

Registration Number: 19BCE2119

Name: Gaurav Kumar Singh

Course: CSE2005 Operating Systems (L7+L8)

OS Assignment 2

Process and Thread Management

- (a) Write a program to create a thread and perform the following (Easy) • Create a thread runner function • Set the thread attributes • Join the parent and thread • Wait for the thread to complete

CODE

```
#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 5

void *wait(void *t) {
    int i;
    long tid;
    tid = (long)t;
    sleep(1);
    printf("Sleeping \n");
    printf("Thread id : %d\n",tid);
    pthread_exit(NULL);
}

int main () {
    int rc;
    int i;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attr;
    void *status;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    for( i = 0; i < NUM_THREADS; i++ ) {
        printf("parent creating thread, %d\n",i);
        rc = pthread_create(&threads[i], &attr, wait, (void *)i );
        if (rc) {
            printf("Error:unable to create thread, %d\n",rc);
            exit(-1);
        }
    }
    pthread_attr_destroy(&attr);
    for( i = 0; i < NUM_THREADS; i++ ) {
        rc = pthread_join(threads[i], &status);
```

```

    if (rc) {
        printf ("Error:unable to join, %d\n",rc);
        exit(-1);
    }
    printf("Parent Process: completed thread id :%d",i) ;
    printf(" exiting with status :%d\n",status);
}
printf ("Parent: program exiting.\n");
pthread_exit(NULL);
}

```

OUTPUT

The screenshot shows a terminal window with a dark background. The top part displays the C code for a program that creates 5 threads. The bottom part shows the compilation output, which includes several warnings from the compiler (gcc) regarding implicit declarations of functions like 'printf' and 'exit', and format specifiers. The code is as follows:

```

#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 5

void *wait(void *t) {
    int i;
    long tid;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attr;
    void *status;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    for( i = 0; i < NUM_THREADS; i++ ) {
        printf("parent creating thread, %d\n",i);
        rc = pthread_create(&threads[i], &attr, wait, (void *)i );
        if (rc) {
            printf("Error:unable to create thread, %d\n",rc);
            exit(-1);
        }
    }
    pthread_attr_destroy(&attr);
    for( i = 0; i < NUM_THREADS; i++ ) {
        rc = pthread_join(threads[i], &status);
        if (rc) {
            printf ("Error:unable to join, %d\n",rc);
            exit(-1);
        }
        printf("Parent Process: completed thread id :%d",i) ;
        printf(" exiting with status :%d\n",status);
    }
    printf ("Parent: program exiting.\n");
    pthread_exit(NULL);
}

int main () {
    int rc;
    int i;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attr;
    void *status;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    for( i = 0; i < NUM_THREADS; i++ ) {
        printf("parent creating thread, %d\n",i);
        rc = pthread_create(&threads[i], &attr, wait, (void *)i );
        if (rc) {
            printf("Error:unable to create thread, %d\n",rc);
            exit(-1);
        }
    }
    pthread_attr_destroy(&attr);
    for( i = 0; i < NUM_THREADS; i++ ) {
        rc = pthread_join(threads[i], &status);
        if (rc) {
            printf ("Error:unable to join, %d\n",rc);
            exit(-1);
        }
        printf("Parent Process: completed thread id :%d",i) ;
        printf(" exiting with status :%d\n",status);
    }
    printf ("Parent: program exiting.\n");
    pthread_exit(NULL);
}

```

The compilation output shows the following warnings:

- la.c:12:19: warning: incompatible implicit declaration of built-in function 'printf'
- la.c:13:1: note: include <stdio.h> or provide a declaration of 'printf'
- la.c:13:30: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long int' [-Wformat-]
- la.c:26:10: warning: incompatible implicit declaration of built-in function 'printf'
- la.c:26:10: note: include <stdio.h> or provide a declaration of 'printf'
- la.c:27:56: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
- la.c:30:11: warning: implicit declaration of function 'exit' [-Wimplicit-function-declaration]
- la.c:30:11: warning: incompatible implicit declaration of built-in function 'exit'
- la.c:31:1: note: include <stdlib.h> or provide a declaration of 'exit'
- la.c:37:17: warning: incompatible implicit declaration of built-in function 'printf'
- la.c:37:17: note: include <stdio.h> or provide a declaration of 'printf'
- la.c:38:17: warning: incompatible implicit declaration of built-in function 'exit'
- la.c:38:17: note: include <stdlib.h> or provide a declaration of 'exit'
- la.c:40:9: warning: incompatible implicit declaration of built-in function 'printf'
- la.c:41:41: warning: format '%d' expects argument of type 'int', but argument 2 has type 'void *' [-Wformat-]
- la.c:43:9: warning: incompatible implicit declaration of built-in function 'printf'
- la.c:43:9: note: include <stdio.h> or provide a declaration of 'printf'

```

gaarav1020@DESKTOP-RORPIEK: ~/DA2
la.c:37:17: warning: incompatible implicit declaration of built-in function 'printf'
37 |         printf("Error:unable to join, %d\n",rc);
    |         ^~~~~~
la.c:37:17: note: include 'stdio.h' or provide a declaration of 'printf'
la.c:38:17: warning: incompatible implicit declaration of built-in function 'exit'
38 |         exit(-1);
    |         ^~~~~
la.c:38:17: note: include 'stdlib.h' or provide a declaration of 'exit'
la.c:40:9: warning: incompatible implicit declaration of built-in function 'printf'
40 |         printf("Parent Process: completed thread id :%d",i) ;
    |         ^~~~~~
la.c:40:9: note: include 'stdio.h' or provide a declaration of 'printf'
la.c:41:41: warning: format '%d' expects argument of type 'int', but argument 2 has type 'void *' [-Wformat-]
41 |         printf(" exiting with status :%d\n",status);
    |         ~~~~~^~~~~~
    |         |         |
    |         int      void *
la.c:43:9: warning: incompatible implicit declaration of built-in function 'printf'
43 |         printf("Parent: program exiting.\n");
    |         ^~~~~~
la.c:43:9: note: include 'stdio.h' or provide a declaration of 'printf'
gaarav1020@DESKTOP-RORPIEK: ~/DA2$ vi la.c
gaarav1020@DESKTOP-RORPIEK: ~/DA2$ ./la
parent creating thread, 0
parent creating thread, 1
parent creating thread, 2
parent creating thread, 3
parent creating thread, 4
sleeping
Thread id : 0
sleeping
Thread id : 1
sleeping
Thread id : 3
sleeping
Thread id : 4
sleeping
Thread id : 2
Parent Process: completed thread id :0 exiting with status :0
Parent Process: completed thread id :1 exiting with status :0
Parent Process: completed thread id :2 exiting with status :0
Parent Process: completed thread id :3 exiting with status :0
Parent Process: completed thread id :4 exiting with status :0
Parent: program exiting.
gaarav1020@DESKTOP-RORPIEK: ~/DA2$

```

(b) Write a program to create a thread to find the factorial of a natural number 'n'. (Medium)

CODE

```

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

void *Factorial(void* var){
    printf("\nThis is thread process with pid:%d\n",getpid());
    int n = (int *)var;
    int fact=1;
    for (int i=n; i>0; i--){
        fact=fact*i;
    }
    printf("Factorial of %d is: %d\n", n, fact);
}

int main() {
    int n; printf("\nThis is parent process with pid:%d",getpid());
    printf("\nBefore Thread Creation.\nThread will now be Created...");
    printf("\nEnter the number whose Factorial you want to calculate: ");
    scanf("%d",&n);
    pthread_t tid;
    pthread_create (&tid,NULL,Factorial,(void*)n);
    pthread_join(tid,NULL);
    pthread_exit(NULL);
    exit(0);
}

