CSE 2005

OPERATING SYSTEMS

Assessment – 2

L7+L8 | PLBG17

WINTER SEMESTER 2020–21

by

SHARADINDU ADHIKARI

19BCE2105

Process and Thread Management (PTM)

(a) Process and Thread Management

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

```
#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);
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:~$ cd /mnt/c/Users/Sharadindu/Desktop
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ sed -i 's/\r//' PTMa.c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat PTMa.c
#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);
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc PTMa.c -o PTMa
PTMa.c: In function ‘wait’:
PTMa.c:11:5: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
   11 |     printf("Sleeping \n");
      |     ^~~~~~
PTMa.c:11:5: warning: incompatible implicit declaration of built-in function ‘printf’
PTMa.c:3:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
    2 | #include <unistd.h>
  +++ |+#include <stdio.h>
    3 |
PTMa.c:12:26: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘long int’ [-Wformat=]
   12 |     printf("Thread id : %d\n" ,tid);
      |                         ~^     ~~~
      |                          |     |
      |                          int   long int
      |                         %ld
PTMa.c: In function ‘main’:
PTMa.c:25:9: warning: incompatible implicit declaration of built-in function ‘printf’
   25 |         printf("parent creating thread, %d\n",i);
      |         ^~~~~~
PTMa.c:25:9: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
PTMa.c:26:55: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   26 |         rc = pthread_create(&threads[i], &attr, wait, (void *)i );
      |                                                        ^
PTMa.c:29:13: warning: implicit declaration of function ‘exit’ [-Wimplicit-function-declaration]
   29 |             exit(-1);
      |             ^~~~
```

```
PTMa.c:29:13: warning: incompatible implicit declaration of built-in function ‘exit’
PTMa.c:3:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
    2 | #include <unistd.h>
  +++ |+#include <stdlib.h>
    3 |
PTMa.c:36:13: warning: incompatible implicit declaration of built-in function ‘printf’
   36 |             printf ("Error:unable to join, %d\n" ,rc);
      |             ^~~~~~
PTMa.c:36:13: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
PTMa.c:37:13: warning: incompatible implicit declaration of built-in function ‘exit’
   37 |             exit(-1);
      |             ^~~~
PTMa.c:37:13: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
PTMa.c:39:9: warning: incompatible implicit declaration of built-in function ‘printf’
   39 |         printf("Parent Process: completed thread id :%d",i) ;
      |         ^~~~~~
PTMa.c:39:9: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
PTMa.c:40:41: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘void *’ [-Wformat=]
   40 |         printf("  exiting with status :%d\n",status);
      |                                         ~^     ~~~~~~
      |                                          |     |
      |                                          int   void *
      |                                         %p
PTMa.c:42:5: warning: incompatible implicit declaration of built-in function ‘printf’
   42 |     printf ("Parent: program exiting.\n");
      |     ^~~~~~
PTMa.c:42:5: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
/usr/bin/ld: /tmp/ccZLs1Xv.o: in function `main':
PTMa.c:(.text+0xd0): undefined reference to `pthread_create'
/usr/bin/ld: PTMa.c:(.text+0x131): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./PTMa
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.
```

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

```c
#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);
}
```

```
sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ sed -i 's/\r//' PTMb.c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat PTMb.c
#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);
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc PTMb.c -o PTMb -lpthread
PTMb.c: In function ‘Factorial’:
PTMb.c:7:13: warning: initialization of ‘int’ from ‘int *’ makes integer from pointer without a cast [-Wint-conversior
    7 |     int n = (int *)var;
      |             ^
PTMb.c: In function ‘main’:
PTMb.c:20:41: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   20 |     pthread_create (&tid,NULL,Factorial,(void*)n);
      |                                         ^
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./PTMb

This is parent process with pid:177
Before Thread Creation.
Thread will now be Created...
Enter the number whose Factorial you want to calculate: 13

This is thread process with pid:177
Factorial of 13 is: 1932053504
```

