INDEX

Q1) Write a program (using fork() and/or exec() commands) where parent and child execute: a) same program, same code. b) same program, different code. - c) before terminating, the parent waits for the child to finish its task.

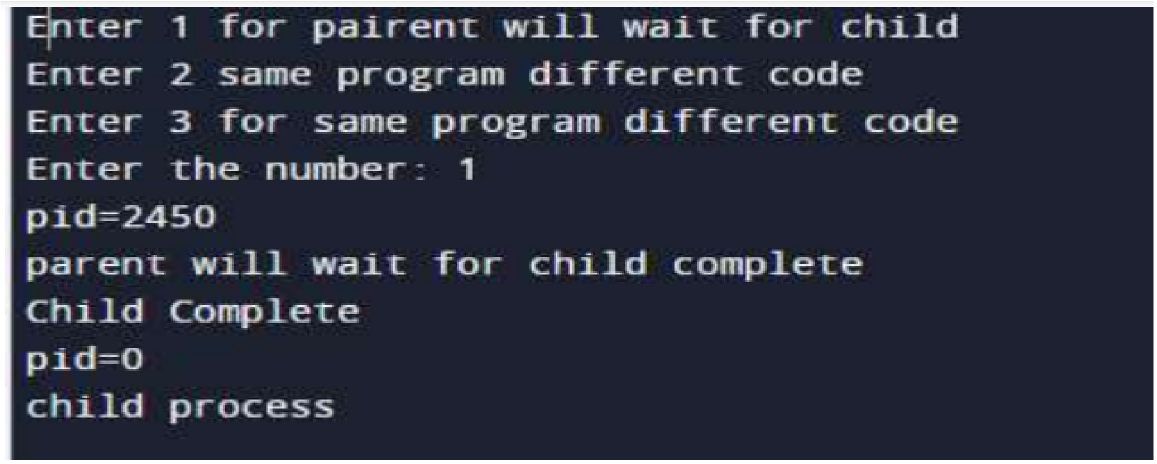
Code:

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include<cstring>
using namespace std;
void parentwait(){
   pid_t pid;
         /* fork a child process */
         pid = fork();
   if (pid < 0)
   {
                  /* error occurred */
                  cout<<"fork not called\n";
                  return;

   }
   else if (pid == 0)
   {
                  /* child process */
                  cout<<"pid"<<pid<<endl;
                  cout<<"child process\n";
   }
   else
   {
                  /* parent process */
                  /* parent will wait for the child to complete */

            cout<<"pid"<<pid<<endl;
      cout<<"parent will wait for child complete"<<endl;
      cout<<"Child Complete\n";
```

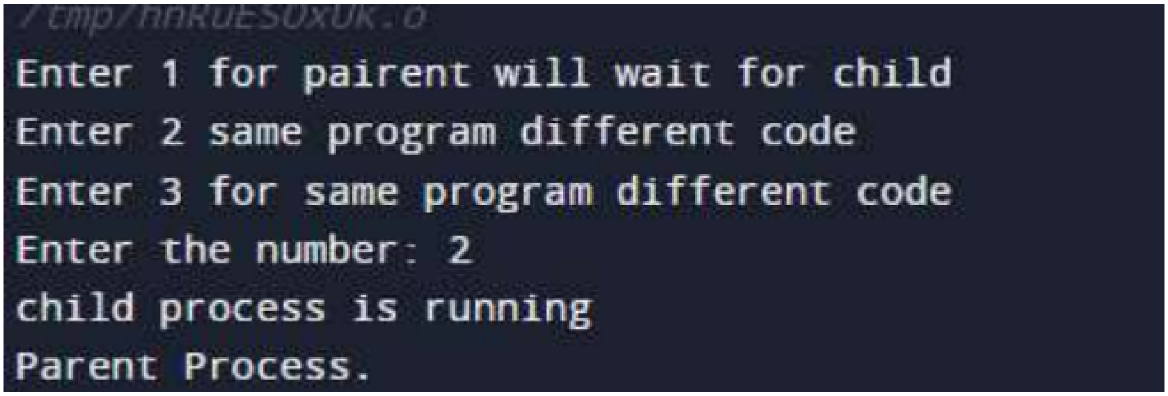
```
    }

}
void spdc(){
pid_t pid;
            /* fork a child process */
            pid = fork();
    if (pid < 0)
    {
                        /* error occurred */
                        cout<<"fork cammand was not called\n";
                        return;
    }
    else if (pid == 0)
    {
                        /* child process */
                        cout<<"child process is running\n";
    }
    else
    {
        cout<<"Parent Process.\n";
    }


}
void spsc(){
    pid_t pid,p;
    p=fork();
    pid=getpid();
            if(p < 0)
            {
        cout<<"Fork Failed";
        return;

            }
    cout<<"Output of Fork id:  "<<p<<endl;
    cout<<"process id is:"<<pid<<endl;;
}
```

```
int main(){
  cout<<"Enter 1 for pairent will wait for child\n"<<"Enter 2 same program different code\n"<<"Enter 3 for same program different code\n";
  int ch;

  cout <<"Enter the number: ";
  cin>>ch;
  if(ch==1){
     parentwait();
  }
  else if(ch==2){
     spdc();
  }
  else if(ch==3){
     spsc();
  }
  else{
     cout<<"invalid choice";
  }



   return 0;
}
```

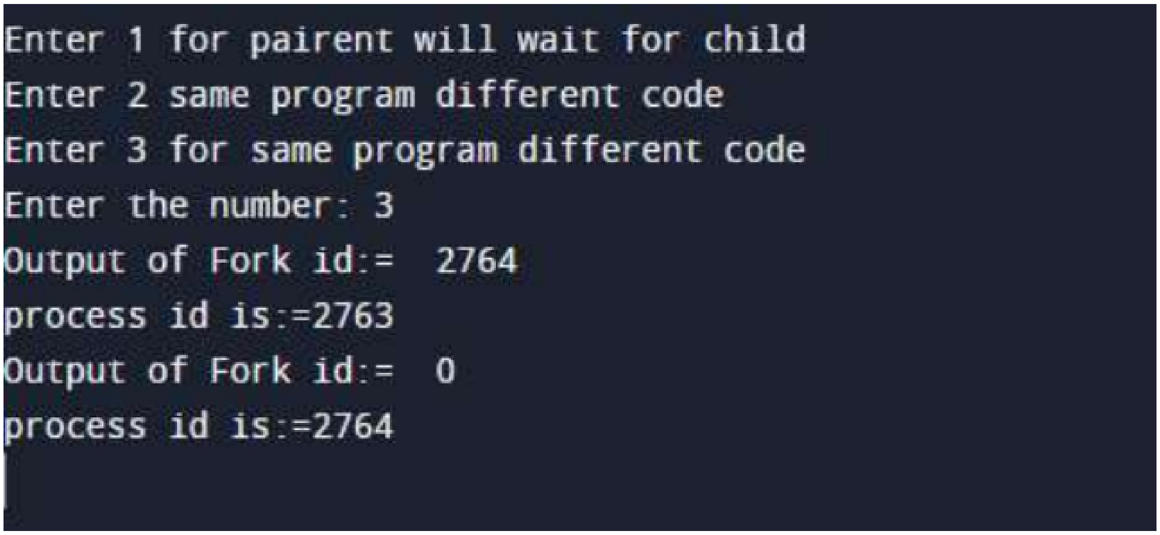
Output:



```
Enter 1 for pairent will wait for child
Enter 2 same program different code
Enter 3 for same program different code
Enter the number: 1
pid=2450
parent will wait for child complete
Child Complete
pid=0
child process
```



```
/tmp/hhRUESOxUK.o
Enter 1 for pairent will wait for child
Enter 2 same program different code
Enter 3 for same program different code
Enter the number: 2
child process is running
Parent Process.
```



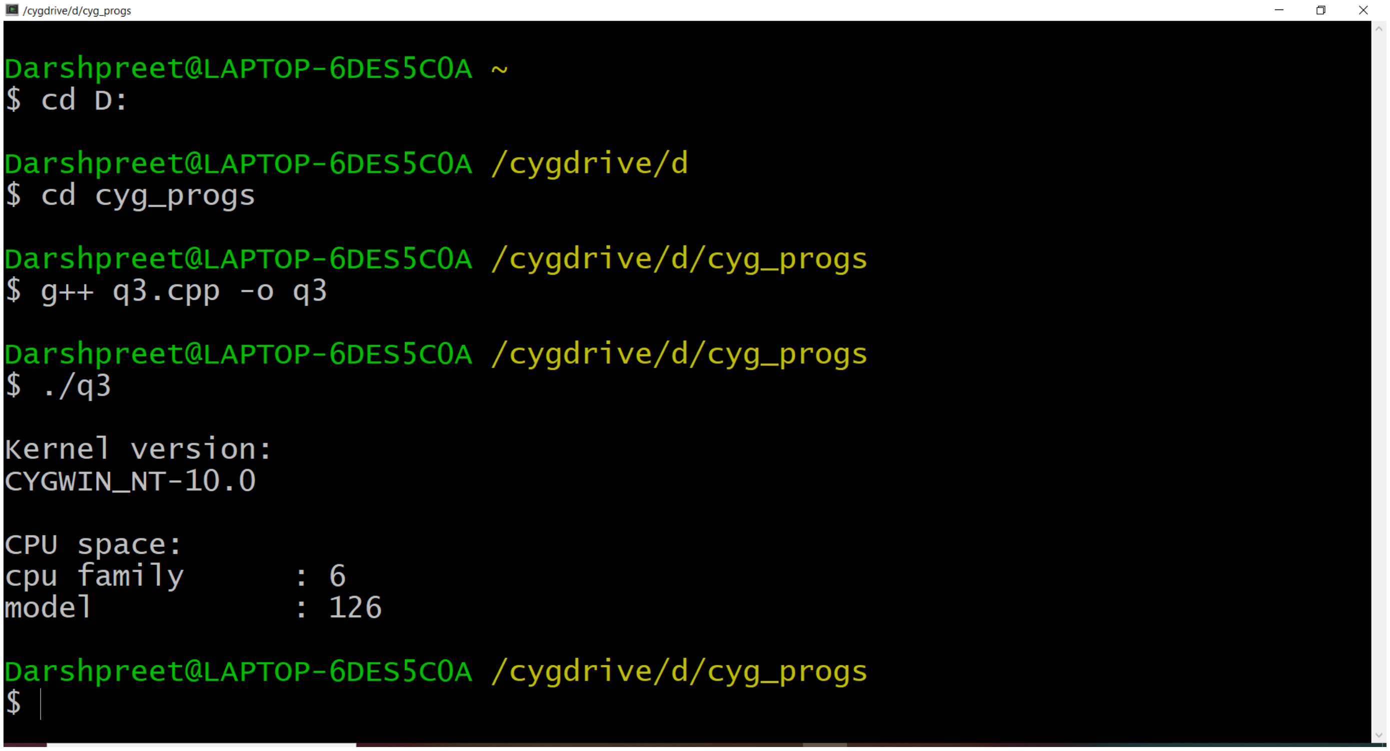
```
Enter 1 for pairent will wait for child
Enter 2 same program different code
Enter 3 for same program different code
Enter the number: 3
Output of Fork id:=  2764
process id is:=2763
Output of Fork id:=  0
process id is:=2764
```

Q2) Write a program to report behaviour of Linux kernel including kernel version, CPU type and model. (CPU information)

Code:

```
#include<iostream>
using namespace std;

int main()
{
	cout<<"\n Kernel version:\n";
	system("uname -s");
	cout<<"\nCPU space: \n";
	system("cat /proc/cpuinfo |awk 'NR==3,NR==4{print}' \n");
	return 0;
}
```

Output:



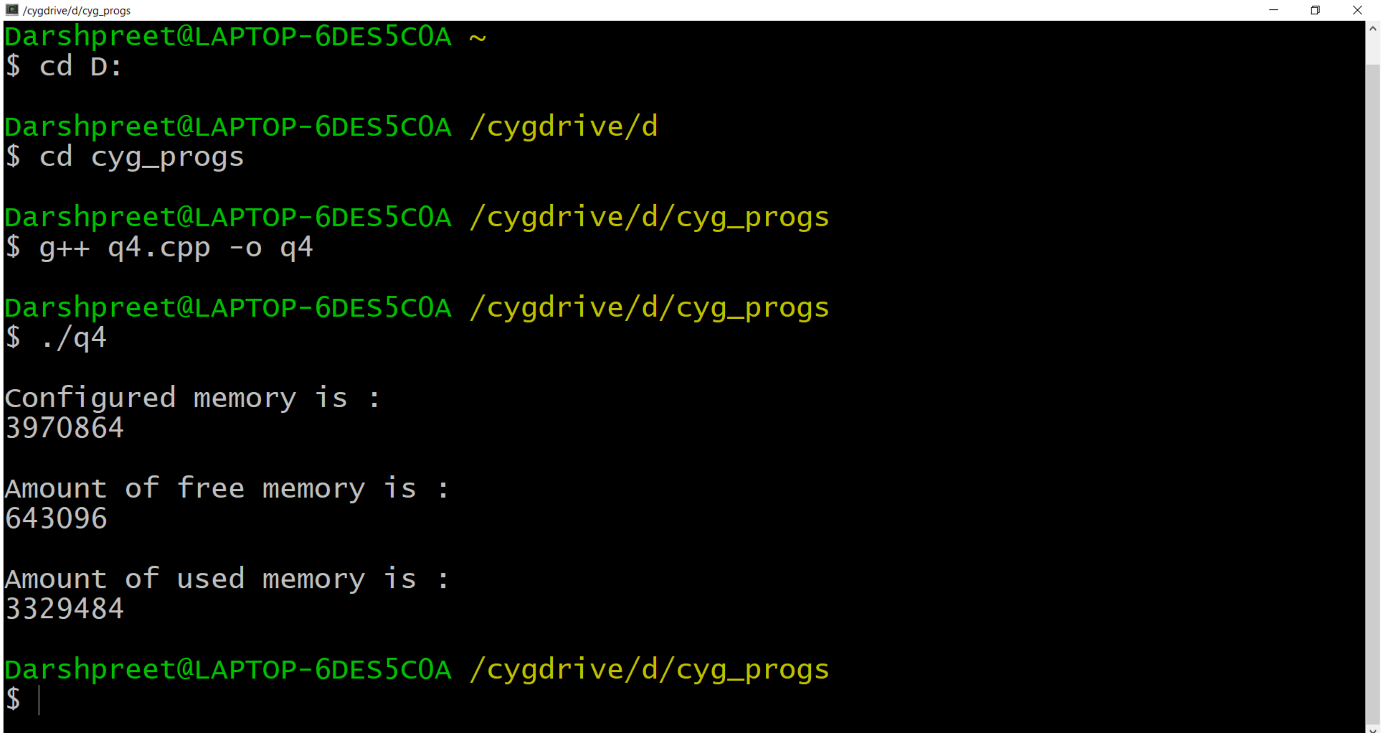
```
Darshpreet@LAPTOP-6DES5C0A ~
$ cd D:

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d
$ cd cyg_progs

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ g++ q3.cpp -o q3

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ ./q3

Kernel version:
CYGWIN_NT-10.0

CPU space:
cpu family      : 6
model           : 126

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ 
```

Q3) Write a program to report behaviour of Linux kernel including information on 19 configured memory, amount of free and used memory. (memory information)

Code:

```
#include<iostream>
using namespace std;
int main()
{
        cout<<"\nConfigured memory is :\n";
        system("cat /proc/meminfo |awk 'NR==1{print $2}'\n");
        cout<<"\nAmount of free memory is :\n";
        system("cat /proc/meminfo |awk 'NR==2{print $2}'\n");
        cout<<"\nAmount of used memory is :\n";
        system("cat /proc/meminfo |awk '{if (NR==1) a=$2; if (NR==2) b=$2 } END
      {print a-b}'\n");
        return 0;
}
```

Output:



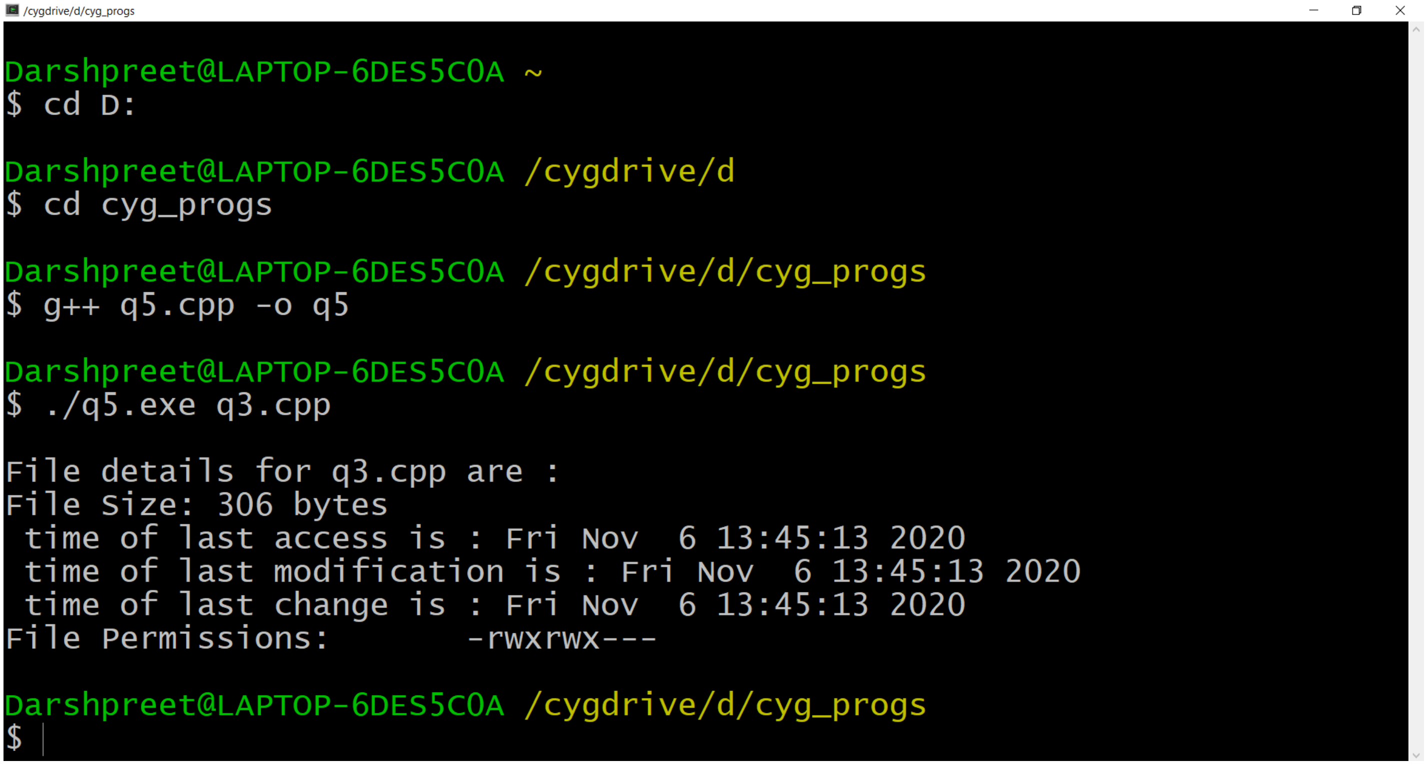
```
Darshpreet@LAPTOP-6DES5C0A ~
$ cd D:

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d
$ cd cyg_progs

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ g++ q4.cpp -o q4

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ ./q4

Configured memory is :
3970864

Amount of free memory is :
643096

Amount of used memory is :
3329484

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ 
```

Q4) Write a program to print file details including owner access permissions, file access time, where file name is given as argument .

Code:

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
using namespace std;
int main(int argc, char** argv)
 {
        if(argc !=2)
        {
           cout<<"\nEnter file name!\n";
           return 1;
        }
        struct stat fileStat;
        if(stat(argv[1],&fileStat)<0)
        return 1;
        cout<<"\nFile details for "<< argv[1]<<" are :\n";
        cout<<"File Size: "<<fileStat.st_size<<" bytes\n";
        cout<<" time of last access is : "<<ctime(&fileStat.st_atime);
        cout<<" time of last modification is : " << ctime(&fileStat.st_mtime);
        cout<<" time of last change is : "<< ctime(&fileStat.st_ctime);
        cout<<"File Permissions: \t";
        cout<<( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
        cout<<( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
        cout<<( (fileStat.st_mode & S_IWUSR) ? "w" : "-");
        cout<<( (fileStat.st_mode & S_IXUSR) ? "x" : "-");
        cout<<( (fileStat.st_mode & S_IRGRP) ? "r" : "-");
        cout<<( (fileStat.st_mode & S_IWGRP) ? "w" : "-");
        cout<<( (fileStat.st_mode & S_IXGRP) ? "x" : "-");
        cout<<( (fileStat.st_mode & S_IROTH) ? "r" : "-");
        cout<<( (fileStat.st_mode & S_IWOTH) ? "w" : "-");
        cout<<( (fileStat.st_mode & S_IXOTH) ? "x" : "-");
        cout<<endl;
        return 0;
}
```

Output:



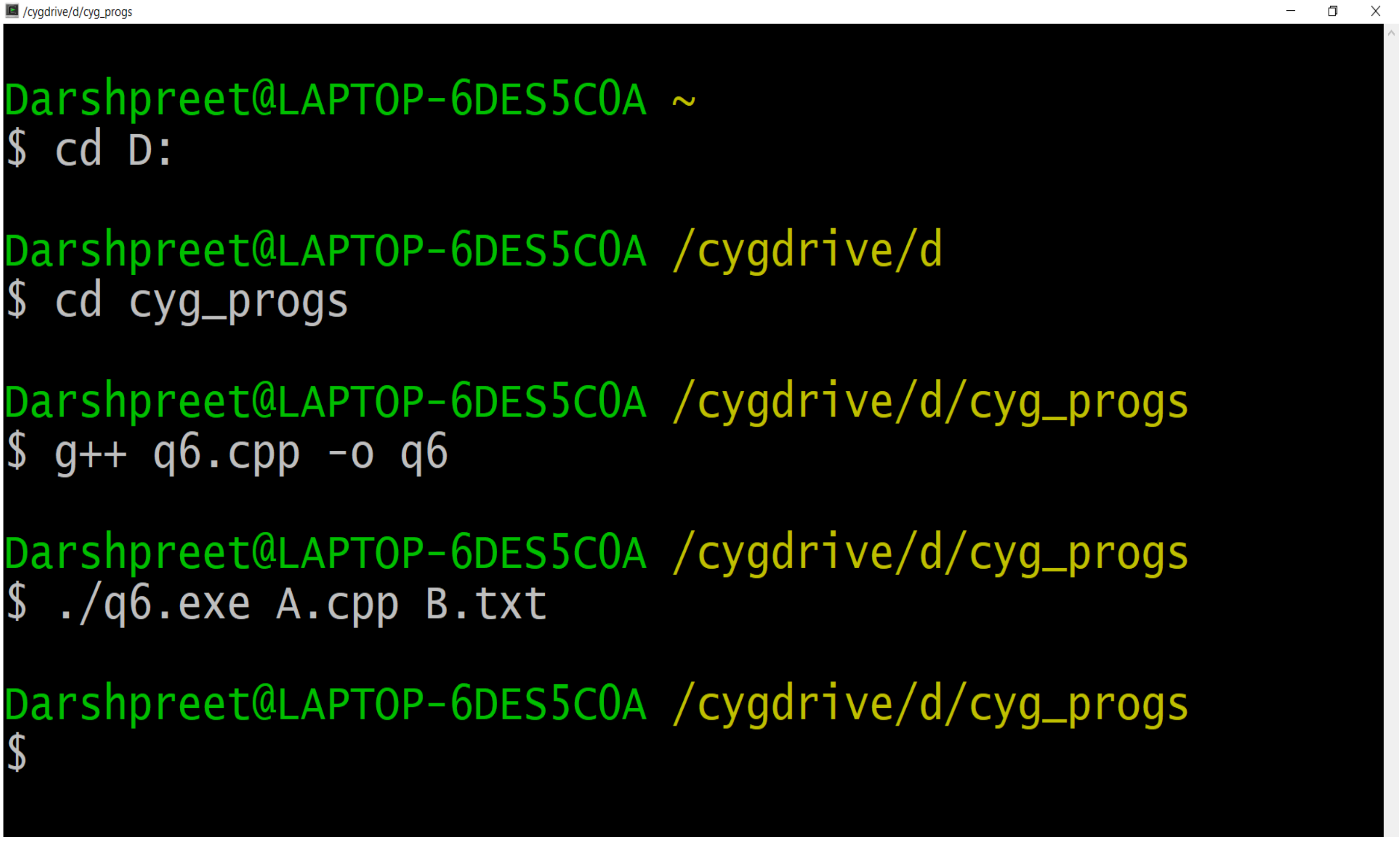
```
Darshpreet@LAPTOP-6DES5C0A ~
$ cd D:

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d
$ cd cyg_progs

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ g++ q5.cpp -o q5

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ ./q5.exe q3.cpp

File details for q3.cpp are :
File Size: 306 bytes
 time of last access is : Fri Nov  6 13:45:13 2020
 time of last modification is : Fri Nov  6 13:45:13 2020
 time of last change is : Fri Nov  6 13:45:13 2020
File Permissions:       -rwxrwx---

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$
```

Q5) Write a program to copy files using system calls.