```

OUTPUT

```

gaarav1020@DESKTOP-RORPIEK: ~/DA2
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void *Factorial(void* var){
    printf("\nThis is thread process with pid:%d\n",getpid());
    int n = (int *)var;
    int fact=1;
    for (int i=n; i>= 1; i--){
        fact=fact*i;
    }
    printf("Factorial of %d is: %d\n", n, fact);
}

int main() {
    int n; printf("\nThis is parent process with pid:%d",getpid());
    printf("\nBefore Thread Creation.\nThread will now be created...");
    printf("\nEnter the number whose Factorial you want to calculate: ");
    scanf("%d",&n);
    pthread_t tid;
    pthread_create (&tid,NULL,Factorial,(void*)n);
    pthread_join(tid,NULL);
    pthread_exit(NULL);
    exit(0);
}

"lb.c" 25L, 701C
6,47-54 All

gaarav1020@DESKTOP-RORPIEK: ~/DA2
ls
lb.c  lb  lb.c
gaarav1020@DESKTOP-RORPIEK:~/DA2$ vi lb.c
gaarav1020@DESKTOP-RORPIEK:~/DA2$ gcc lb.c -o lb -lpthread
lb.c: In function 'Factorial':
lb.c:7:10: warning: initialization of 'int' from 'int **' makes integer from pointer without a cast [-Wint-conversion]
   7 | int n = (int *)var;
     |         ^
lb.c: In function 'main':
lb.c:20:45: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   20 | pthread_create (&tid,NULL,Factorial,(void*)n);
     |                                     ^
gaarav1020@DESKTOP-RORPIEK:~/DA2$ ./lb

This is parent process with pid:45
Before Thread Creation.
Thread will now be Created...
Enter the number whose Factorial you want to calculate: 6

This is thread process with pid:45
Factorial of 6 is: 720
gaarav1020@DESKTOP-RORPIEK:~/DA2$

```

- (c) Assume that two processes named client and server running in the system. It is required that these two processes should communicate with each other using shared memory concept. The server writes alphabets from a..z to the shared memory .the client should read the alphabets from the shared memory and convert it to A...Z. Write a program to demonstrate the above mentioned scenario. (Medium)

CODE SERVER

```

#include<iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

```

```

using namespace std;
int main()
{
    key_t my_key = ftok("shmfile",65);
    int shmid = shmget(my_key,1024,0666|IPC_CREAT);
    char *str = (char*) shmat(shmid,(void*)0,0);
    cout<<"Write data:";
    fgets(str, 50, stdin);
    printf("Data written in memory: %s\n",str);
    shmdt(str);
    return 0;
}

```

OUTPUT SERVER

```

gaurav1020@DESKTOP-RORPIEK: ~/DA2
#include<iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;
int main()
{
    key_t my_key = ftok("shmfile",65);
    int shmid = shmget(my_key,1024,0666|IPC_CREAT);
    char *str = (char*) shmat(shmid,(void*)0,0);
    cout<<"Write data:";
    fgets(str, 50, stdin);
    printf("Data written in memory: %s\n",str);
    shmdt(str);
    return 0;
}

"lc_5.cpp" 16L, 363C
12,23-30 All

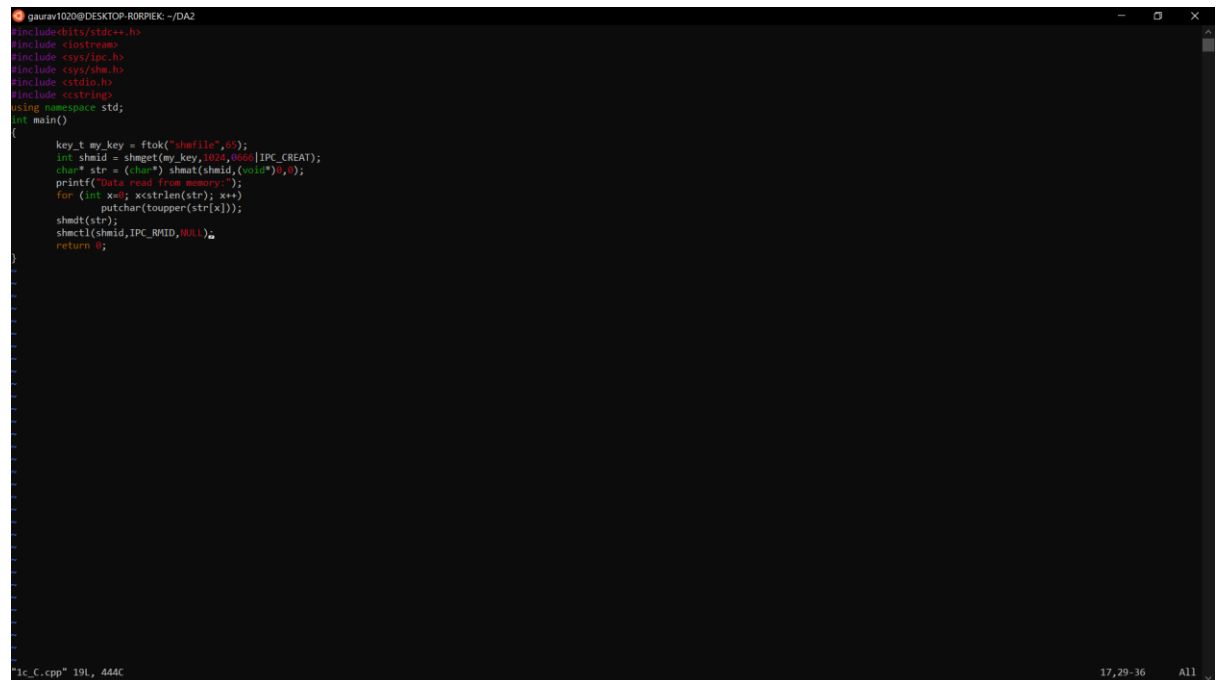
gaurav1020@DESKTOP-RORPIEK: ~/DA2
gaurav1020@DESKTOP-RORPIEK:~/DA2$ g++ 1c_5.cpp -o 1c_5
gaurav1020@DESKTOP-RORPIEK:~/DA2$ ./1c_5
Write data:abcdefghijklmnopqrstuvwxyz
Data written in memory: abcdefghijklmnopqrstuvwxyz
gaurav1020@DESKTOP-RORPIEK:~/DA2$

```

CODE CLIENT

```
#include<bits/stdc++.h>
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <cstring>
using namespace std;
int main()
{
    key_t my_key = ftok("shmfile",65);
    int shmid = shmget(my_key,1024,0666|IPC_CREAT);
    char* str = (char*) shmat(shmid,(void*)0,0);
    printf("Data read from memory:");
    for (int x=0; x<strlen(str); x++)
        putchar(toupper(str[x]));
    shmdt(str);
    shmctl(shmid,IPC_RMID,NULL);
    return 0;
}
```

OUTPUT CLIENT



```
gaurav1020@DESKTOP-RORPEK: ~/DA2
#include<bits/stdc++.h>
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <cstring>
using namespace std;
int main()
{
    key_t my_key = ftok("shmfile",65);
    int shmid = shmget(my_key,1024,0666|IPC_CREAT);
    char* str = (char*) shmat(shmid,(void*)0,0);
    printf("Data read from memory:");
    for (int x=0; x<strlen(str); x++)
        putchar(toupper(str[x]));
    shmdt(str);
    shmctl(shmid,IPC_RMID,NULL);
    return 0;
}
```

17,29-36 All

```
gaurav1020@DESKTOP-RORPIEK: ~/DA2
gaurav1020@DESKTOP-RORPIEK:~/DA2$ vi 1c_c.cpp
gaurav1020@DESKTOP-RORPIEK:~/DA2$ g++ 1c_c.cpp -o 1c_C
gaurav1020@DESKTOP-RORPIEK:~/DA2$ ./1c_C
Data read from memory: ABCDEFGHIJKLMNOPQRSTUVWXYZ
gaurav1020@DESKTOP-RORPIEK:~/DA2$
```

- (d) Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers 90 81 78 95 79 72 85 , the program will report the average value as 82. The minimum value as 72. The maximum value as 95. The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited. (High)

