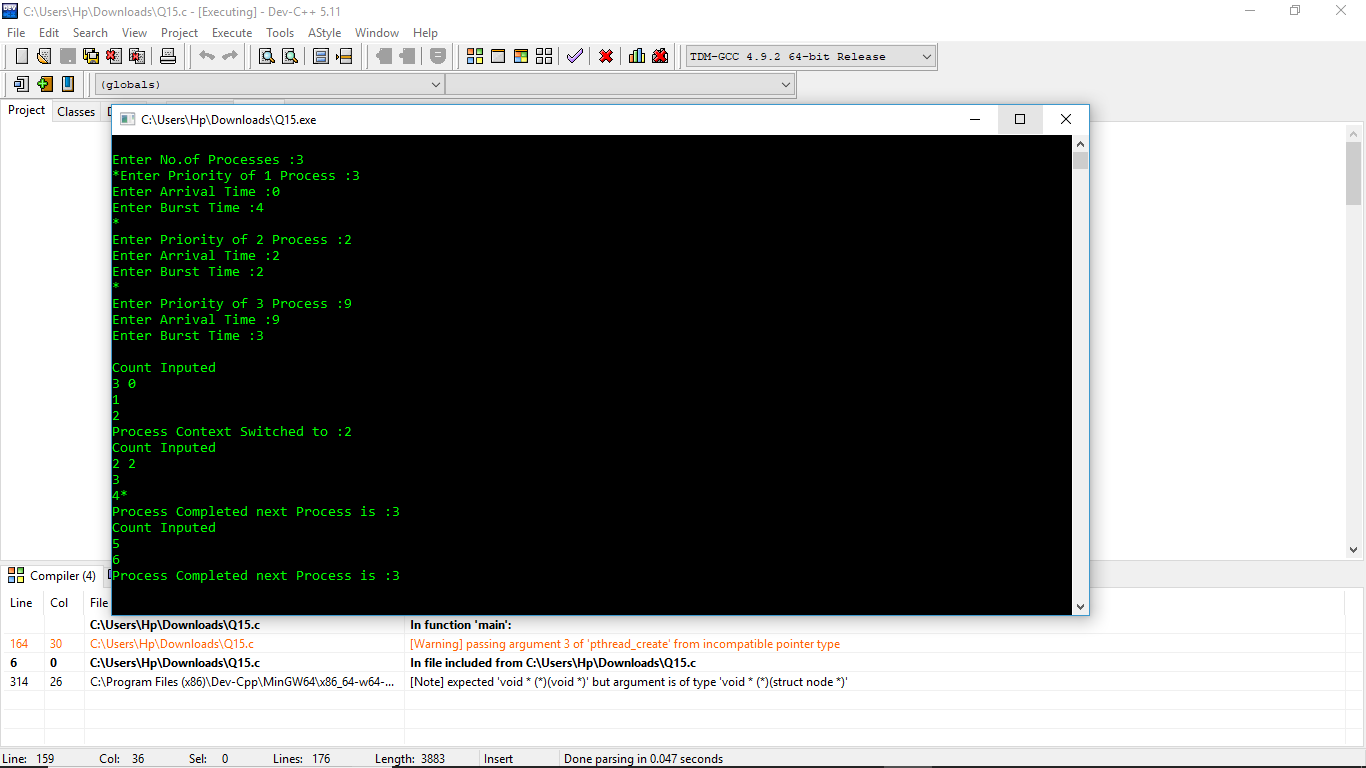


1. **Test Cases :**

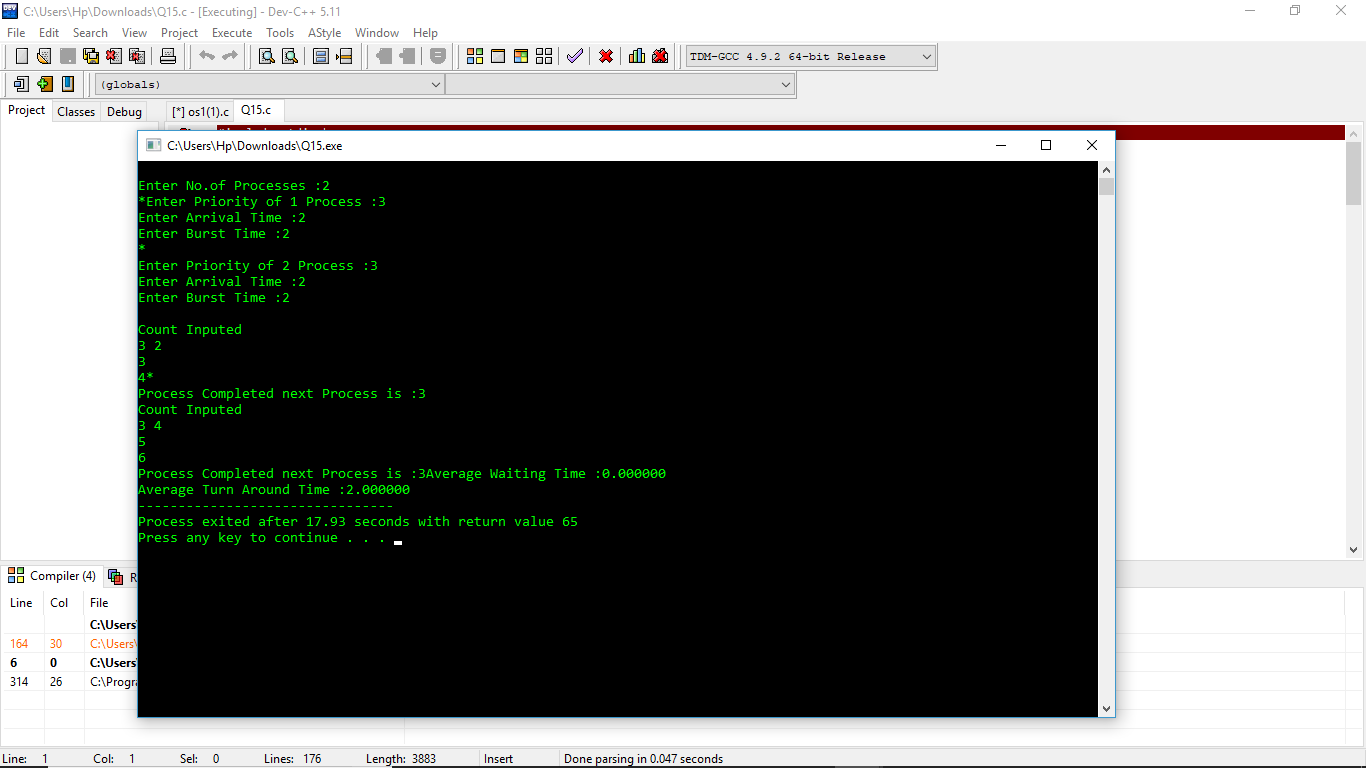
When all processes are running at different Arrivall time

****

When few Processes run and Complete their Execution and remaining processes are waiting in Queue Until they got their arrival time to execute



When all processes are having same priority



1. **Boundary Conditions :**
2. If Arrival time is less than 0 then that process would not be Created and it will be discarded.
3. Priority can be Negative.
4. Burst Time Shouldn’t be Negative.
5. N Cannot be Negative.
6. **Description :**

I Used Threads to Run Processes and Semaphores to Avoid Process Context Switching.

1. **GitHub Link :**

https://github.com/mochiraviteja