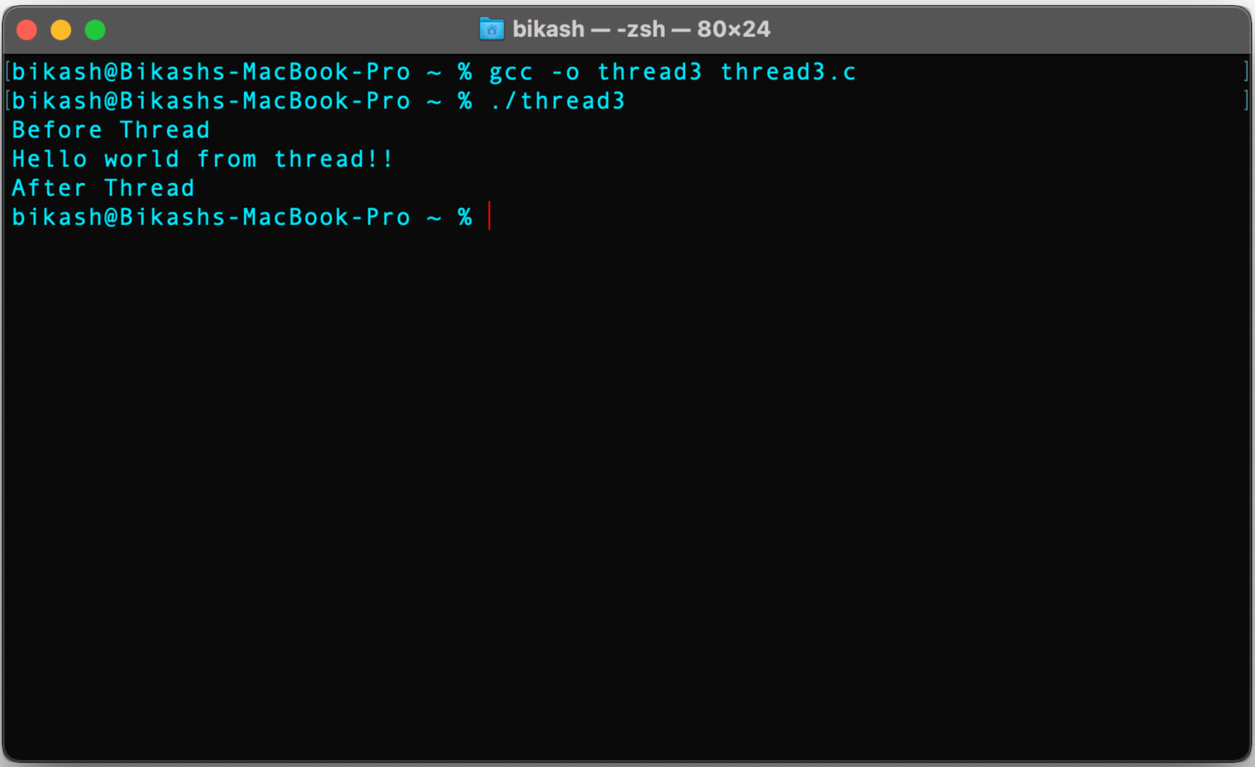
Process and thread management:
Q. Write a program to create a thread.

CODE:

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

// A normal C function that is executed as a thread
// when its name is specified in pthread_create()
void *myThreadFun(void *vargp)
{
        sleep(1);
        printf("Hello world from thread!! \n");
        return NULL;
}