CODE

```
#include<stdio.h>

#include<pthread.h>

int arr[50],n,i;

void *th()

{

    float sum=0;

    float average;

    printf("enter your number :=");

    scanf("%d",&n);

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

    {

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

    }

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

```

        {
            sum=sum+arr[i];
        }
        average=sum/n;
        printf("The average value is:%f",average);
    }
void *th1()
{
    int temp=arr[0];
    for(int i=1;i<n;i++)
    {
        if(temp>arr[i])
        {
            temp=arr[i];
        }
    }
    printf("\nThe Minimum value is:=%d",temp);
}
void *th2()
{
    int temp=arr[0];
    for(int i=1;i<n;i++)
    {
        if(temp<arr[i])
        {
            temp=arr[i];
        }
    }
    printf("\nThe Maximum value is:=%d",temp);
}
int main()
{
    int n,i;
    pthread_t t1;
    pthread_t t2;
    pthread_t t3;
    n=pthread_create(&t1,NULL,&th,NULL);
    pthread_join(t1,NULL);
    n=pthread_create(&t2,NULL,&th1,NULL);
    pthread_join(t2,NULL);
    n=pthread_create(&t3,NULL,&th2,NULL);
    pthread_join(t3,NULL);

}

```

OUTPUT


```
Select gaurav1020@DESKTOP-RORPIEK: ~/DA2
printf("enter your number :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
    scanf("%d",&arr[i]);
}
for(i=0;i<n;i++)
{
    sum=sum+arr[i];
}
average=sum/n;
printf(" the average value is:%f",average);
}

void *th1()
{
    int temp=arr[0];
    for(int i=1;i<n;i++)
    {
        if(temp>arr[i])
        {
            temp=arr[i];
        }
    }
    printf("\nThe Minimum value is:%d",temp);
}

void *th2()
{
    int temp=arr[0];
    for(int i=1;i<n;i++)
    {
        if(temp<arr[i])
        {
            temp=arr[i];
        }
    }
    printf("\nThe Maximum value is:%d",temp);
}

int main()
{
    int n,i;
    pthread_t t1;
    pthread_t t2;
    pthread_t t3;
    nthread_create(&t1,NULL,&th,NULL);
    pthread_join(t1,NULL);
    nthread_create(&t2,NULL,&th1,NULL);
    pthread_join(t2,NULL);
    nthread_create(&t3,NULL,&th2,NULL);
    pthread_join(t3,NULL);
}

gaurav1020@DESKTOP-RORPIEK: ~/DA2
gaurav1020@DESKTOP-RORPIEK:~/DA2$ vi 1d.c
gaurav1020@DESKTOP-RORPIEK:~/DA2$ gcc 1d.c -o 1d -lpthread
gaurav1020@DESKTOP-RORPIEK:~/DA2$ ./1d
enter your number :4
12
45
34
23
The average value is:20.500000
The Minimum value is:12
The Maximum value is:45gaurav1020@DESKTOP-RORPIEK:~/DA2$
```

CPU Scheduling

- (a) Implement the various process scheduling algorithms such as FCFS, SJF, Priority (Non Preemptive). (Easy)

CODE FCFS

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
#include <stdlib.h>
```

```

#include <string.h>

struct Process {
    int p_id;
    int AT;
    int BT;
    int CT;
    int TAT;
    int WT;
};

void display(struct Process Process_Array[], int n){
    for (int i=0; i<n;i++) {
        int Pno=i+1;
        printf("\n\nProcess %d\n",Pno);
        printf("p_id=%d\n",Process_Array[i].p_id);
        printf("AT=%d\n",Process_Array[i].AT);
        printf("BT=%d\n",Process_Array[i].BT);
        printf("CT=%d\n",Process_Array[i].CT);
        printf("TAT=%d\n",Process_Array[i].TAT);
        printf("WT=%d\n",Process_Array[i].WT);
    }
}

void getStats(struct Process Process_Array[], int n){
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i=0; i<n;i++) {
        printf("Enter PID of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].p_id);
        printf("Enter Arrival Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].AT);
        printf("Enter Burst Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].BT);
    }
}

```

```

}

void calcCT(struct Process Process_Array[],int n){
    int timeline=0;
    for (int i=0; i<n;i++) {
        if (timeline<Process_Array[i].AT) {
            timeline=Process_Array[i].AT;
        }
        timeline = timeline + Process_Array[i].BT;
        Process_Array[i].CT = timeline;
    }
}

void calcTAT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[],int n){
    float sumTAT=0;
    for (int i=0; i<n;i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[],int n){
    float sumWT=0;
    for (int i=0; i<n;i++) {

```

```

        sumWT = sumWT + Process_Array[i].WT;
    }

    printf("\n\nAverage Waiting time= %.3f\n", (sumWT/n));
}

int main(){

    int n;

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

    scanf ("%d",&n);

    struct Process Process_Array[100];

    getStats(Process_Array, n);

    calcCT(Process_Array, n);

    calcTAT(Process_Array, n);

    calcWT(Process_Array, n);

    display(Process_Array, n);

    calcAvgTAT(Process_Array, n);

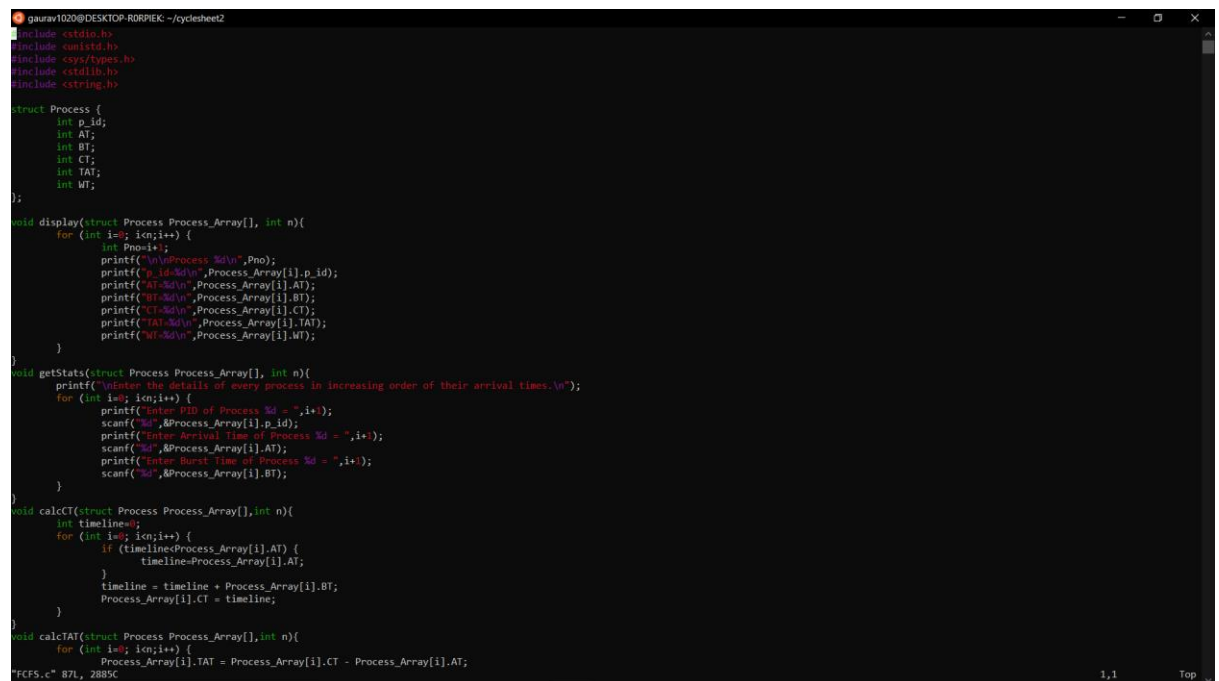
    calcAvgWT(Process_Array, n);

    return 0;

}

