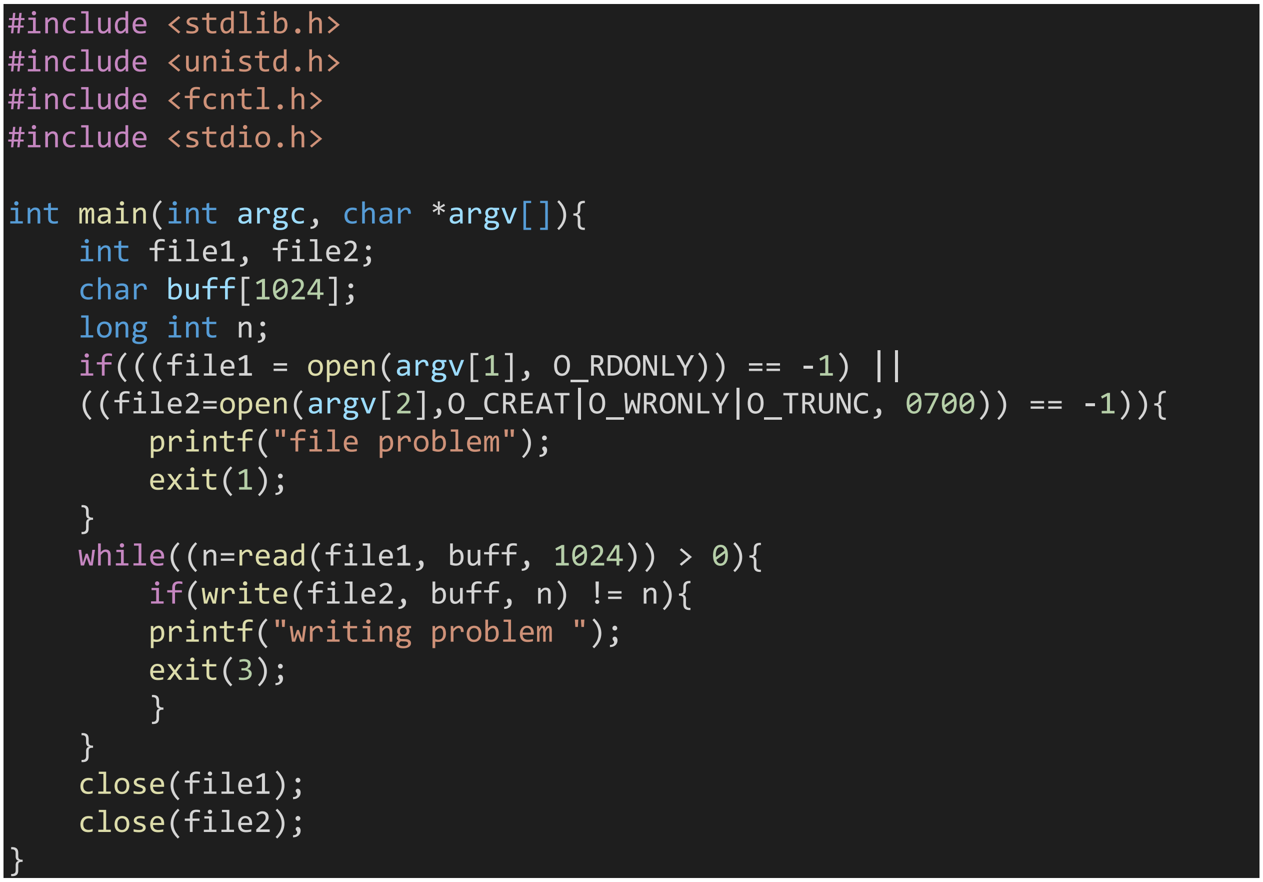
Sidharth

2K18/MC/114

Experiment 1

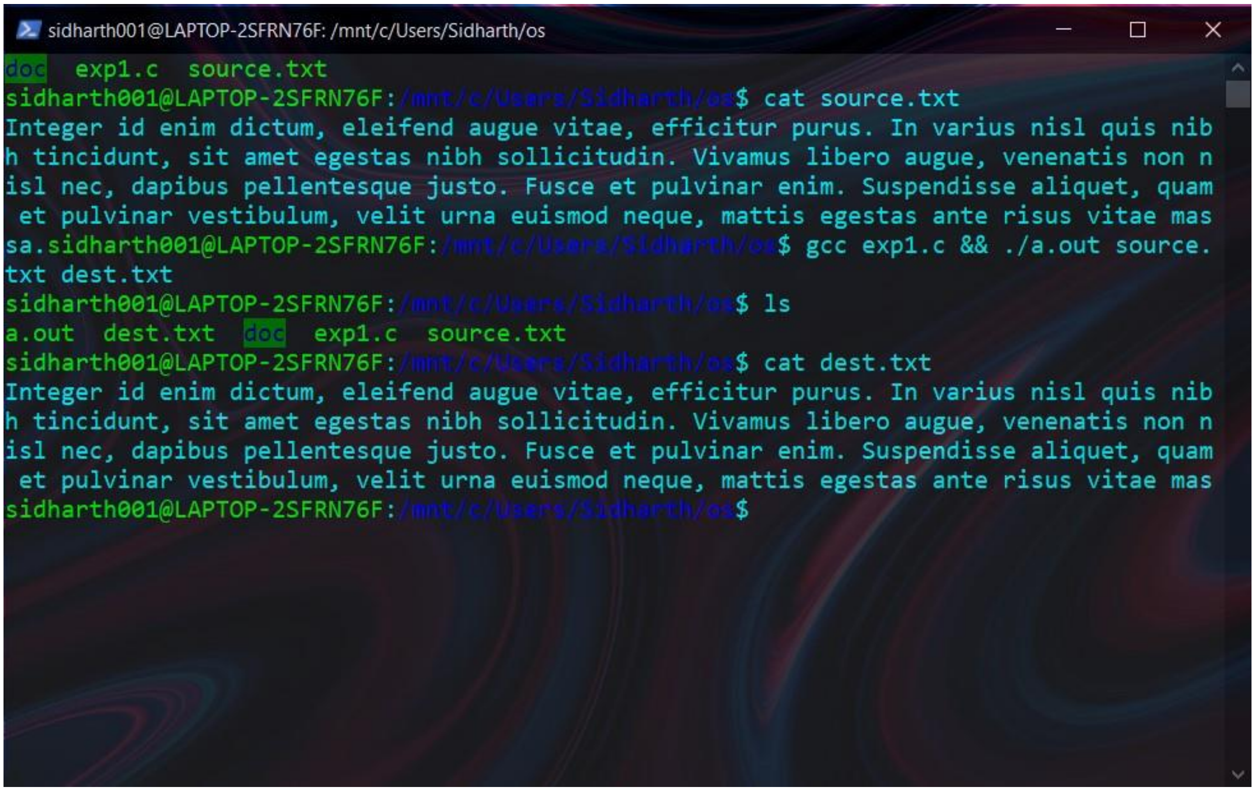
Aim: To Write a C program to read a file and copy the contents into a new file using system calls.

Code:



```c
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char *argv[]){
    int file1, file2;
    char buff[1024];
    long int n;
    if(((file1 = open(argv[1], O_RDONLY)) == -1) ||
    ((file2=open(argv[2],O_CREAT|O_WRONLY|O_TRUNC, 0700)) == -1)){
        printf("file problem");
        exit(1);
    }
    while((n=read(file1, buff, 1024)) > 0){
        if(write(file2, buff, n) != n){
        printf("writing problem ");
        exit(3);
        }
    }
    close(file1);
    close(file2);
}
```

