**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);

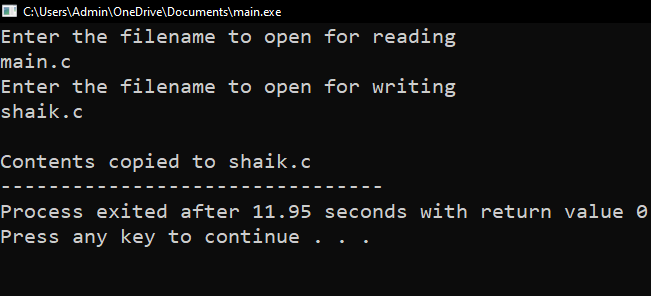
fclose(fptr1);

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++)

{

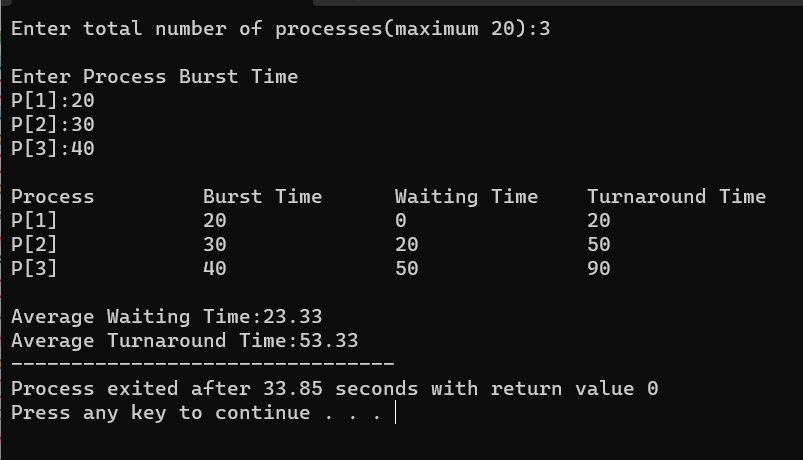
tat[i]=bt[i]+wt[i]; avwt+=wt[i]; avtat+=tat[i];printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

} 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>

struct Process {

int id;

int burst\_time;

};

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

int remaining\_time[n];

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

processes[i].id = i + 1;

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

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

remaining\_time[i] = processes[i].burst\_time;

}

int current\_time = 0;

printf("\nProcess execution order:\n");

while (1) {

int smallest = -1;

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

if (remaining\_time[i] > 0) {

if (smallest == -1 || remaining\_time[i] < remaining\_time[smallest]) {

smallest = i;

}

}

}

if (smallest == -1) {

break;

}

printf("Process %d (Burst Time: %d) is executing from time %d to ", processes[smallest].id, processes[smallest].burst\_time, current\_time);

current\_time += 1;

remaining\_time[smallest] -= 1;

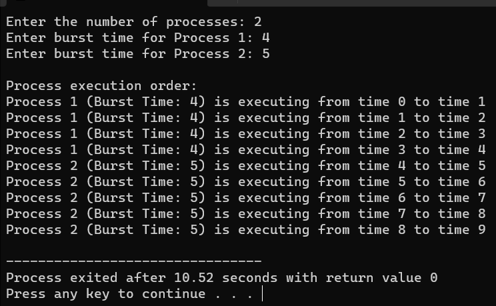
printf("time %d\n", current\_time);

}

return 0;

}

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];

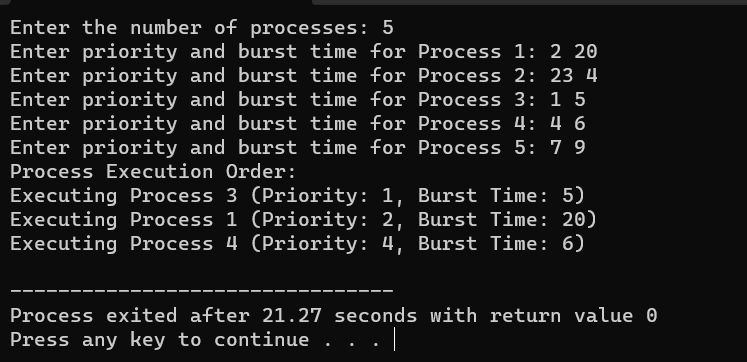
}

n--;

}

return 0;