```

OUTPUT FCFS



```

gaurav1020@DESKTOP-RORPEK: ~/cyclesheet2
#include <stdio.h>
#include <conio.h>
#include <csys/types.h>
#include <stdlib.h>
#include <string.h>

struct Process {
    int p_id;
    int AT;
    int BT;
    int CT;
    int TAT;
    int WT;
};

void display(struct Process Process_Array[], int n){
    for (int i=0; i<n; i++) {
        int Pno=i+1;
        printf("\nProcess %d\n", Pno);
        printf("p_id=%d\n", Process_Array[i].p_id);
        printf("at=%d\n", Process_Array[i].AT);
        printf("bt=%d\n", Process_Array[i].BT);
        printf("ct=%d\n", Process_Array[i].CT);
        printf("TAT=%d\n", Process_Array[i].TAT);
        printf("WT=%d\n", Process_Array[i].WT);
    }
}

void getStats(struct Process Process_Array[], int n){
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i=0; i<n; i++) {
        printf("Enter PID of Process %d = ", i+1);
        scanf("%d", &Process_Array[i].p_id);
        printf("Enter Arrival Time of Process %d = ", i+1);
        scanf("%d", &Process_Array[i].AT);
        printf("Enter Burst Time of Process %d = ", i+1);
        scanf("%d", &Process_Array[i].BT);
    }
}

void calcCT(struct Process Process_Array[], int n){
    int timeline=0;
    for (int i=0; i<n; i++) {
        if (timeline<Process_Array[i].AT) {
            timeline=Process_Array[i].AT;
        }
        timeline = timeline + Process_Array[i].BT;
        Process_Array[i].CT = timeline;
    }
}

void calcTAT(struct Process Process_Array[], int n){
    for (int i=0; i<n; i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[], int n){
    int sumWT=0;
    for (int i=0; i<n; i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f\n", (sumWT/n));
}

int main(){
    int n;
    printf("\nEnter the number of processes in the system:");
    scanf ("%d",&n);
    struct Process Process_Array[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}

```

```
gaurav1020@DESKTOP-RORPIEK: ~/cyclesheet2
gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$ gcc FCFS.c -o FCFS
gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$ ./FCFS

Enter the number of processes in the system:3

Enter the details of every process in increasing order of their arrival times.
Enter PID of Process 1 = 1
Enter Arrival Time of Process 1 = 0
Enter Burst Time of Process 1 = 12
Enter PID of Process 2 = 2
Enter Arrival Time of Process 2 = 1
Enter Burst Time of Process 2 = 23
Enter PID of Process 3 = 3
Enter Arrival Time of Process 3 = 11
Enter Burst Time of Process 3 = 22

Process 1
p_id=1
AT=0
BT=12
CT=12
TAT=12
WT=0

Process 2
p_id=2
AT=1
BT=23
CT=35
TAT=34
WT=11

Process 3
p_id=3
AT=11
BT=22
CT=57
TAT=46
WT=24

Average Turnaround time= 30.667
Average Waiting time= 11.667
gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$
```

CODE SJF

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
struct Process {
```

```
    int p_id;
```

```
    int AT;
```

```
    int BT;
```

```
    int CT;
```

```
    int TAT;
```

```
    int WT;
```

```
    int flag;
```

```
};
```

```
void display(struct Process Process_Array[], int n){
```

```

for (int i=0; i<n;i++) {
    int Pno=i+1;
    printf("\n\nProcess %d\n",Pno);
    printf("p_id=%d\n",Process_Array[i].p_id);
    printf("AT=%d\n",Process_Array[i].AT);
    printf("BT=%d\n",Process_Array[i].BT);
    printf("CT=%d\n",Process_Array[i].CT);
    printf("TAT=%d\n",Process_Array[i].TAT);
    printf("WT=%d\n",Process_Array[i].WT);
}
}

void getStats(struct Process Process_Array[], int n){
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i=0; i<n;i++) {
        printf("Enter PID of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].p_id);
        printf("Enter Arrival Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].AT);
        printf("Enter Burst Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].BT);
        Process_Array[i].flag=0;
    }
}

int MinBT(struct Process Process_Array[],int n){
    int min=0;
    for (int i=0; i<n; i++){
        if (Process_Array[i].BT<Process_Array[min].BT && Process_Array[i].flag==0){
            min=i;
        }
    }
    return min;
}

```

```

}

void calcCT(struct Process Process_Array[], struct Process Ready[], int n){
    int timeline=0;
    int counter=0;
    int min=0;
    int j=0;
    while (counter!=n){
        j=0;
        for (int i=0; i<n; i++){
            if(Process_Array[i].AT<=timeline&& Process_Array[i].flag==0){
                Ready[j]=Process_Array[i];
                j++;
            }
        }
        if (j==0){
            timeline++;
        }
        else{
            min=MinBT(Ready,j);
            for(int i=0; i<n; i++){
                if (Process_Array[i].p_id==Ready[min].p_id){
                    timeline=timeline+Process_Array[i].BT;
                    Process_Array[i].CT=timeline;
                    Process_Array[i].flag=1;
                    counter++;
                }
            }
        }
    }
}

void calcTAT(struct Process Process_Array[],int n){

```

```

        for (int i=0; i<n;i++) {
            Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
        }
    }

void calcWT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[],int n){
    float sumTAT=0;
    for (int i=0; i<n;i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[],int n){
    float sumWT=0;
    for (int i=0; i<n;i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}

int main(){
    int n;
    printf("\nEnter the number of processes in the system");
    scanf ("%d",&n);
    struct Process Process_Array[100];
    struct Process Ready[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, Ready, n);

```



```

    calcTAT(Process_Array, n);

    calcWT(Process_Array, n);

    display(Process_Array, n);

    calcAvgTAT(Process_Array, n);

    calcAvgWT(Process_Array, n);

    return 0;
}

```

OUTPUT SJF

```

gauravi1020@DESKTOP-RORPIEK: ~/cyclesheet2
    for(int i=0; i<n; i++){
        if (Process_Array[i].p_id==Ready[min].p_id){
            timeline=timeline+Process_Array[i].BT;
            Process_Array[i].CT=timeline;
            Process_Array[i].flag=1;
            counter++;
        }
    }
}

void calcTAT(struct Process Process_Array[], int n){
    for (int i=0; i<n; i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[], int n){
    for (int i=0; i<n; i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[], int n){
    float sumTAT=0;
    for (int i=0; i<n; i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[], int n){
    float sumWT=0;
    for (int i=0; i<n; i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}

int main(){
    int n;
    printf("\nEnter the number of processes in the system");
    scanf ("%d", &n);
    struct Process Process_Array[100];
    struct Process Ready[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, Ready, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}

gauravi1020@DESKTOP-RORPIEK: ~/cyclesheet2$ gcc SJF.c -o SJF
gauravi1020@DESKTOP-RORPIEK: ~/cyclesheet2$ ./SJF

Enter the number of processes in the system3

Enter the details of every process in increasing order of their arrival times.
Enter PID of Process 1 = 1
Enter Arrival Time of Process 1 = 0
Enter Burst Time of Process 1 = 12
Enter PID of Process 2 = 2
Enter Arrival Time of Process 2 = 1
Enter Burst Time of Process 2 = 23
Enter PID of Process 3 = 3
Enter Arrival Time of Process 3 = 11
Enter Burst Time of Process 3 = 22

Process 1
p_id=1
AT=0
BT=12
CT=12
TAT=12
WT=0

Process 2
p_id=2
AT=1
BT=23
CT=24
TAT=56
WT=33

Process 3
p_id=3
AT=11
BT=22
CT=34
TAT=23
WT=1

Average Turnaround time= 30.333
Average Waiting time= 11.333gauravi1020@DESKTOP-RORPIEK: ~/cyclesheet2$

```

CODE PRIORITY

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
struct Process {
```

```
    int p_id;
```

```
    int AT;
```

```
    int BT;
```

```
    int rem_BT;
```

```
    int CT;
```

```
    int TAT;
```

```
    int WT;
```

```
    int Priority;
```

```
};
```

```
void display(struct Process Process_Array[], int n){
```