Output:



```
doc  exp1.c  source.txt
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ cat source.txt
Integer id enim dictum, eleifend augue vitae, efficitur purus. In varius nisl quis nib
h tincidunt, sit amet egestas nibh sollicitudin. Vivamus libero augue, venenatis non n
isl nec, dapibus pellentesque justo. Fusce et pulvinar enim. Suspendisse aliquet, quam
 et pulvinar vestibulum, velit urna euismod neque, mattis egestas ante risus vitae mas
sa.sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ gcc exp1.c && ./a.out source.
txt dest.txt
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ ls
a.out  dest.txt  doc  exp1.c  source.txt
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ cat dest.txt
Integer id enim dictum, eleifend augue vitae, efficitur purus. In varius nisl quis nib
h tincidunt, sit amet egestas nibh sollicitudin. Vivamus libero augue, venenatis non n
isl nec, dapibus pellentesque justo. Fusce et pulvinar enim. Suspendisse aliquet, quam
 et pulvinar vestibulum, velit urna euismod neque, mattis egestas ante risus vitae mas
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```

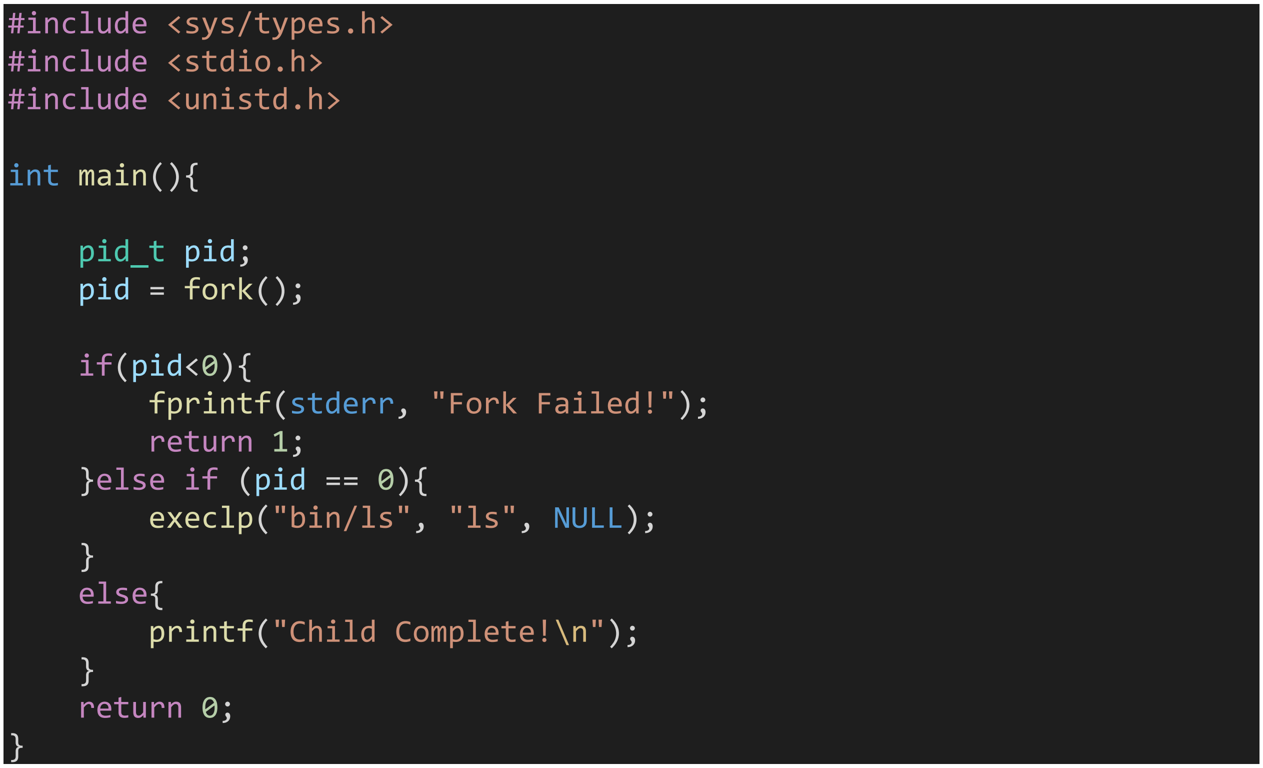
Sidharth

2K18/MC/114

Experiment 2

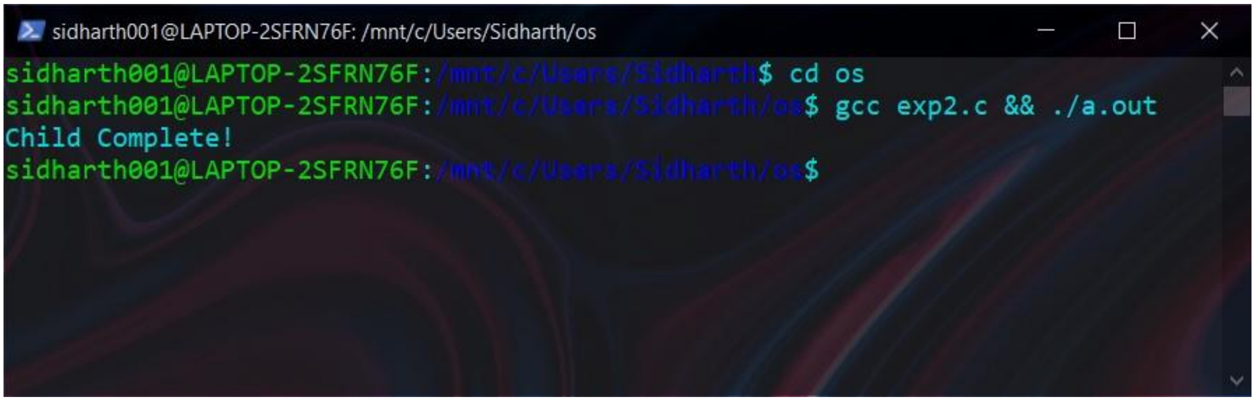
Aim: Process creation and termination using fork(), exit() and wait() system calls.

Code:



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

int main(){

    pid_t pid;
    pid = fork();

    if(pid<0){
        fprintf(stderr, "Fork Failed!");
        return 1;
    }else if (pid == 0){
        execlp("bin/ls", "ls", NULL);
    }
    else{
        printf("Child Complete!\n");
    }
    return 0;
}
```

Output:



```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ gcc exp2.c && ./a.out
Child Complete!
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```

Sidharth

2K18/MC/114

Experiment 3

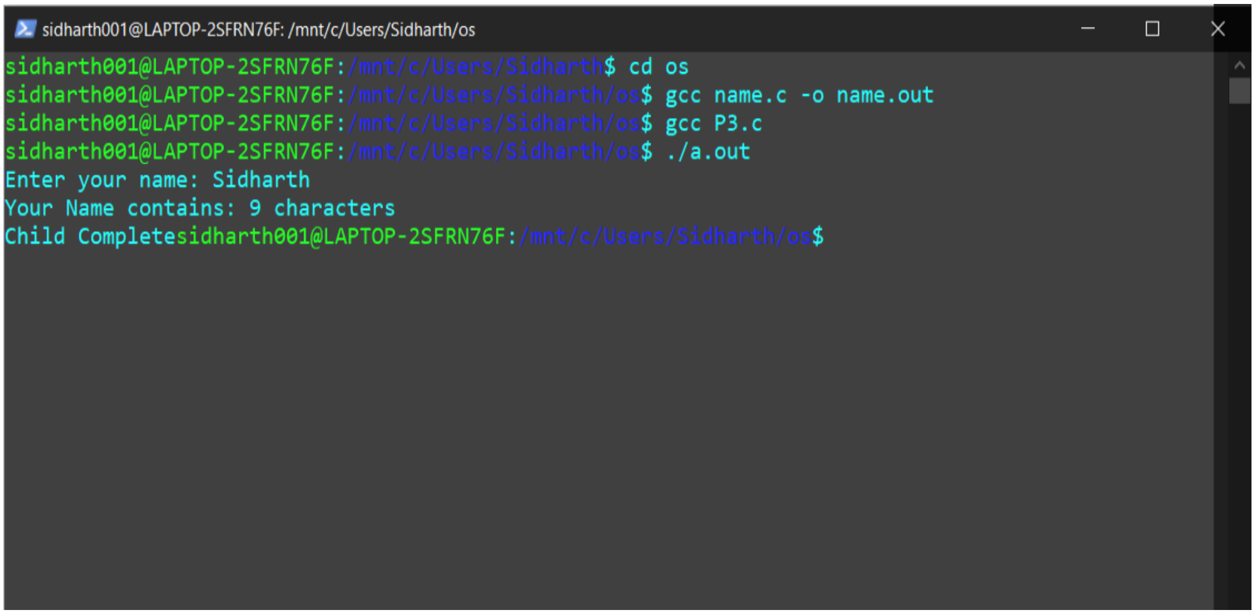
Aim: Write a C program that creates a new child process. The child process should be assigned to do the task of finding the length of your name.

Code:

```
// Program to find length of the string  "name.c"

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>


int  main(){

   char  name[30];

   printf("Enter your name: ");

   fgets(name, 30, stdin);

   int count = 0;

   for(int  i=0; name[i] != '\0'; i++)  count++;

   printf("Your Name contains: %d characters\n", count);
```

```
    return  0;
}

// Program for Child Process "P3.c"
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(){
    pid_t pid;
    pid = fork();
    if (pid < 0){
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0){

execlp("/mnt/c/Users/Sidharth/os/name.out","./mnt/c/Users/Sidharth/os/name.out",NULL);
    }
    else{
        wait(NULL);
        printf("Child Complete");
    }
    return 0;
}
```

Output:



```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ gcc name.c -o name.out
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ gcc P3.c
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ ./a.out
Enter your name: Sidharth
Your Name contains: 9 characters
Child Completesidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```

Sidharth

2K18/MC/114

Experiment 4

Aim: Write a program to implement following Disk Scheduling Algorithms.

Algorithm:

FCFS

Elevator's Algorithm

Input - Head Start Position - 101, Total number of cylinders = 300

1. Queue 1 - 22, 43, 55, 21, 220, 2, 33, 55, 221, 231,121, 157, 189
2. Queue 2 - 31, 157, 242, 134, 79, 145, 178, 244, 267, 288

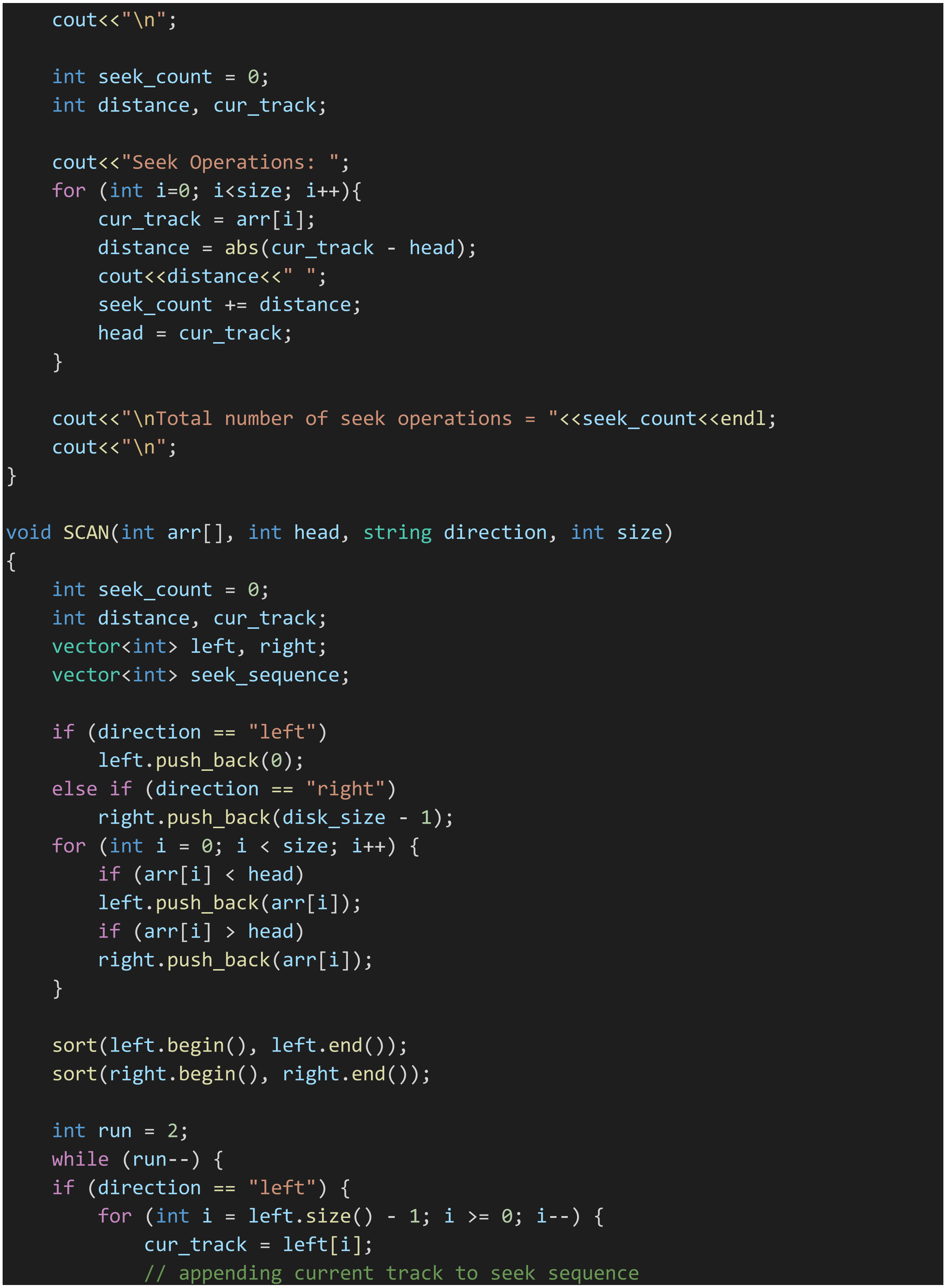
Generate the output as

Number of cylinders traversed for FCFS and Elevators algorithm for both the input queues

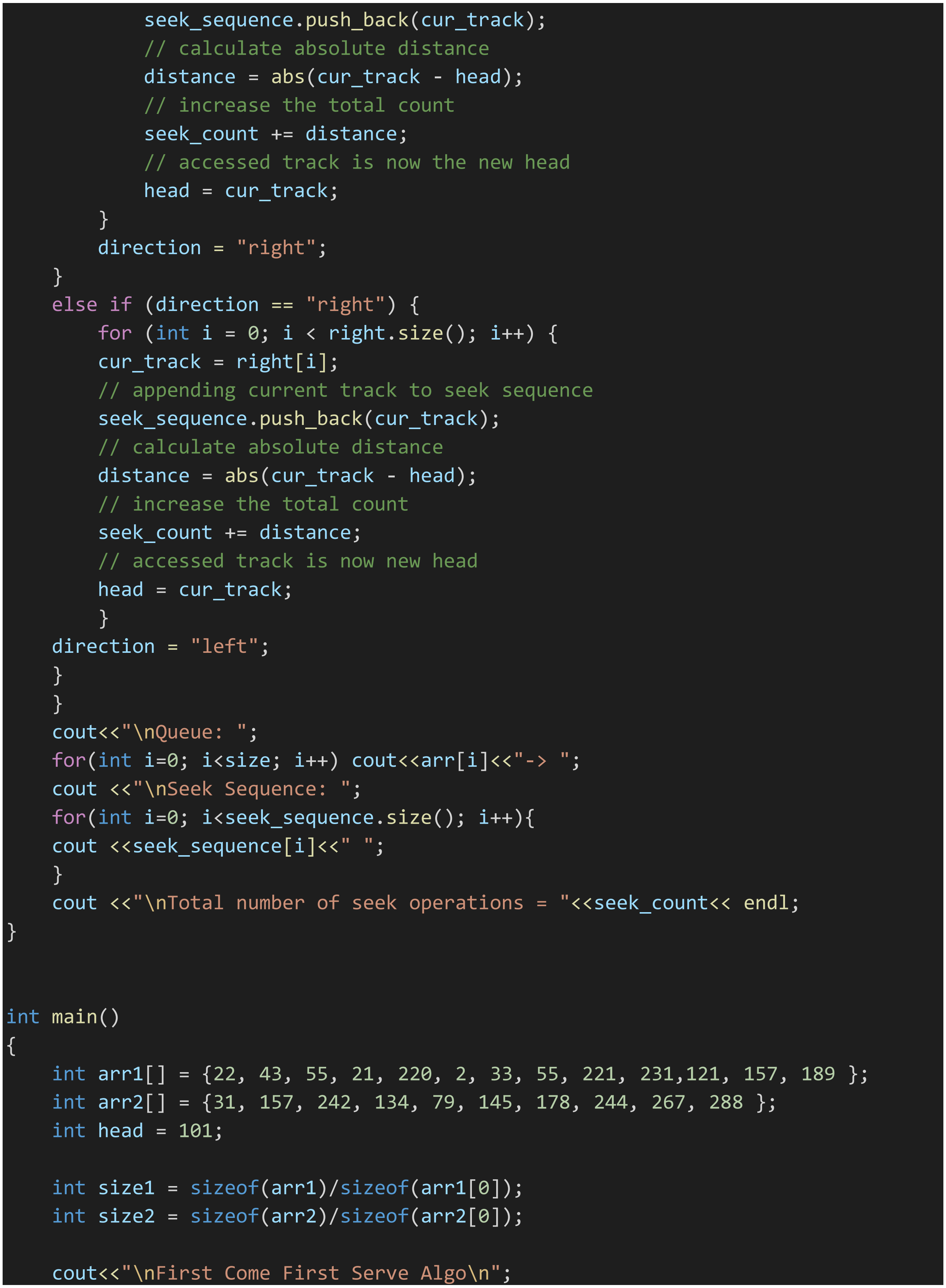
The order of cylinders traversed for FCFS and Elevators algorithm for both the input queues

Code:

```
#include <bits/stdc++.h>
using namespace std;

int disk_size = 300;

void FCFS(int arr[], int head, int size)
{
    cout<<"Queue: ";
    for(int i=0; i<size; i++){
        cout<<arr[i]<<"-> ";
    }
```

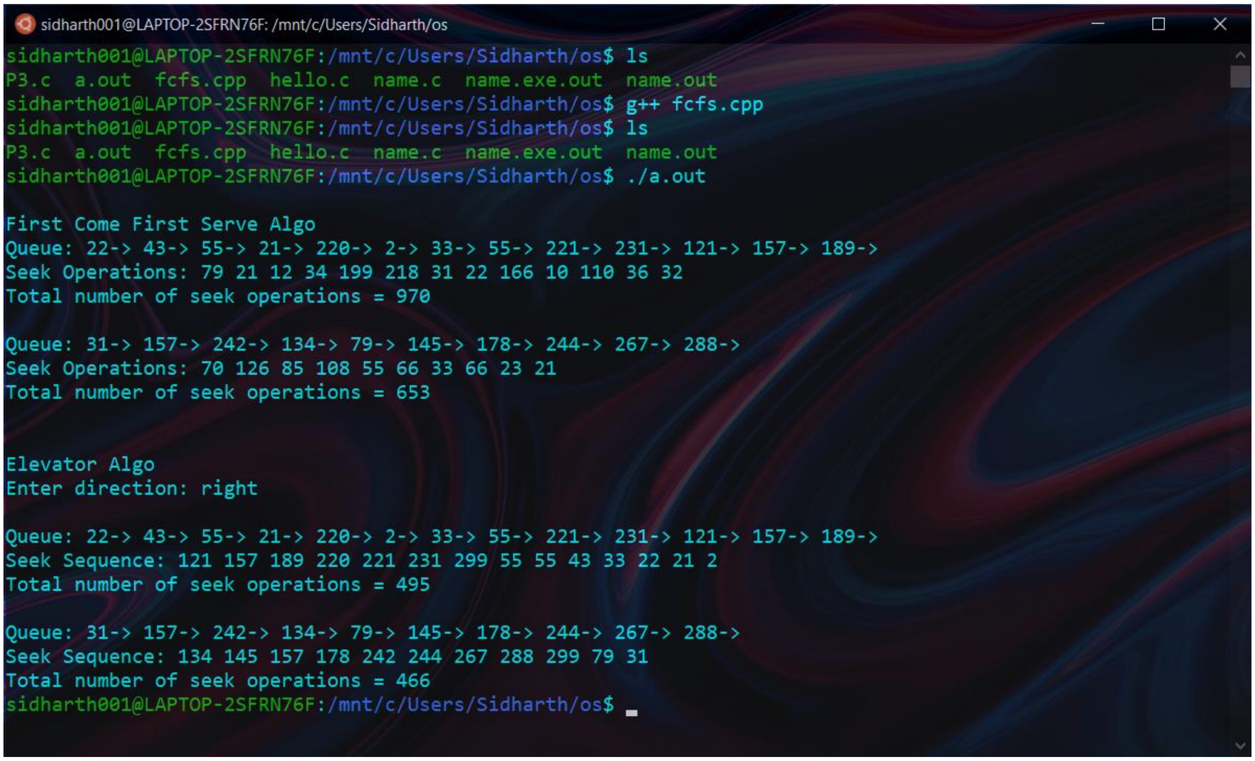


```
    cout<<"\n";

    int seek_count = 0;
    int distance, cur_track;

    cout<<"Seek Operations: ";
    for (int i=0; i<size; i++){
        cur_track = arr[i];
        distance = abs(cur_track - head);
        cout<<distance<<" ";
        seek_count += distance;
        head = cur_track;
    }

    cout<<"\nTotal number of seek operations = "<<seek_count<<endl;
    cout<<"\n";
}

void SCAN(int arr[], int head, string direction, int size)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;

    if (direction == "left")
        left.push_back(0);
    else if (direction == "right")
        right.push_back(disk_size - 1);
    for (int i = 0; i < size; i++) {
        if (arr[i] < head)
        left.push_back(arr[i]);
        if (arr[i] > head)
        right.push_back(arr[i]);
    }

    sort(left.begin(), left.end());
    sort(right.begin(), right.end());

    int run = 2;
    while (run--) {
    if (direction == "left") {
        for (int i = left.size() - 1; i >= 0; i--) {
            cur_track = left[i];
            // appending current track to seek sequence
```



```
            seek_sequence.push_back(cur_track);
            // calculate absolute distance
            distance = abs(cur_track - head);
            // increase the total count
            seek_count += distance;
            // accessed track is now the new head
            head = cur_track;
        }
        direction = "right";
    }
    else if (direction == "right") {
        for (int i = 0; i < right.size(); i++) {
        cur_track = right[i];
        // appending current track to seek sequence
        seek_sequence.push_back(cur_track);
        // calculate absolute distance
        distance = abs(cur_track - head);
        // increase the total count
        seek_count += distance;
        // accessed track is now new head
        head = cur_track;
        }
    direction = "left";
    }
    }
    cout<<"\nQueue: ";
    for(int i=0; i<size; i++) cout<<arr[i]<<"-> ";
    cout <<"\nSeek Sequence: ";
    for(int i=0; i<seek_sequence.size(); i++){
    cout <<seek_sequence[i]<<" ";
    }
    cout <<"\nTotal number of seek operations = "<<seek_count<< endl;
}


int main()
{
    int arr1[] = {22, 43, 55, 21, 220, 2, 33, 55, 221, 231,121, 157, 189 };
    int arr2[] = {31, 157, 242, 134, 79, 145, 178, 244, 267, 288 };
    int head = 101;

    int size1 = sizeof(arr1)/sizeof(arr1[0]);
    int size2 = sizeof(arr2)/sizeof(arr2[0]);

    cout<<"\nFirst Come First Serve Algo\n";
    FCFS(arr1, head, size1);
    FCFS(arr2, head, size2);

    string direction;
    cout<<"\nElevator Algo\nEnter direction: ";
    cin>>direction;

    SCAN(arr1, head, direction, size1);
    SCAN(arr2, head, direction, size2);

    return 0;
}
```

Output:



sidharth001@LAPTOP-2SFRN76F: /mnt/c/Users/Sidharth/os

```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ ls
P3.c  a.out  fcfs.cpp  hello.c  name.c  name.exe.out  name.out
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ g++ fcfs.cpp
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ ls
P3.c  a.out  fcfs.cpp  hello.c  name.c  name.exe.out  name.out
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ ./a.out

First Come First Serve Algo
Queue: 22-> 43-> 55-> 21-> 220-> 2-> 33-> 55-> 221-> 231-> 121-> 157-> 189->
Seek Operations: 79 21 12 34 199 218 31 22 166 10 110 36 32
Total number of seek operations = 970

Queue: 31-> 157-> 242-> 134-> 79-> 145-> 178-> 244-> 267-> 288->
Seek Operations: 70 126 85 108 55 66 33 66 23 21
Total number of seek operations = 653


Elevator Algo
Enter direction: right

Queue: 22-> 43-> 55-> 21-> 220-> 2-> 33-> 55-> 221-> 231-> 121-> 157-> 189->
Seek Sequence: 121 157 189 220 221 231 299 55 55 43 33 22 21 2
Total number of seek operations = 495

Queue: 31-> 157-> 242-> 134-> 79-> 145-> 178-> 244-> 267-> 288->
Seek Sequence: 134 145 157 178 242 244 267 288 299 79 31
Total number of seek operations = 466
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ _
```

Sidharth

2K18/MC/114

Experiment 5

Aim: Implement FCFS and Non preemptive Shortest Job first algorithm.

Input: Sequence of Processes with their arrival time and burst time

Output: Sequence of processes executing Average Waiting time and Average Turnaround time

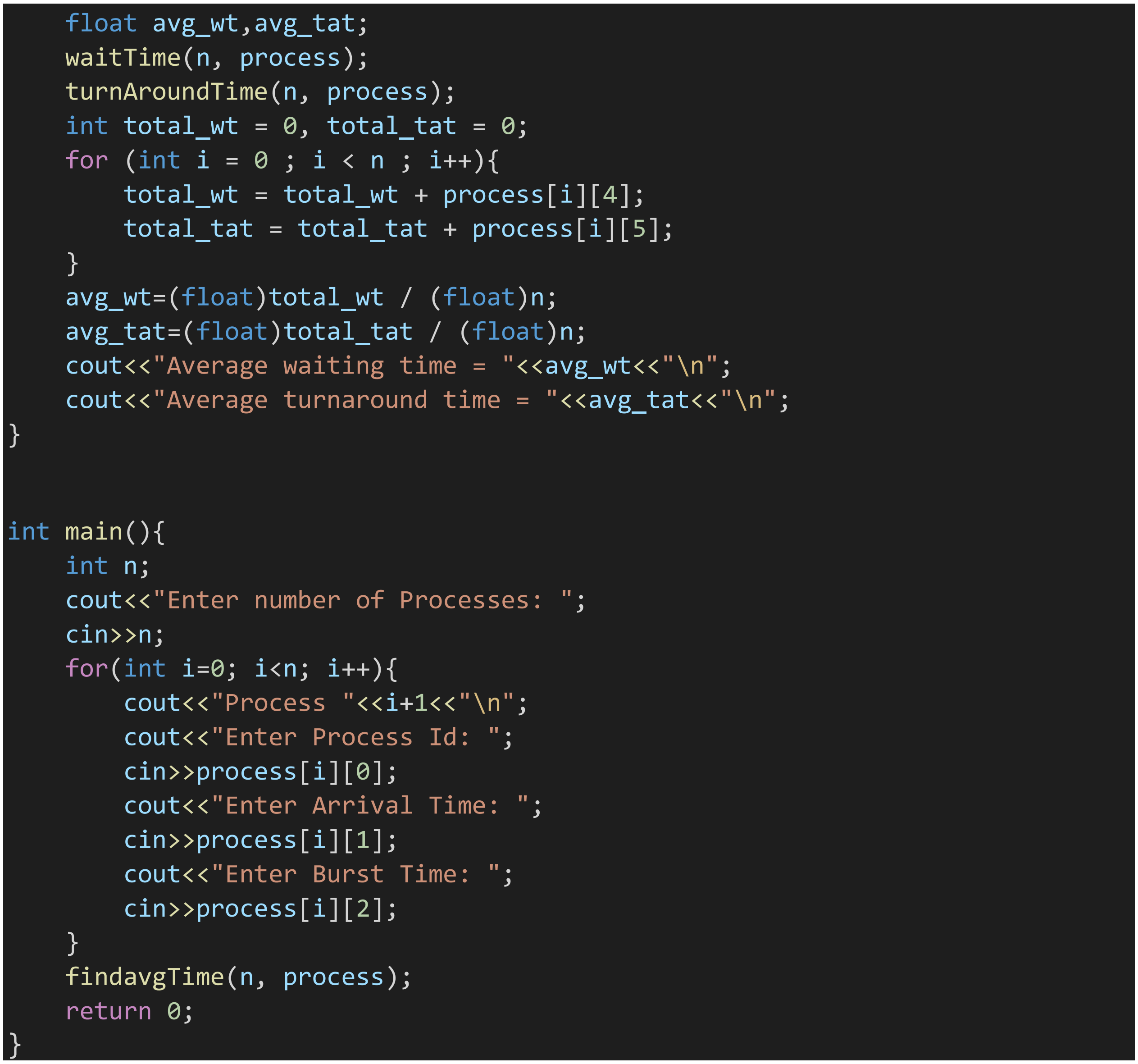
Example:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

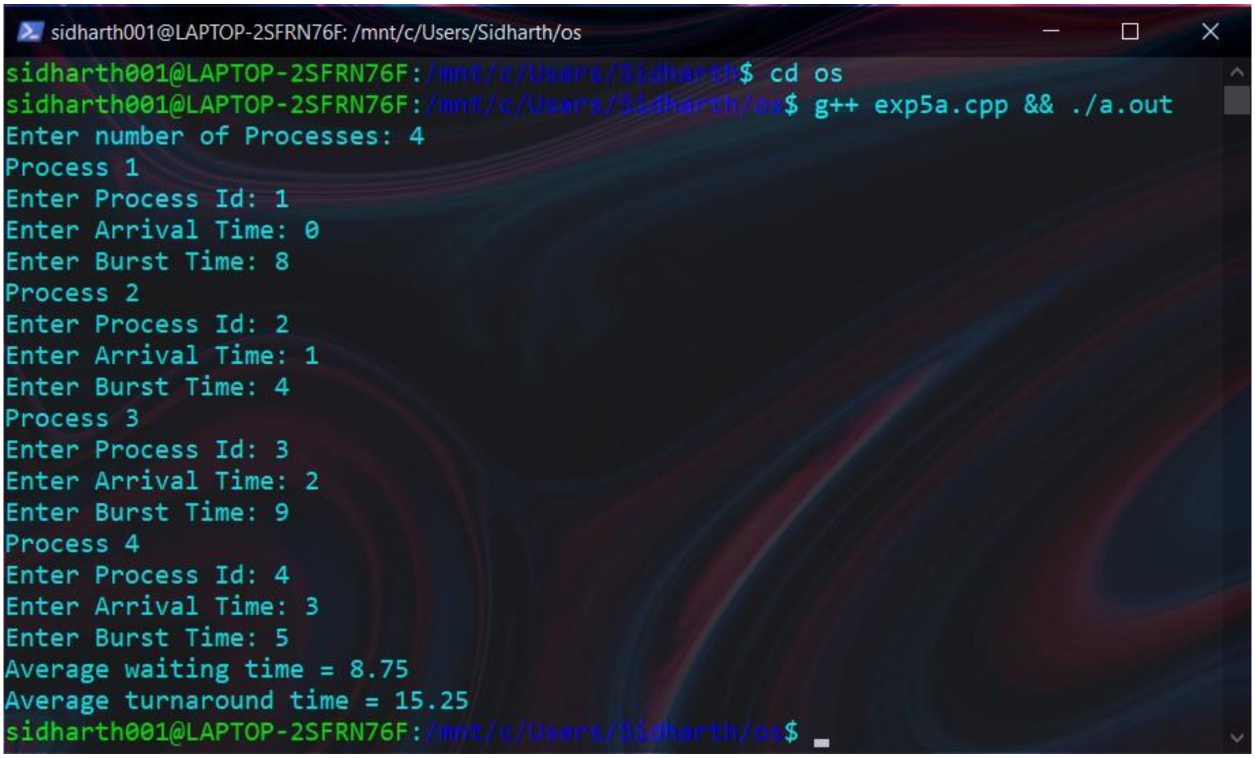
Code:

FCFS:

```
#include<iostream>
using namespace std;
int process[20][6];

void waitTime(int n, int process[][6]){
    for (int i = 1; i < n ; i++){
        process[i][3] = process[i-1][3] + process[i-1][2];
        process[i][4] = process[i][3] - process[i][1];
        if (process[i][4] < 0)  process[i][4] = 0;
    }
}

void turnAroundTime(int n, int process[][6]){
    for (int i=0; i<n; i++)  process[i][5] = process[i][2] + process[i][4];
}

void findavgTime(int n, int process[][6]){
```

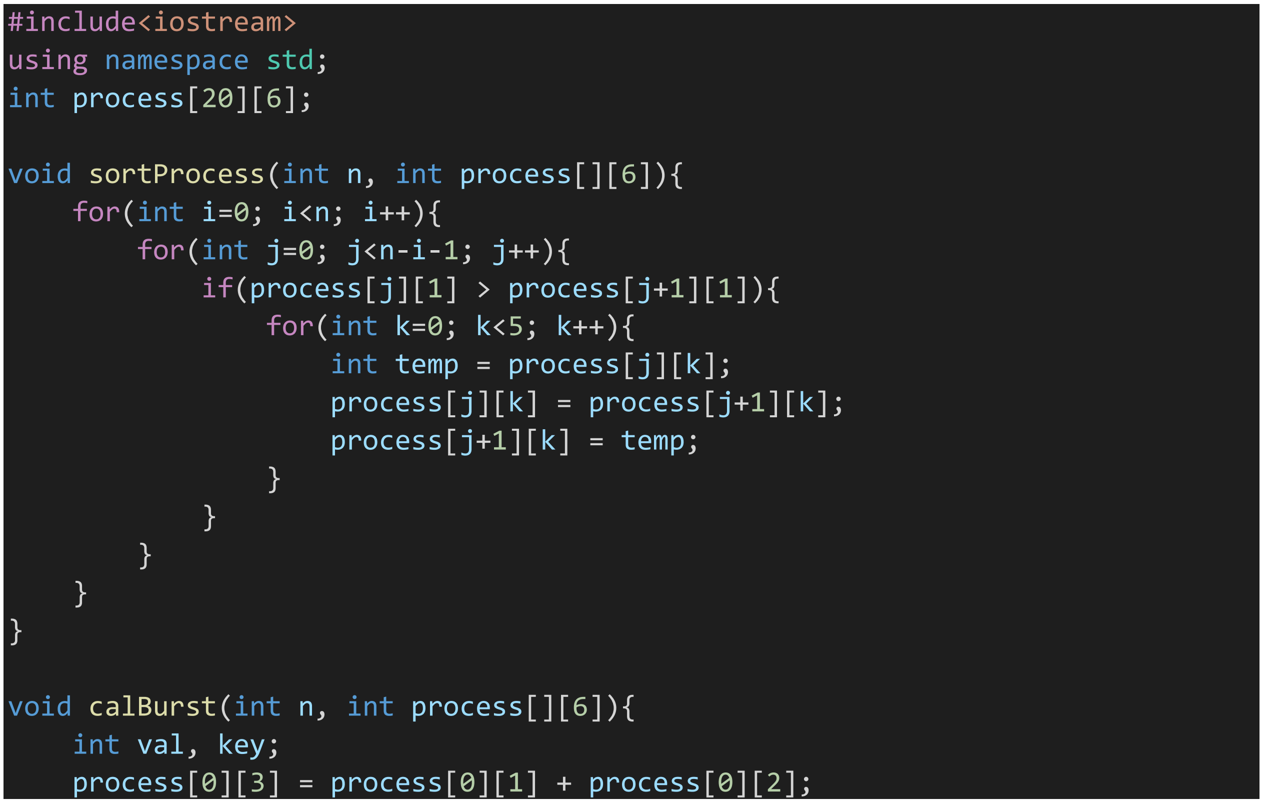


```
    float avg_wt,avg_tat;
    waitTime(n, process);
    turnAroundTime(n, process);
    int total_wt = 0, total_tat = 0;
    for (int i = 0 ; i < n ; i++){
        total_wt = total_wt + process[i][4];
        total_tat = total_tat + process[i][5];
    }
    avg_wt=(float)total_wt / (float)n;
    avg_tat=(float)total_tat / (float)n;
    cout<<"Average waiting time = "<<avg_wt<<"\n";
    cout<<"Average turnaround time = "<<avg_tat<<"\n";
}


int main(){
    int n;
    cout<<"Enter number of Processes: ";
    cin>>n;
    for(int i=0; i<n; i++){
        cout<<"Process "<<i+1<<"\n";
        cout<<"Enter Process Id: ";
        cin>>process[i][0];
        cout<<"Enter Arrival Time: ";
        cin>>process[i][1];
        cout<<"Enter Burst Time: ";
        cin>>process[i][2];
    }
    findavgTime(n, process);
    return 0;
}
```

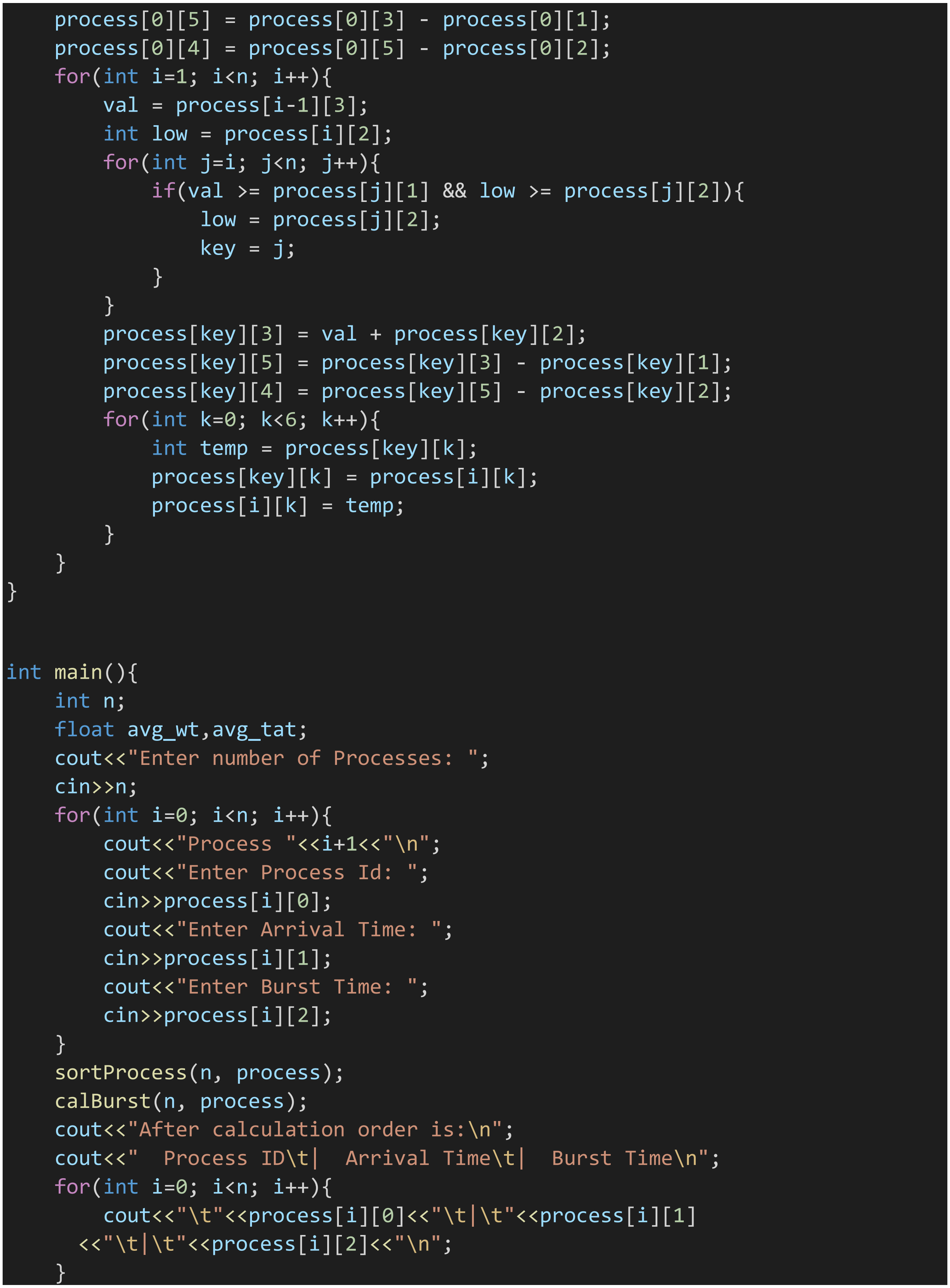


```
sidharth001@LAPTOP-2SFRN76F: /mnt/c/Users/Sidharth/os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ g++ exp5a.cpp && ./a.out
Enter number of Processes: 4
Process 1
Enter Process Id: 1
Enter Arrival Time: 0
Enter Burst Time: 8
Process 2
Enter Process Id: 2
Enter Arrival Time: 1
Enter Burst Time: 4
Process 3
Enter Process Id: 3
Enter Arrival Time: 2
Enter Burst Time: 9
Process 4
Enter Process Id: 4
Enter Arrival Time: 3
Enter Burst Time: 5
Average waiting time = 8.75
Average turnaround time = 15.25
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ _
```

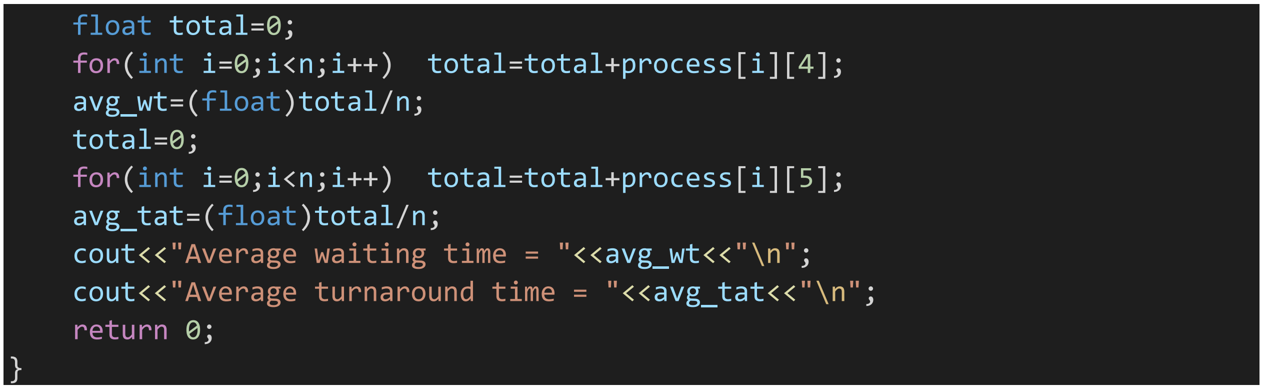
Non preemptive SJF:



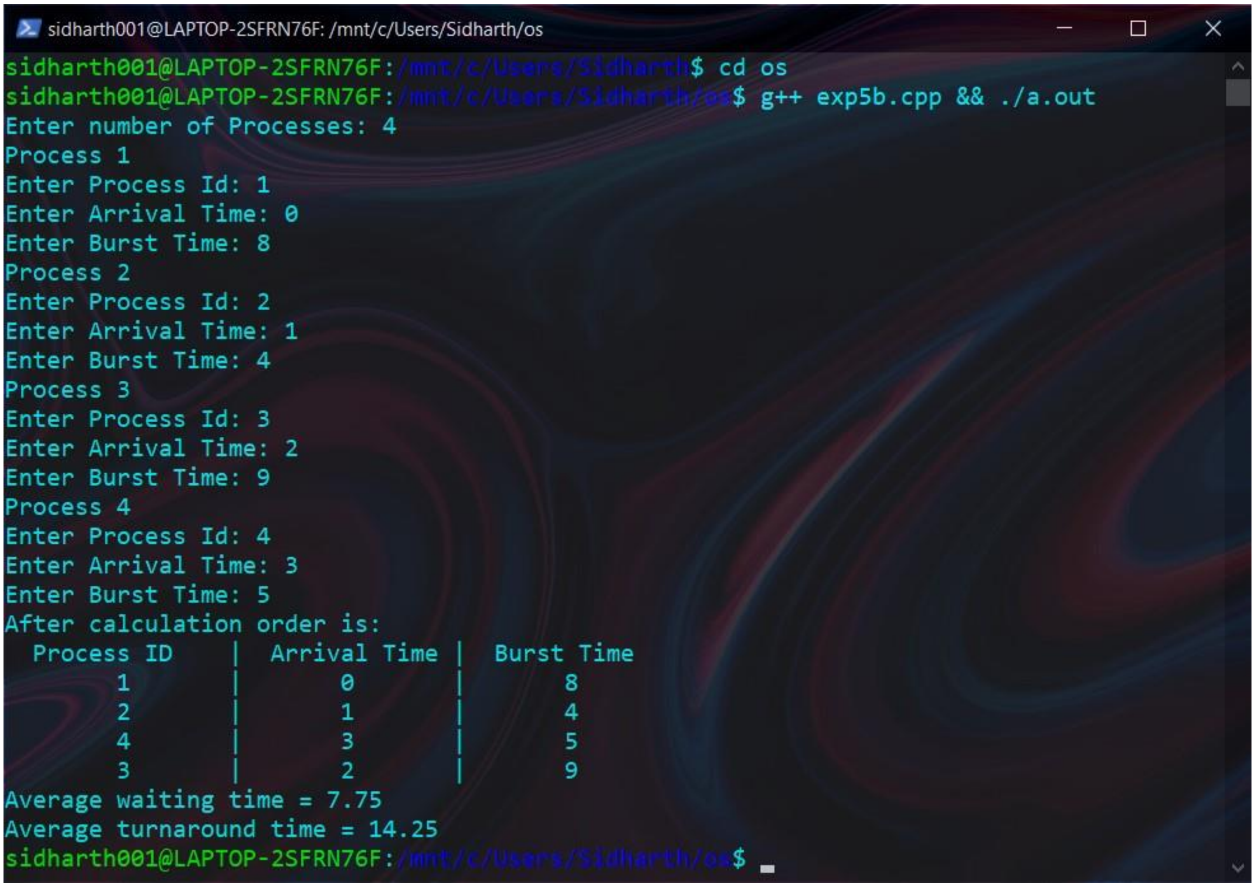
```
#include<iostream>
using namespace std;
int process[20][6];

void sortProcess(int n, int process[][6]){
    for(int i=0; i<n; i++){
        for(int j=0; j<n-i-1; j++){
            if(process[j][1] > process[j+1][1]){
                for(int k=0; k<5; k++){
                    int temp = process[j][k];
                    process[j][k] = process[j+1][k];
                    process[j+1][k] = temp;
                }
            }
        }
    }
}

void calBurst(int n, int process[][6]){
    int val, key;
    process[0][3] = process[0][1] + process[0][2];
```



```
    process[0][5] = process[0][3] - process[0][1];
    process[0][4] = process[0][5] - process[0][2];
    for(int i=1; i<n; i++){
        val = process[i-1][3];
        int low = process[i][2];
        for(int j=i; j<n; j++){
            if(val >= process[j][1] && low >= process[j][2]){
                low = process[j][2];
                key = j;
            }
        }
        process[key][3] = val + process[key][2];
        process[key][5] = process[key][3] - process[key][1];
        process[key][4] = process[key][5] - process[key][2];
        for(int k=0; k<6; k++){
            int temp = process[key][k];
            process[key][k] = process[i][k];
            process[i][k] = temp;
        }
    }
}


int main(){
    int n;
    float avg_wt,avg_tat;
    cout<<"Enter number of Processes: ";
    cin>>n;
    for(int i=0; i<n; i++){
        cout<<"Process "<<i+1<<"\n";
        cout<<"Enter Process Id: ";
        cin>>process[i][0];
        cout<<"Enter Arrival Time: ";
        cin>>process[i][1];
        cout<<"Enter Burst Time: ";
        cin>>process[i][2];
    }
    sortProcess(n, process);
    calBurst(n, process);
    cout<<"After calculation order is:\n";
    cout<<"  Process ID\t|  Arrival Time\t|  Burst Time\n";
    for(int i=0; i<n; i++){
        cout<<"\t"<<process[i][0]<<"\t|\t"<<process[i][1]
      <<"\t|\t"<<process[i][2]<<"\n";
    }
```



```
    float total=0;
    for(int i=0;i<n;i++)  total=total+process[i][4];
    avg_wt=(float)total/n;
    total=0;
    for(int i=0;i<n;i++)  total=total+process[i][5];
    avg_tat=(float)total/n;
    cout<<"Average waiting time = "<<avg_wt<<"\n";
    cout<<"Average turnaround time = "<<avg_tat<<"\n";
    return 0;
}
```



sidharth001@LAPTOP-2SFRN76F: /mnt/c/Users/Sidharth/os

```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ g++ exp5b.cpp && ./a.out
Enter number of Processes: 4
Process 1
Enter Process Id: 1
Enter Arrival Time: 0
Enter Burst Time: 8
Process 2
Enter Process Id: 2
Enter Arrival Time: 1
Enter Burst Time: 4
Process 3
Enter Process Id: 3
Enter Arrival Time: 2
Enter Burst Time: 9
Process 4
Enter Process Id: 4
Enter Arrival Time: 3
Enter Burst Time: 5
After calculation order is:
  Process ID    |  Arrival Time |  Burst Time
        1       |       0       |      8
        2       |       1       |      4
        4       |       3       |      5
        3       |       2       |      9
Average waiting time = 7.75
Average turnaround time = 14.25
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```

Sidharth

2K18/MC/114

Experiment 6

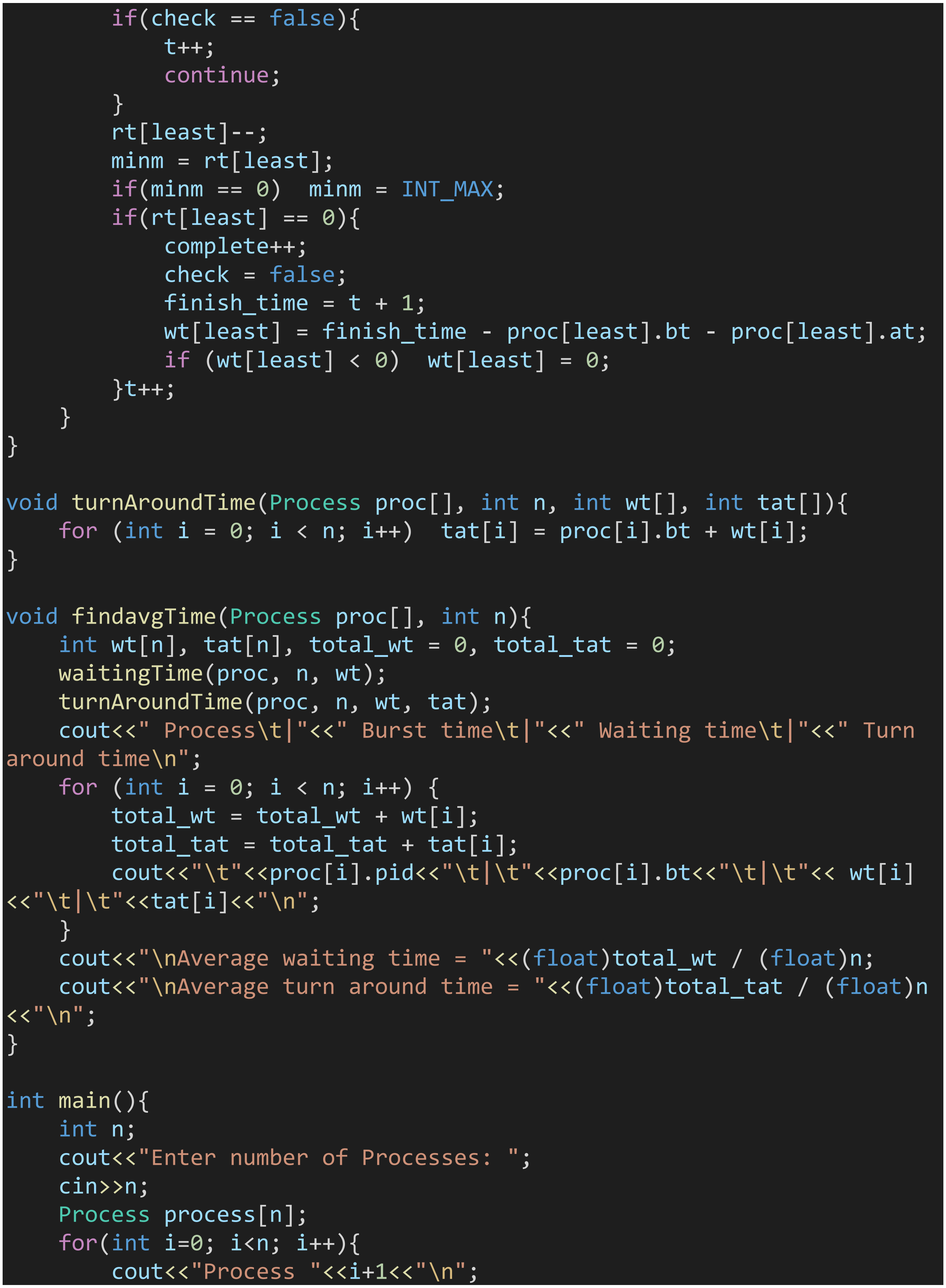
Aim: Write a program to implement Shortest Job First (Preemptive version).

Example:

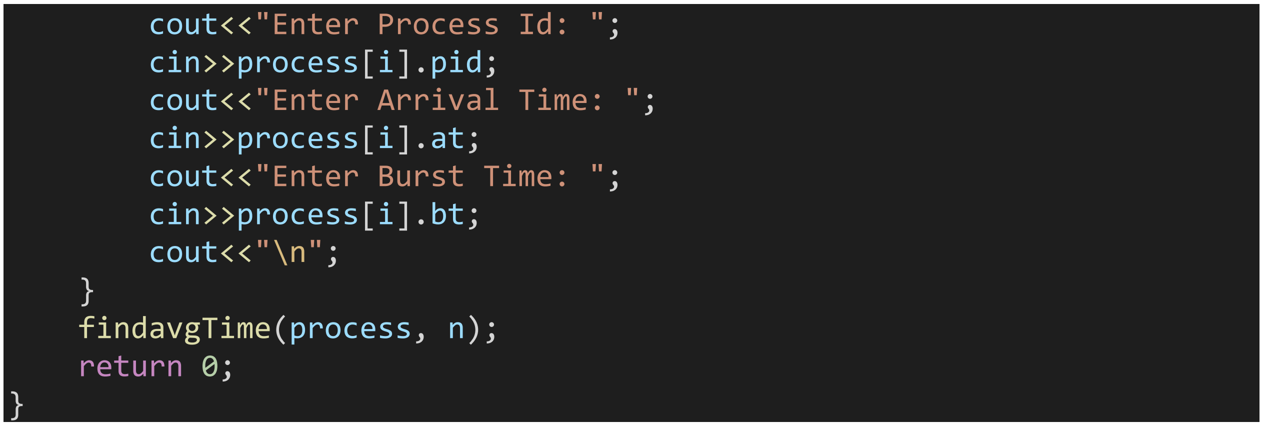
Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Code:

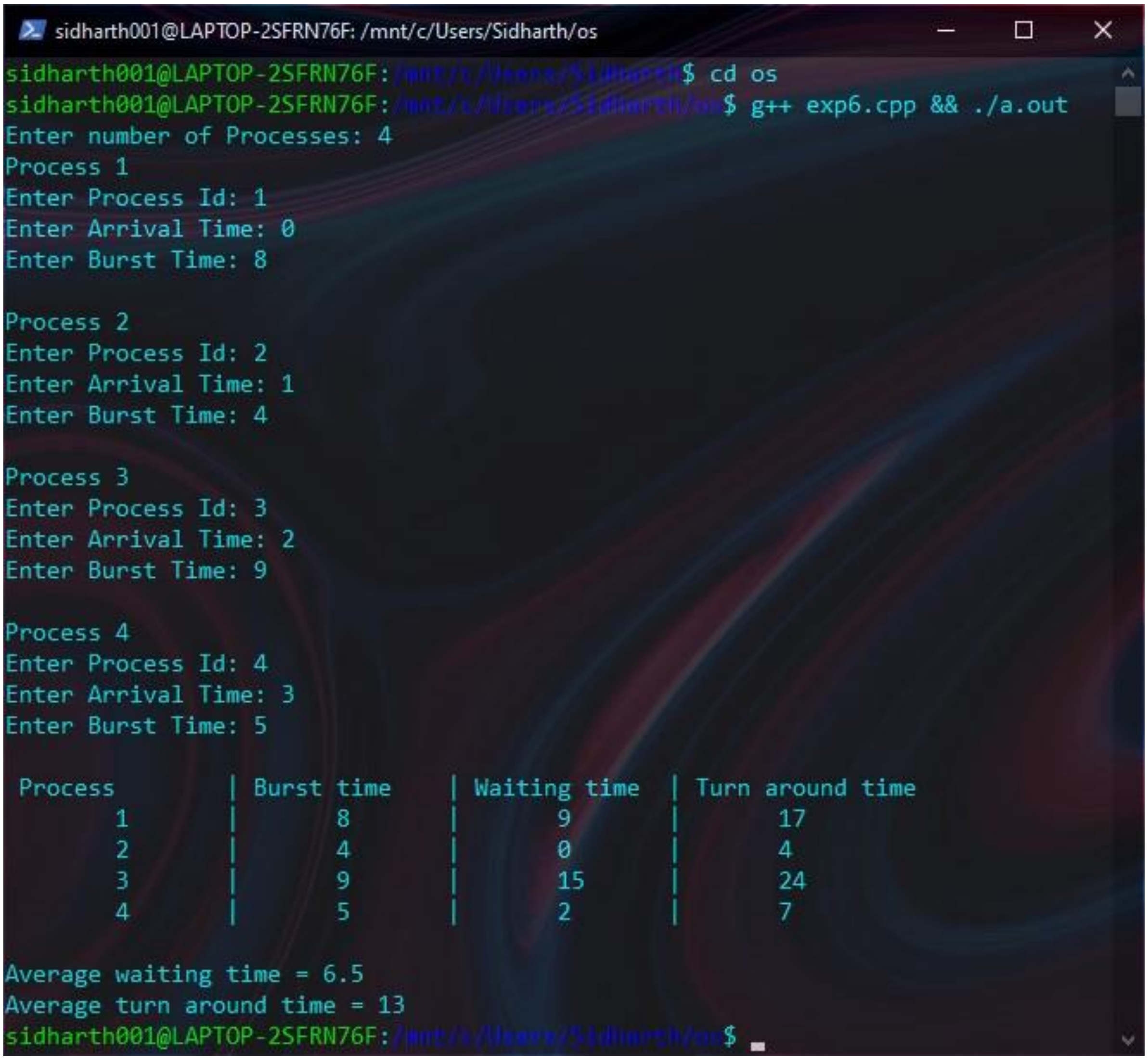
```
#include <bits/stdc++.h>
using namespace std;

struct Process{
    int pid;
    int bt;
    int at;
};

void waitingTime(Process proc[], int n, int wt[]){
    int rt[n];
    for (int i = 0; i < n; i++)  rt[i] = proc[i].bt;
    int complete = 0, t = 0, minm = INT_MAX;
    int least = 0, finish_time;
    bool check = false;
    while (complete != n){
        for(int j = 0; j < n; j++){
            if((proc[j].at <= t) && (rt[j] < minm) && rt[j] > 0){
                minm = rt[j];
                least = j;
                check = true;
            }
        }
```



```
        if(check == false){
            t++;
            continue;
        }
        rt[least]--;
        minm = rt[least];
        if(minm == 0)  minm = INT_MAX;
        if(rt[least] == 0){
            complete++;
            check = false;
            finish_time = t + 1;
            wt[least] = finish_time - proc[least].bt - proc[least].at;
            if (wt[least] < 0)  wt[least] = 0;
        }t++;
    }
}

void turnAroundTime(Process proc[], int n, int wt[], int tat[]){
    for (int i = 0; i < n; i++)  tat[i] = proc[i].bt + wt[i];
}

void findavgTime(Process proc[], int n){
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    waitingTime(proc, n, wt);
    turnAroundTime(proc, n, wt, tat);
    cout<<" Process\t|"<<" Burst time\t|"<<" Waiting time\t|"<<" Turn
around time\n";
    for (int i = 0; i < n; i++) {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout<<"\t"<<proc[i].pid<<"\t|\t"<<proc[i].bt<<"\t|\t"<< wt[i]
<<"\t|\t"<<tat[i]<<"\n";
    }
    cout<<"\nAverage waiting time = "<<(float)total_wt / (float)n;
    cout<<"\nAverage turn around time = "<<(float)total_tat / (float)n
<<"\n";
}

int main(){
    int n;
    cout<<"Enter number of Processes: ";
    cin>>n;
    Process process[n];
    for(int i=0; i<n; i++){
        cout<<"Process "<<i+1<<"\n";
```



```
        cout<<"Enter Process Id: ";
        cin>>process[i].pid;
        cout<<"Enter Arrival Time: ";
        cin>>process[i].at;
        cout<<"Enter Burst Time: ";
        cin>>process[i].bt;
        cout<<"\n";
    }
    findavgTime(process, n);
    return 0;
}
```



```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ g++ exp6.cpp && ./a.out
Enter number of Processes: 4
Process 1
Enter Process Id: 1
Enter Arrival Time: 0
Enter Burst Time: 8

Process 2
Enter Process Id: 2
Enter Arrival Time: 1
Enter Burst Time: 4

Process 3
Enter Process Id: 3
Enter Arrival Time: 2
Enter Burst Time: 9

Process 4
Enter Process Id: 4
Enter Arrival Time: 3
Enter Burst Time: 5

 Process        | Burst time     | Waiting time   | Turn around time
       1        |     8          |      9         |      17
       2        |     4          |      0         |      4
       3        |     9          |      15        |      24
       4        |     5          |      2         |      7

Average waiting time = 6.5
Average turn around time = 13
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```

Sidharth

2K18/MC/114

Experiment 7

Aim: Write a program to implement Round Robin Scheduling Algorithm.

