**Name :Saravanavelu.D Reg no:192224249**

**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(){

Pid\_t child\_pid=fork();

If (child\_pid == -1){

Perror(“fork failed);

Return 1;

}

else{

printf(" parent Process : PID= %d, child PID=%d\n", getpid() ),child\_pid);}

return 0;

}

INPUT AND OUTPUT:



**2.To 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);

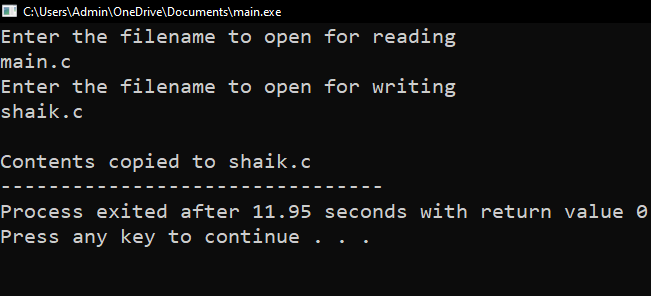
fclose(fptr1);

fclose(fptr2);

return 0;

}

INPUT AND OUTPUT:



**3.ToDesign a CPU scheduling program with C using First Come First Served technique with the following considerations. a. All processes are activated at time 0. b. Assume that no process waits on I/O devices.**

PROGRAM:

#include<stdio.h>

void main()

{

int n,bt[20],wt[20],tat[20],i,j; float avwt=0,avtat=0;printf("Enter total number of processes(maximum 20):");scanf("%d",&n);

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

{

printf("P[%d]:",i+1);

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

} wt[0]=0;

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

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

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time"); for(i=0;i<n;i++)

{

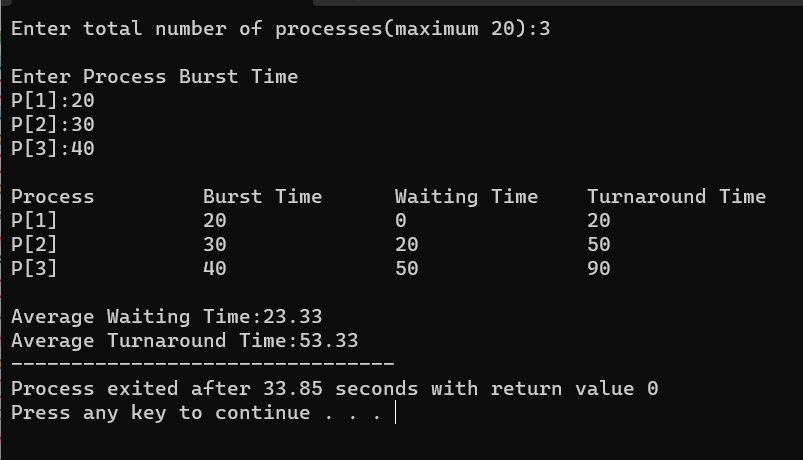
tat[i]=bt[i]+wt[i]; avwt+=wt[i]; avtat+=tat[i];printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

} avwt/=i; avtat/=i;printf("\n\nAverage Waiting Time:%.2f",avwt);

printf("\nAverage Turnaround Time:%.2f",avtat);

}

INPUT AND OUTPUT:



**4.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,floatavg\_wt,avg\_tat,avg\_wt;

printf("Enter number of process:");

scanf("%d",&n);

printf("\n Enter 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("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");for(i=0;i<n;i++)

{

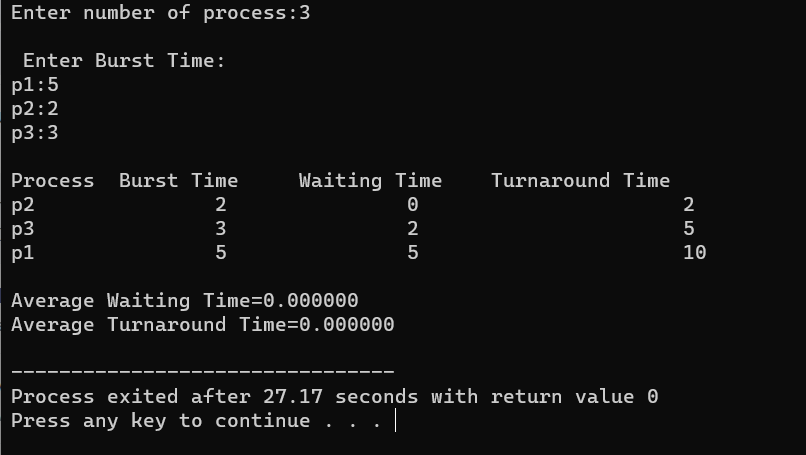
tat[i]=bt[i]+wt[i]; total+=tat[i];printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}avg\_tat=(float)total/n;printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