int main()
{
        pthread_t thread_id;
        printf("Before Thread\n");
        pthread_create(&thread_id, NULL, myThreadFun, 
NULL);
        pthread_join(thread_id, NULL);
        printf("After Thread\n");
        exit(0);
}
```

OUTPUT:



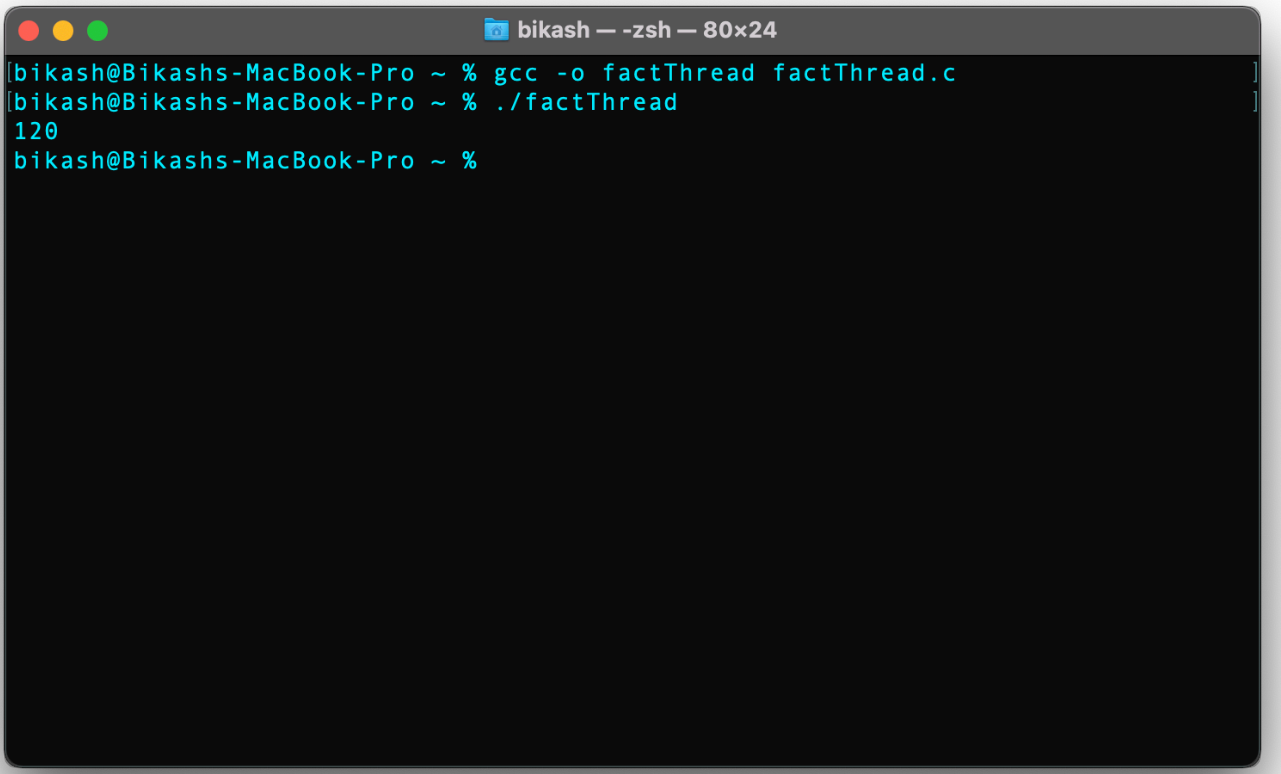
```
bikash@Bikashs-MacBook-Pro ~ % gcc -o thread3 thread3.c
bikash@Bikashs-MacBook-Pro ~ % ./thread3
Before Thread
Hello world from thread!!
After Thread
bikash@Bikashs-MacBook-Pro ~ %
```

(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 <pthread.h>

int n=5;
int *pN = &n;
void* fact(void *n){
 int *N = (int*)n;
 int result=1;
 for(int i=*N; i>=1; i--){
   result *= i;
}
 printf("%d\n", result);
}

int main(){
  pthread_t ID;

  pthread_create(&ID, NULL, &fact, (void*)pN);
  pthread_join(ID, NULL);
  exit(EXIT_SUCCESS);
  return 0;
}
```

OUTPUT:



```
[bikash@Bikashs-MacBook-Pro ~ % gcc -o factThread factThread.c                ]
[bikash@Bikashs-MacBook-Pro ~ % ./factThread                                  ]
 120
 bikash@Bikashs-MacBook-Pro ~ %
```