(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)**

```
//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); // ftok function is used to generate unique
key
    int shmid = shmget(my_key,1024,0666|IPC_CREAT); // shmget returns an ide in shmid
    char *str = (char*) shmat(shmid,(void*)0,0); // shmat to join to shared memory
    cout<<"Write Data : ";
    fgets(str, 50, stdin);
    printf("Data written in memory: %s\n",str);
    //detach from shared memory
    shmdt(str);

    return 0;
}
```

```
//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); // ftok function is used to generate unique
key
    int shmid = shmget(my_key,1024,0666|IPC_CREAT); // shmget returns an ide in shmid
    char *str = (char*) shmat(shmid,(void*)0,0); // shmat to join to shared memory
    printf("Data read from memory:");
    for (int x=0; x<strlen(str); x++)
        putchar(toupper(str[x]));
    shmdt(str);
    shmctl(shmid,IPC_RMID,NULL); // destroy the shared memory

    return 0;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat PTMcServer.cpp
//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); // ftok function is used to generate unique key
    int shmid = shmget(my_key,1024,0666|IPC_CREAT); // shmget returns an ide in shmid
    char *str = (char*) shmat(shmid,(void*)0,0); // shmat to join to shared memory
    cout<<"Write Data : ";
    fgets(str, 50, stdin);
    printf("Data written in memory: %s\n",str);
    //detach from shared memory
    shmdt(str);

    return 0;
}sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ g++ PTMcServer.cpp -o PTMcServer
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./PTMcServer
Write Data : qwertyuioplkjhgfdsazxcvbnm
Data written in memory: qwertyuioplkjhgfdsazxcvbnm

sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat PTMcClient.cpp
//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); // ftok function is used to generate unique key
    int shmid = shmget(my_key,1024,0666|IPC_CREAT); // shmget returns an ide in shmid
    char *str = (char*) shmat(shmid,(void*)0,0); // shmat to join to shared memory
    printf("Data read from memory:");
    for (int x=0; x<strlen(str); x++)
        putchar(toupper(str[x]));
    shmdt(str);
    shmctl(shmid,IPC_RMID,NULL); // destroy the shared memory

    return 0;
}sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ g++ PTMcClient.cpp -o PTMcClient
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./PTMcClient
Data read from memory:QWERTYUIOPLKJHGFDSAZXCVBNM
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

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)**

```
#include<stdio.h>
#include<pthread.h>
int arr[50],n,i;
void *th()
{
    int sum=0;
    float average;
    printf("Enter the limit:");
    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);
    //printf("\n done and my value is %d",n);
```

```
    n=pthread_create(&t2,NULL,&th1,NULL);
    pthread_join(t2,NULL);
    //printf("\n done and my value is %d",n);
    n=pthread_create(&t3,NULL,&th2,NULL);
    pthread_join(t3,NULL);
    //printf("\n done and my value is %d",n);
    return 0;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat PTMd.c
#include<stdio.h>
#include<pthread.h>
int arr[50],n,i;
void *th()
{
    int sum=0;
    float average;
    printf("Enter the limit:");
    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);
    //printf("\n done and my value is %d",n);
    n=pthread_create(&t2,NULL,&th1,NULL);
    pthread_join(t2,NULL);
    //printf("\n done and my value is %d",n);
    n=pthread_create(&t3,NULL,&th2,NULL);
    pthread_join(t3,NULL);
    //printf("\n done and my value is %d",n);
    return 0;
}sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$  gcc PTMd.c -o PTMd -lpthread
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$  ./PTMd
Enter the limit:9
13
45
2
78
139
62
47
88
91
The Average value is:62.000000
The Minimum  value is:=2
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

CPU Scheduling (CS)

CPU Scheduling

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

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){
```

```c
    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;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CSaFCFS.c
#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;
}
```

```
sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CSaFCFS.c -o CSaFCFS
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CSaFCFS

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
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

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;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CSaSJF.c
#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;
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CSaSJF.c -o CSaSJF
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CSaSJF

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=57
TAT=56
WT=33


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


Average Turnaround time= 30.333
```

Priority (Non-preemptive):

```
#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;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CSaPriority.c
#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;
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CSaPriority.c -o CSaPriority
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CSaPriority

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 = 2


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=2
CT=46
TAT=44
WT=33


Average Turnaround time= 30.000
```

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

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;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CSbPriority.c
#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;
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CSbPriority.c -o CSbPriority
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CSbPriority

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= 19.333sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

Roundrobin:

```
#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){
```

```c
    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);
    struct Process temp=dequeue(Ready);
    getStats(Process_Array, n);
    calcCT(Process_Array,Ready, Time_Quantum, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```c
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CSbRR.c
#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);
    }
}
```

sharadindu.adhikari2019@vitstudent.ac.in

```
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 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);
    struct Process temp=dequeue(Ready);
    getStats(Process_Array, n);
    calcCT(Process_Array,Ready, Time_Quantum, n);
    calcTAT(Process_Array, n);
    calcWT(Process_Array, n);
    display(Process_Array, n);
    calcAvgTAT(Process_Array, n);
    calcAvgWT(Process_Array, n);
    return 0;
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CSbRR.c -o CSbRR
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CSbRR

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
```

(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 (primitive and non-pre-emptive) **(High)**

FCFS:

```
#include<iostream>
using namespace std;
struct prodet
{
    int bt;
    int art;
    int ft;
    int tat;
    int wt;
}ar[20], tmp;
int main()
{
    int n,avwt=0,avtat=0,i,j;
    cout<<"Enter total number of patients:";
    cin>>n;
    cout<<"\nEnter Patient Time\n";
    for(i=0;i<n;i++)
    {
        cout<<"P["<<i+1<<"]:";
        cin>>ar[i].bt;
    }
    cout<<"\nEnter Patient Arrival Time\n";
    for(i=0;i<n;i++)
    {
        cout<<"P["<<i+1<<"]:";
        cin>>ar[i].art;
    }
    for(int i=0;i<n-1; i++)
    {
        for(j = 0;j <n-1; j++)
        {
            if(ar[j].art>ar[j+1].art)
            {
                tmp = ar[j];
                ar[j]=ar[j+1];
                ar[j+1]=tmp;
            }
        }
    }
    ar[0].ft=ar[0].bt+ar[0].art;
```

```
    for(i=1;i<n;i++)
    {
        ar[i].ft=ar[i-1].ft+ar[i].bt;
    }
    cout<<endl<<"********************"<<endl;
    cout<<"\nProcess\t\tTime\tArival Time";
    for(i=0;i<n;i++)
    {
        cout<<"\nP["<<i+1<<"]"<<"\t\t"<<ar[i].bt<<"\t\t"<<ar[i].art;
    }
    cout<<"\nProcess\t\tTime\tWaiting Time\tTurnaround Time";
    for(i=0;i<n;i++)
    {
        ar[i].tat=ar[i].ft - ar[i].art;
        ar[i].wt = ar[i].tat - ar[i].bt;
        avwt+=ar[i].wt;
        avtat+=ar[i].tat;
        cout<<"\nP["<<i+1<<"]"<<"\t\t"<<ar[i].bt<<"\t\t"<<ar[i].wt<<"\t\t"<<ar[i].tat;
    }
    avwt/=i;
    avtat/=i;
    cout<<"\n\nAverage Waiting Time:"<<avwt;
    cout<<"\nAverage Turnaround Time:"<<avtat;
    return 0;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CScFCFS.c
#include<iostream>
using namespace std;
struct prodet
{
    int bt;
    int art;
    int ft;
    int tat;
    int wt;
}ar[20], tmp;
int main()
{
    int n,avwt=0,avtat=0,i,j;
    cout<<"Enter total number of patients:";
    cin>>n;
    cout<<"\nEnter Patient Time\n";
    for(i=0;i<n;i++)
    {
        cout<<"P["<<i+1<<"]:";
        cin>>ar[i].bt;
    }
    cout<<"\nEnter Patient Arrival Time\n";
    for(i=0;i<n;i++)
    {
        cout<<"P["<<i+1<<"]:";
        cin>>ar[i].art;
    }
    for(int i=0;i<n-1; i++)
    {
        for(j = 0;j <n-1; j++)
        {
            if(ar[j].art>ar[j+1].art)
            {
                tmp = ar[j];
                ar[j]=ar[j+1];
                ar[j+1]=tmp;
            }
        }
    }
    ar[0].ft=ar[0].bt+ar[0].art;
    for(i=1;i<n;i++)
    {
        ar[i].ft=ar[i-1].ft+ar[i].bt;
    }
```

```
    cout<<endl<<"*******************"<<endl;
    cout<<"\nProcess\t\tTime\tArival Time";
    for(i=0;i<n;i++)
    {
        cout<<"\nP["<<i+1<<"]"<<"\t\t"<<ar[i].bt<<"\t\t"<<ar[i].art;
    }
    cout<<"\nProcess\t\tTime\tWaiting Time\tTurnaround Time";
    for(i=0;i<n;i++)
    {
        ar[i].tat=ar[i].ft - ar[i].art;
        ar[i].wt = ar[i].tat - ar[i].bt;
        avwt+=ar[i].wt;
        avtat+=ar[i].tat;
        cout<<"\nP["<<i+1<<"]"<<"\t\t"<<ar[i].bt<<"\t\t"<<ar[i].wt<<"\t\t"<<ar[i].tat;
    }
    avwt/=i;
    avtat/=i;
    cout<<"\n\nAverage Waiting Time:"<<avwt;
    cout<<"\nAverage Turnaround Time:"<<avtat;
    return 0;
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ g++ CScFCFS.c -o CScFCFS
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CScFCFS
Enter total number of patients:5

Enter Patient Time
P[1]:11
P[2]:13
P[3]:15
P[4]:19
P[5]:21

Enter Patient Arrival Time
P[1]:12
P[2]:15
P[3]:16
P[4]:13
P[5]:19

*******************

Process         Time    Arival Time
P[1]            11              12
P[2]            19              13
P[3]            13              15
P[4]            15              16
P[5]            21              19
Process         Time    Waiting Time    Turnaround Time
P[1]            11              0               11
P[2]            19              10              29
P[3]            13              27              40
P[4]            15              39              54
P[5]            21              51              72

Average Waiting Time:25
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

SJF (Preemptive):

```
#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 PATIENTS:");
    scanf("%d",&n);
    printf("\nEnter Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;//contains process number
    }//sorting burst time in ascending order using selection sort
    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;//waiting time for first process will be zero
//calculate waiting time
    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; //average waiting time
    total=0;
    printf("\nProcess\t\tPATIENT Time\tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i]; //calculate turnaround time
        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);
    return 0;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CScSJFp.c
#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 PATIENTS:");
scanf("%d",&n);
printf("\nEnter Time:\n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
p[i]=i+1;//contains process number
}//sorting burst time in ascending order using selection sort
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;//waiting time for first process will be zero
//calculate waiting time
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; //average waiting time
total=0;
printf("\nProcess\t\tPATIENT Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i]; //calculate turnaround time
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);
return 0;
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CScSJFp.c -o CScSJFp
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CScSJFp
Enter number of PATIENTS:5

Enter Time:
p1:13
p2:15
p3:17
p4:19
p5:22

Process         PATIENT Time    Waiting Time    Turnaround Time
p1               13              0                    13
p2               15              13                   28
p3               17              28                   45
p4               19              45                   64
p5               22              64                   86

Average Waiting Time=30.000000
Average Turnaround Time=47.200001
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

SJF (Non-preemptive):

```
#include<stdio.h>
typedef struct nonpresjf
{
    int at,bt,ft,tat,wt;
}nonpresjf;
nonpresjf p[20],p1[20];
int main()
{
    int i,limit,nextval,m,min,n;
    p[0].wt=p[0].tat=0;
    printf("\nEnter the no of PATIENTS:");
    scanf("%d",&n);
    printf("\nEnter Patient Arrival Time\n");
    for(i=1;i<=n;i++)
    {
        printf("P[%d]:",i);
        scanf("%d",&p[i].at);
    }
    limit=p[1].at;
    printf("\nEnter Patient Burst Time\n");
    for(i=1;i<=n;i++)
    {
        printf("P[%d]:",i);
        scanf("%d",&p[i].bt);
    }
    for(i=1;i<=n;i++)
        limit+=p[i].bt;
    for(i=1;i<=n;i++)
        p1[i]=p[i];
    printf("\n\nGantt chart is as follows:");
    printf("%d",p[1].at);
    nextval=p[1].at;
    m=1;
    do
    {
        min = 9999;
        for(i=1;p[i].at<=nextval && i<=n ;i++)
            if(p[i].bt<min && p[i].bt>0)
            {
                m=i;
                min=p[i].bt;
            }
        nextval+=p[m].bt;
        p[m].bt=0;
        printf("->P%d->%d",m,nextval);
        if(p[m].bt==0)
        {
            p[m].ft=nextval;
            p[m].tat=p[m].ft-p[m].at;
            p[m].wt=p[m].tat-p1[m].bt;
            p[0].tat+=p[m].tat;
            p[0].wt+=p[m].wt;
        }
    }while(nextval<limit);
    p[0].tat=p[0].tat/n;
    p[0].wt=p[0].wt/n;
    printf("\n\n-----TABLE---------\n");
    printf("\nProcess\tAT\tBT\tFT\tTAT\tWT\n");
    for(i=1;i<=n;i++)

printf("\nP%d\t%d\t%d\t%d\t%d\t%d\n",i,p[i].at,p1[i].bt,p[i].ft,p[i].tat,p[i].wt);
    printf("\n\n---------\n");
    return 0;
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CScSJFnp.c
#include<stdio.h>
typedef struct nonpresjf
{
int at,bt,ft,tat,wt;
}nonpresjf;
nonpresjf p[20],p1[20];
int main()
{
int i,limit,nextval,m,min,n;
p[0].wt=p[0].tat=0;
printf("\nEnter the no of PATIENTS:");
scanf("%d",&n);
printf("\nEnter Patient Arrival Time\n");
for(i=1;i<=n;i++)
{
printf("P[%d]:",i);
scanf("%d",&p[i].at);
}
limit=p[1].at;
printf("\nEnter Patient Burst Time\n");
for(i=1;i<=n;i++)
{
printf("P[%d]:",i);
scanf("%d",&p[i].bt);
}
for(i=1;i<=n;i++)
limit+=p[i].bt;
for(i=1;i<=n;i++)
p1[i]=p[i];
printf("\n\nGantt chart is as follows:");
printf("%d",p[1].at);
nextval=p[1].at;
m=1;
do
{
min = 9999;
for(i=1;p[i].at<=nextval && i<=n ;i++)
if(p[i].bt<min && p[i].bt>0)
{
m=i;
min=p[i].bt;
}
nextval+=p[m].bt;
p[m].bt=0;
printf("->P%d->%d",m,nextval);
if(p[m].bt==0)
{
p[m].ft=nextval;
p[m].tat=p[m].ft-p[m].at;
p[m].wt=p[m].tat-p1[m].bt;
p[0].tat+=p[m].tat;
p[0].wt+=p[m].wt;
}
}while(nextval<limit);
p[0].tat=p[0].tat/n;
p[0].wt=p[0].wt/n;
printf("\n\n-----TABLE---------\n");
printf("\nProcess\tAT\tBT\tFT\tTAT\tWT\n");
for(i=1;i<=n;i++)
printf("\nP%d\t%d\t%d\t%d\t%d\t%d\n",i,p[i].at,p1[i].bt,p[i].ft,p[i].tat,p[i].wt);
printf("\n\n---------\n");
return 0;
}
```

```
sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CScSJFnp.c -o CScSJFnp
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CScSJFnp

Enter the no of PATIENTS:5

Enter Patient Arrival Time
P[1]:12
P[2]:13
P[3]:11
P[4]:15
P[5]:19

Enter Patient Burst Time
P[1]:33
P[2]:21
P[3]:14
P[4]:15
P[5]:13


Gantt chart is as follows:12->P1->45->P5->58->P3->72->P4->87->P2->108

-----TABLE---------

Process AT      BT      FT      TAT     WT

P1      12      33      45      33      0

P2      13      21      108     95      74

P3      11      14      72      61      47

P4      15      15      87      72      57

P5      19      13      58      39      26


---------
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```

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

```
#include <stdio.h>
int main()
{
    int n, m, i, j, k;
    n = 5;
    m = 3;
    int alloc[5][3] = { { 0, 1, 0 }, // P0
                        { 2, 0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4
    int max[5][3] = { { 7, 5, 3 }, // P0
                      { 3, 2, 2 }, // P1
                      { 9, 0, 2 }, // P2
                      { 2, 2, 2 }, // P3
                      { 4, 3, 3 } }; // P4

    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);
}
```

sharad@Rohans-Workstation: /mnt/c/Users/Sharadindu/Desktop

```
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ cat CSd.c
#include <stdio.h>
int main()
{
    int n, m, i, j, k;
    n = 5;
    m = 3;
    int alloc[5][3] = { { 0, 1, 0 }, // P0
                        { 2, 0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4
   int max[5][3] = { { 7, 5, 3 }, // P0
                      { 3, 2, 2 }, // P1
                      { 9, 0, 2 }, // P2
                      { 2, 2, 2 }, // P3
                      { 4, 3, 3 } }; // P4

    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);
}
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ gcc CSd.c -o CSd -lpthread
sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$ ./CSd
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2sharad@Rohans-Workstation:/mnt/c/Users/Sharadindu/Desktop$
```