OUTPUT:



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

PROGRAM:

#include <stdio.h>

struct Process {

int id;

int priority;

int burst\_time;

};

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[n];

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

processes[i].id = i + 1;

printf("Enter priority and burst time for Process %d: ", i + 1);

scanf("%d %d", &processes[i].priority, &processes[i].burst\_time);

}

printf("Process Execution Order:\n");

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

int highest\_priority\_idx = 0;

for (int j = 1; j < n; j++) {

if (processes[j].priority < processes[highest\_priority\_idx].priority) {

highest\_priority\_idx = j;

}

}

printf("Executing Process %d (Priority: %d, Burst Time: %d)\n",

processes[highest\_priority\_idx].id, processes[highest\_priority\_idx].priority, processes[highest\_priority\_idx].burst\_time);

for (int j = highest\_priority\_idx; j < n - 1; j++) {

processes[j] = processes[j + 1];

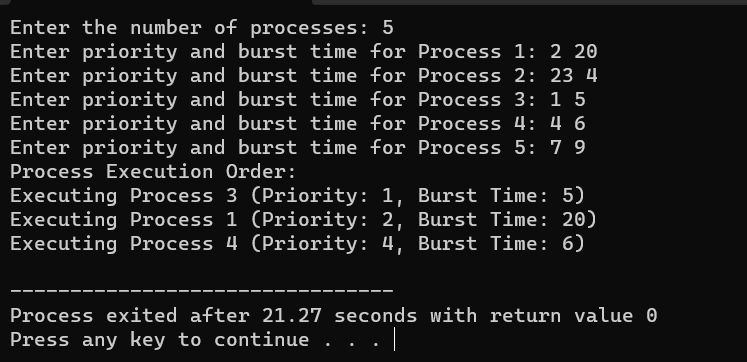
}

n--;

}

return 0;

**}OUTPUT:**



**6.To Construct a C program to simulate producer-consumer problem using semaphores**

PROGRAM:

#include <stdio.h>

struct Process {

int id, arrival\_time, burst\_time, priority;

};

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[n];

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

processes[i].id = i + 1;

printf("Enter arrival time, burst time, and priority for process %d: ", processes[i].id);

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

}

int current\_time = 0;

int total\_time = 0;

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

total\_time += processes[i].burst\_time;

}

printf("Gantt Chart: ");

while (current\_time < total\_time) {

int highest\_priority = 9999; // A high value to represent the lowest priority

int selected\_process = -1;

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

if (processes[i].arrival\_time <= current\_time && processes[i].burst\_time > 0 && processes[i].priority < highest\_priority) {

highest\_priority = processes[i].priority;

selected\_process = i;

}

}

if (selected\_process == -1) {

printf("Idle ");

current\_time++;

} else {

printf("P%d ", processes[selected\_process].id);

processes[selected\_process].burst\_time--;

current\_time++;

}

}

printf("\n\nProcess\tArrival Time\tBurst Time\tPriority\n");

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

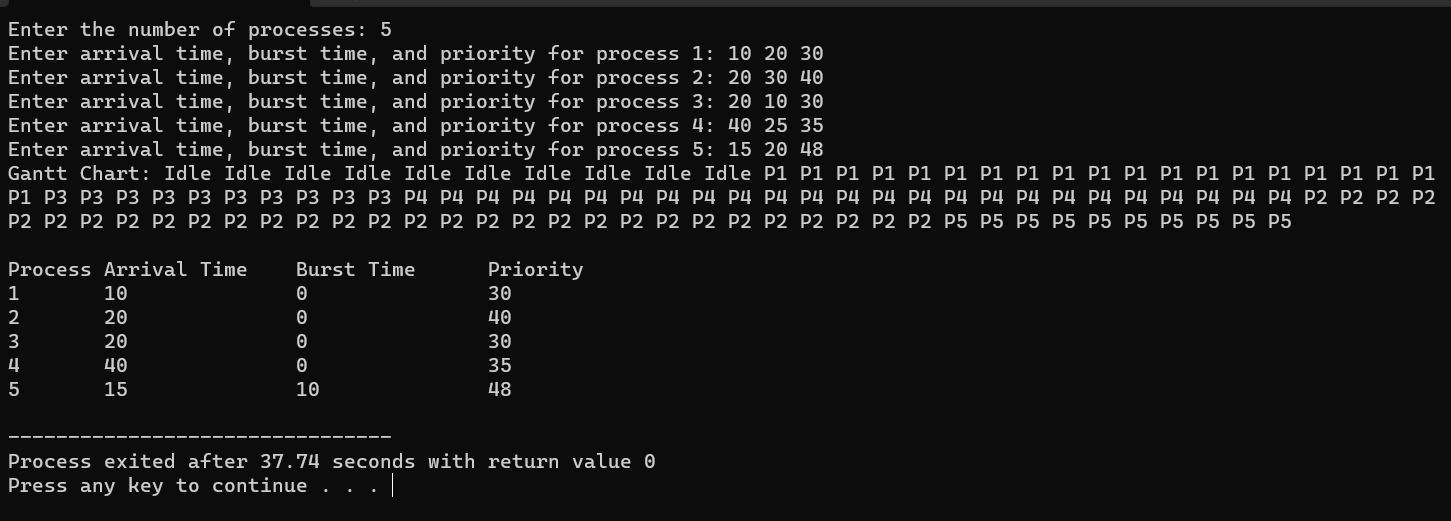
printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival\_time, processes[i].burst\_time, processes[i].priority);