**}OUTPUT:**



**6.** **Construct a C program to implement pre-emptive priority scheduling algorithm**.

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++) {

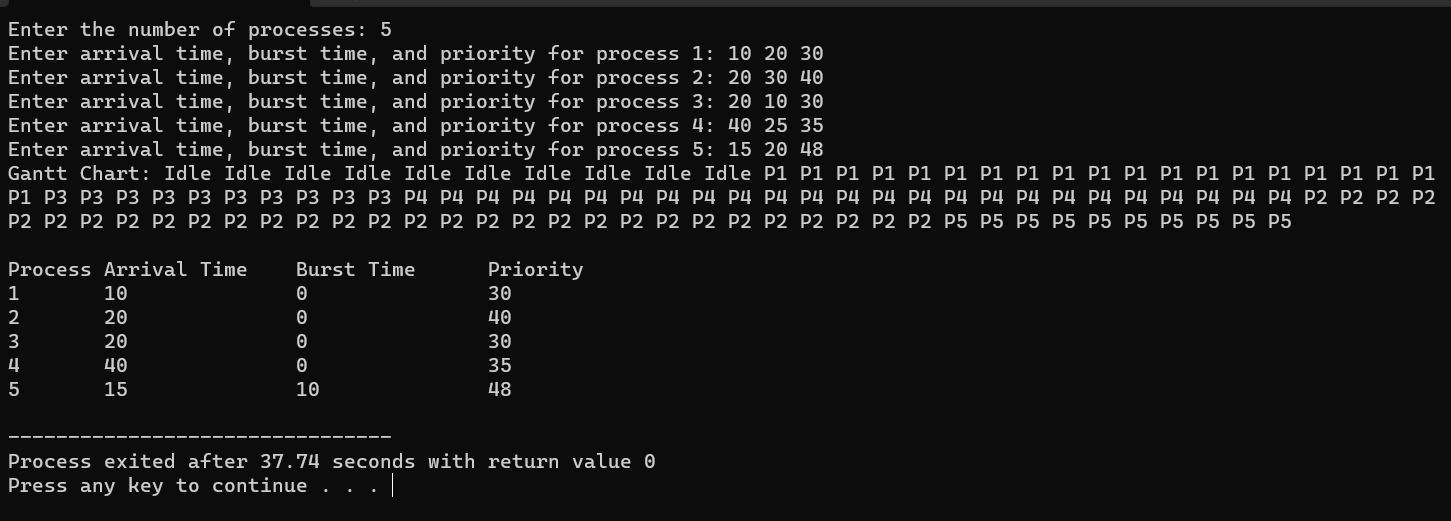
printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival\_time, processes[i].burst\_time, processes[i].priority);

}

return 0;

}

**OUTPUT:**

****

**7.** **Construct a C program to implement non-preemptive SJF algorithm**

PROGRAM:

#include <stdio.h>

struct Process {

int id, arrival\_time, 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 arrival time and burst time for process %d: ", processes[i].id);

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

}

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

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

if (processes[i].burst\_time > processes[j].burst\_time) {

struct Process temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

int waiting\_time = 0, turnaround\_time = 0;

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

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

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

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

}

float avg\_waiting\_time = (float)waiting\_time / n;

float avg\_turnaround\_time = (float)turnaround\_time / n;

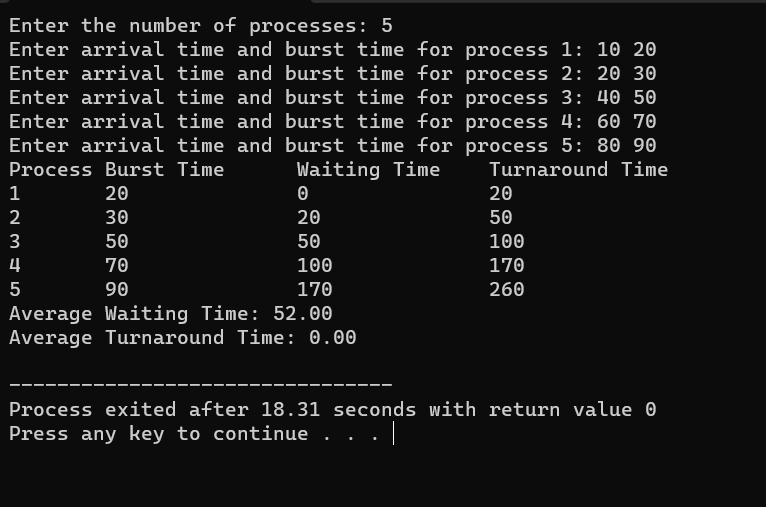
printf("Average Waiting Time: %.2f\n", avg\_waiting\_time);

printf("Average Turnaround Time: %.2f\n");

return 0;

}

OUTPUT:



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

**PROGRAM:**

**#include <stdio.h>**

**struct Process {**

**int id, burstTime, remainingTime;**

**};**

**void roundRobin(struct Process processes[], int n, int quantum) {**

**int time = 0, completed = 0;**

**while (completed < n) {**

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

**if (processes[i].remainingTime > 0) {**

**int execTime = (processes[i].remainingTime < quantum) ? processes[i].remainingTime : quantum;**

**processes[i].remainingTime -= execTime;**

**time += execTime;**

**printf("Process %d runs for %d units.\n", processes[i].id, execTime);**

**if (processes[i].remainingTime == 0) {**

**completed++;**

**printf("Process %d has completed.\n", processes[i].id);**

**}**

**}**

**}**

**}**

**}**

**int main() {**

**int n, quantum;**

**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 burst time for process %d: ", processes[i].id);**

**scanf("%d", &processes[i].burstTime);**

**processes[i].remainingTime = processes[i].burstTime;**

**}**

**printf("Enter time quantum: ");**

**scanf("%d", &quantum);**

**printf("\nRound Robin Scheduling:\n");**

**roundRobin(processes, n, quantum);**

**return 0;**

**}**

OUTPUT:

