# Operating Systems

**Name:-** Shivang Shukla

**Roll No :-** 21058570054

**Semester:-** III

**Session:-** 2022-23

**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.**

**a)**

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

int main()
{
fork();
printf("Today is 18 october 2021");

}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc q6.c -o q6.out && ./q6.out
Today is 18 october 2021Today is 18 october 2021shiv42@shiv42-Inspiron-5570:~/Documents/College t
Today is 18 october 2021Today is 18 october 2021shiv42@shiv42-Inspiron-5570:~/Documents/College t
Today is 18 october 2021Today is 18 october 2021shiv42@shiv42-Inspiron-5570:~/Documents/College t
Today is 18 october 2021Today is 18 october 2021shiv42@shiv42-Inspiron-5570:~/Documents/College t
Today is 18 october 2021Today is 18 october 2021shiv42@shiv42-Inspiron-5570:~/Documents/College P
rograms/Sem 3/OS$
```

**b)**
```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
int pid;
pid=fork();
if(pid < 0)
{
printf("\n Error ");
exit(1);
}
else if(pid==0)
{
printf("\n Hello I am the child process");
printf("\n My pid is %d",getpid());
exit(0);
}
else
{
printf("\n Hello I am the parent process ");
printf("\n My actual pid is %d",getpid());

exit(1);
}
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc q5.c -o q5.out && ./q5.out

 Hello I am the parent process
 My actual pid is 7974
 Hello I am the child process
 My pid is 7975shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc q5.c -o q5t
out && ./q5.out
 Hello I am the parent process
 My actual pid is 7981
 Hello I am the child process
 My pid is 7982shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$
```

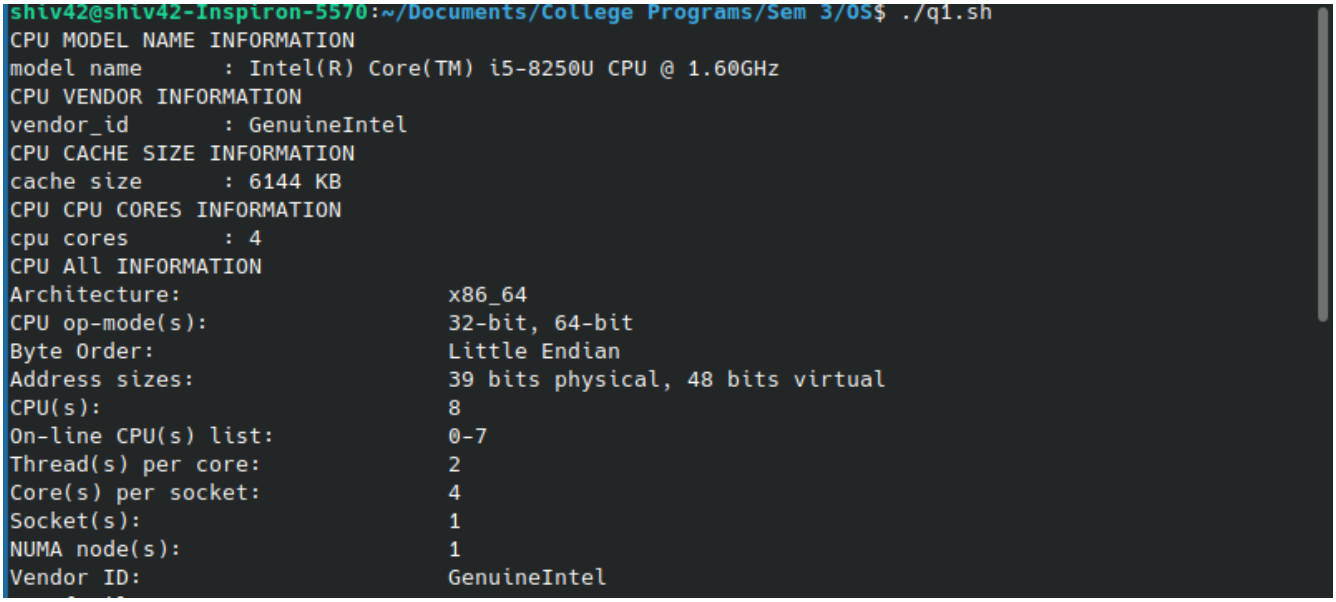
**c)**

```
#include<stdio.h>
#include<unistd.h>
#include <sys/wait.h>
int main()
{
int pid;
pid=fork();
if(pid<0) {
printf(" error");
return 1;
}
else if (pid==0) {
printf("\nChild process created");
printf("\nProcess id of child is %d", getpid()); }
else if(pid>0) {
printf("\nParent process created");
printf("\nProcess id of Parent is %d", getpid()); wait(NULL);
}
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc q8.c -o q8.out && ./q8.out

Parent process created

Child process created
Process id of child is 8107Process id of Parent is 8106shiv42@shiv42-Inspiron-5570:~/Documents/Ct
llege Programs/Sem 3/OS$ gcc q8.c -o q8.out && ./q8.out
Parent process created

Child process created
Process id of child is 8114Process id of Parent is 8113shiv42@shiv42-Inspiron-5570:~/Documents/Ct
llege Programs/Sem 3/OS$ gcc q8.c -o q8.out && ./q8.out
Parent process created
```

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

```
echo "CPU MODEL NAME INFORMATION"
cat /proc/cpuinfo | grep 'model name' | uniq
echo "CPU VENDOR INFORMATION"
cat /proc/cpuinfo | grep 'vendor' | uniq
echo "CPU CACHE SIZE INFORMATION"
cat /proc/cpuinfo | grep 'cache size' | uniq
echo "CPU CPU CORES INFORMATION"
cat /proc/cpuinfo | grep 'cpu cores' | uniq
echo "CPU All INFORMATION"
lscpu
```




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

```
echo "Question 2 MEMORY INFORMATION"

echo "TOTAL MEMORY Information"
cat /proc/meminfo | grep 'MemTotal' | uniq
echo "MEMORY FREE Information"
cat /proc/meminfo | grep 'MemFree' | uniq
echo "MEMORY Available"
cat /proc/meminfo | grep 'MemAvailable' | uniq
```

```
Question 2 MEMORY INFORMATION
TOTAL MEMORY Information
MemTotal:       12158896 kB
MEMORY FREE Information
MemFree:         6096832 kB
MEMORY Available
MemAvailable:    8704652 kB
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$
```

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

```
echo -n "Enter file name : "
read file

# find out if file has write permission or not
[ -w $file ] && W="Write = yes" || W="Write = No"

# find out if file has excute permission or not
[ -x $file ] && X="Execute = yes" || X="Execute = No"

# find out if file has read permission or not
[ -r $file ] && R="Read = yes" || R="Read = No"

echo "$file permissions"
echo "$W"
echo "$R"
echo "$X"

FILE="$1"

# use stat command to get info
echo "Time of last access : "
stat -c %x $file
echo "Time of last modification : "
stat -c %y $file
echo "Time of last change : "
stat -c %z $file
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ ./q3.sh
Enter file name : a.txt
a.txt permissions
Write = yes
Read = yes
Execute = No
Time of last access :
2022-12-03 14:31:30.874548464 +0530
Time of last modification :
2022-12-03 14:31:30.878548538 +0530
Time of last change :
2022-12-03 14:31:30.878548538 +0530
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$
```

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

```
#include <sys/types.h>
#include <sys/uio.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#define BUFSIZE 512
int main (int argc, char** argv)
{
int from, to, nr, nw, n;
char buf[BUFSIZE];
if ((from=open(argv[1], O_RDONLY)) < 0) {
perror("Error opening source file");
exit(1);
}
if ((to=creat(argv[2], 0666)) < 0) {
perror("Error creating destination file");
exit(2);
}
while((nr=read(from, buf, sizeof( buf))) != 0) {
if (nr < 0) {
perror("Error reading source file");
exit(3);
}
nw=0;
do {
if ((n=write(to, &buf[nw], nr-nw)) < 0) {
perror("Error writing destination file");
exit(4);
```

```
}
nw += n;
}
while (nw < nr);
}
close(from );
close(to);
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc q4.c -o q4.out && ./q4.out
 a.txt b.txt
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$
```

**Q6) Write a program to implement FCFS scheduling algorithm.**

```
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;
#define MAX 50

class FCFS{
private:
int siz;
float pid[MAX][3];
public:
FCFS(int s);
void swapprocess(int a,int b);
void calculate();
void insort();
void display();
void compute_values();
};
FCFS::FCFS(int s = 0){
siz = s;
cout << "Enter the arrival and burst times for: \n";
for(int k = 0;k < s;k++){
pid[k][0] = k+1;
cout << "Process " << k+1 << ": ";
scanf("%f %f",&pid[k][1],&pid[k][2]);cout << endl;
}
//Specific to FCFS
insort();
}
void FCFS::swapprocess(int a,int b){
float* fir = pid[a];float* sec = pid[b];
```

```
int mid[3];
for(int t = 0;t < siz;t++){
mid[t] = sec[t];
sec[t] = fir[t];
fir[t] = mid[t];
}
}
void FCFS::insort(){
for(int i = 0;i < siz;i++){
for(int j = i+1;j < siz;j++){
if(pid[i][1]>pid[j][1])
swapprocess(i,j);
}}
}

void FCFS::calculate(){
float turnaround[siz];float waitime[siz];float resptime[siz];float awt = 0,atat = 0;
cout << "Process table: \n";
// Response time
resptime[0] = 0;
for(int m = 1;m < siz;m++)
resptime[m] = resptime[m-1]+pid[m-1][2];
// Waiting time
for(int m = 0;m < siz;m++){
waitime[m] = resptime[m]-pid[m][1];
awt+=waitime[m];
}
awt/=siz;
// T.A.T time
for(int m = 0;m < siz;m++){
turnaround[m] = resptime[m]+pid[m][2]-pid[m][1];
atat+=turnaround[m];
}
atat/=siz;
cout << "| PID | Waiting |Turnaround |" << endl;
cout << "| | Time \t|\t Time |" << endl;
for(int n = 0;n < siz;n++)
cout << " " << pid[n][0] << "\t\t\t" << waitime[n] << "\t\t\t" << turnaround[n] << endl;
cout << *(new string(40,'=')) << endl;
cout << "Average waiting time: " << awt << endl;
cout << "Average turnaround time: " << atat << endl;
cout << *(new string(40,'=')) << endl;
}
void FCFS::display(){
cout << "Process table: \n";
for(int k = 0;k < siz;k++){
cout << "Process " << pid[k][0] << ": ";
```

```
cout << pid[k][1] << " " << pid[k][2] << endl;
}
}
int main(){
int n;
cout << "Enter the no. of processes: ";cin >> n;
FCFS ff(n);
ff.calculate();
}
```

```
/tmp/76AnUdGmCcK.o
Enter the no. of processes: 5
Enter the arrival and burst times for:
Process 1:  0 2
Process 2:  1 6
Process 3:  2 4
Process 4:  3 9
Process 5:  6 12
Process table:
| PID |   Waiting   |Turnaround   |
|     |    Time     |    Time       |
  1         0           2
  2         1           7
  3         6           10
  4         9           18
  5         15          27
========================================
Average waiting time: 6.2
Average turnaround time: 12.8
========================================
```

**Q7) Write a program to implement RR scheduling algorithm.**

```
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;
#define MAX 50
#define QUANTA 3

class RR{
private:
int siz;
float pid[MAX][3];
public:
RR(int s);
void swapprocess(int a,int b);
void calculate();
void insort();
void display();
void comptimeute_values();
};
RR::RR(int s = 0){
siz = s;
cout << "Enter the arrival and burst times for: \n";
for(int k = 0;k < s;k++){
pid[k][0] = k+1;
cout << "Process " << k+1 << ": ";
scanf("%f %f",&pid[k][1],&pid[k][2]);cout << endl;
}
//Specific to RR
insort();
}
void RR::swapprocess(int a,int b){
float* fir = pid[a];float* sec = pid[b];
float mid[3];
for(int t = 0;t < 3;t++){
mid[t] = sec[t];
sec[t] = fir[t];
fir[t] = mid[t];
}
}
void RR::insort(){
for(int i = 0;i < siz;i++){
for(int j = i+1;j < siz;j++){
if(pid[i][1]>pid[j][1])
swapprocess(i,j);
}}
```

```
}

void RR::calculate(){
float remtime[siz];
//Initialize remaining process time
for(int k = 0;k < siz;k++)
remtime[k] = pid[k][2];
float waitime[siz];float comptime[siz];float turnaround[siz];

float awt = 0,atat = 0;
int btsum = 0;int gcsum = 0,id = 0;

for(int i = 0;i < siz;i++)
btsum+=pid[i][2];
cout << btsum << endl;
while(gcsum<btsum){
if(id==siz)
id = 0;
//cout << gcsum << endl;
//cout << remtime[id] << QUANTA << endl;
//Check if process is alive
if(remtime[id]!=0){
//Last burst
if(remtime[id]<=QUANTA){
gcsum+=remtime[id];
comptime[id] = gcsum;
remtime[id] = 0;
}
//Regular burst
else{
gcsum+=QUANTA;
remtime[id]-=QUANTA;
}
}
id++;
}
// T.A.T
for(int m = 0;m < siz;m++){
turnaround[m] = comptime[m]-pid[m][1];
atat+=turnaround[m];
}
// W.T
for(int m = 0;m < siz;m++){
waitime[m] = turnaround[m]-pid[m][2];
awt+=waitime[m];
}
awt/=siz;atat/=siz;
cout << "| PID | Waiting |Turnaround |" << endl;
cout << "| | Time |\t Time |" << endl;
```

```
for(int n = 0;n < siz;n++)
cout << " " << pid[n][0] << "\t\t" << waitime[n] << "\t\t" << turnaround[n] << endl;
cout << *(new string(40,'=')) << endl;
cout << "Average waiting time: " << awt << endl;
cout << "Average turnaround time: " << atat << endl;
cout << *(new string(40,'=')) << endl;

}
void RR::display(){
cout << "Process table: \n";
for(int k = 0;k < siz;k++){
cout << "Process " << pid[k][0] << ": ";
cout << pid[k][1] << " " << pid[k][2] << endl;
}
}
int main(){
int n;
cout << "Enter the no. of processes: ";cin >> n;
RR rr(n);
rr.display();
rr.calculate();
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3$ g++ rr.cpp -o rr.out && .
/rr.out
Enter the no. of processes: 3
Enter the arrival and burst times for:
Process 1:  2 3

Process 2:  0 2

Process 3:  1 4

Process table:
Process 2: 0 2
Process 3: 1 4
Process 1: 2 3
9
| PID |   Waiting   |Turnaround   |
|     |    Time     |     Time    |
  2            0              2
  3            4              8
  1            3              6
========================================
Average waiting time: 2.33333
Average turnaround time: 5.33333
========================================
```

**Q8) Write a program to implement SJF scheduling algorithm.**

```
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;
#define MAX 50

class FCFS{
private:
int siz;
float pid[MAX][3];
public:
FCFS(int s);
void swapprocess(int a,int b);
void calculate();
void insort();
void display();
void compute_values();
};
FCFS::FCFS(int s = 0){
siz = s;
cout << "Enter the arrival and burst times for: \n";
for(int k = 0;k < s;k++){
pid[k][0] = k+1;
cout << "Process " << k+1 << ": ";
scanf("%f %f",&pid[k][1],&pid[k][2]);cout << endl;
}
//Specific to FCFS
insort();
}
void FCFS::swapprocess(int a,int b){
float* fir = pid[a];float* sec = pid[b];
int mid[3];
for(int t = 0;t < siz;t++){
mid[t] = sec[t];
sec[t] = fir[t];
fir[t] = mid[t];
}
};
void FCFS::insort(){
for(int i = 0;i < siz;i++){
for(int j = i+1;j < siz;j++){
if(pid[i][1]>pid[j][1])
swapprocess(i,j);
}}
}

void FCFS::calculate(){
```

```
float turnaround[siz];float waitime[siz];float comptime[siz];float awt = 0,atat = 0;
cout << "Process table: \n";

// C.T
comptime[0] = pid[0][1] + pid[0][2];
for(int m = 1;m < siz;m++){
if(pid[m][1]<comptime[m-1])
comptime[m] = comptime[m-1]+pid[m][2];//Push the process adjacently
else
comptime[m] = pid[m][1]+pid[m][2];//Account for idle time
}
// T.A.T
for(int m = 0;m < siz;m++){
turnaround[m] = comptime[m]-pid[m][1];
atat+=turnaround[m];
}
atat/=siz;
// W.T
for(int m = 0;m < siz;m++){
waitime[m] = turnaround[m]-pid[m][2];
awt+=waitime[m];
}
awt/=siz;
cout << "| PID | Waiting |Turnaround |" << endl;
cout << "| | Time \t|\t Time |" << endl;
for(int n = 0;n < siz;n++)
cout << " " << pid[n][0] << "\t\t\t" << waitime[n] << "\t\t\t" << turnaround[n] << endl;
cout << *(new string(40,'=')) << endl;
cout << "Average waiting time: " << awt << endl;
cout << "Average turnaround time: " << atat << endl;
cout << *(new string(40,'=')) << endl;
}
void FCFS::display(){
cout << "Process table: \n";
for(int k = 0;k < siz;k++){
cout << "Process " << pid[k][0] << ": ";
cout << pid[k][1] << " " << pid[k][2] << endl;
}
}
int main(){
int n;
cout << "Enter the no. of processes: ";cin >> n;
FCFS ff(n);
ff.calculate();
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3$ g++ sjf.cpp -o sjf.out &&
 ./sjf.out
Enter the no. of processes: 3
Enter the arrival and burst times for:
Process 1:  2 3

Process 2:  0 2

Process 3:  1 4

Process table:
| PID |   Waiting   |Turnaround   |
|     |    Time  | Time       |
  2             0               2
  3             1               5
  1             4               7
=======================================
Average waiting time: 1.66667
Average turnaround time: 4.66667
=======================================
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3$
```

**Q9) Write a program to implement priority scheduling(without pre-emption) algorithm.**

```
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;
#define MAX 50

class PP{
   private:
      int siz;
      float pid[MAX][4];
   public:
      PP(int s);
      void swapprocess(int a,int b);
      void calculate();
      void insort();
      void display();
      void compute_values();
};
PP::PP(int s = 0){
   siz = s;
   cout << "Enter the arrival, burst times and priorities for: \n";
   for(int k = 0;k < s;k++){
      pid[k][0] = k+1;
      cout << "Process " << k+1 << ":  ";
      scanf("%f  %f  %f",&pid[k][1],&pid[k][2],&pid[k][3]);cout << endl;
   }
   //Specific to PP
   insort();
}
void PP::swapprocess(int a,int b){
   float* fir = pid[a];float* sec = pid[b];
   int mid[4];
   for(int t = 0;t < 4;t++){
      mid[t] = sec[t];
      sec[t] = fir[t];
      fir[t] = mid[t];
   }

};
void PP::insort(){
   for(int i = 0;i < siz;i++){
      for(int j = i+1;j < siz;j++){
         if(pid[i][3]>pid[j][3])
            swapprocess(i,j);

   }}
```

```
}

void PP::calculate(){
    float turnaround[siz];float waitime[siz];float comptime[siz];float awt = 0,atat = 0;
    cout << "Process table: \n";int min = 0;


     // C.T
    comptime[0] = pid[0][1] + pid[0][2];
    for(int m = 1;m < siz;m++){
        if(pid[m][1]<comptime[m-1])
            comptime[m] = comptime[m-1]+pid[m][2];//Push the process adjacently
        else
            comptime[m] = pid[m][1]+pid[m][2];//Account for idle time
    }
    // T.A.T
    for(int m = 0;m < siz;m++){
        turnaround[m] = comptime[m]-pid[m][1];
        atat+=turnaround[m];
    }
    atat/=siz;

    // W.T
    for(int m = 0;m < siz;m++){
        waitime[m] = turnaround[m]-pid[m][2];
        awt+=waitime[m];
    }
    awt/=siz;

    cout << "| PID |   Waiting   |Turnaround   |" << endl;
    cout << "|     |    Time  | Time       |" << endl;
    for(int n = 0;n < siz;n++)
        cout << "  " << pid[n][0] << "\t\t" << waitime[n] << "\t\t" << turnaround[n] << endl;
    cout << *(new string(40,'=')) << endl;
    cout << "Average waiting time: " << awt << endl;
   cout << "Average turnaround time: " << atat << endl;
   cout << *(new string(40,'=')) << endl;
}
void PP::display(){
    cout << "Process table: \n";
    for(int k = 0;k < siz;k++){
        cout << "Process " << pid[k][0] << ": ";
        cout << pid[k][1] << " " << pid[k][2] << endl;
    }
}
int main(){
    int n;
    cout << "Enter the no. of processes: ";cin >> n;
    PP ff(n);
```

```
    ff.calculate();
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3$ g++ pp.cpp -o pp.out && .
/pp.out
Enter the no. of processes: 3
Enter the arrival, burst times and priorities for:
Process 1:  0 2 1

Process 2:  0 3 4

Process 3:  0 5 2

Process table:
| PID |   Waiting   |Turnaround   |
|     |    Time  | Time       |
  1             0              2
  3             2              7
  2             7              10
=======================================
Average waiting time: 3
Average turnaround time: 6.33333
=======================================
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3$
```

**Q10) Write a program to implement priority scheduling(with pre-emption) algorithm.**

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


class Priority_Preemptive{
    private:
        int size,atat = 0,awt = 0;
        int* arrivalTime;
        int* burstTime;
        int* priority;int* waitime;
        int* completionTime;int* turnaround;
        int* temp;
        void swap(int& n1,int& n2){
            int temp = n1;
            n1 = n2;
            n2 = temp;
        }
        bool isZero(){
            bool flag = true;
            for(int i=0;i<size;i++){
                if(temp[i]!=0){
                    flag = false;
                    return flag;
                }
            }
            return flag;
        }
        void sort(int* arr,int size){
            for(int i=0;i<size-1;i++){
                for(int j=0;j<size-i-1;j++){
                    if(priority[arr[j]]>priority[arr[j+1]]){
                        swap(arr[j],arr[j+1]);
                    }
                }
            }
            // for(int i=0;i<size;i++){
            //     cout<<arr[i]<<",";
            // }
            // cout<<endl;
        }
        void pop(int* arr,int& size){
            for(int i=0;i<size-1;i++){
                arr[i] = arr[i+1];
            }
            size--;
            // for(int i=0;i<size;i++){
            //     cout<<arr[i]<<",";
            // }
```

```
        // cout<<endl;
    }
public:
    Priority_Preemptive(int size){
        this->size = size;
        arrivalTime = new int[size];
        burstTime = new int[size];
        priority = new int[size];
        completionTime = new int[size];
        waitime = new int[size];
        turnaround = new int[size];
        temp = new int[size];
    }
    ~Priority_Preemptive(){
        delete []arrivalTime;
        delete []burstTime;
        delete []priority;
        delete []completionTime;
    }
    void input(){
        for(int i=0;i<size;i++){
            cout<<"Enter the arrival time of process "<<(i+1)<<": ";
            cin>>arrivalTime[i];

            cout<<"Enter the burst time of process "<<(i+1)<<": ";
            cin>>burstTime[i];
            temp[i] = burstTime[i];

            cout<<"Enter the priority of process "<<(i+1)<<": ";
            cin>>priority[i];
        }
    }
    void display(){
        cout << "| PID |   Arrival   |Burst   |" << endl;
cout << "|     |    Time    |\t Time    |" << endl;
for(int n = 0;n < size;n++)
    cout << "  " << n << "\t\t" << arrivalTime[n] << "\t\t" << burstTime[n] << endl;
cout << *(new string(40,'=')) << endl;
    }
    void CompletionTime(){
        int iter = 0; //points to next arrival
        int counter = 0; //gantt chart
        int* q = new int[size]; //ready queue
        int queue_iter = 0; //ready queue iterator
        for(int i=0;i<size-1;i++){
            for(int j =0;j<size-i-1;j++){
                if(arrivalTime[j]>arrivalTime[j+1])
                    swap(arrivalTime[j],arrivalTime[j+1]);
            }
```

```
        }
        do{
            if(counter<arrivalTime[iter]){
                counter++;
                continue;
            }
            else if(iter<=size-1){
                q[queue_iter] = iter;
                queue_iter++;
                iter++;
            }
            if(iter>size-1)
                iter = size;
            sort(q,queue_iter); //sort according to priority
            int current = q[0];
            // cout<<current<<endl;
            while(counter<arrivalTime[iter]){
                counter++;
                if(temp[current] == 0){
                    completionTime[current] = counter;
                    pop(q,queue_iter);
                    break;
                }
                temp[current]--;
            }

            if(counter>=arrivalTime[iter] && (temp[current]!=0) && iter>=size-1){
                counter++;
                temp[current]--;
                if(temp[current] == 0){
                    completionTime[current] = counter;
                    pop(q,queue_iter);
                }
            }
        }while(!isZero());
    }
void calculate(){
    for(int m = 0;m < size;m++){
    turnaround[m] = completionTime[m]-arrivalTime[m];
    atat+=turnaround[m];
}
// W.T
for(int m = 0;m < size;m++){
    waitime[m] = turnaround[m]-burstTime[m];
    awt+=waitime[m];
}
awt/=size;atat/=size;

cout << "| PID |   Waiting   |Turnaround   |" << endl;
```

```
    cout << "|    |    Time    |\t Time    |" << endl;
    for(int n = 0;n < size;n++)
        cout << "  " << n << "\t" << waitime[n] << "\t" << turnaround[n] << endl;
    cout << *(new string(40,'=')) << endl;
    cout << "Average waiting time: " << awt << endl;
   cout << "Average turnaround time: " << atat << endl;
   cout << *(new string(40,'=')) << endl;
}
};
int main(){
    int size = 4;
    Priority_Preemptive obj(size);
    obj.input();
    obj.CompletionTime();
    obj.calculate();
    obj.display();
}
```

```
Enter the priority of process 1: 5
Enter the arrival time of process 2: 0
Enter the burst time of process 2: 1
Enter the priority of process 2: 2
Enter the arrival time of process 3: 0
Enter the burst time of process 3: 6
Enter the priority of process 3: 7
Enter the arrival time of process 4: 0
Enter the burst time of process 4: 4
Enter the priority of process 4: 3
| PID |   Waiting   |Turnaround   |
|     |    Time     |     Time    |
  0     5       8
  1     0       1
  2     8       14
  3     1       5
========================================
Average waiting time: 3
Average turnaround time: 7
========================================
| PID |   Arrival   |Burst   |
|     |    Time     |     Time    |
  0             0               3
  1             0               1
  2             0               6
  3             0               4
========================================
```

**Q11) Write a program to implement SRJF algorithm.**

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

```
class SJF{
    private:
        int size;
        int* arrivalTime;
        int* burstTime;
        int* priority;
        int* completionTime;
        int* temp;
        void swap(int& n1,int& n2){
            int temp = n1;
            n1 = n2;
            n2 = temp;
        }
        bool isZero(){
            bool flag = true;
            for(int i=0;i<size;i++){
                if(temp[i]!=0){
                    flag = false;
                    return flag;
                }
            }
            return flag;
        }
        void sort(int* arr,int size){
            // cout<<"Hello"<<endl;
            for(int i=0;i<size-1;i++){
                for(int j=0;j<size-i-1;j++){
                    if(burstTime[arr[j]]>burstTime[arr[j+1]]){
                        swap(arr[j],arr[j+1]);
                    }
                }
            }
        }
        void pop(int* arr,int& size){
            for(int i=0;i<size-1;i++){
                arr[i] = arr[i+1];
            }
            size--;
        }
    public:
        SJF(int size){
            this->size = size;
            arrivalTime = new int[size];
            burstTime = new int[size];
            priority = new int[size];
            completionTime = new int[size];
            temp = new int[size];
        }
        ~SJF(){
```

```
    delete []arrivalTime;
    delete []burstTime;
    delete []priority;
    delete []completionTime;
}
void input(){
    for(int i=0;i<size;i++){
        cout<<"Enter the arrival time of process "<<(i+1)<<": ";
        cin>>arrivalTime[i];

        cout<<"Enter the burst time of process "<<(i+1)<<": ";
        cin>>burstTime[i];
        temp[i] = burstTime[i];

        cout<<"Enter the priority of process "<<(i+1)<<": ";
        cin>>priority[i];
    }
}
void display(){
    cout<<"Arrival Time:";
    for(int i=0;i<size;i++){
        cout<<arrivalTime[i]<<",";
    }
    cout<<endl;

    cout<<"Burst Time:";
    for(int i=0;i<size;i++){
        cout<<burstTime[i]<<",";
    }
    cout<<endl;

    cout<<"priority Time:";
    for(int i=0;i<size;i++){
        cout<<priority[i]<<",";
    }
    cout<<endl;

    cout<<"completion Time:";
    for(int i=0;i<size;i++){
        cout<<completionTime[i]<<",";
    }
    cout<<endl;
}
void CompletionTime(){
    int iter = 0; //points to next arrival
    int counter = 0; //gantt chart
    int* q = new int[size]; //ready queue
    int queue_iter = 0; //ready queue iterator
    for(int i=0;i<size-1;i++){
```

```
            for(int j =0;j<size-i-1;j++){
                if(arrivalTime[j]>arrivalTime[j+1]){
                    swap(arrivalTime[j],arrivalTime[j+1]);
                }
            }
        }
        do{
            if(counter<arrivalTime[iter]){
                counter++;
                continue;
            }
            else if(iter<=size-1){
                q[queue_iter] = iter;
                queue_iter++;
                iter++;
            }
            if(iter>size-1)
                iter = size;
            sort(q,queue_iter); //sort according to burst time
            int current = q[0];
            // cout<<current<<endl;
            while(counter<arrivalTime[iter]){
                counter++;
                temp[current]--;
                if(temp[current] == 0){
                    completionTime[current] = counter;
                    pop(q,queue_iter);
                    break;
                }
            }

            if(counter>=arrivalTime[iter] && (temp[current]!=0) && iter>=size-1){
                counter++;
                temp[current]--;
                if(temp[current] == 0){
                    completionTime[current] = counter;
                    pop(q,queue_iter);
                }
            }
        }while(!isZero());
    }
};
int main(){
    int size = 5;
    SJF obj(size);
    obj.input();
    obj.CompletionTime();
    obj.display();
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <pthread.h>

int sum;
void *run(void *param);

int main(int argc,char *argv[]){
pthread_t tid;
pthread_attr_t attr;
if(argc != 2){
printf("Error !");
return 1;
}
if(atoi(argv[1])<0){
printf("no. should be +ive");
return 1;
}
pthread_attr_init(&attr);
pthread_create(&tid,&attr,run,argv[1]);
pthread_join(tid,NULL);
printf("Sum = %d \n",sum);
}
void* run(void *param){
int i,upper;sum = 0;
upper = atoi((char* ) param);
if(upper>0){
for(i = 1;i <= upper;++i)
sum+=i;
}
pthread_exit(0);
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc -pthread  q12.cpp -o q12.o
ut && ./q12.out 5
Sum = 15
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc -pthread  q12.cpp -o q12.o
ut && ./q12.out 3
Sum = 6
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ gcc -pthread  q12.cpp -o q12.o
ut && ./q12.out 10
Sum = 55
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$
```

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

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

class Node{
private:
int partition=0;
string allocated;
public:
int size;
void print(){
cout<<"Partition is: "<<partition<<endl;
cout<<allocated<<endl;
}
friend class Allocation;
};
class Allocation{
private:
int* arr1;
int size;
public:
Allocation(int size){
this->size = size;
arr1 = new int[size];
}
void input(){
for(int i=0;i<size;i++){
cout<<"Enter the size of partition "<<(i+1)<<": ";
cin>>arr1[i];
}
}
```

```
bool best_fit(Node* proc){
bool flag =false;
int min = INT_MAX;
int partition;
for(int i=0;i<size;i++){
if(arr1[i]>=proc->size){
flag = true;
int temp = arr1[i] - proc->size;
if(temp<min){
min = temp;
proc->partition = i;
}
}
}
if(proc->partition==0 && !flag){
proc->allocated = "unallocated";
}
else{
arr1[proc->partition] = min;
proc->allocated = "allocated";
}
return flag;
}
bool worst_fit(Node* proc){
bool flag=true;
int max = INT_MIN;
for(int i=0;i<size;i++){
if(arr1[i]>=proc->size && arr1[i]>=max){
max = arr1[i];
proc->partition = i;
}
}
if(proc->partition==0){
flag = false;
}
arr1[proc->partition] = max;
return flag;
}
bool first_fit(Node* proc){
bool flag = true;
for(int i=0;i<size;i++){
if(arr1[i] >= proc->size){
arr1[i] = arr1[i]-proc->size;
proc->partition = i;
break;
}
}
```

```
if(proc->partition==0)
flag = false;
return flag;
}

};
void display(Node** arr,int size){
for(int i=0;i<size;i++){
arr[i]->print();
}
}
int main(){
int p;
cout<<"Enter the number of process: ";
cin>>p;

Allocation obj(p);
obj.input();

Node** arr = new Node*[p];
for(int i=0;i<p;i++){
Node* proc = new Node;
cout<<"Enter the size of the process: ";
cin>>proc->size;

if(obj.best_fit(proc)){
arr[i] = proc;
proc->print();
}
else{
proc->print();
}
}
return 0;
}
```

```
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$ g++ q13.cpp -o q13.out && ./q1
3.out
Enter the number of process: 3
Enter the size of partition 1: 1000
Enter the size of partition 2: 300
Enter the size of partition 3: 500
Enter the size of the process: 200
Partition is: 1
allocated
Enter the size of the process: 400
Partition is: 2
allocated
Enter the size of the process: 300
Partition is: 0
allocated
shiv42@shiv42-Inspiron-5570:~/Documents/College Programs/Sem 3/OS$
```