(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-Side:

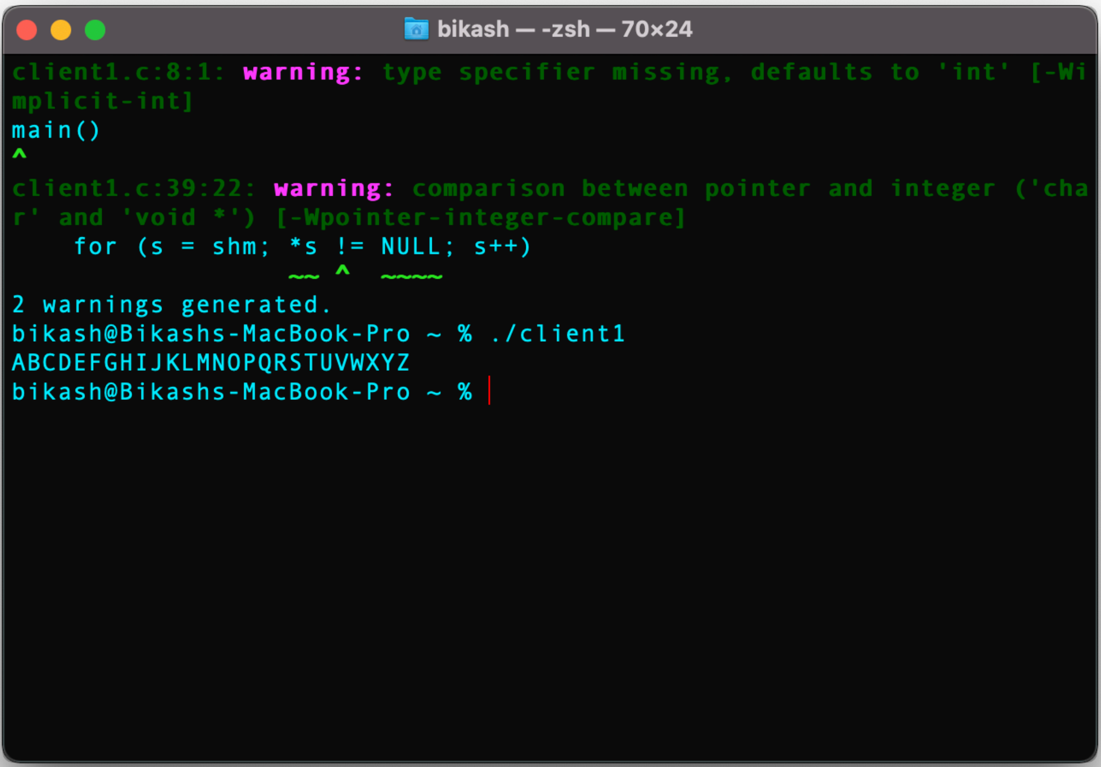
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#include<unistd.h>
#define SHMSZ 1024
main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s;
    /*
     * We'll name our shared memory segment
     * "5678".
    */
    key = 5678;
    /*
     * Create the segment.
     */
    if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) <
0) {
        perror("shmget");
        exit(1);
    }
    /*
     * Now we attach the segment to our data space.
     */
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }


    s = shm;
    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
    *s = NULL;
    /*
     * Finally, we wait until the other process
     * changes the first character of our memory
     * to '*', indicating that it has read what
     * we put there.
     */
    while (*shm != '*')
        sleep(1);
    exit(0);
}
```

Client-Side:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#define SHMSZ 1024

main()
{
    int shmid;
    key_t key;
    char *shm, *s;

    /*
     * We need to get the segment named
     * "5678", created by the server.
    */
    key = 5678;

    /*
     * Locate the segment.
     */
    if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    /*
```

```
     * Now we attach the segment to our data space.
     */
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    /*
     * Now read what the server put in the memory.
     */
    for (s = shm; *s != NULL; s++)
        putchar(*s-97+65);
    putchar('\n');

    /*
     * Finally, change the first character of the
     * segment to '*', indicating we have read
     * the segment.
     */
    *shm = '*';

    exit(0);
}
```

OUTPUT:



```
client1.c:8:1: warning: type specifier missing, defaults to 'int' [-Wi
mplicit-int]
main()
^
client1.c:39:22: warning: comparison between pointer and integer ('cha
r' and 'void *') [-Wpointer-integer-compare]
    for (s = shm; *s != NULL; s++)
                  ~~ ^  ~~~~
2 warnings generated.
bikash@Bikashs-MacBook-Pro ~ % ./client1
ABCDEFGHIJKLMNOPQRSTUVWXYZ
bikash@Bikashs-MacBook-Pro ~ %
```

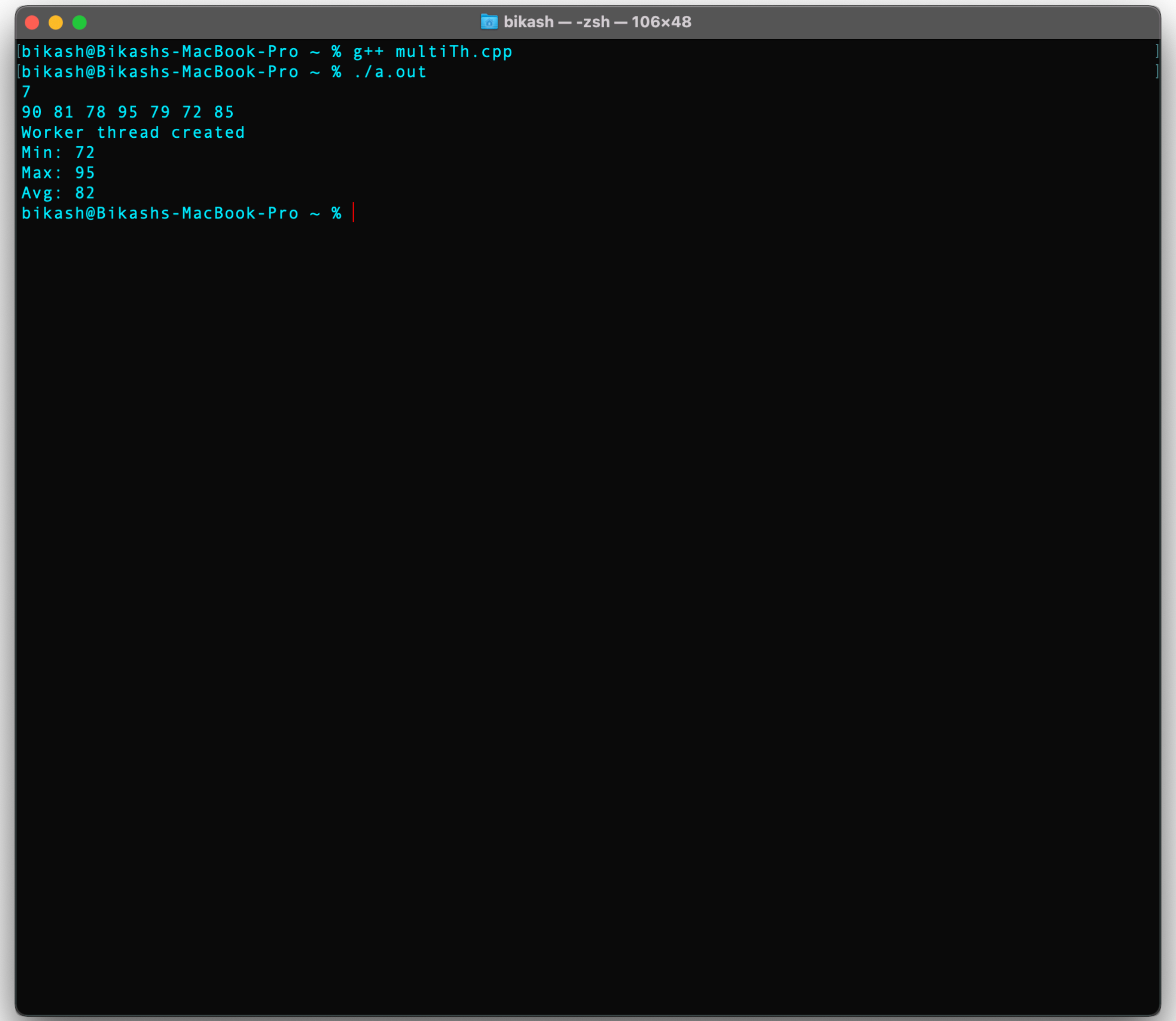
d) Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the 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