}

return 0;

}

**OUTPUT:**

****

**7.To construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen.**

PROGRAM:

#include<stdio.h>

Int main ()

{

int i,j,n,a[50],frame[10],nf,k,avail,count=0;

printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

printf("\n ENTER THE PAGE

NUMBER:\n");for(i=1;i<=n;i++)scanf("%d",&a[i]);

printf("\n ENTER THE NUMBER OF FRAMES :");

scanf("%d",&nf);for(i=0;i<nf;i++)frame[i]= -1;j=0;

printf("\tref string\t page frames\n");for(i=1;i<=n;i++)

{

printf("%d\t\t",a[i]);avail=0;for(k=0;

k<nf;k++)

if(frame[k]==a[i])avail=1;

if (avail==0)

{

frame[j]=a[i];j=(j+1)%nf;c

ount++;for(k=0;k<nf;k++)

printf("%d\t",frame[k]);

}

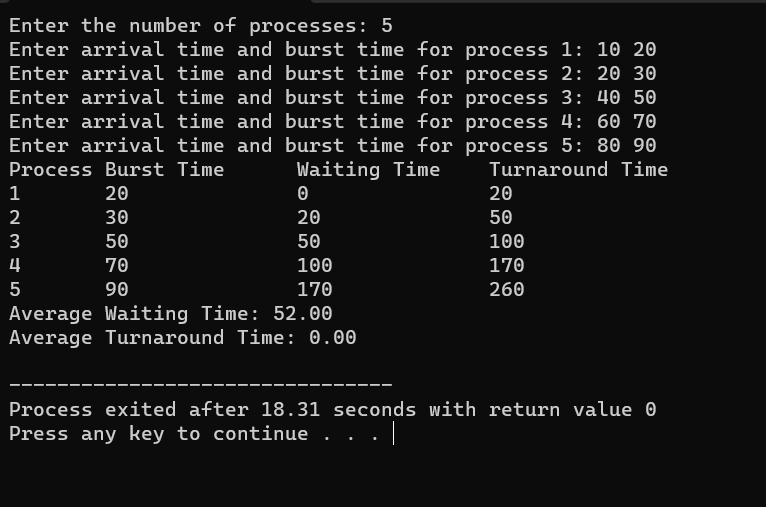
printf("\n");

}

printf("Page Fault Is %d",count);

}

OUTPUT:



**8. To Construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen**

**PROGRAM:**

#include<stdio.h>

int findLRU(int time[], int n){int i, minimum = time[0], pos = 0;for(i = 1; i < n; ++i){if(time[i] < minimum){minimum = time[i];pos = i;

}}

return pos;

}

int main()

{int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j,pos, faults = 0;printf("Enter number of frames: ");scanf("%d", &no\_of\_frames);printf("Enter number of pages: ");scanf("%d", &no\_of\_pages);printf("Enter reference string: ");

for(i = 0; i < no\_of\_pages; ++i){scanf("%d", &pages[i]);

}for(i = 0; i < no\_of\_frames; ++i){frames[i] = -1;

}for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;for(j = 0; j < no\_of\_frames; ++j){if(frames[j] == pages[i]){counter++;time[j] = counter;flag1 = flag2 = 1;

break;

}

}if(flag1 == 0){for(j = 0; j < no\_of\_frames; ++j){ if(frames[j] == -1){ counter++; faults++; frames[j] = pages[i]; time[j] = counter; flag2 = 1; break;

}}}if(flag2 == 0){ pos = findLRU(time, no\_of\_frames); counter++; faults++; frames[pos] = pages[i];

time[pos] = counter;

} printf("\n"); for(j = 0; j < no\_of\_frames; ++j){

printf("%d\t", frames[j]);

}}

printf("\n\nTotal Page Faults = %d", faults); return 0;

}

OUTPUT:

