# 1.Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.

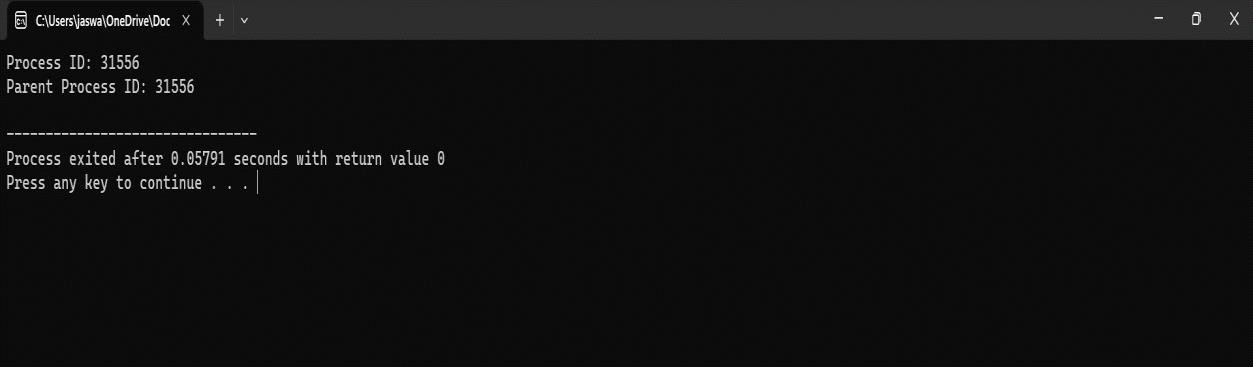
# PROGRAM:

#include<stdio.h> #include<unistd.h> int main()

{

printf("Process ID: %d\n", getpid() ); printf("Parent Process ID: %d\n", getpid() ); return 0;

}

 Output:

# Identify the system calls to copy the content of one file to another and illustrate the same using a C program

## PROGRAM:

#include <stdio.h> #include <stdlib.h>

int main()

{

FILE \*fptr1, \*fptr2; char filename[100], c;

printf("Enter the filename to open for reading \n"); scanf("%s", filename);

fptr1 = fopen(filename, "r"); if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename); exit(0);

}

printf("Enter the filename to open for writing \n"); scanf("%s", filename);

fptr2 = fopen(filename, "w"); if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename); exit(0);

}

c = fgetc(fptr1); while (c != EOF)

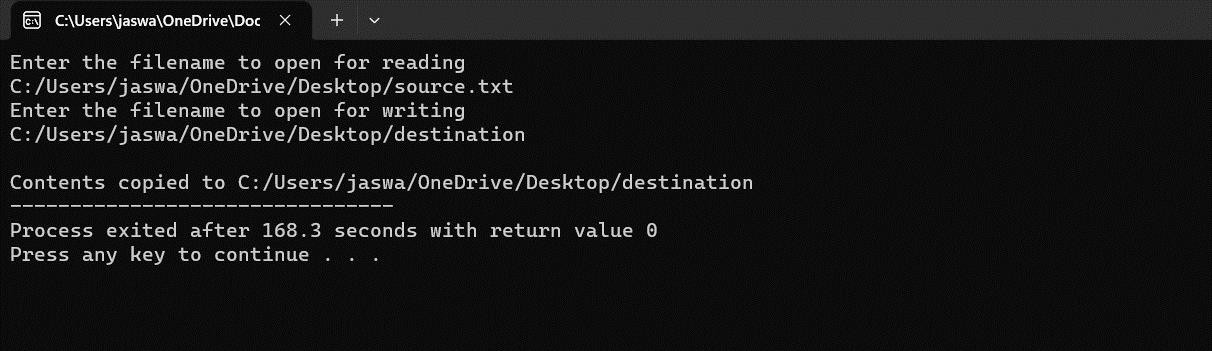
{

fputc(c, fptr2); c = fgetc(fptr1);

}

printf("\nContents copied to %s", filename);

fclose(fptr1); fclose(fptr2); return 0;

}

# Design a CPU scheduling program with C using First Come First Served technique with the following considerations.

1. **All processes are activated at time 0.**

# Assume that no process waits on I/O devices.

## PROGRAM:

#include <stdio.h> int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp; float avg\_wt, avg\_tat;

printf("Enter number of process: "); scanf("%d", &n);

printf("Enter Burst Time:\n"); for (i = 0; i < n; i++) {

printf("P%d: ", i + 1); scanf("%d", &A[i][1]); A[i][0] = i + 1;

}

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

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

if (A[j][1] < A[index][1]) index = j;

temp = A[i][1]; A[i][1] = A[index][1]; A[index][1] = temp;

temp = A[i][0]; A[i][0] = A[index][0]; A[index][0] = temp;

}

A[0][2] = 0;

for (i = 1; i < n; i++) { A[i][2] = 0;

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

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n; total = 0;

printf("P BT WT TAT\n");

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

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],A[i][1],

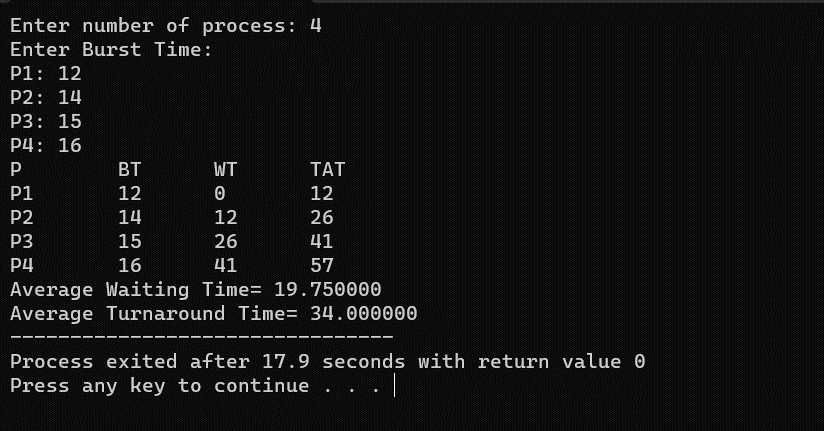
A[i][2], A[i][3]);

}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt); printf("\nAverage Turnaround Time= %f", avg\_tat);

}



# Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.

## PROGRAM:

#include<stdio.h> int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp; float avg\_wt,avg\_tat;

printf("Enter number of process:"); scanf("%d",&n);

printf("nEnter Burst Time:\n"); for(i=0;i<n;i++)

{

printf("p%d:",i+1);

scanf("%d",&bt[i]); p[i]=i+1;

}

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

{

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

{

if(bt[j]<bt[pos]) pos=j;

}

temp=bt[i]; bt[i]=bt[pos]; bt[pos]=temp;

temp=p[i]; p[i]=p[pos]; p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n; total=0;

printf("nProcesst Burst Time tWaiting TimetTurnaround Time\n"); for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i]; total+=tat[i];

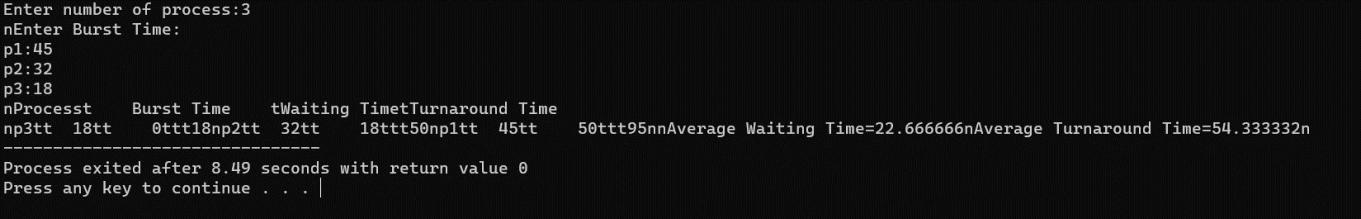
printf("np%dtt %dtt %dttt%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total/n;

printf("nnAverage Waiting Time=%f",avg\_wt); printf("nAverage Turnaround Time=%fn",avg\_tat);

}



## **Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.**

## PROGRAM:

#include<stdio.h> int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp; float avg\_wt,avg\_tat;

printf("Enter number of process:"); scanf("%d",&n);

printf("nEnter Burst Time:\n"); for(i=0;i<n;i++)

{

printf("p%d:",i+1);

scanf("%d",&bt[i]); p[i]=i+1;

}

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

{

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

{

if(bt[j]<bt[pos]) pos=j;

}

temp=bt[i]; bt[i]=bt[pos]; bt[pos]=temp;

temp=p[i]; p[i]=p[pos]; p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n; total=0;

printf("nProcesst Burst Time tWaiting TimetTurnaround Time\n"); for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i]; total+=tat[i];

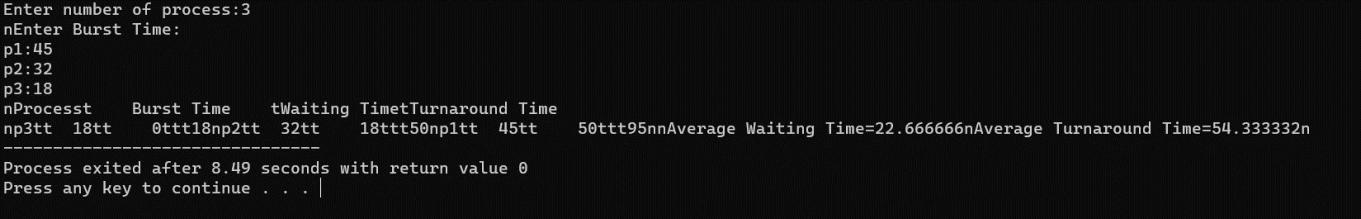
printf("np%dtt %dtt %dttt%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total/n;

printf("nnAverage Waiting Time=%f",avg\_wt); printf("nAverage Turnaround Time=%fn",avg\_tat);

}



## Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

## Program:-

#include<stdio.h>

struct priority\_scheduling { char process\_name;

int burst\_time; int waiting\_time;

int turn\_around\_time; int priority;

};

int main() {

int number\_of\_process; int total = 0;

struct priority\_scheduling temp\_process; int ASCII\_number = 65;

int position;

float average\_waiting\_time; float average\_turnaround\_time;

printf("Enter the total number of Processes: "); scanf("%d", & number\_of\_process);

struct priority\_scheduling process[number\_of\_process]; printf("\nPlease Enter the Burst Time and Priority of each process:\n"); for (int i = 0; i < number\_of\_process; i++) {

process[i].process\_name = (char) ASCII\_number;

printf("\nEnter the details of the process %c \n", process[i].process\_name);

printf("Enter the burst time: "); scanf("%d", & process[i].burst\_time); printf("Enter the priority: "); scanf("%d", & process[i].priority); ASCII\_number++; }

for (int i = 0; i < number\_of\_process; i++) { position = i;

for (int j = i + 1; j < number\_of\_process; j++) {

if (process[j].priority > process[position].priority) position = j; }

temp\_process = process[i]; process[i] = process[position]; process[position] = temp\_process; } process[0].waiting\_time = 0;

for (int i = 1; i < number\_of\_process; i++) { process[i].waiting\_time = 0;

for (int j = 0; j < i; j++) {

process[i].waiting\_time += process[j].burst\_time; } total += process[i].waiting\_time; }

average\_waiting\_time = (float) total / (float) number\_of\_process;

total = 0;

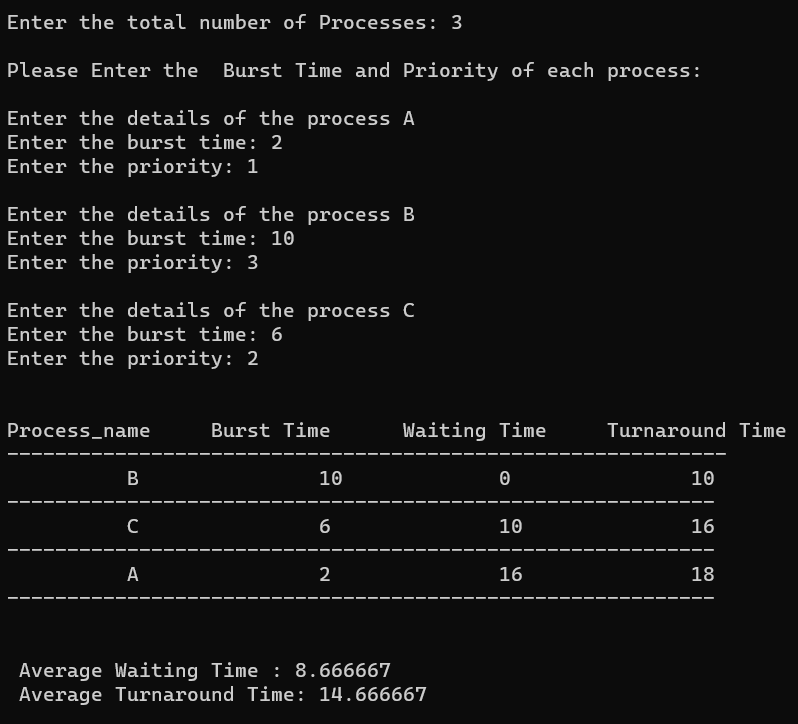
printf("\n\nProcess\_name \t Burst Time \t Waiting Time \t Turnaround Time\n");

printf(" \n"); for (int i = 0; i < number\_of\_process; i++) {

process[i].turn\_around\_time = process[i].burst\_time + process[i].waiting\_time;

printf("\t %c \t\t %d \t\t %d \t\t %d", process[i].process\_name, process[i].burst\_time, process[i].waiting\_time, process[i].turn\_around\_time);

printf("\n \n"); } average\_turnaround\_time = (float) total / (float) number\_of\_process; printf("\n\n Average Waiting Time : %f", average\_waiting\_time); printf("\n Average Turnaround Time: %f\n", average\_turnaround\_time); return 0;



## Construct a C program to simulate Round Robin scheduling algorithm with C.

**Program:-** #include<stdio.h> #include<conio.h> int main()

{

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10]; float avg\_wt, avg\_tat;

printf(" Total number of process in the system: "); scanf("%d", &NOP);

y = NOP;

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1); printf(" Arrival time is: \t");

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

printf(" \nBurst time is: \t"); scanf("%d", &bt[i]); temp[i] = bt[i];

}

printf("Enter the Time Quantum for the process: \t"); scanf("%d", &quant);

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time "); for(sum=0, i = 0; y!=0; )

{

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

{

sum = sum + temp[i]; temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant; sum = sum + quant;

}

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

{

y--;

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum- at[i], sum-at[i]-bt[i]);

wt = wt+sum-at[i]-bt[i]; tat = tat+sum-at[i]; count =0;

}

if(i==NOP-1)

{

i=0;

}

else if(at[i+1]<=sum)

{

i++;

}

else

{

i=0;

}

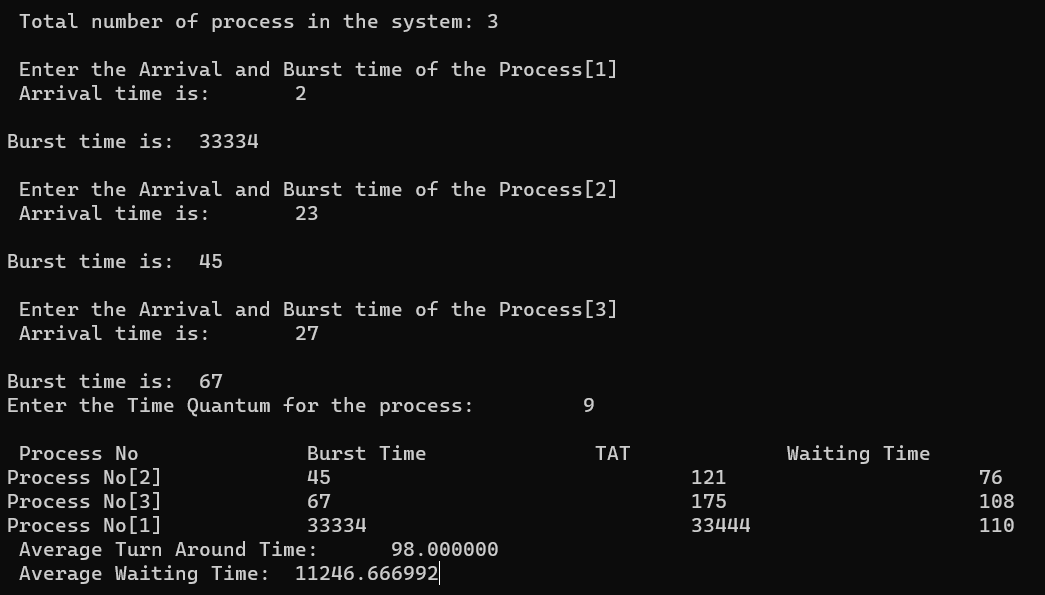
}

avg\_wt = wt \* 1.0/NOP; avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt); printf("\n Average Waiting Time: \t%f", avg\_tat); getch();

}

**Output:-**



# Construct a C program to implement non-preemptive SJF algorithm

## PROGRAM:

#include<stdio.h> int main()

{

int at[10],bt[10],pr[10]; int

n,i,j,temp,time=0,count,over=0,sum\_wait=0,sum\_turnaround=0,start;

float avgwait,avgturn;

printf("Enter the number of processes\n"); scanf("%d",&n);

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

{

%d\n",i+1);

}

printf("Enter the arrival time and execution time for process

scanf("%d%d",&at[i],&bt[i]); pr[i]=i+1;

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

{

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

{

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

{

temp=at[i]; at[i]=at[j]; at[j]=temp; temp=bt[i];

bt[i]=bt[j]; bt[j]=temp; temp=pr[i]; pr[i]=pr[j]; pr[j]=temp;

}

}

}

printf("\n\nProcess\t|Arrival time\t|Execution time\t|Start time\t|End time\t|waiting time\t|Turnaround time\n\n");

while(over<n)

{

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

{

if(at[i]<=time) count++;

else break;

}

if(count>1)

{

for(i=over;i<over+count-1;i++)

{

for(j=i+1;j<over+count;j++)

{

if(bt[i]>bt[j])

{

temp=at[i]; at[i]=at[j]; at[j]=temp; temp=bt[i]; bt[i]=bt[j]; bt[j]=temp; temp=pr[i];

pr[i]=pr[j]; pr[j]=temp;

}

}

}

}

start=time; time+=bt[over];

printf("p[%d]\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\n",pr[over],

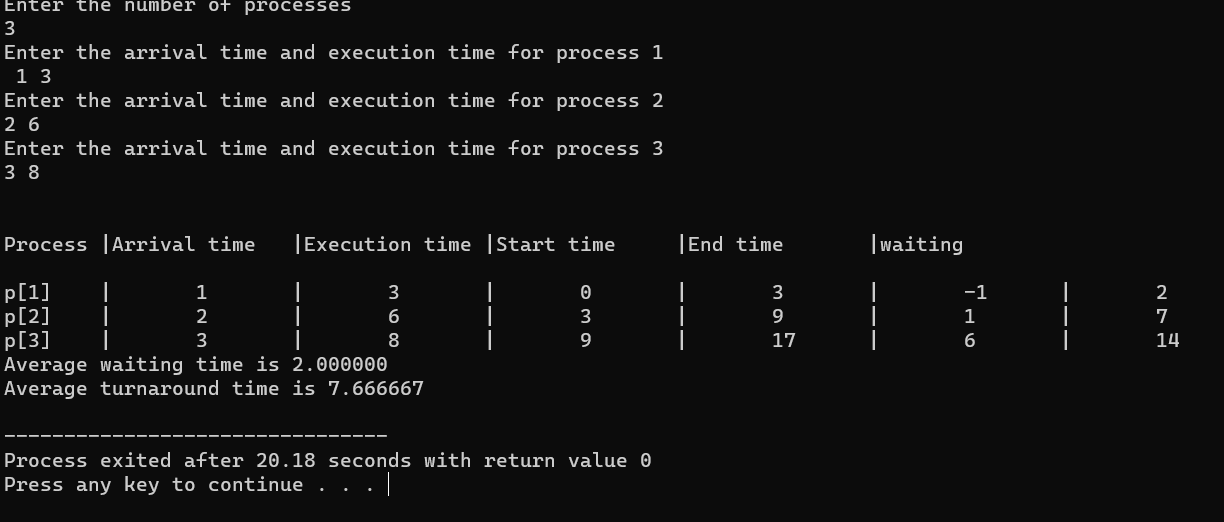
at[over],bt[over],start,time,time-at[over]- bt[over],time-at[over]);

sum\_wait+=time-at[over]-bt[over];

sum\_turnaround+=time-at[over]; over++;

}

avgwait=(float)sum\_wait/(float)n; avgturn=(float)sum\_turnaround/(float)n; printf("Average waiting time is %f\n",avgwait); printf("Average turnaround time is %f\n",avgturn); return 0;

}

# 8. Construct a C program to simulate Round Robin scheduling algorithm with C.

## PROGRAM:

#include<stdio.h> #include<conio.h> int main() {

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

printf(" Total number of process in the system: "); scanf("%d", &NOP);

y = NOP;

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

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1); printf(" Arrival time is: \t");

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

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

temp[i] = bt[i]; }

printf("Enter the Time Quantum for the process: \t"); scanf("%d", &quant);

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time "); for(sum=0, i = 0; y!=0; )

{

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

{

sum = sum + temp[i]; temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant; sum = sum + quant;

}

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

{

y--;

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum- at[i], sum-at[i]-bt[i]);

wt = wt+sum-at[i]-bt[i]; tat = tat+sum-at[i]; count =0;

}

if(i==NOP-1)

{

i=0;