CODE:

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

using namespace std;

int n, mini, maxi, avg;
int *arr;

//thread 1 - find min

void *Fmin(void* arg){
    mini = arr[0];
    for(int i=1; i<n; i++){
       if(mini > arr[i]){
         mini = arr[i];
       }
    }
    return NULL;
}


//thread 2- find max

void *Fmax(void *arg){

    maxi = arr[0];
    for(int i=1; i<n; i++){
       if(maxi < arr[i]){
         maxi = arr[i];
       }
    }
    return NULL;
}
```

```
//thread 3- find average

void *Favg(void *arg){
     avg=0;
     for(int i=0; i<n; i++){
       avg += arr[i];
     }
     avg = avg/n;
     return NULL;
}


int main(){
      cin>>n;
      arr = new int[n];
      for(int i=0; i<n; i++){
        cin>>arr[i];
      }
      pthread_t ID1, ID2, ID3;
      pthread_create(&ID1, NULL, &Fmin, NULL);
      pthread_create(&ID2, NULL, &Fmax, NULL);
      pthread_create(&ID3, NULL, &Favg, NULL);

      cout<<"Worker thread created\n";
      pthread_join(ID1, NULL);
      pthread_join(ID2, NULL);
      pthread_join(ID3, NULL);

      cout<<"Min: "<<mini<<endl;
      cout<<"Max: "<<maxi<<endl;
      cout<<"Avg: "<<avg<<endl;
      exit(EXIT_SUCCESS);
      return 0;
}
```

OUTPUT:



```
bikash@Bikashs-MacBook-Pro ~ % g++ multiTh.cpp
bikash@Bikashs-MacBook-Pro ~ % ./a.out
7
90 81 78 95 79 72 85
Worker thread created
Min: 72
Max: 95
Avg: 82
bikash@Bikashs-MacBook-Pro ~ %
```

#CPU Scheduling

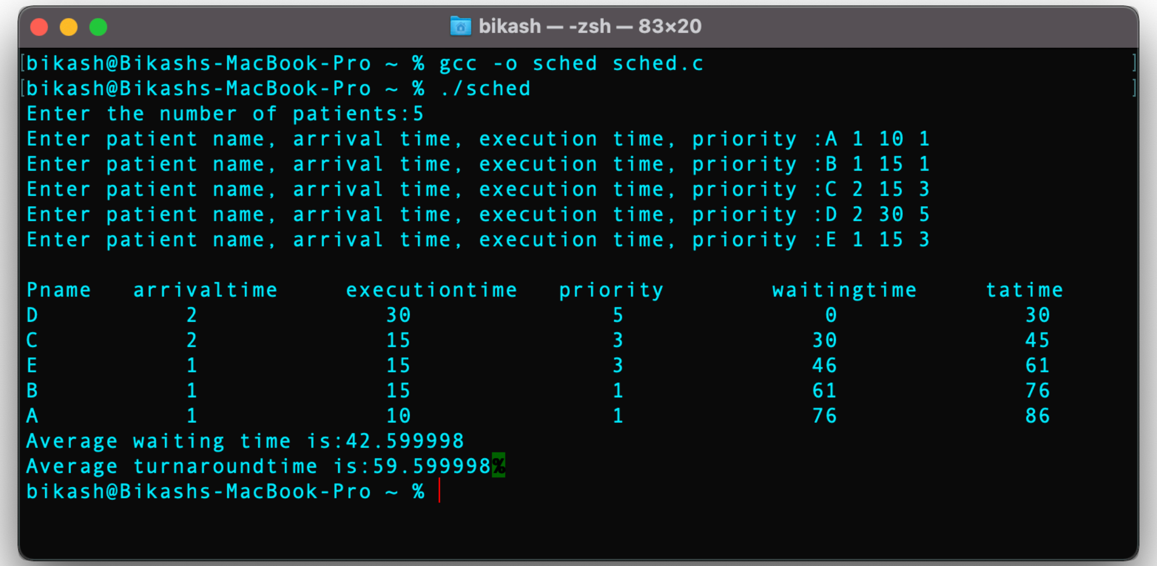
Q. (c) Consider a corporate hospital where we have n number of patients waiting for consultation. The amount of time required to serve a patient may vary, say 10 to 30 minutes. If a patient arrives with an emergency, he /she should be attended immediately before other patients, which may increase the 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)

CODE:

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

int main(){
  int
et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10
];
  char ch;
  int totwt=0,totta=0;
  float awt,ata;
  char pn[10][10],t[10];

  printf("Enter the number of patients:");
  scanf("%d",&n);
  for(i=0; i<n; i++){
    printf("Enter patient name, arrival time, execution
time, priority :");
    scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
    }
  for(i=0; i<n; i++)
   for(j=0; j<n; j++){
    if(p[i]>p[j]){
      temp=p[i];
      p[i]=p[j];
      p[j]=temp;
      temp=at[i];
      at[i]=at[j];
      at[j]=temp;
      temp=et[i];
      et[i]=et[j];
      et[j]=temp;
      strcpy(t,pn[i]);
      strcpy(pn[i],pn[j]);
      strcpy(pn[j],t);
     }}
   for(i=0; i<n; i++){
      if(i==0){
        st[i]=at[i];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];}
      else{
        st[i]=ft[i-1];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];
      }
    totwt+=wt[i];
    totta+=ta[i];
    }
    awt=(float)totwt/n;
    ata=(float)totta/n;

printf("\nPname\tarrivaltime\texecutiontime\tpriority\twa
itingtime\ttatime");

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

printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i
],et[i],p[i],wt[i],ta[i]);

    printf("\nAverage waiting time is:%f",awt);
    printf("\nAverage turnaroundtime is:%f”,ata);
    return 0;
}
```

OUTPUT:


bikash — -zsh — 83×20

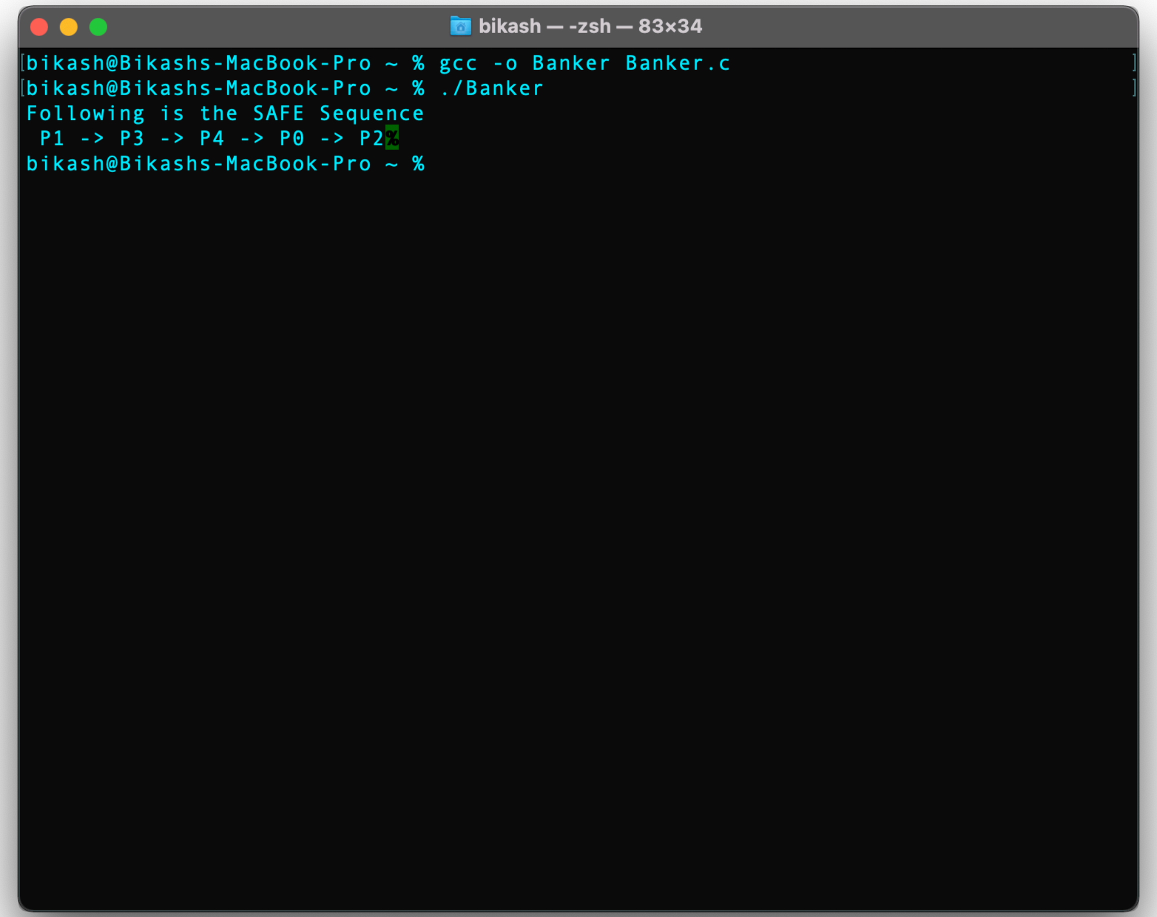
```
bikash@Bikashs-MacBook-Pro ~ % gcc -o sched sched.c
bikash@Bikashs-MacBook-Pro ~ % ./sched
Enter the number of patients:5
Enter patient name, arrival time, execution time, priority :A 1 10 1
Enter patient name, arrival time, execution time, priority :B 1 15 1
Enter patient name, arrival time, execution time, priority :C 2 15 3
Enter patient name, arrival time, execution time, priority :D 2 30 5
Enter patient name, arrival time, execution time, priority :E 1 15 3

Pname   arrivaltime     executiontime   priority        waitingtime     tatime
D           2              30              5                0              30
C           2              15              3               30              45
E           1              15              3               46              61
B           1              15              1               61              76
A           1              10              1               76              86
Average waiting time is:42.599998
Average turnaroundtime is:59.599998%
bikash@Bikashs-MacBook-Pro ~ %
```

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

CODE:
```
// Banker's Algorithm
#include <stdio.h>
int main()
{
        // P0, P1, P2, P3, P4 are the Process names here

        int n, m, i, j, k;
        n = 5; // Number of processes
        m = 3; // Number of resources
        int alloc[5][3] = { { 0, 1, 0 }, // P0 //
Allocation Matrix
                            { 2, 0, 0 }, // P1
                            { 3, 0, 2 }, // P2
                            { 2, 1, 1 }, // P3
                            { 0, 0, 2 }}; // P4

        int max[5][3] = { { 7, 5, 3 }, // P0 // MAX
Matrix
                          { 3, 2, 2 }, // P1
                          { 9, 0, 2 }, // P2
                          { 2, 2, 2 }, // P3
                          { 4, 3, 3 } }; // P4

        int avail[3] = { 3, 3, 2 }; // Available
Resources

        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:



```
bikash@Bikashs-MacBook-Pro ~ % gcc -o Banker Banker.c
bikash@Bikashs-MacBook-Pro ~ % ./Banker
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2%
bikash@Bikashs-MacBook-Pro ~ %
```