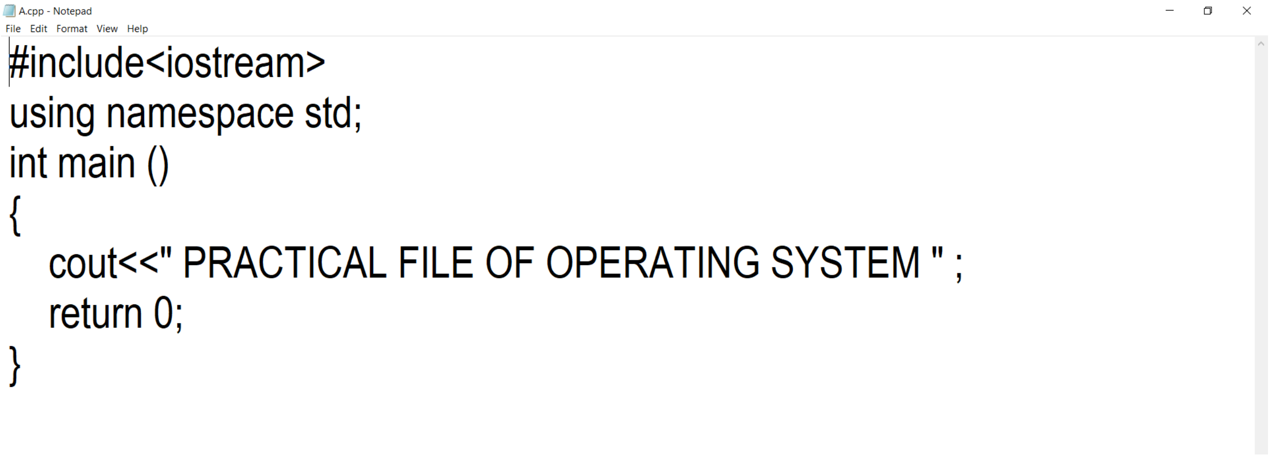
Code:

```
#include <iostream>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include<unistd.h>
#include<sys/types.h>
#define BUFF_SIZE 1024
using namespace std;
int main(int argc, char* argv[])
{
  int srcFD, destFD, nbread, nbwrite ;
 char *buff[BUFF_SIZE];

 if(argc != 3 || argv[1] == "--help")
 {
   cout<<"\nUsage: cpcmd source_file destination_file\n";
   exit(EXIT_FAILURE);
  }

  srcFD = open(argv[1],O_RDONLY);
  if(srcFD == -1)
  {
    cout<<"\nError opening file "<<argv[1]<<" errno = \n"<<errno;
    exit(EXIT_FAILURE);
  }
  destFD = open(argv[2],O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR |S_IRGRP | S_IWGRP |
S_IROTH | S_IWOTH);
  if(destFD == -1)
 {
       cout<<"\nError opening file "<<argv[2]<<" errno = \n"<<errno;
       exit(EXIT_FAILURE);
  }
  while((nbread = read(srcFD,buff,BUFF_SIZE)) > 0)
 {
    if(write(destFD,buff,nbread) != nbread)
    cout<<"\nError in writing data to \n"<<argv[2];
  }
     if(nbread == -1)
     cout<<"\nError in reading data from \n"<<argv[1];
     if(close(srcFD) == -1)
     cout<<"\nError in closing file \n"<<argv[1];
     if(close(destFD) == -1)
      cout<<"\nError in closing file \n"<<argv[2];
      exit(EXIT_SUCCESS);
}
```

Output:



```
Darshpreet@LAPTOP-6DES5C0A ~
$ cd D:

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d
$ cd cyg_progs

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ g++ q6.cpp -o q6

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$ ./q6.exe A.cpp B.txt

Darshpreet@LAPTOP-6DES5C0A /cygdrive/d/cyg_progs
$
```



```
#include<iostream>
using namespace std;
int main ()
{
   cout<<" PRACTICAL FILE OF OPERATING SYSTEM " ;
   return 0;
}
```

Q6) Write a program to implement FCFS scheduling algorithm.

Code:

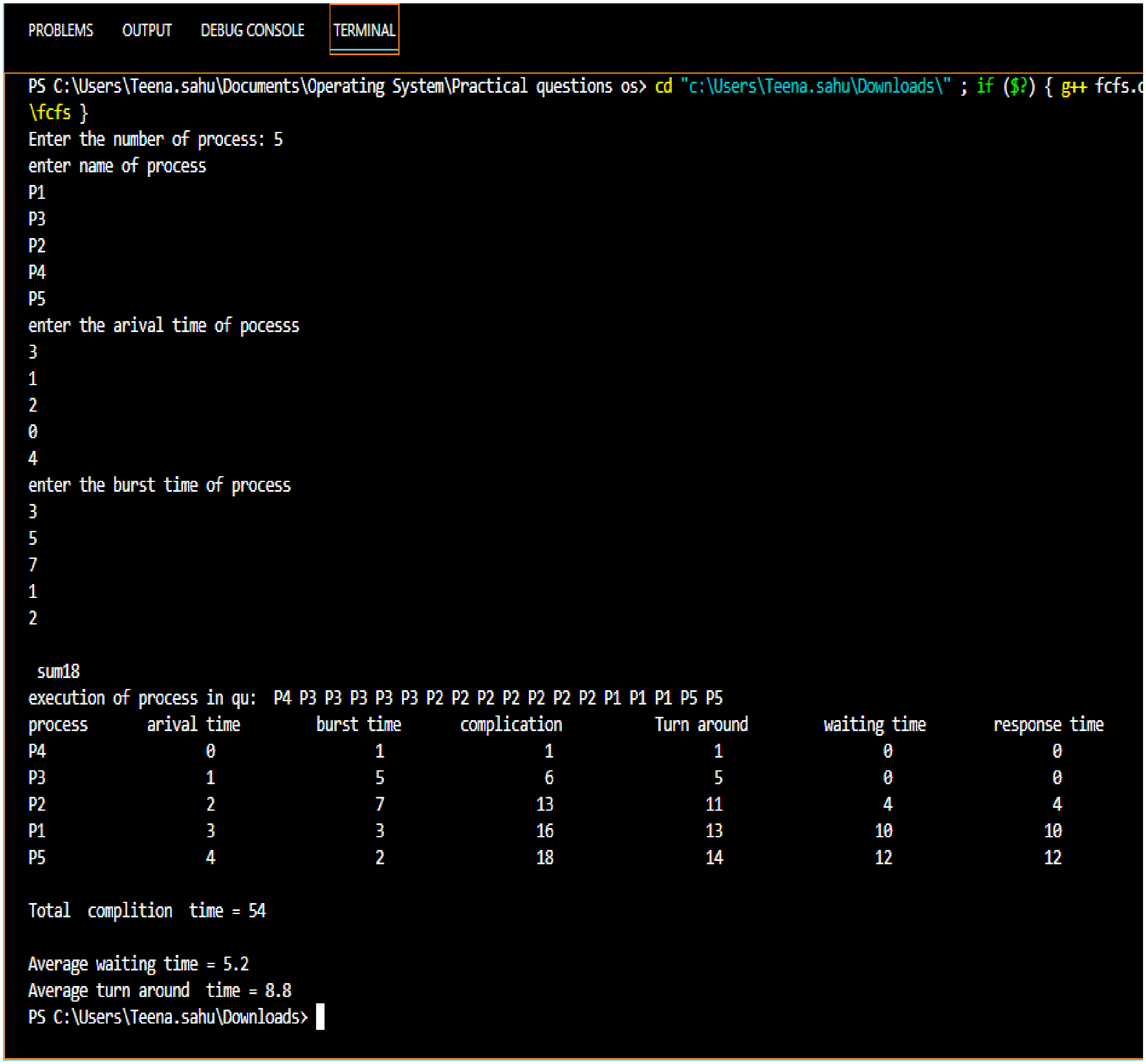
```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cstring>
using namespace std;
int execute1(string s, string exe[], int index, int bt)
{
    for (int i = index; i < bt; i++)
    {
        exe[i] = s;
    }

    return bt;
}
int main()
{
    int size;
    cout << "Enter the number of process: ";
    cin >> size;
    string process[size];
    cout << "enter name of process \n";
    for (int i = 0; i < size; i++)
    {
        cin >> process[i];
    }
    cout << "enter the arival time of pocesss \n";
```

```
  int time[size];
  for (int i = 0; i < size; i++)
  {
    cin >> time[i];
  }
  int bt[size];
  cout << "enter the burst time of process \n";

  for (int i = 0; i < size; i++)
  {
    cin >> bt[i];
  }
    int temptime;
  string tempstr;
 for(int i=0;i<size;i++){
  for(int j=i+1;j<size;j++){
    if(time[j]<time[i]){
      temptime=time[i];
      time[i]=time[j];
      time[j]=temptime;
      temptime=bt[i];
      bt[i]=bt[j];
      bt[j]=temptime;
      tempstr=process[i];
      process[i]=process[j];
      process[j]=tempstr;
    }
  }
```

```
}




 int sum = 0;
 for (int i = 0; i < size; i++)
 {
     sum = sum + bt[i];
 }
 int temparival[size];
 for (int i = 0; i < size; i++)
 {
     temparival[i] = time[i];
 }
 int bt2 = 0;
 string exe[sum];
 int temp = 0, p ;
 int j = 0;
 temp = temparival[0];

 int index = 0;
 while (j < size)
 {
     temp = temparival[0];
     p=0;
     for (int i = 0; i < size; i++)
     {
        if (temp > temparival[i])
        {
            temp = temparival[i];
            p = i;
        }
    }
    temparival[p] = 100;
    bt2 = bt2 + bt[p];
    // cout<<"\nproces "<<process[p]<<endl;

    int b = execute1(process[p], exe, index, bt2);
    // cout << "index" << index << endl;
    index = b;
    j++;
}
cout<<"\n sum"<<sum<<endl;
cout << "execution of process in qu:  ";
for (int i = 0; i < sum; i++)
{
    cout << exe[i] << " ";
}
int complition[size];
int Tat[size];
int waiting[size];
int rt[size];
for(int i=0;i<size;i++){
    for(int j=sum-1;i>=0;j--){
        if(process[i]==exe[j]){
                complition[i]=j+1;
                break;
            }
        }
    }
    for(int i=0;i<size;i++){
        Tat[i]=(complition[i]-time[i]);
        waiting[i]=Tat[i]-bt[i];
    }
    for(int i=0;i<size;i++){
        for(int j=0;j<sum;j++)
        {
            if(process[i]==exe[j]){
                rt[i]=j-time[i];
                break;
            }
        }
    }
        cout << "\nprocess       "
         << "arival time       "  << "  burst time       "<<"complication     " <<"      Turn around      "<<"   waiting
time        "<<"response time"<< endl;
    for (int i = 0; i < size; i++)
    {



        cout << process[i] << setw(20) << time[i] << setw(20) << bt[i] <<setw(20) << complition[i]<<setw(20) <<
Tat[i]<<setw(20) << waiting[i]<<setw(20) << rt[i]<< endl;
    }
    float sum1=0,sum2=0,sum3=0;
```

```
for (int i=0;i<size;i++){
   sum1=sum1+waiting[i];
   sum2=sum2+Tat[i];
   sum3=sum3+complition[i];
}
float avg=sum1/size;
float avgtat=sum2/size;
     cout<<"\nTotal  complition  time = "<<sum3<<endl;
   cout<<"\nAverage waiting time = "<<avg<<endl;
     cout<<"Average turn around  time = "<<avgtat<<endl;

   return 0;
}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Operating System\Practical questions os> cd "c:\Users\Teena.sahu\Downloads\" ; if ($?) { g++ fcfs.c
\fcfs }
Enter the number of process: 5
enter name of process
P1
P3
P2
P4
P5
enter the arival time of pocesss
3
1
2
0
4
enter the burst time of process
3
5
7
1
2

 sum18
execution of process in qu:  P4 P3 P3 P3 P3 P3 P2 P2 P2 P2 P2 P2 P2 P1 P1 P1 P5 P5
process       arival time         burst time       complication          Turn around          waiting time         response time
P4                 0                   1                  1                    1                    0                    0
P3                 1                   5                  6                    5                    0                    0
P2                 2                   7                 13                   11                    4                    4
P1                 3                   3                 16                   13                   10                   10
P5                 4                   2                 18                   14                   12                   12

Total  complition  time = 54

Average waiting time = 5.2
Average turn around  time = 8.8
PS C:\Users\Teena.sahu\Downloads> 
```

Q7) Write a program to implement Round Robin scheduling algorithm.