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

```
        int Pno=i+1;
```

```
        printf("\n\nProcess %d\n",Pno);
```

```
        printf("p_id=%d\n",Process_Array[i].p_id);
```

```
        printf("AT=%d\n",Process_Array[i].AT);
```

```
        printf("BT=%d\n",Process_Array[i].BT);
```

```
        printf("Priority=%d\n",Process_Array[i].Priority);
```

```
        printf("CT=%d\n",Process_Array[i].CT);
```

```
        printf("TAT=%d\n",Process_Array[i].TAT);
```

```
        printf("WT=%d\n",Process_Array[i].WT);
```

```
    }
```

```
}
```

```

void getStats(struct Process Process_Array[], int n){
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i=0; i<n;i++) {
        printf("Enter PID of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].p_id);
        printf("Enter Arrival Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].AT);
        printf("Enter Burst Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].BT);
        Process_Array[i].rem_BT=Process_Array[i].BT;
        printf("Enter Priority of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].Priority);
    }
}

int readyQueueManagement(struct Process Process_Array[], struct Process Ready[],int
timeline, int n){
    int i=0, j=0;
    while (Process_Array[i].AT<=timeline && i<n){
        if (Process_Array[i].rem_BT !=0){
            Ready[j]=Process_Array[i];
            j++;
        }
        i++;
    }
    return j;
}

```

```

int MaxPriority(struct Process Ready[], int j){
    int max=0;
    for(int i=0; i<j;i++) {
        if (Ready[i].Priority>Ready[max].Priority){
            max=i;
        }
    }
}

```

```

    }
}
return max;
}

```

```

void calcCT(struct Process Process_Array[], struct Process Ready[],int n){
    int timeline=0;
    int counter = 0;
    while (counter !=n) {
        int j=readyQueueManagement(Process_Array, Ready, timeline, n);
        if (j==0){
            timeline++;
        }
        else{
            int max= MaxPriority(Ready, j);
            for(int i=0;i<n;i++) {
                if (Ready[max].p_id==Process_Array[i].p_id){
                    Process_Array[i].rem_BT=0;
                    timeline=timeline+Process_Array[i].BT;
                    Process_Array[i].CT=timeline;
                    counter++;
                }
            }
        }
    }
}

```

```

void calcTAT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

```

```

}

void calcWT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[],int n){
    float sumTAT=0;
    for (int i=0; i<n;i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[],int n){
    float sumWT=0;
    for (int i=0; i<n;i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}

int main(){
    int n;

    printf("\nEnter the number of Processes in the system:");

    scanf ("%d",&n);

    struct Process Process_Array[100];
    struct Process Ready[100];

    getStats(Process_Array, n);
    calcCT(Process_Array,Ready, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
}

```

```

    calcAvgTAT(Process_Array, n);

    calcAvgWT(Process_Array, n);

    return 0;

}

```

OUTPUT PRIORITY

```

gaurav1020@DESKTOP-RORPEK: ~/cyclesheet2
for(int i=0; i<n; i++) {
    if (Ready[max].p_id==Process_Array[i].p_id){
        Process_Array[i].rem_BT=0;
        timeline=timeline+Process_Array[i].BT;
        Process_Array[i].CT=timeline;
        counter++;
    }
}

void calcTAT(struct Process Process_Array[], int n){
    for (int i=0; i<n; i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[], int n){
    for (int i=0; i<n; i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[], int n){
    float sumTAT=0;
    for (int i=0; i<n; i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[], int n){
    float sumWT=0;
    for (int i=0; i<n; i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}

int main(){
    int n;
    printf("\nEnter the number of Processes in the system:");
    scanf ("%d", &n);
    struct Process Process_Array[100];
    struct Process Ready[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, Ready, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}

Priority_MP.c" 127L, 4339C
125,1 Bot

```

```
gaurav1020@DESKTOP-RORPIEK: ~/cyclesheet2
gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$ ./Priority_MP
Enter the number of Processes in the system:3
Enter the details of every process in increasing order of their arrival times.
Enter PID of Process 1 = 1
Enter Arrival Time of Process 1 = 0
Enter Burst Time of Process 1 = 12
Enter Priority of Process 1 = 1
Enter PID of Process 2 = 2
Enter Arrival Time of Process 2 = 1
Enter Burst Time of Process 2 = 23
Enter Priority of Process 2 = 4
Enter PID of Process 3 = 3
Enter Arrival Time of Process 3 = 2
Enter Burst Time of Process 3 = 11
Enter Priority of Process 3 = 3

Process 1
p_id=1
AT=0
BT=12
Priority=1
CT=12
TAT=12
WT=0

Process 2
p_id=2
AT=1
BT=23
Priority=4
CT=35
TAT=34
WT=11

Process 3
p_id=3
AT=2
BT=11
Priority=3
CT=46
TAT=44
WT=33

Average Turnaround time= 30.000
Average Waiting time= 14.667gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$
```

- (b) Implement the various process scheduling algorithms such as Priority, Round Robin (preemptive). (Medium)

CODE PRIORITY

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>
```

```
struct Process {

    int p_id;

    int AT;

    int BT;

    int rem_BT;

    int CT;

    int TAT;

    int WT;
```

```

        int Priority;
    };

void display(struct Process Process_Array[], int n){
    for (int i=0; i<n;i++) {
        int Pno=i+1;
        printf("\n\nProcess %d\n",Pno);
        printf("p_id=%d\n",Process_Array[i].p_id);
        printf("AT=%d\n",Process_Array[i].AT);
        printf("BT=%d\n",Process_Array[i].BT);
        printf("Priority=%d\n",Process_Array[i].Priority);
        printf("CT=%d\n",Process_Array[i].CT);
        printf("TAT=%d\n",Process_Array[i].TAT);
        printf("WT=%d\n",Process_Array[i].WT);
    }
}

void getStats(struct Process Process_Array[], int n){
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i=0; i<n;i++) {
        printf("Enter PID of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].p_id);
        printf("Enter Arrival Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].AT);
        printf("Enter Burst Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].BT);
        Process_Array[i].rem_BT=Process_Array[i].BT;
        printf("Enter Priority of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].Priority);
    }
}

int readyQueueManagement(struct Process Process_Array[], struct Process Ready[],int
timeline, int n){

```



```

int i=0, j=0;
while (Process_Array[i].AT<=timeline && i<n){
    if (Process_Array[i].rem_BT !=0){
        Ready[j]=Process_Array[i];
        j++;
    }
    i++;
}
return j;
}

```

```

int MaxPriority(struct Process Ready[], int j){
    int max=0;
    for(int i=0; i<j;i++) {
        if (Ready[i].Priority>Ready[max].Priority){
            max=i;
        }
    }
    return max;
}

```

```

void calcCT(struct Process Process_Array[], struct Process Ready[],int n){
    int timeline=0;
    int counter = 0;
    while (counter !=n) {
        int j=readyQueueManagement(Process_Array, Ready, timeline, n);
        if (j==0){
            timeline++;
        }
        else{
            int max= MaxPriority(Ready, j);

```

```

        for(int i=0;i<n;i++) {
            if (Ready[max].p_id==Process_Array[i].p_id){
                Process_Array[i].rem_BT=Process_Array[i].rem_BT-1;
                timeline=timeline+1;
                if (Process_Array[i].rem_BT==0){
                    Process_Array[i].CT=timeline;
                    counter++;
                }
            }
        }
    }
}

void calcTAT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[],int n){
    float sumTAT=0;
    for (int i=0; i<n;i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[],int n){

```

```

float sumWT=0;

for (int i=0; i<n;i++) {
    sumWT = sumWT + Process_Array[i].WT;
}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}

int main(){
    int n;

    printf("\nEnter the number of Processes in the system:");

    scanf ("%d",&n);

    struct Process Process_Array[100];
    struct Process Ready[100];

    getStats(Process_Array, n);
    calcCT(Process_Array,Ready, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);

    return 0;
}

```

OUTPUT PRIORITY

```
gaurav1020@DESKTOP-RORPIEK: ~/cyclesheet2
    Process_Array[i].rem_BT=Process_Array[i].rem_BT-1;
    timeline=timeline+1;
    if (Process_Array[i].rem_BT==0){
        Process_Array[i].CT=timeline;
        counter++;
    }
}
}
}

void calcTAT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++){
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++){
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[],int n){
    float sumTAT=0;
    for (int i=0; i<n;i++){
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[],int n){
    float sumWT=0;
    for (int i=0; i<n;i++){
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting times %.3f", (sumWT/n));
}

int main(){
    int n;
    printf("\nEnter the number of Processes in the system:");
    scanf ("%d",&n);
    struct Process Process_Array[100];
    struct Process Ready[100];
    getStats(Process_Array, n);
    calcCT(Process_Array,Ready, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}

gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$ ./Priority_Scheduling
Enter the number of Processes in the system:3
Enter the details of every process in increasing order of their arrival times.
Enter PID of Process 1 = 1
Enter Arrival Time of Process 1 = 0
Enter Burst Time of Process 1 = 12
Enter Priority of Process 1 = 1
Enter PID of Process 2 = 2
Enter Arrival Time of Process 2 = 1
Enter Burst Time of Process 2 = 23
Enter Priority of Process 2 = 4
Enter PID of Process 3 = 3
Enter Arrival Time of Process 3 = 11
Enter Burst Time of Process 3 = 22
Enter Priority of Process 3 = 3

Process 1
p_id=1
AT=0
BT=12
Priority=1
CT=57
TAT=57
WT=45

Process 2
p_id=2
AT=1
BT=23
Priority=4
CT=24
TAT=23
WT=0

Process 3
p_id=3
AT=11
BT=22
Priority=3
CT=46
TAT=35
WT=13

Average Turnaround time= 38.333
Average Waiting time= 19.333gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$
```

CODE ROUND ROBIN

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
struct Process {
```

```
int p_id;

int AT;

int BT;

int rem_BT;

int CT;

int TAT;

int WT;

}buffer;
```

```
void SeT(struct Process Ready[], int n){

    buffer.p_id=-1;

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

        Ready[i].p_id=-1;

        Ready[i].AT=-1;

        Ready[i].BT=-1;

        Ready[i].rem_BT=-1;

        Ready[i].CT=-1;

        Ready[i].TAT=-1;

        Ready[i].WT=-1;

    }

}
```

```
void display(struct Process Process_Array[], int n){

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

        int Pno=i+1;

        printf("\n\nProcess %d\n",Process_Array[i].p_id);

        printf("p_id=%d\n",Process_Array[i].p_id);

        printf("AT=%d\n",Process_Array[i].AT);

        printf("BT=%d\n",Process_Array[i].BT);

        printf("CT=%d\n",Process_Array[i].CT);

        printf("TAT=%d\n",Process_Array[i].TAT);

    }

}
```

```

        printf("WT=%d\n",Process_Array[i].WT);
        printf("Rem_BT=%d\n",Process_Array[i].rem_BT);
    }
}

void getStats(struct Process Process_Array[], int n){
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i=0; i<n;i++) {
        printf("Enter PID of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].p_id);
        printf("Enter Arrival Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].AT);
        printf("Enter Burst Time of Process %d = ",i+1);
        scanf("%d",&Process_Array[i].BT);
        Process_Array[i].rem_BT=Process_Array[i].BT;
    }
}

```

```

void queue(struct Process Ready[], struct Process Process_Array){
    int i=0;
    while(Ready[i].p_id!=-1){
        i++;
    }
    Ready[i]=Process_Array;
}

```

```

struct Process dequeue(struct Process Ready[]){
    struct Process temp=Ready[0];
    int i=0;
    while(Ready[i].p_id!=-1){
        i++;
    }
}

```

```

    }
    int j=0;
    for (j; j<i+1;j++){
        Ready[j]=Ready[j+1];
    }
    return temp;
}

```

```

int Ready_No(struct Process Ready[]){
    int i=0;
    while(Ready[i].p_id!=-1){
        i++;
    }
    return i;
}

```

```

void calcCT(struct Process Process_Array[], struct Process Ready[], int Time_Quantum, int n){
    int timeline=0;
    int counter=0;
    int Process_counter=0;
    while(counter !=n){
        for(int i=Process_counter;i<n;i++){
            if(Process_Array[i].AT<=timeline){
                queue(Ready, Process_Array[i]);
                Process_counter++;
            }
        }
        int ReadyQueueCheck=Ready_No(Ready);
        if(ReadyQueueCheck==0){
            timeline++;
        }
        else{

```

```

struct Process temp=dequeue(Ready);
if(temp.rem_BT<=Time_Quantum){
    for(int i=0;i<n;i++){
        if(Process_Array[i].p_id==temp.p_id){
            timeline=timeline+Process_Array[i].rem_BT;
            Process_Array[i].rem_BT=0;
            Process_Array[i].CT=timeline;
            counter++;
        }
    }
}
else{
    for(int i=0;i<n;i++){
        if(Process_Array[i].p_id==temp.p_id){
            timeline=timeline+Time_Quantum;
            for(int i=Process_counter;i<n;i++){
                if(Process_Array[i].AT<=timeline){
                    queue(Ready, Process_Array[i]);
                    Process_counter++;
                }
            }
            Process_Array[i].rem_BT=Process_Array[i].rem_BT-Time_Quantum;
            queue(Ready, Process_Array[i]);
        }
    }
}
}
}

void calcTAT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {

```



```

        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[],int n){
    for (int i=0; i<n;i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[],int n){
    float sumTAT=0;
    for (int i=0; i<n;i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}

void calcAvgWT(struct Process Process_Array[],int n){
    float sumWT=0;
    for (int i=0; i<n;i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}

int main(){
    int n, Time_Quantum;

    printf("\nEnter Number of Processes in the system: ");
    scanf ("%d",&n);

    printf("Enter Time Quantum: ");
    scanf ("%d",&Time_Quantum);

    struct Process Process_Array[100];
    struct Process Ready[100];

    SeT(Ready, 100);

```

}

OUTPUT ROUND ROBIN

	173,1	Bot
--	-------	-----

```
gaurav1020@DESKTOP-RORPIEK: ~/cyclesheet2
gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$ gcc Round_Robin.c -o Round_Robin
gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$ ./Round_Robin

Enter Number of Processes in the system: 3
Enter Time Quantum: 2

Enter the details of every process in increasing order of their arrival times.
Enter PID of Process 1 = 1
Enter Arrival Time of Process 1 = 0
Enter Burst Time of Process 1 = 12
Enter PID of Process 2 = 2
Enter Arrival Time of Process 2 = 1
Enter Burst Time of Process 2 = 23
Enter PID of Process 3 = 3
Enter Arrival Time of Process 3 = 11
Enter Burst Time of Process 3 = 22

Process 1
p_id=1
AT=0
BT=12
CT=26
TAT=26
WT=14
rem_BT=0

Process 2
p_id=2
AT=1
BT=23
CT=53
TAT=52
WT=29
rem_BT=0

Process 3
p_id=3
AT=11
BT=22
CT=57
TAT=46
WT=24
rem_BT=0

Average Turnaround time= 41.333
Average Waiting time= 22.333gaurav1020@DESKTOP-RORPIEK:~/cyclesheet2$
```

- (c) Consider a corporate hospital where we have n number of patients waiting for consultation. The amount of time required to serve a patient may vary, say 10 to 30 minutes. If a patient arrives with an emergency, he /she should be attended immediately before other patients, which may increase the waiting time of other patients. If you are given this problem with the following algorithms how would you devise an effective scheduling so that it optimizes the overall performance such as minimizing the waiting time of all patients. [Single queue or multi-level queue can be used]. Consider the availability of single and multiple doctors • Assign top priority for patients with emergency case, women, children, elders, and youngsters. • Patients coming for review may take less time than others. This can be taken into account while using SJF. 1. FCFS 2. SJF (High)

CODE FCFS

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
struct Process {
    int p_id;
    int AT;
    int BT;
    int rem_BT;
    int CT;
    int TAT;
```

```

        int WT;
        int Priority;
    };

void display(struct Process Process_Array[], int n) {
    for (int i = 0; i < n; i++) {
        int Pno = i + 1;
        printf("\n\nProcess %d\n", Pno);
        printf("p_id=%d\n", Process_Array[i].p_id);
        printf("AT=%d\n", Process_Array[i].AT);
        printf("BT=%d\n", Process_Array[i].BT);
        printf("Priority=%d\n", Process_Array[i].Priority);
        printf("CT=%d\n", Process_Array[i].CT);
        printf("TAT=%d\n", Process_Array[i].TAT);
        printf("WT=%d\n", Process_Array[i].WT);
    }
}

void getStats(struct Process Process_Array[], int n) {
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i = 0; i < n; i++) {
        Process_Array[i].p_id = i + 1;
        printf("Enter Arrival Time of Patient %d = ", i + 1);
        scanf("%d", & Process_Array[i].AT);
        printf("Enter Diagnosis Time of Patient %d = ", i + 1);
        scanf("%d", & Process_Array[i].BT);
        Process_Array[i].rem_BT = Process_Array[i].BT;
        printf("Enter Priority of Patient (0 for normal and 1 for emergency) %d = ", i + 1);
        scanf("%d", & Process_Array[i].Priority);
    }
}

int ATArray(struct Process Process_Array[], int ATArray[], int n){
    for(int i=0; i<n ; i++){
        ATArray[i]= Process_Array[i].AT;
    }
}

int Ready_to_run(struct Process Process_Array[], int n, int timeline){
    int r=0;
    for (int i=0;i<n; i++){
        if (Process_Array[i].AT<=timeline && Process_Array[i].rem_BT!=0){
            r++;
        }
    }
    return r;
}

void calcCT(struct Process Process_Array[], int n) {
    int timeline = 0;
    int counter = 0;
    int ATArray[100];

```

```

ATArray(Process_Array, ATArray, n);
while (counter != n) {
    int runner=-1;
    int j= Ready_to_run(Process_Array, n, timeline);
    if (j == 0) {
        timeline++;
    } else {
        int flag=0;
        for (int i=0; i<n; i++){
            if (Process_Array[i].AT<=timeline && Process_Array[i].rem_BT!=0 &&
Process_Array[i].Priority==1){
                flag=1;
            }
        }
        if (flag==1){
            for (int i=0; i<n; i++){
                if (Process_Array[i].AT<=timeline && Process_Array[i].rem_BT!=0 &&
Process_Array[i].Priority==1){
                    runner=i;
                    break;
                }
            }
        }
        else {
            for (int i=0; i<n; i++){
                if (Process_Array[i].AT<=timeline && Process_Array[i].rem_BT!=0){
                    runner=i;
                    break;
                }
            }
        }
        Process_Array[runner].rem_BT=Process_Array[runner].rem_BT-1;
        timeline++;
        if (Process_Array[runner].rem_BT==0){
            Process_Array[runner].CT=timeline;
            counter++;
        }
    }
}

void calcTAT(struct Process Process_Array[], int n) {
    for (int i = 0; i < n; i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[], int n) {
    for (int i = 0; i < n; i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

```

```

    }
}
void calcAvgTAT(struct Process Process_Array[], int n) {
    float sumTAT = 0;
    for (int i = 0; i < n; i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT / n));
}
void calcAvgWT(struct Process Process_Array[], int n) {
    float sumWT = 0;
    for (int i = 0; i < n; i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT / n));
}
int main() {
    int n;
    printf("\nEnter the number of Processes in the system:");
    scanf("%d", & n);
    struct Process Process_Array[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}

```

OUTPUT FCFS

```
        }
        }
        }
        Process_Array[runner].rem_BT=Process_Array[runner].rem_BT-1;
        timeline++;
        if (Process_Array[runner].rem_BT==0){
            Process_Array[runner].CT=timeline;
            counter++;
        }
    }
}

void calcTAT(struct Process Process_Array[], int n) {
    for (int i = 0; i < n; i++) {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[], int n) {
    for (int i = 0; i < n; i++) {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[], int n) {
    float sumTAT = 0;
    for (int i = 0; i < n; i++) {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT / n));
}

void calcAvgWT(struct Process Process_Array[], int n) {
    float sumWT = 0;
    for (int i = 0; i < n; i++) {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT / n));
}

int main() {
    int n;
    printf("\nEnter the number of Processes in the system:");
    scanf("%d", &n);
    struct Process Process_Array[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}
```

```
gaarav1020@DESKTOP-RORPIEK: ~/DA2
gaarav1020@DESKTOP-RORPIEK:~/DA2$ vi 2cFCFS.c
gaarav1020@DESKTOP-RORPIEK:~/DA2$ gcc 2cFCFS.c -o 2cFCFS
gaarav1020@DESKTOP-RORPIEK:~/DA2$ ./2cFCFS

Enter the number of Processes in the system:3

Enter the details of every process in increasing order of their arrival times.
Enter Arrival Time of Patient 1 = 0
Enter Diagnosis Time of Patient 1 = 12
Enter Priority of Patient (0 for normal and 1 for emergency) 1 = 0
Enter Arrival Time of Patient 2 = 2
Enter Diagnosis Time of Patient 2 = 23
Enter Priority of Patient (0 for normal and 1 for emergency) 2 = 1
Enter Arrival Time of Patient 3 = 10
Enter Diagnosis Time of Patient 3 = 8
Enter Priority of Patient (0 for normal and 1 for emergency) 3 = 0

Process 1
p_id=1
AT=0
BT=12
Priority=0
CT=35
TAT=35
WT=23

Process 2
p_id=2
AT=2
BT=23
Priority=1
CT=25
TAT=23
WT=0

Process 3
p_id=3
AT=10
BT=8
Priority=0
CT=43
TAT=33
WT=25

Average Turnaround time= 30.333
Average Waiting time= 16.000gaarav1020@DESKTOP-RORPIEK:~/DA2$
```

CODE SJF

{Context: If the patients are for normal diagnosis then they are attended based on their diagnosis time. During that time if a patient with emergency condition arrives, treatment of ongoing patient with normal diagnosis is stopped immediately and the patient in emergency condition is diagnosed completely. If another patient with emergency condition arrives when treatment of an ongoing patient with emergency condition is going on, the treatment of ongoing patient is not halted and the next emergency case is dealt with after the current emergency is resolved.}

```

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
struct Process {
int p_id;
int AT;
int BT;
int CT;
int TAT;
int WT;
int flag;
int rem_BT;
int Priority;
};
void display(struct Process Process_Array[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int Pno = i + 1;
        printf("\n\nProcess %d\n", Pno);
        printf("p_id=%d\n", Process_Array[i].p_id);
        printf("AT=%d\n", Process_Array[i].AT);
        printf("BT=%d\n", Process_Array[i].BT);
        printf("CT=%d\n", Process_Array[i].CT);
        printf("TAT=%d\n", Process_Array[i].TAT);
        printf("WT=%d\n", Process_Array[i].WT);