}

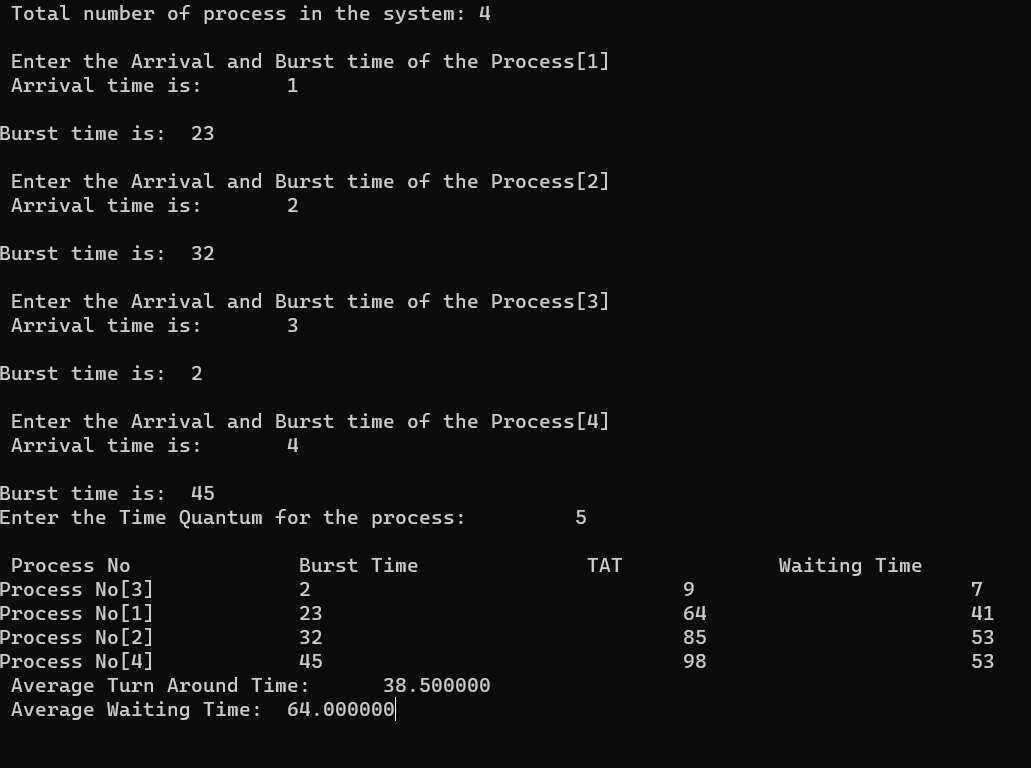
else if(at[i+1]<=sum) { i++; }

else { i=0; }

}

avg\_wt = wt \* 1.0/NOP; avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt); printf("\n Average Waiting Time: \t%f", avg\_tat); getch();

}

# 9 Illustrate the concept of inter-process communication using shared memory with a C program

# PROGRAM:

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <sys/ipc.h> #include <sys/shm.h>

#define SHM\_SIZE 1024 // Size of the shared memory segment int main() {

key\_t key = ftok("shmfile", 65); // Generate a unique key for the shared

memory segment

// Create a new shared memory segment (or get the identifier of an existing one)

int shmid = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666); if (shmid == -1) {

perror("shmget"); exit(EXIT\_FAILURE);

}

// Attach the shared memory segment to the process address space

char \*shm\_ptr = (char\*)shmat(shmid, NULL, 0); if (shm\_ptr == (char\*)(-1)) {

perror("shmat"); exit(EXIT\_FAILURE);

}

// Write data to the shared memory strcpy(shm\_ptr, "Hello, shared memory!");

// Detach the shared memory segment from the process if (shmdt(shm\_ptr) == -1) {

perror("shmdt"); exit(EXIT\_FAILURE);

}

printf("Data written to shared memory: %s\n", shm\_ptr);

// Optional: Remove the shared memory segment if (shmctl(shmid, IPC\_RMID, NULL) == -1) {

perror("shmctl"); exit(EXIT\_FAILURE);

}

return 0;

}

## OUTPUT:

**10. Illustrate the concept of inter-process communication using message queue with a c program**

## PROGRAM :

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h>

struct message { long msg\_type;

char msg\_text[100];

};

int main() {

key\_t key = ftok("msgqfile", 65); // Generate a unique key for the message queue

// Create a new message queue (or get the identifier of an existing one) int msgid = msgget(key, IPC\_CREAT | 0666);

if (msgid == -1) { perror("msgget");

exit(EXIT\_FAILURE);

}

struct message msg;

msg.msg\_type = 1; // Message type (can be any positive number)

// Producer: Send a message to the message queue strcpy(msg.msg\_text, "Hello, message queue!");

if (msgsnd(msgid, (void\*)&msg, sizeof(msg.msg\_text), IPC\_NOWAIT) == -1) {

perror("msgsnd"); exit(EXIT\_FAILURE);

}

printf("Producer: Data sent to message queue: %s\n", msg.msg\_text);

// Consumer: Receive a message from the message queue

if (msgrcv(msgid, (void\*)&msg, sizeof(msg.msg\_text), 1, 0) == -1) { perror("msgrcv");

exit(EXIT\_FAILURE);

}

printf("Consumer: Data received from message queue: %s\n", msg.msg\_text);

// Remove the message queue

if (msgctl(msgid, IPC\_RMID, NULL) == -1) { perror("msgctl");

exit(EXIT\_FAILURE);

}

return 0;

}

## OUTPUT :