Code:

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstring>
#include <vector>
using namespace std;
int execute1(string s, string exe[], int index, int bt, int *arival)
{
    for (int i = index; i < bt; i++)
    {
        exe[i] = s;
    }

    *arival = bt;
    return bt;
}
int main()
{

    vector<int> myvec;
    vector<int>::iterator it;
    it = myvec.begin();
    int size;
    cout << "Enter the number of process: ";
```

```
cin >> size;
string process[size];
cout << "enter name of process \n";
for (int i = 0; i < size; i++)
{
   cin >> process[i];
}
cout << "enter the arival time of pocesss \n";
int time[size];
for (int i = 0; i < size; i++)
{
   cin >> time[i];
}
int bt[size];
cout << "enter the burst time of process \n";

for (int i = 0; i < size; i++)
{
   cin >> bt[i];
}
int timeqt;
cout << "\n Enter the time quntum of algorithm : ";
cin >> timeqt;
int temptime;
string tempstr;
for (int i = 0; i < size; i++)
{
   for (int j = i + 1; j < size; j++)
```

```
    {
        if (time[j] < time[i])
        {
            temptime = time[i];
            time[i] = time[j];
            time[j] = temptime;
            temptime = bt[i];
            bt[i] = bt[j];
            bt[j] = temptime;
            tempstr = process[i];
            process[i] = process[j];
            process[j] = tempstr;
        }
    }
}

int sum = 0, g, b = 0;
for (int i = 0; i < size; i++)
{
    sum = sum + bt[i];
}
int temparival[size];
int tempbt[size];
for (int i = 0; i < size; i++)
{
    temparival[i] = time[i];
    tempbt[i] = bt[i];
}
```

```
int bt2 = 0;
string exe[sum];
int arival = 0;
int temp = 0, p;
int j = 0;
temp = temparival[0];
int a = 0;
int index = 0;
while (j != size)
{
    j = 0;

    if (a == 0)
    {
        a = 1;
        temp = temparival[0];
        p = 0;
        for (int i = 0; i < size; i++)
        {
            if (temp > temparival[i])
            {
                temp = temparival[i];
                p = i;
            }
        }

        if (timeqt > tempbt[p])
        {
```

```
    bt2 = bt2 + tempbt[p];
    tempbt[p] = 0;

    b = execute1(process[p], exe, index, bt2, &arival);
    index = b;
}
else
{
    bt2 = bt2 + timeqt;

    tempbt[p] = tempbt[p] - timeqt;
    b = execute1(process[p], exe, index, bt2, &arival);
    index = b;
}
for (int k = 0; k < size; k++)
{
    if (temparival[k] <= arival && tempbt[k] == bt[k])
    {
        int t;
        t = count(myvec.begin(), myvec.end(), k);
        if (t == 0)
        {
            myvec.push_back(k);
        }
    }
}
if (tempbt[p] != 0)
```

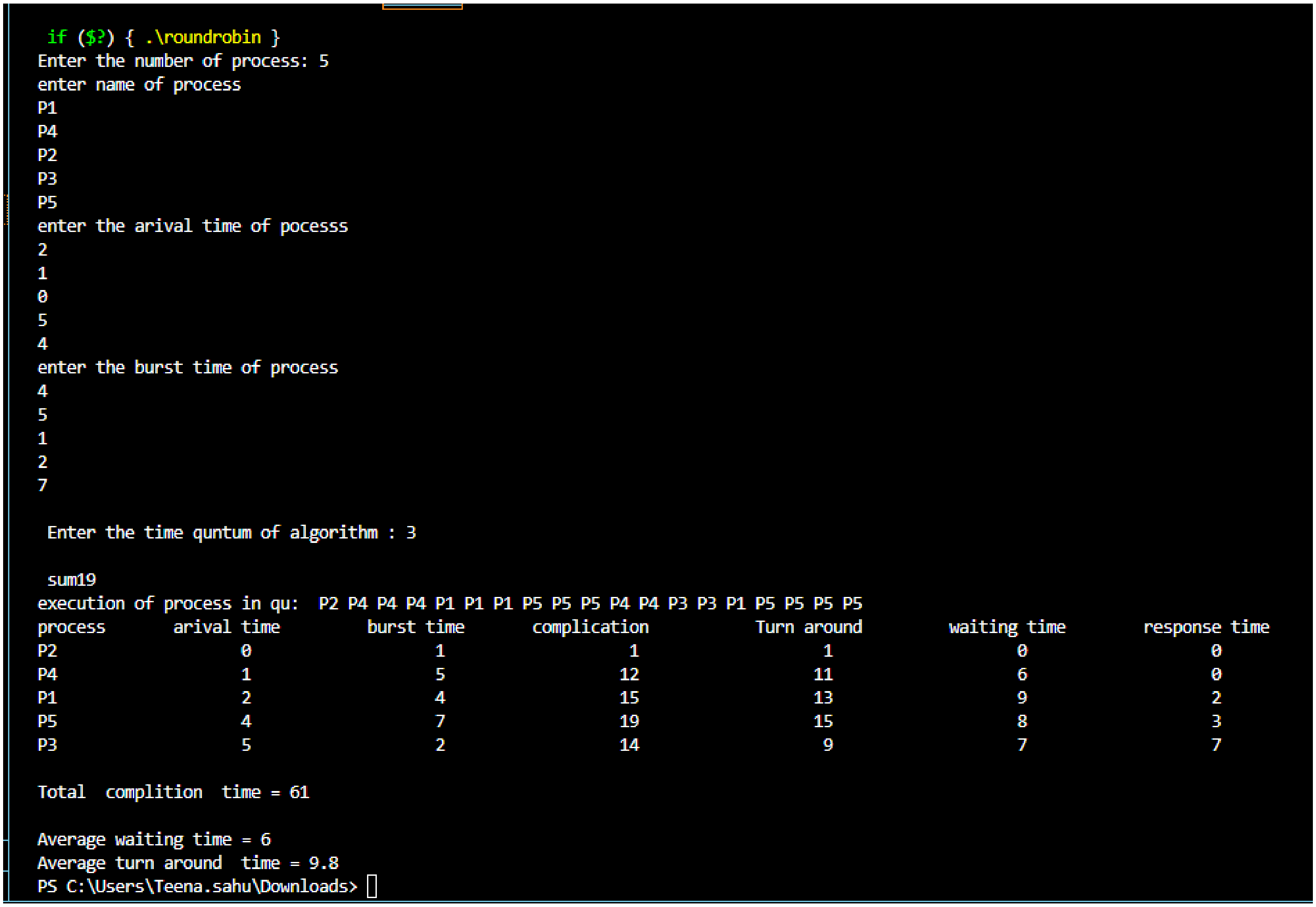
```
    {
        myvec.push_back(p);
    }
}
else
{

    g = myvec.front();

    it = myvec.begin();
    myvec.erase(it);

    if (timeqt > tempbt[g])
    {
        bt2 = bt2 + tempbt[g];
        tempbt[g] = 0;

        b = execute1(process[g], exe, index, bt2, &arival);
        index = b;
    }
    else
    {
        bt2 = bt2 + timeqt;
        tempbt[g] = tempbt[g] - timeqt;

        b = execute1(process[g], exe, index, bt2, &arival);
        index = b;
    }
```

```
        for (int k = 0; k < size; k++)
        {
            if (temparival[k] <= arival && tempbt[k] == bt[k])
            {
                int t;
                t = count(myvec.begin(), myvec.end(), k);
                if (t == 0)
                {
                    myvec.push_back(k);
                }
            }
        }
        if (tempbt[g] != 0)
        {
            myvec.push_back(g);
        }
    }
    for (int i = 0; i < size; i++)
    {
        if (tempbt[i] == 0)
        {
            j++;
        }
    }
}
cout << "\n sum" << sum << endl;
cout << "execution of process in qu:  ";
for (int i = 0; i < sum; i++)
```

```
{
    cout << exe[i] << " ";
}
int complition[size];
int Tat[size];
int waiting[size];
int rt[size];
for (int i = 0; i < size; i++)
{
    for (int j = sum - 1; i >= 0; j--)
    {
        if (process[i] == exe[j])
        {
            complition[i] = j + 1;
            break;
        }
    }
}
for (int i = 0; i < size; i++)
{
    Tat[i] = (complition[i] - time[i]);
    waiting[i] = Tat[i] - bt[i];
}
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < sum; j++)
    {
        if (process[i] == exe[j])
```

```
            {
                rt[i] = j - time[i];
                break;
            }
        }
    }
    cout << "\nprocess       "
         << "arival time       "
         << "  burst time       "
         << "complication     "
         << "      Turn around      "
         << "   waiting time        "
         << "response time" << endl;
    for (int i = 0; i < size; i++)
    {

        cout << process[i] << setw(20) << time[i] << setw(20) << bt[i] << setw(20) << complition[i] << setw(20)
<< Tat[i] << setw(20) << waiting[i] << setw(20) << rt[i] << endl;
    }
    float sum1 = 0, sum2 = 0, sum3 = 0;

    for (int i = 0; i < size; i++)
    {
        sum1 = sum1 + waiting[i];
        sum2 = sum2 + Tat[i];
        sum3 = sum3 + complition[i];
    }
    float avg = sum1 / size;
    float avgtat = sum2 / size;
    cout << "\nTotal  complition  time = " << sum3 << endl;
    cout << "\nAverage waiting time = " << avg << endl;
    cout << "Average turn around  time = " << avgtat << endl;

    return 0;
}
```

Output:



```
 if ($?) { .\roundrobin }
Enter the number of process: 5
enter name of process
P1
P4
P2
P3
P5
enter the arival time of pocesss
2
1
0
5
4
enter the burst time of process
4
5
1
2
7

 Enter the time quntum of algorithm : 3

 sum19
execution of process in qu:  P2 P4 P4 P4 P1 P1 P1 P5 P5 P5 P4 P4 P3 P3 P1 P5 P5 P5 P5
process       arival time         burst time       complication            Turn around          waiting time          response time
P2                  0                   1                   1                   1                   0                   0
P4                  1                   5                  12                  11                   6                   0
P1                  2                   4                  15                  13                   9                   2
P5                  4                   7                  19                  15                   8                   3
P3                  5                   2                  14                   9                   7                   7

Total  complition  time = 61

Average waiting time = 6
Average turn around  time = 9.8
PS C:\Users\Teena.sahu\Downloads> 
```

Q8) Write a program to implement SJF scheduling algorithm.

Code:

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cstring>
using namespace std;
int execute1(string s, string exe[], int index, int bt, int *arival  )
{
   for (int i = index; i < bt; i++)
   {
      exe[i] = s;
   }
*arival=bt;
   return bt;
}
int main()
{
   int size;
   cout << "Enter the number of process: ";
   cin >> size;
   string process[size];
   cout << "enter name of process \n";
   for (int i = 0; i < size; i++)
   {
      cin >> process[i];
   }
```

```
  cout << "enter the arival time of pocesss \n";
  int time[size];
  for (int i = 0; i < size; i++)
  {
     cin >> time[i];
  }
  int bt[size];
  cout << "enter the burst time of process \n";

  for (int i = 0; i < size; i++)
  {
     cin >> bt[i];
  }

  int temptime;
  string tempstr;
for(int i=0;i<size;i++){
  for(int j=i+1;j<size;j++){
     if(time[j]<time[i]){
        temptime=time[i];
        time[i]=time[j];
        time[j]=temptime;
        temptime=bt[i];
        bt[i]=bt[j];
        bt[j]=temptime;
        tempstr=process[i];
        process[i]=process[j];
        process[j]=tempstr;
```

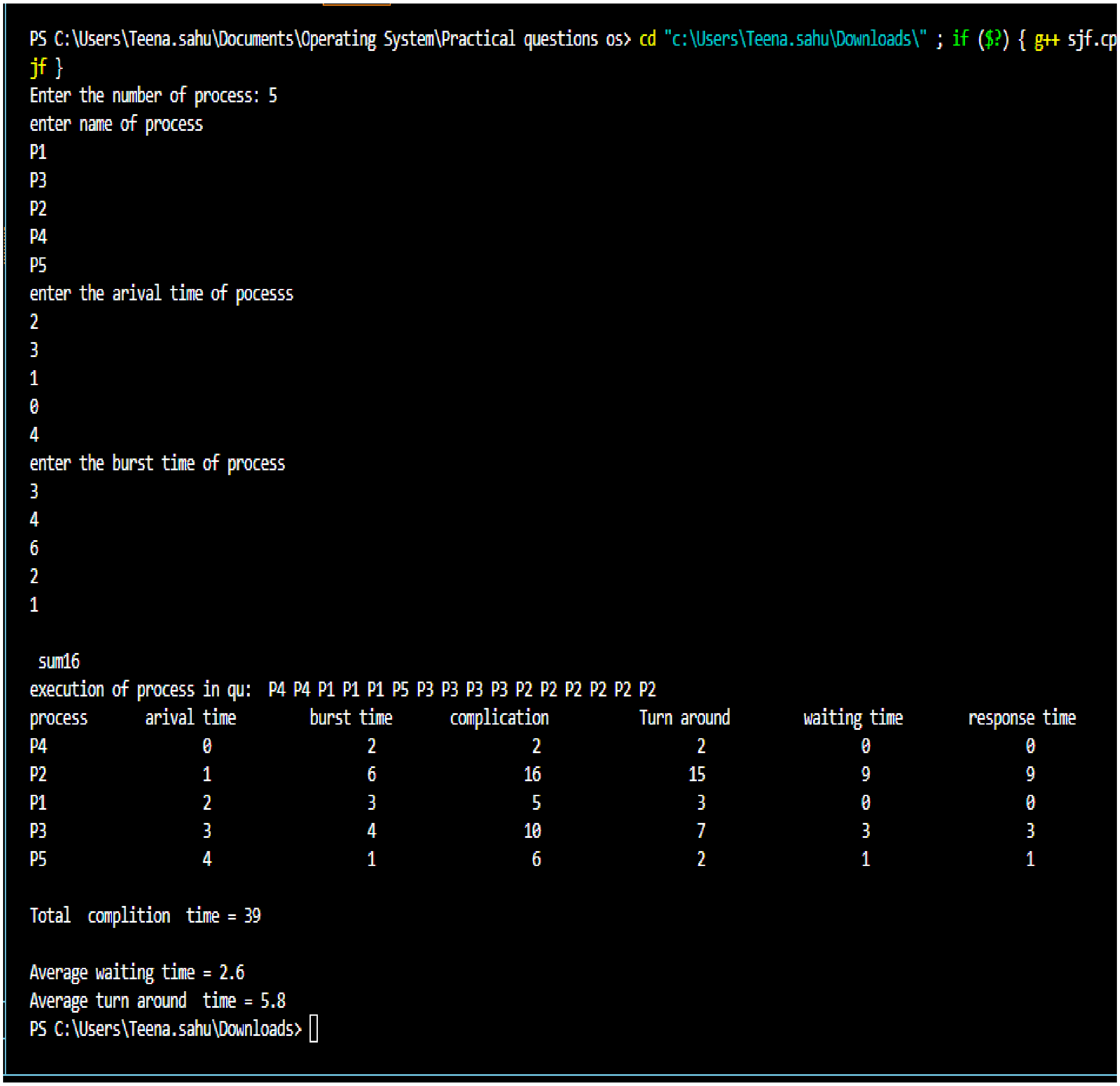
```
    }
  }
}



  int sum = 0,b;
  for (int i = 0; i < size; i++)
  {
    sum = sum + bt[i];
  }
  int temparival[size];
  int tempbt[size];
  for (int i = 0; i < size; i++)
  {
    temparival[i] = time[i];
    tempbt[i]=bt[i];
  }
  int bt2 = 0,c=0;
  string exe[sum];
  int temp = 0, p ;
  int j = 0;
  temp = temparival[0];

  int index = 0;
  int a=0;
  int arival=0;
  while (j < size)
  {  if(a==0){
```

```
temp = temparival[0];

p=0;

for (int i = 0; i < size; i++)

{

   if (temp > temparival[i])

   {

      temp = temparival[i];

      p = i;

   }

}

temparival[p] = 100;

bt2 = bt2 + bt[p];

// cout<<"\nproces "<<process[p]<<endl;


 b = execute1(process[p], exe, index, bt2,&arival);

tempbt[p]=0;

// cout << "index" << index << endl;

index = b;

a=1;

}

else {int tempbt1;

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


     if(  temparival[i]<=arival &&  temparival[i]!=100){

      tempbt1=tempbt[i];

      c=i;

         for(int k=0;k<size;k++){

            if(temparival[k]<=arival && tempbt1>tempbt[k]){
```

```
            tempbt1=tempbt[k];
            c=k;

          }
        }

     }
        if(tempbt[c]!=0){
           bt2 = bt2 + bt[c];
          b = execute1(process[c], exe, index, bt2,&arival);
          index=b;
           tempbt[c]=0;
        temparival[c]=100;
        }
    }

  }
  j++;
}
cout<<"\n sum"<<sum<<endl;
cout << "execution of process in qu:  ";
for (int i = 0; i < sum; i++)
{
  cout << exe[i] << " ";
}
int complition[size];
int Tat[size];
int waiting[size];
```

```
    int rt[size];
    for(int i=0;i<size;i++){
        for(int j=sum-1;i>=0;j--){
            if(process[i]==exe[j]){
                complition[i]=j+1;
                break;
            }
        }
    }
    for(int i=0;i<size;i++){
        Tat[i]=(complition[i]-time[i]);
        waiting[i]=Tat[i]-bt[i];
    }
    for(int i=0;i<size;i++){
        for(int j=0;j<sum;j++)
        {
            if(process[i]==exe[j]){
                rt[i]=j-time[i];
                break;
            }
        }
    }
        cout << "\nprocess       "
         << "arival time       "  << "  burst time       "<<"complication     " <<"      Turn around      "<<"   waiting
time        "<<"response time"<< endl;
    for (int i = 0; i < size; i++)
    {
        cout << process[i] << setw(20) << time[i] << setw(20) << bt[i] <<setw(20) << complition[i]<<setw(20) <<
Tat[i]<<setw(20) << waiting[i]<<setw(20) << rt[i]<< endl;

    }

    float sum1=0,sum2=0,sum3=0;


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

    sum1=sum1+waiting[i];

    sum2=sum2+Tat[i];

    sum3=sum3+complition[i];

}

float avg=sum1/size;

float avgtat=sum2/size;

      cout<<"\nTotal  complition  time = "<<sum3<<endl;

    cout<<"\nAverage waiting time = "<<avg<<endl;

      cout<<"Average turn around  time = "<<avgtat<<endl;


    return 0;

}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Operating System\Practical questions os> cd "c:\Users\Teena.sahu\Downloads\" ; if ($?) { g++ sjf.cp
jf }
Enter the number of process: 5
enter name of process
P1
P3
P2
P4
P5
enter the arival time of pocesss
2
3
1
0
4
enter the burst time of process
3
4
6
2
1

 sum16
execution of process in qu:  P4 P4 P1 P1 P1 P5 P3 P3 P3 P3 P2 P2 P2 P2 P2 P2
process       arival time        burst time      complication          Turn around         waiting time        response time
P4                  0                   2                2                     2                   0                    0
P2                  1                   6               16                    15                   9                    9
P1                  2                   3                5                     3                   0                    0
P3                  3                   4               10                     7                   3                    3
P5                  4                   1                6                     2                   1                    1

Total  complition  time = 39

Average waiting time = 2.6
Average turn around  time = 5.8
PS C:\Users\Teena.sahu\Downloads> 
```

Q9) Write a program to implement non-preemptive priority based scheduling algorithm.

Code:

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstring>
#include <vector>
using namespace std;
int execute1(string s, string exe[], int index, int bt, int *arival)
{
    for (int i = index; i < bt; i++)
    {
        exe[i] = s;
    }

    *arival = bt;
    return bt;
}
int main()
{

    vector<int> myvec;
    vector<int>::iterator it;
    it = myvec.begin();
    int size;
    cout << "Enter the number of process: ";
    cin >> size;
    string process[size];
```

```
cout << "enter name of process \n";
for (int i = 0; i < size; i++)
{
    cin >> process[i];
}
cout << "enter the arival time of pocesss \n";
int time[size];
for (int i = 0; i < size; i++)
{
    cin >> time[i];
}
int bt[size];
cout << "enter the burst time of process \n";

for (int i = 0; i < size; i++)
{
    cin >> bt[i];
}
int priority[size];
cout << "\n Enter priority of process:\n ";
   for (int i = 0; i < size; i++)
{
    cin >> priority[i];
}
int temptime;
string tempstr;
for (int i = 0; i < size; i++)
{
    for (int j = i + 1; j < size; j++)
    {
        if (time[j] < time[i])
        {
            temptime = time[i];
            time[i] = time[j];
            time[j] = temptime;
            temptime = bt[i];
            bt[i] = bt[j];
            bt[j] = temptime;
             temptime = priority[i];
            priority[i] = priority[j];
            priority[j] = temptime;
            tempstr = process[i];
            process[i] = process[j];
            process[j] = tempstr;
        }
    }
}

int sum = 0, g, b = 0;
for (int i = 0; i < size; i++)
{
    sum = sum + bt[i];
}
int temparival[size];
int tempbt[size];
for (int i = 0; i < size; i++)
```

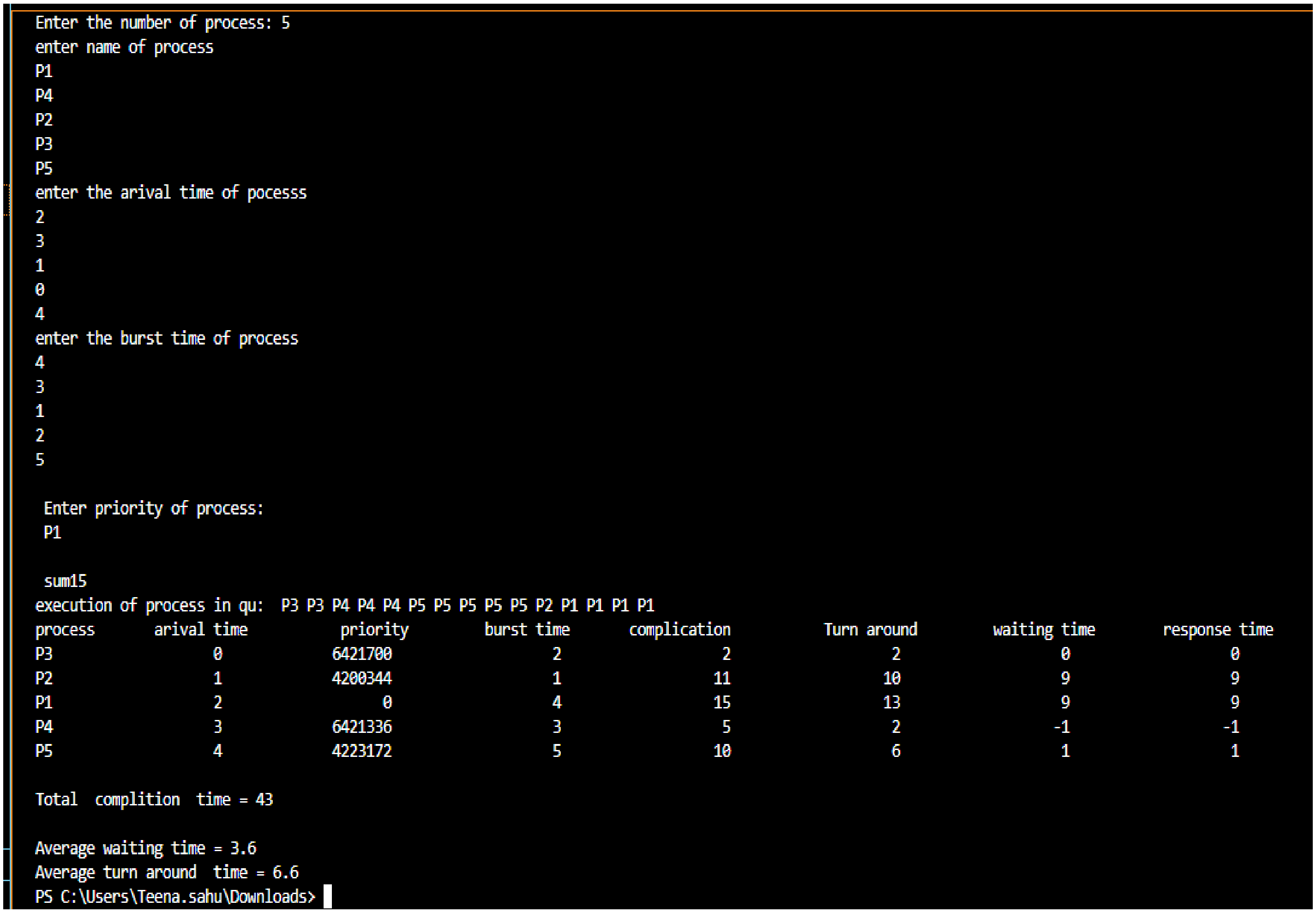
```
{
   temparival[i] = time[i];
   tempbt[i] = bt[i];
}
int bt2 = 0;
string exe[sum];
int arival = 0;
int temp = 0, p;
int j = 0;

int a = 0;
int index = 0;
a=0;
while (j != size)
{

 j=0;
   if(a==0){
   bt2=bt2+tempbt[0];
 b = execute1(process[0], exe, index, bt2, &arival);
 index=b;
 tempbt[0]=0;

 a=1;
 }
 else{for(int i=1;i<size;i++){
   if(arival>=temparival[i] && tempbt[i]!=0){
   temp=priority[i];
```

```
        p=i;
        for(int t=1;t<size;t++){
        for(int k=t+1;k<size;k++){
            if(temp<priority[k] && tempbt[k]!=0 ){
                temp=priority[k];
                p=k;
            }
        }
        if(tempbt[p]!=0){
    bt2=bt2+tempbt[p];
     b = execute1(process[p], exe, index, bt2, &arival);
     index=b;
     tempbt[p]=0;

     }}
        }
     }


for(int i=0;i<size;i++){
    if(tempbt[i]==0){
    j++;
}}



    }}
    cout << "\n sum" << sum << endl;
cout << "execution of process in qu:  ";
for (int i = 0; i < sum; i++)
{
    cout << exe[i] << " ";
}
int complition[size];
int Tat[size];
int waiting[size];
int rt[size];
for (int i = 0; i < size; i++)
{
    for (int j = sum - 1; i >= 0; j--)
    {
        if (process[i] == exe[j])
        {
            complition[i] = j + 1;
            break;
        }
    }
}
for (int i = 0; i < size; i++)
{
    Tat[i] = (complition[i] - time[i]);
    waiting[i] = Tat[i] - bt[i];
}
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < sum; j++)
```

```
        {
            if (process[i] == exe[j])
            {
                rt[i] = j - time[i];
                break;
            }
        }
    }
    cout << "\nprocess      "
         << "arival time      " << "    priority      "
         << "  burst time      "
         << "complication    "
         << "      Turn around     "
         << "   waiting time       "
         << "response time" << endl;
    for (int i = 0; i < size; i++)
    {

        cout << process[i] << setw(20) << time[i] << setw(20)<<priority[i]<<setw(20) << bt[i] << setw(20) << complition[i] << setw(20) << Tat[i] << setw(20) << waiting[i] << setw(20) << rt[i] << endl;
    }
    float sum1 = 0, sum2 = 0, sum3 = 0;

    for (int i = 0; i < size; i++)
    {
        sum1 = sum1 + waiting[i];
        sum2 = sum2 + Tat[i];
        sum3 = sum3 + complition[i];
```

```
}
float avg = sum1 / size;
float avgtat = sum2 / size;
cout << "\nTotal  complition  time = " << sum3 << endl;
cout << "\nAverage waiting time = " << avg << endl;
cout << "Average turn around  time = " << avgtat << endl;


return 0;}
```

<u>Output:</u>



```
Enter the number of process: 5
enter name of process
P1
P4
P2
P3
P5
enter the arival time of pocesss
2
3
1
0
4
enter the burst time of process
4
3
1
2
5

 Enter priority of process:
 P1

 sum15
execution of process in qu:  P3 P3 P4 P4 P4 P5 P5 P5 P5 P5 P2 P1 P1 P1 P1
process      arival time          priority        burst time      complication          Turn around         waiting time         response time
P3                0               6421700                2               2                   2                   0                    0
P2                1               4200344                1              11                  10                   9                    9
P1                2                     0                4              15                  13                   9                    9
P4                3               6421336                3               5                   2                  -1                   -1
P5                4               4223172                5              10                   6                   1                    1

Total  complition  time = 43

Average waiting time = 3.6
Average turn around  time = 6.6
PS C:\Users\Teena.sahu\Downloads>
```

Q10) Write a program to implement preemptive priority based scheduling algorithm.

Code:

```
#include <iostream>

#include <iomanip>

#include <vector>

#include <algorithm>

#include <cstring>

#include <vector>

using namespace std;

int execute1(string s, string exe[], int index, int bt, int *arival)

{

   for (int i = index; i < bt; i++)

   {

      exe[i] = s;

   }


   *arival = bt;

   return bt;

}

int main()

{


   vector<int> myvec;

   vector<int>::iterator it;

   it = myvec.begin();

   int size;

   cout << "Enter the number of process: ";

   cin >> size;
```

```
string process[size];
cout << "enter name of process \n";
for (int i = 0; i < size; i++)
{
   cin >> process[i];
}
cout << "enter the arival time of pocesss \n";
int time[size];
for (int i = 0; i < size; i++)
{
   cin >> time[i];
}
int bt[size];
cout << "enter the burst time of process \n";

for (int i = 0; i < size; i++)
{
   cin >> bt[i];
}
int priority[size];
cout << "\n Enter priority of process:\n ";
  for (int i = 0; i < size; i++)
{
   cin >> priority[i];
}
int temptime;
string tempstr;
for (int i = 0; i < size; i++)
```

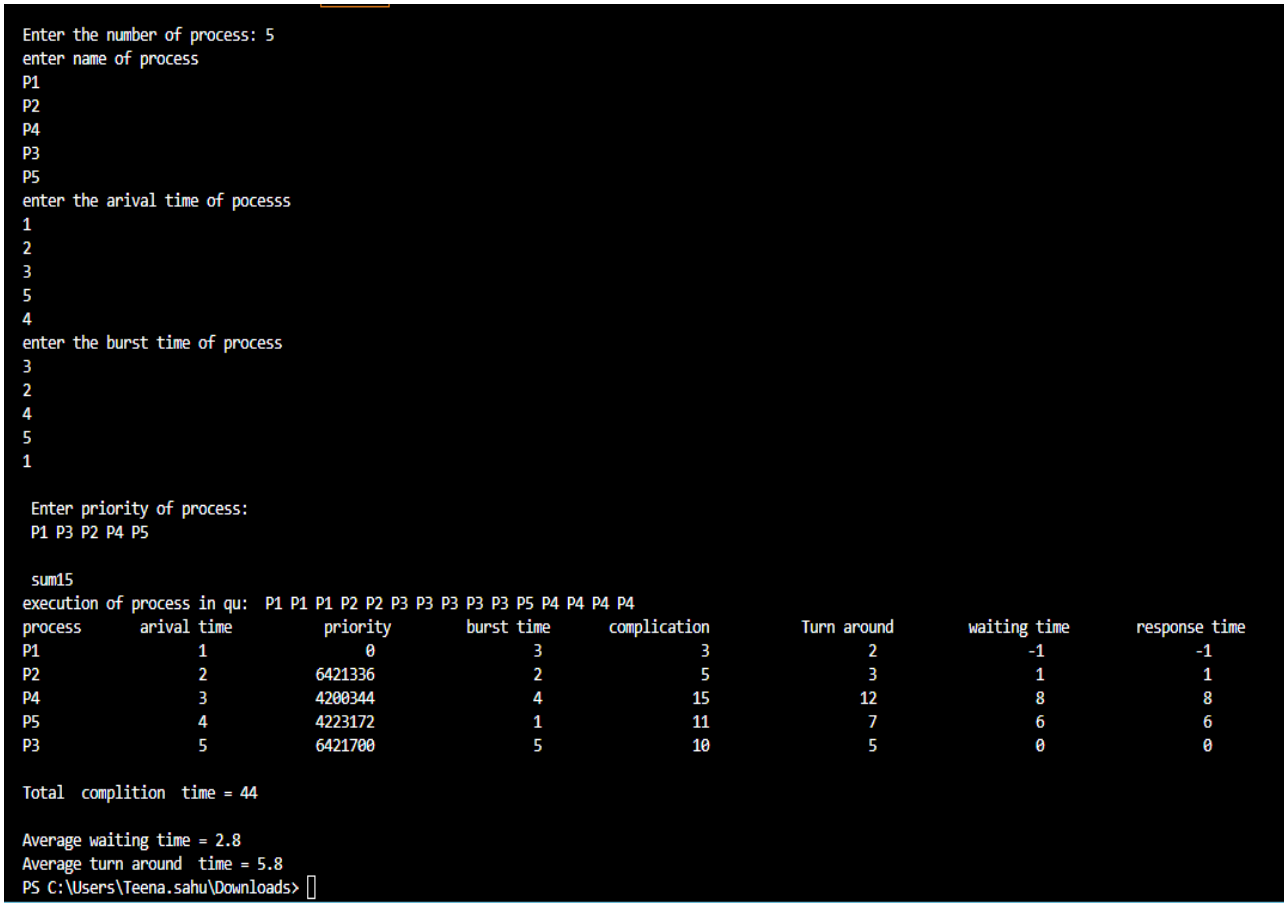
```
{
    for (int j = i + 1; j < size; j++)
    {
        if (time[j] < time[i])
        {
            temptime = time[i];
            time[i] = time[j];
            time[j] = temptime;
            temptime = bt[i];
            bt[i] = bt[j];
            bt[j] = temptime;
             temptime = priority[i];
            priority[i] = priority[j];
            priority[j] = temptime;
            tempstr = process[i];
            process[i] = process[j];
            process[j] = tempstr;
        }
    }
}

int sum = 0, g, b = 0;
for (int i = 0; i < size; i++)
{
    sum = sum + bt[i];
}
int temparival[size];
int tempbt[size];
```

```
for (int i = 0; i < size; i++)
{
   temparival[i] = time[i];
   tempbt[i] = bt[i];
}
int bt2 = 0;
string exe[sum];
int arival = 0;
int temp = 0, p;
int j = 0;

int a = 0;
int index = 0;
a=0;
while (j != size)
{

 j=0;
   if(a==0){
   bt2=bt2+1;
 b = execute1(process[0], exe, index, bt2, &arival);
 index=b;
 tempbt[0]=tempbt[0]-1;

 a=1;
 }
 else{for(int i=0;i<size;i++){
   if(arival>=temparival[i] && tempbt[i]!=0){
        temp=priority[i];
        p=i;
        for(int t=0;t<size;t++){
        for(int k=t+1;k<size;k++){
            if(temp<priority[k] && tempbt[k]!=0 && arival>=temparival[k] ){
                temp=priority[k];
                p=k;
            }
        }
        if(tempbt[p]!=0){
    bt2=bt2+1;
     b = execute1(process[p], exe, index, bt2, &arival);
     index=b;
       tempbt[p]=tempbt[p]-1;
     }}
        }
     }


for(int i=0;i<size;i++){
    if(tempbt[i]==0){
    j++;
}}




    }}
    cout << "\n sum" << sum << endl;
cout << "execution of process in qu:  ";
for (int i = 0; i < sum; i++)
{
    cout << exe[i] << " ";
}
int complition[size];
int Tat[size];
int waiting[size];
int rt[size];
for (int i = 0; i < size; i++)
{
    for (int j = sum - 1; i >= 0; j--)
    {
        if (process[i] == exe[j])
        {
            complition[i] = j + 1;
            break;
        }
    }
}
for (int i = 0; i < size; i++)
{
    Tat[i] = (complition[i] - time[i]);
    waiting[i] = Tat[i] - bt[i];
}
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < sum; j++)
```

```
        {
            if (process[i] == exe[j])
            {
                rt[i] = j - time[i];
                break;
            }
        }
    }
    cout << "\nprocess       "
         << "arival time       " << "    priority       "
         << "  burst time       "
         << "complication     "
         << "      Turn around      "
         << "   waiting time        "
         << "response time" << endl;
    for (int i = 0; i < size; i++)
    {

        cout << process[i] << setw(20) << time[i] << setw(20)<<priority[i]<<setw(20) << bt[i] << setw(20) << complition[i] << setw(20) << Tat[i] << setw(20) << waiting[i] << setw(20) << rt[i] << endl;
    }
    float sum1 = 0, sum2 = 0, sum3 = 0;

    for (int i = 0; i < size; i++)
    {
        sum1 = sum1 + waiting[i];
        sum2 = sum2 + Tat[i];
        sum3 = sum3 + complition[i];
```

```
}
float avg = sum1 / size;
float avgtat = sum2 / size;
cout << "\nTotal  complition  time = " << sum3 << endl;
cout << "\nAverage waiting time = " << avg << endl;
cout << "Average turn around  time = " << avgtat << endl;

return 0;
```

Output:



```
Enter the number of process: 5
enter name of process
P1
P2
P4
P3
P5
enter the arival time of pocesss
1
2
3
5
4
enter the burst time of process
3
2
4
5
1

 Enter priority of process:
 P1 P3 P2 P4 P5

 sum15
execution of process in qu:  P1 P1 P1 P2 P2 P3 P3 P3 P3 P3 P5 P4 P4 P4 P4
process       arival time          priority        burst time      complication          Turn around         waiting time         response time
P1                      1                 0                 3                 3                   2                   -1                   -1
P2                      2           6421336                 2                 5                   3                    1                    1
P4                      3           4200344                 4                15                  12                    8                    8
P5                      4           4223172                 1                11                   7                    6                    6
P3                      5           6421700                 5                10                   5                    0                    0

Total  complition  time = 44

Average waiting time = 2.8
Average turn around  time = 5.8
PS C:\Users\Teena.sahu\Downloads> 
```

Q11) Write a program to implement SRJF scheduling algorithm.

Code:

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cstring>
using namespace std;
int execute1(string s, string exe[], int index, int bt, int *arival  )
{
   for (int i = index; i < bt; i++)
   {
      exe[i] = s;
   }
*arival=bt;
   return bt;
}
int main()
{
   int size;
   cout << "Enter the number of process: ";
   cin >> size;
   string process[size];
   cout << "enter name of process \n";
   for (int i = 0; i < size; i++)
   {
      cin >> process[i];
   }
   cout << "enter the arival time of pocesss \n";
```

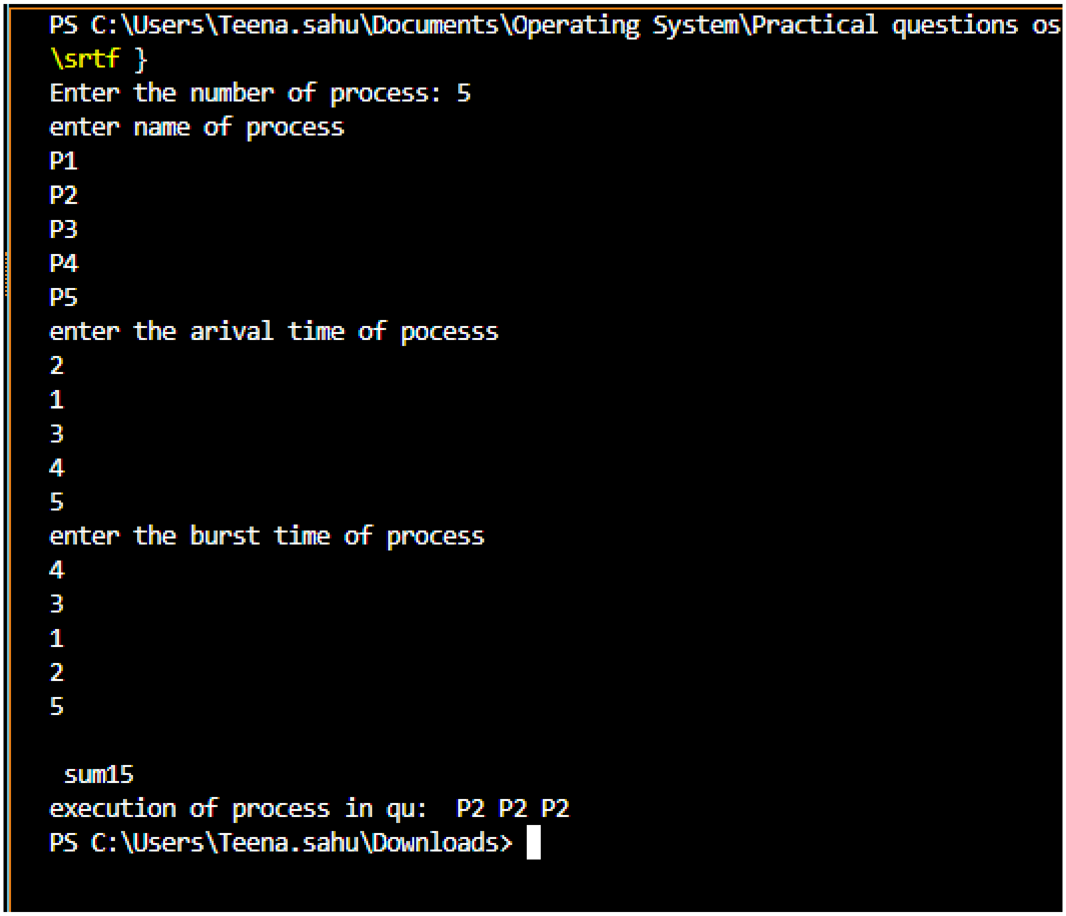
```
  int time[size];
  for (int i = 0; i < size; i++)
  {
      cin >> time[i];
  }
  int bt[size];
  cout << "enter the burst time of process \n";

  for (int i = 0; i < size; i++)
  {
      cin >> bt[i];
  }
  int temptime;
  string tempstr;
 for(int i=0;i<size;i++){
  for(int j=i+1;j<size;j++){
      if(time[j]<time[i]){
          temptime=time[i];
          time[i]=time[j];
          time[j]=temptime;
          temptime=bt[i];
          bt[i]=bt[j];
          bt[j]=temptime;
          tempstr=process[i];
          process[i]=process[j];
          process[j]=tempstr;
      }
  }
```

```
}



  int sum = 0,b;
  for (int i = 0; i < size; i++)
  {
     sum = sum + bt[i];
  }
  int temparival[size];
  int tempbt[size];
  for (int i = 0; i < size; i++)
  {
     temparival[i] = time[i];
     tempbt[i]=bt[i];
  }
  int bt2 = 0,c=0;
  string exe[sum];
  int temp = 0, p ;
  int j = 0;
  temp = temparival[0];

  int index = 0;
  int a=0;
  int arival=0;
  while (j < sum)
  {  if(a==0){
     temp = temparival[0];
```

```
p=0;
for (int i = 0; i < size; i++)
{
   if (temp > temparival[i])
   {
      temp = temparival[i];
      p = i;
   }
}
// temparival[p] = 100;
bt2 = bt2 + 1;
// cout<<"\nproces "<<process[p]<<endl
 b = execute1(process[p], exe, index, bt2,&arival);
tempbt[p]=tempbt[p]-1;
// cout << "index" << index << endl;
index = b;
a=1;
if(tempbt[p]==0){
   temparival[p]=100;
}
}
else {int tempbt1;



      tempbt1=tempbt[0];
      c=0;
         for(int k=0;k<size;k++){
                if((temparival[k]<=arival && tempbt1>tempbt[k])&& tempbt[k]!=0){


                    tempbt1=tempbt[k];

                    c=k;


                }

            }




             if(tempbt[c]!=0){

                  bt2 = bt2 + 1;

                  tempbt[c]=tempbt[c]-1;

                b = execute1(process[c], exe, index, bt2,&arival);

                index=b;


            }

                  if(tempbt[c]==0){

        temparival[c]=100;

    }




    }

    j++;

}

cout<<"\n sum"<<sum<<endl;

cout << "execution of process in qu:  ";

for (int i = 0; i < sum; i++)

{
    cout << exe[i] << " ";
}
int complition[size];
int Tat[size];
int waiting[size];
int rt[size];
for(int i=0;i<size;i++){
    for(int j=sum-1;i>=0;j--){
        if(process[i]==exe[j]){
            complition[i]=j+1;
            break;
        }
    }
}
for(int i=0;i<size;i++){
    Tat[i]=(complition[i]-time[i]);
    waiting[i]=Tat[i]-bt[i];
}
for(int i=0;i<size;i++){
    for(int j=0;j<sum;j++)
    {
        if(process[i]==exe[j]){
            rt[i]=j-time[i];
            break;
        }
    }
}
    cout << "\nprocess       "
```

```
        << "arival time       "  << "  burst time       "<<"complication     " <<"      Turn around      "<<"   waiting
time        "<<"response time"<< endl;
   for (int i = 0; i < size; i++)
   {



        cout << process[i] << setw(20) << time[i] << setw(20) << bt[i] <<setw(20) << complition[i]<<setw(20) <<
Tat[i]<<setw(20) << waiting[i]<<setw(20) << rt[i]<< endl;
   }
   float sum1=0,sum2=0,sum3=0;

for (int i=0;i<size;i++){
   sum1=sum1+waiting[i];
   sum2=sum2+Tat[i];
   sum3=sum3+complition[i];
}
float avg=sum1/size;
float avgtat=sum2/size;
     cout<<"\nTotal  complition  time = "<<sum3<<endl;
   cout<<"\nAverage waiting time = "<<avg<<endl;
     cout<<"Average turn around  time = "<<avgtat<<endl;

   return 0;
}
```

Output:



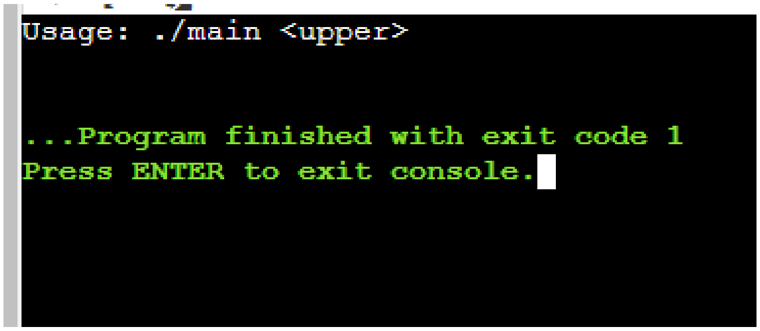
```
PS C:\Users\Teena.sahu\Documents\Operating System\Practical questions os
\srtf }
Enter the number of process: 5
enter name of process
P1
P2
P3
P4
P5
enter the arival time of pocesss
2
1
3
4
5
enter the burst time of process
4
3
1
2
5

 sum15
execution of process in qu:  P2 P2 P2
PS C:\Users\Teena.sahu\Downloads>
```

Q12) Write a program to calculate sum of n numbers using thread library.

Code:

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

using namespace std;

long long sum;

void *runner(void *number);

int main(int argc, char **argv)
{
  if (argc != 2)
  {
    cerr << "Usage: ./main <upper>" << endl;
    exit(1);
  }

  if (atoi(argv[1]) < 0)
  {
    cerr << "Argument must be non-negative." << endl;
    exit(1);
  }

  pthread_t tid;
  pthread_attr_t attr;
  pthread_attr_init(&attr);

  pthread_create(&tid, &attr, runner, (void *)argv[1]);

  pthread_join(tid, NULL);


  cout << "Sum from 1 to " << atoi(argv[1])

       << " is " << sum << endl;


  return 0;

}


void *runner(void *upper)

{

  int num = atoi((const char *)(upper));

  for (int i = 1; i <= num; i++)

    sum += i;

  pthread_exit(0);

  return nullptr;

}
```

<u>Output:</u>



```
Usage: ./main <upper>


...Program finished with exit code 1
Press ENTER to exit console.
```

Q13) Write a program to implement first-fit, best-fit and worst-fit allocation strategies.

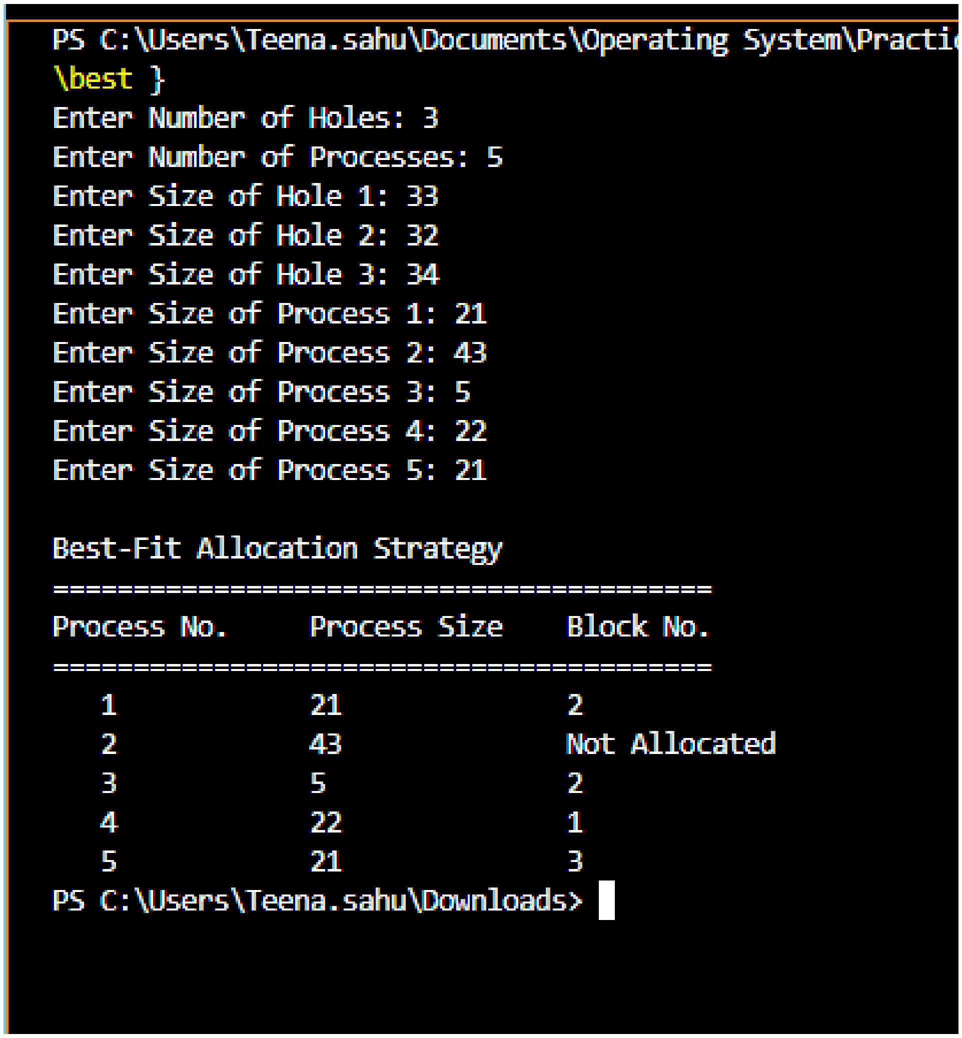
–Best-Fit

Code:

```
#include <cstring>

#include <iostream>

#define MAX_SIZE 100


using namespace std;


void bestFit(int blockSize[], int m,

             int processSize[], int n)

{

  int allocation[n];


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

    allocation[i] = -1;


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

  {

    int bestIdx = -1;

    for (int j = 0; j < m; j++)

    {

      if (blockSize[j] >= processSize[i])

      {

        if (bestIdx == -1)

          bestIdx = j;

        else if (blockSize[bestIdx] > blockSize[j])

          bestIdx = j;

      }
    }

    if (bestIdx != -1)
    {
      allocation[i] = bestIdx;
      blockSize[bestIdx] -= processSize[i];
    }
  }

  cout << "\nBest-Fit Allocation Strategy\n";
  cout << "==============================================\n";
  cout << "Process No.\tProcess Size\tBlock No.\n";
  cout << "==============================================\n";
  for (int i = 0; i < n; i++)
  {
    cout << "   " << i + 1 << "\t\t" << processSize[i] << "\t\t";
    if (allocation[i] != -1)
      cout << allocation[i] + 1;
    else
      cout << "Not Allocated";
    cout << endl;
  }
}

int main()
{
  int holes, processes;
  int holeSizes[MAX_SIZE], processSizes[MAX_SIZE];
```

```
 cout << "Enter Number of Holes: ";
 cin >> holes;
 cout << "Enter Number of Processes: ";
 cin >> processes;

 for (int i = 0; i < holes; i++)
 {
  cout << "Enter Size of Hole " << (i + 1) << ": ";
  cin >> holeSizes[i];
 }

 for (int i = 0; i < processes; i++)
 {
  cout << "Enter Size of Process " << (i + 1) << ": ";
  cin >> processSizes[i];
 }

 bestFit(holeSizes, holes, processSizes, processes);

 return 0;
}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Operating System\Practi
\best }
Enter Number of Holes: 3
Enter Number of Processes: 5
Enter Size of Hole 1: 33
Enter Size of Hole 2: 32
Enter Size of Hole 3: 34
Enter Size of Process 1: 21
Enter Size of Process 2: 43
Enter Size of Process 3: 5
Enter Size of Process 4: 22
Enter Size of Process 5: 21

Best-Fit Allocation Strategy
=========================================
Process No.     Process Size    Block No.
=========================================
   1            21              2
   2            43              Not Allocated
   3            5               2
   4            22              1
   5            21              3
PS C:\Users\Teena.sahu\Downloads> 
```

–First-Fit

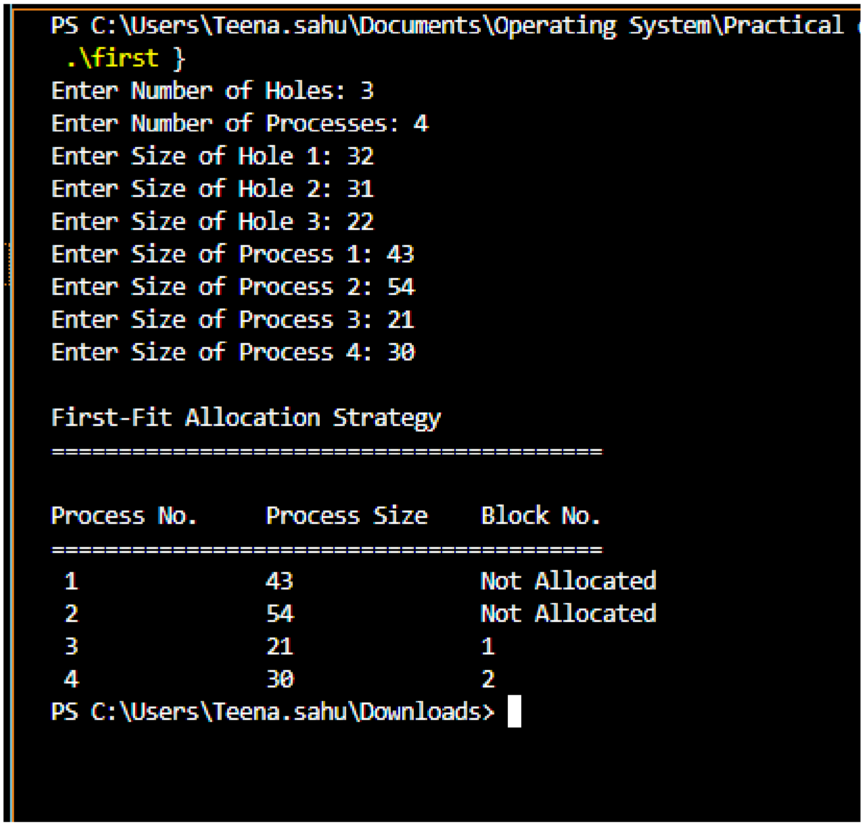
Code:

```
#include <cstring>
#include <iostream>
#define MAX_SIZE 100

using namespace std;

void firstFit(int blockSize[], int m,
              int processSize[], int n)
{
  int allocation[n];

  for (int i = 0; i < n; i++)
    allocation[i] = -1;

  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++)
    {
      if (blockSize[j] >= processSize[i])
      {
        allocation[i] = j;
        blockSize[j] -= processSize[i];
        break;
      }
    }
  }

  cout << "\nFirst-Fit Allocation Strategy\n";
```

```
 cout << "================================================\n";
 cout << "\nProcess No.\tProcess Size\tBlock No.\n";
 cout << "================================================\n";
 for (int i = 0; i < n; i++)
 {
  cout << " " << i + 1 << "\t\t"
      << processSize[i] << "\t\t";
  if (allocation[i] != -1)
   cout << allocation[i] + 1;
  else
   cout << "Not Allocated";
  cout << endl;
 }
}

int main()
{
 int holes, processes;
 int holeSizes[MAX_SIZE], processSizes[MAX_SIZE];

 cout << "Enter Number of Holes: ";
 cin >> holes;
 cout << "Enter Number of Processes: ";
 cin >> processes;

 for (int i = 0; i < holes; i++)
 {
  cout << "Enter Size of Hole " << (i + 1) << ": ";
```

```
    cin >> holeSizes[i];
  }

  for (int i = 0; i < processes; i++)
  {
    cout << "Enter Size of Process " << (i + 1) << ": ";
    cin >> processSizes[i];
  }

  firstFit(holeSizes, holes, processSizes, processes);

  return 0;
}
```

Output:



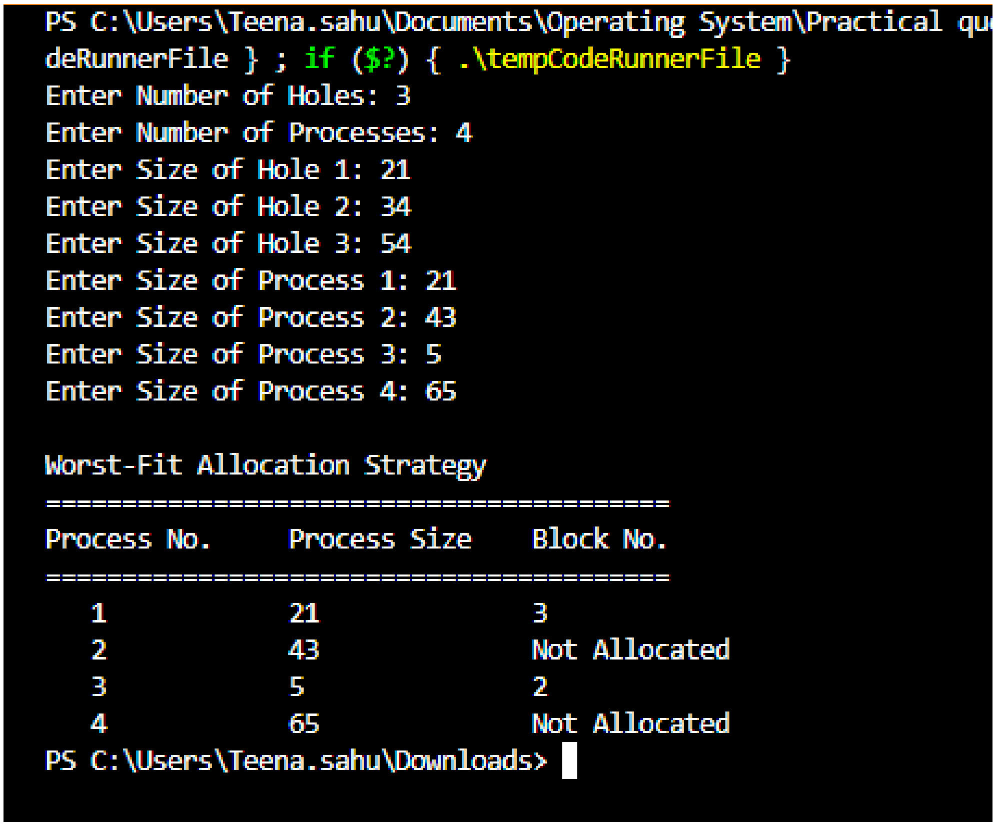
```
PS C:\Users\Teena.sahu\Documents\Operating System\Practical 
 .\first }
Enter Number of Holes: 3
Enter Number of Processes: 4
Enter Size of Hole 1: 32
Enter Size of Hole 2: 31
Enter Size of Hole 3: 22
Enter Size of Process 1: 43
Enter Size of Process 2: 54
Enter Size of Process 3: 21
Enter Size of Process 4: 30

First-Fit Allocation Strategy
=========================================

Process No.     Process Size    Block No.
=========================================
 1              43              Not Allocated
 2              54              Not Allocated
 3              21              1
 4              30              2
PS C:\Users\Teena.sahu\Downloads>
```

–Worst-Fit

Code:

```
#include <iostream>
#define MAX_SIZE 100

using namespace std;

void worstFit(int blockSize[], int m,
              int processSize[], int n)
{
  int allocation[n];

  for (int i = 0; i < n; i++)
    allocation[i] = -1;

  for (int i = 0; i < n; i++)
  {
    int wstIdx = -1;
    for (int j = 0; j < m; j++)
    {
      if (blockSize[j] >= processSize[i])
      {
        if (wstIdx == -1)
          wstIdx = j;
        else if (blockSize[wstIdx] < blockSize[j])
          wstIdx = j;
      }
    }

    if (wstIdx != -1)
    {
      allocation[i] = wstIdx;
      blockSize[wstIdx] -= processSize[i];
    }
  }

  cout << "\nWorst-Fit Allocation Strategy\n";
  cout << "=============================================\n";
  cout << "Process No.\tProcess Size\tBlock No.\n";
  cout << "=============================================\n";
  for (int i = 0; i < n; i++)
  {
    cout << "   " << i + 1 << "\t\t" << processSize[i] << "\t\t";
    if (allocation[i] != -1)
      cout << allocation[i] + 1;
    else
      cout << "Not Allocated";
    cout << endl;
  }
}

int main()
{
  int holes, processes;
  int holeSizes[MAX_SIZE], processSizes[MAX_SIZE];

  cout << "Enter Number of Holes: ";
  cin >> holes;
  cout << "Enter Number of Processes: ";
  cin >> processes;

  for (int i = 0; i < holes; i++)
  {
    cout << "Enter Size of Hole " << (i + 1) << ": ";
    cin >> holeSizes[i];
  }

  for (int i = 0; i < processes; i++)
  {
    cout << "Enter Size of Process " << (i + 1) << ": ";
    cin >> processSizes[i];
  }

  worstFit(holeSizes, holes, processSizes, processes);

  return 0;
}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Operating System\Practical qu
deRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter Number of Holes: 3
Enter Number of Processes: 4
Enter Size of Hole 1: 21
Enter Size of Hole 2: 34
Enter Size of Hole 3: 54
Enter Size of Process 1: 21
Enter Size of Process 2: 43
Enter Size of Process 3: 5
Enter Size of Process 4: 65

Worst-Fit Allocation Strategy
=========================================
Process No.     Process Size    Block No.
=========================================
   1            21              3
   2            43              Not Allocated
   3            5               2
   4            65              Not Allocated
PS C:\Users\Teena.sahu\Downloads>
```