    }
}

void getStats(struct Process Process_Array[], int n)
{
    printf("\nEnter the details of every process in increasing order of their arrival times.\n");
    for (int i = 0; i < n; i++)
    {
        Process_Array[i].p_id=i+1;
        printf("Enter Arrival Time of Patient %d = ", i + 1);
        scanf("%d", &Process_Array[i].AT);
        printf("Enter Diagnosis Time of Patient %d = ", i + 1);
        scanf("%d", &Process_Array[i].BT);
        printf("Enter Priority of Patient %d (0 for normal, 1 for emergency)= ", i + 1);
        scanf("%d", &Process_Array[i].Priority);
        Process_Array[i].flag = 0;
        Process_Array[i].rem_BT = Process_Array[i].BT;

    }
}

```



```

}

int MinBT(struct Process Process_Array[], int n)
{
    int min = 0;
    for (int i = 0; i < n; i++)
    {
        if (Process_Array[i].rem_BT < Process_Array[min].rem_BT &&
Process_Array[i].flag == 0)
        {
            min = i;
        }
    }

    return min;
}

void calcCT(struct Process Process_Array[], struct Process Ready[], int n)
{
    int flag=0;
    int timeline = 0;
    int counter = 0;
    int min = 0;
    int j = 0;
    while (counter != n)
    {
        int runner=-1;
        int flag=0;
        j = 0;
        for (int i = 0; i < n; i++)
        {
            if (Process_Array[i].AT <= timeline && Process_Array[i].flag == 0)
            {
                Ready[j] = Process_Array[i];
                j++;
            }
        }

        if (j == 0)
        {
            timeline++;
        }
        else
        {
            for (int i=0; i<j; i++){
                if (Ready[i].Priority==1){
                    flag=1;
                    runner=i;

```

```

        break;
    }
    }
    if (flag==1) {
    for (int i=0; i<n; i++){
        if (Ready[runner].p_id==Process_Array[i].p_id){
            runner=i;
            break;
        }
    }
    Process_Array[runner].flag=1;
    Process_Array[runner].rem_BT=0;
    timeline=timeline+Process_Array[runner].BT;
    counter++;
    Process_Array[runner].CT=timeline;
    }
    if (flag==0){
    min = MinBT(Ready, j);
    for (int i = 0; i < n; i++)
    {
        if (Process_Array[i].p_id == Ready[min].p_id)
        {
            runner=i;
            break;
        }
    }
    Process_Array[runner].rem_BT=Process_Array[runner].rem_BT-1;
    timeline++;
    if (Process_Array[runner].rem_BT==0){
        Process_Array[runner].CT=timeline;
        Process_Array[runner].flag=1;
        counter++;
    }
    }
    }
}

void calcTAT(struct Process Process_Array[], int n)
{
    for (int i = 0; i < n; i++)
    {
        Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
    }
}

void calcWT(struct Process Process_Array[], int n)
{

```

```

        for (int i = 0; i < n; i++)
        {
            Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
        }
    }

void calcAvgTAT(struct Process Process_Array[], int n)
{
    float sumTAT = 0;
    for (int i = 0; i < n; i++)
    {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }

    printf("\n\nAverage Turnaround time= %.3f", (sumTAT / n));
}

void calcAvgWT(struct Process Process_Array[], int n)
{
    float sumWT = 0;
    for (int i = 0; i < n; i++)
    {
        sumWT = sumWT + Process_Array[i].WT;
    }

    printf("\n\nAverage Waiting time= %.3f", (sumWT / n));
}

int main()
{
    int n;
    printf("\nEnter the number of processes in the system:");
    scanf("%d", &n);
    struct Process Process_Array[100];
    struct Process Ready[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, Ready, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}

```

OUTPUT SJF

```
Select gaurav1020@DESKTOP-R0RPIEK: ~/DA2
for (int i = 0; i < n; i++)
{
    Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
}

void calcWT(struct Process Process_Array[], int n)
{
    for (int i = 0; i < n; i++)
    {
        Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
    }
}

void calcAvgTAT(struct Process Process_Array[], int n)
{
    float sumTAT = 0;
    for (int i = 0; i < n; i++)
    {
        sumTAT = sumTAT + Process_Array[i].TAT;
    }
    printf("\n\nAverage Turnaround time= %.3f", (sumTAT / n));
}

void calcAvgWT(struct Process Process_Array[], int n)
{
    float sumWT = 0;
    for (int i = 0; i < n; i++)
    {
        sumWT = sumWT + Process_Array[i].WT;
    }
    printf("\n\nAverage Waiting time= %.3f", (sumWT / n));
}

int main()
{
    int n;
    printf("\nEnter the number of processes in the system:");
    scanf("%d", &n);
    struct Process Process_Array[100];
    struct Process Ready[100];
    getStats(Process_Array, n);
    calcCT(Process_Array, Ready, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}

gaurav1020@DESKTOP-R0RPIEK: ~/DA2$ vi 2cSJF.c
gaurav1020@DESKTOP-R0RPIEK: ~/DA2$ gcc 2cSJF.c -o 2cSJF
gaurav1020@DESKTOP-R0RPIEK: ~/DA2$ ./2cSJF

Enter the number of processes in the system:3

Enter the details of every process in increasing order of their arrival times.
Enter Arrival Time of Patient 1 = 0
Enter Diagnosis Time of Patient 1 = 10
Enter Priority of Patient 1 (0 for normal, 1 for emergency)= 0
Enter Arrival Time of Patient 2 = 2
Enter Diagnosis Time of Patient 2 = 23
Enter Priority of Patient 2 (0 for normal, 1 for emergency)= 1
Enter Arrival Time of Patient 3 = 15
Enter Diagnosis Time of Patient 3 = 4
Enter Priority of Patient 3 (0 for normal, 1 for emergency)= 0

Process 1
p_id=1
AT=0
BT=10
CT=37
TAT=37
WT=27

Process 2
p_id=2
AT=2
BT=23
CT=25
TAT=23
WT=0

Process 3
p_id=3
AT=15
BT=4
CT=29
TAT=14
WT=10

Average Turnaround time= 24.667
Average Waiting time= 12.333gaurav1020@DESKTOP-R0RPIEK:~/DA2$
```

- (d) Simulate with a program to provide deadlock avoidance of Banker's Algorithm including Safe state and additional resource request (High).

CODE

```
int main()
{
    int n, m, i, j, k;
    n = 5;
    m = 3;
    int alloc[5][3] = { { 0, 1, 0 },
```

```

{ 2, 0, 0 },
{ 3, 0, 2 },
{ 2, 1, 1 },
{ 0, 0, 2 } };
int max[5][3] = { { 7, 5, 3 },
{ 3, 2, 2 },
{ 9, 0, 2 },
{ 2, 2, 2 },
{ 4, 3, 3 } };
int avail[3] = { 3, 3, 2 };
int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
    f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
        need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {
            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]){
                    flag = 1;
                    break;
                }
            }
            if (flag == 0) {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
        }
    }
}
printf("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
    printf(" P%d ->", ans[i]);
printf(" P%d", ans[n - 1]);
return (0);
}

```

OUTPUT

```
gaurav1020@DESKTOP-RORPIEK: ~/DA2
int main()
{
    int n, m, i, j, k;
    n = 3;
    m = 3;
    int alloc[i][j] = { { 0, 1, 0 },
        { 2, 0, 0 },
        { 2, 0, 1 },
        { 2, 1, 1 },
        { 0, 0, 2 } };
    int max[i][j] = { { 2, 0, 3 },
        { 3, 2, 1 },
        { 0, 0, 2 },
        { 2, 2, 2 },
        { 4, 3, 1 } };
    int avail[i] = { 3, 3, 2 };
    int fin, ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf("%d ", ans[i]);
    printf("%d", ans[n - 1]);
    return (0);
}

2d.c" 52L, 1501C
52,1 Bot

gaurav1020@DESKTOP-RORPIEK: ~/DA2
gaurav1020@DESKTOP-RORPIEK:~/DA2$ vi 2d.c
gaurav1020@DESKTOP-RORPIEK:~/DA2$ gcc 2d.c -o 2d
gaurav1020@DESKTOP-RORPIEK:~/DA2$ ./2d
Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2gaurav1020@DESKTOP-RORPIEK:~/DA2$
```