The program accepts following 4 inputs –

1. Processes

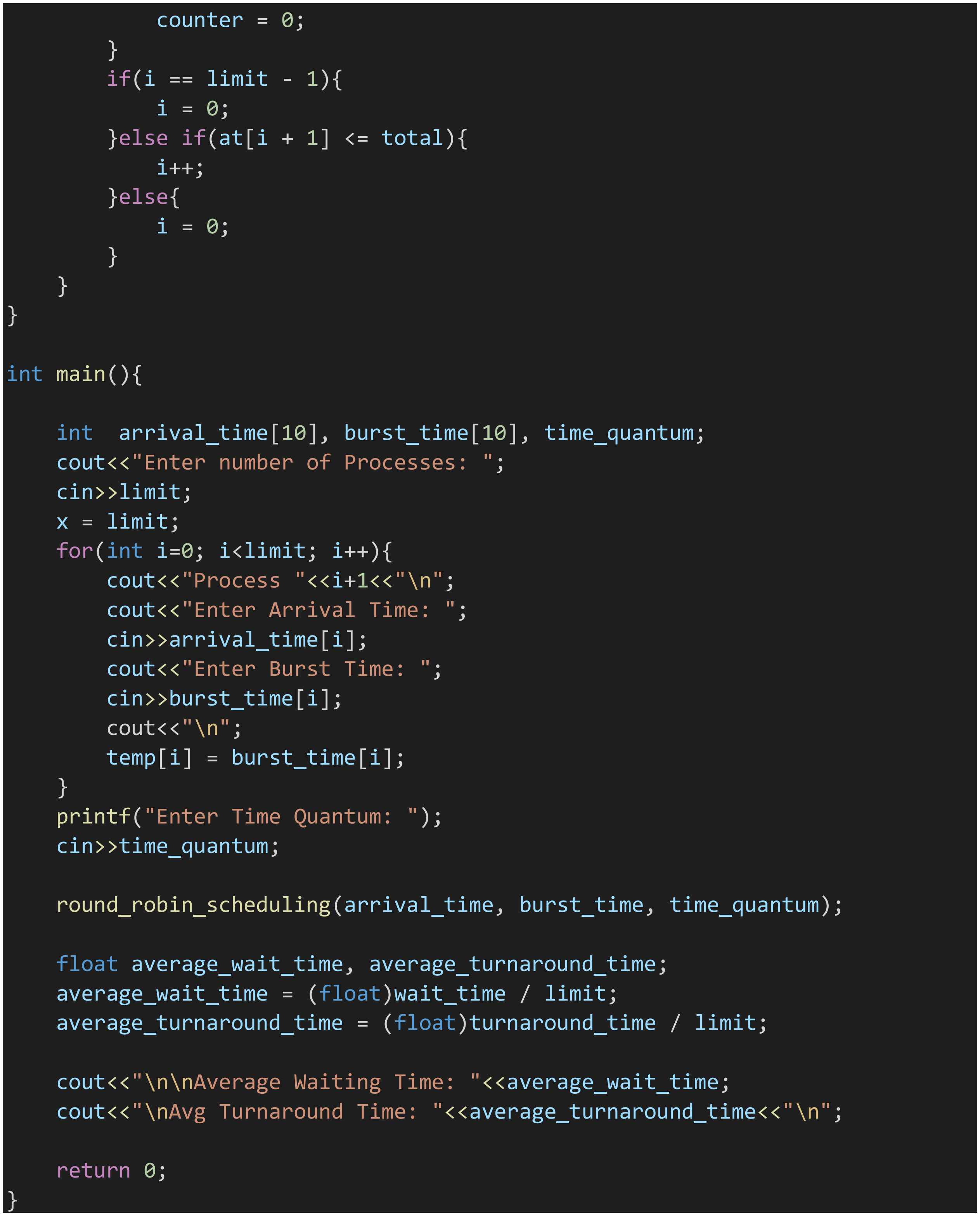
2. Corresponding Arrival Time

3. Corresponding Burst Time

4. Maximum Time Quantum

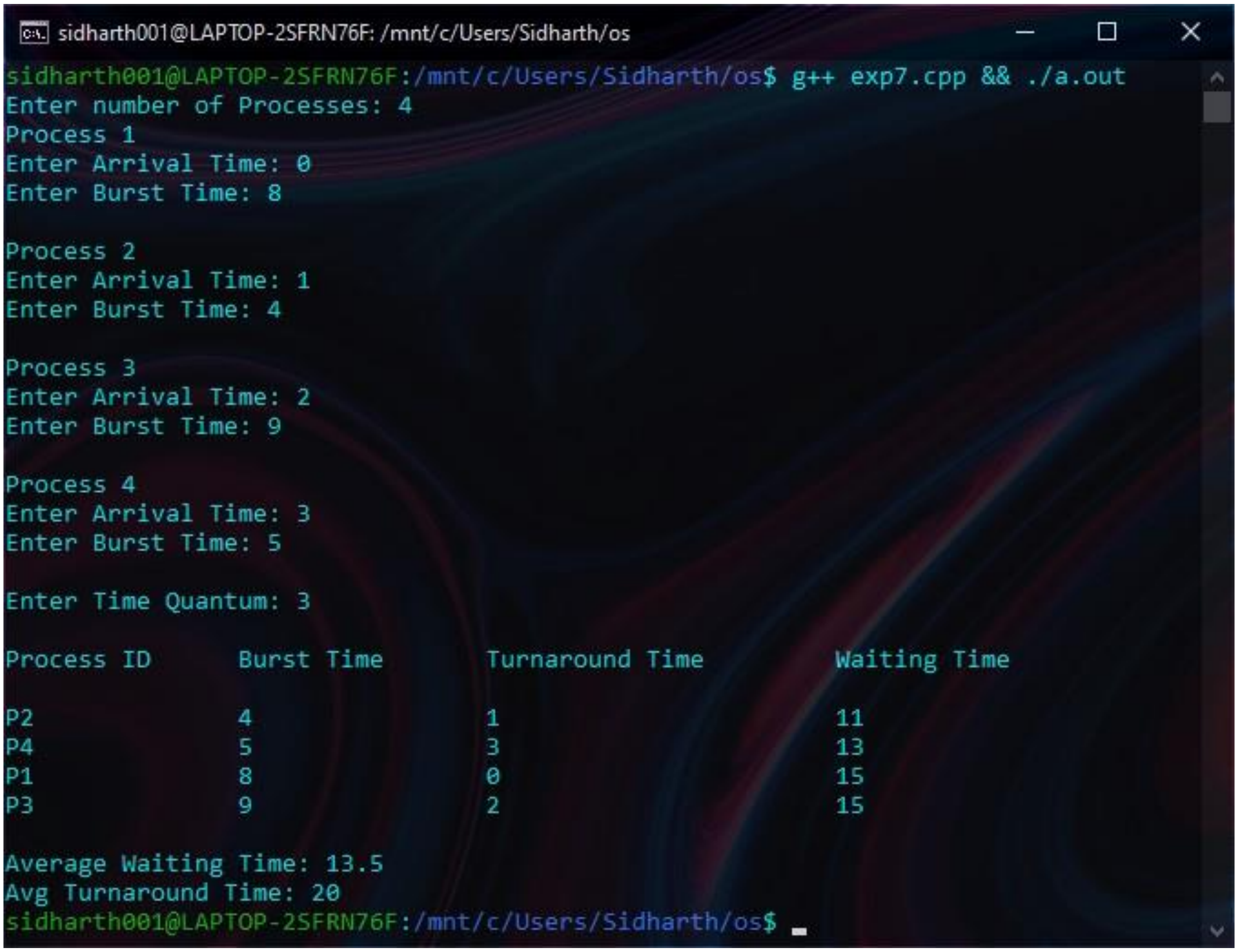
Code:

```
#include <stdio.h>
#include <iostream>
using namespace std;

int i, limit, total = 0, x, counter = 0;
int wait_time = 0, turnaround_time = 0, temp[10];

void round_robin_scheduling(int at[], int bt[], int qtime){
    cout<<"\nProcess ID\tBurst Time\t Turnaround Time\t Waiting Time\n";
    for(total = 0, i = 0; x != 0;){
        if(temp[i] <= qtime && temp[i] > 0){
            total = total + temp[i];
            temp[i] = 0;
            counter = 1;
        }else if(temp[i] > 0){
            temp[i] = temp[i] - qtime;
            total = total + qtime;
        }
        if(temp[i] == 0 && counter == 1){
            x--;
            cout<<"\nP"<<i+1<<"\t\t"<<bt[i]<<"\t\t "<<at[i]<<"\t\t\t "<<total
 - at[i] - bt[i];
            wait_time = wait_time + total - at[i] - bt[i];
            turnaround_time = turnaround_time + total - at[i];
```



```
            counter = 0;
        }
        if(i == limit - 1){
            i = 0;
        }else if(at[i + 1] <= total){
            i++;
        }else{
            i = 0;
        }
    }
}

int main(){

    int  arrival_time[10], burst_time[10], time_quantum;
    cout<<"Enter number of Processes: ";
    cin>>limit;
    x = limit;
    for(int i=0; i<limit; i++){
        cout<<"Process "<<i+1<<"\n";
        cout<<"Enter Arrival Time: ";
        cin>>arrival_time[i];
        cout<<"Enter Burst Time: ";
        cin>>burst_time[i];
        cout<<"\n";
        temp[i] = burst_time[i];
    }
    printf("Enter Time Quantum: ");
    cin>>time_quantum;

    round_robin_scheduling(arrival_time, burst_time, time_quantum);

    float average_wait_time, average_turnaround_time;
    average_wait_time = (float)wait_time / limit;
    average_turnaround_time = (float)turnaround_time / limit;

    cout<<"\n\nAverage Waiting Time: "<<average_wait_time;
    cout<<"\nAvg Turnaround Time: "<<average_turnaround_time<<"\n";

    return 0;
}
```

Output:



```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ g++ exp7.cpp && ./a.out
Enter number of Processes: 4
Process 1
Enter Arrival Time: 0
Enter Burst Time: 8

Process 2
Enter Arrival Time: 1
Enter Burst Time: 4

Process 3
Enter Arrival Time: 2
Enter Burst Time: 9

Process 4
Enter Arrival Time: 3
Enter Burst Time: 5

Enter Time Quantum: 3

Process ID      Burst Time       Turnaround Time         Waiting Time

P2              4                1                       11
P4              5                3                       13
P1              8                0                       15
P3              9                2                       15

Average Waiting Time: 13.5
Avg Turnaround Time: 20
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ _
```

Sidharth

2K18/MC/114

Experiment 8

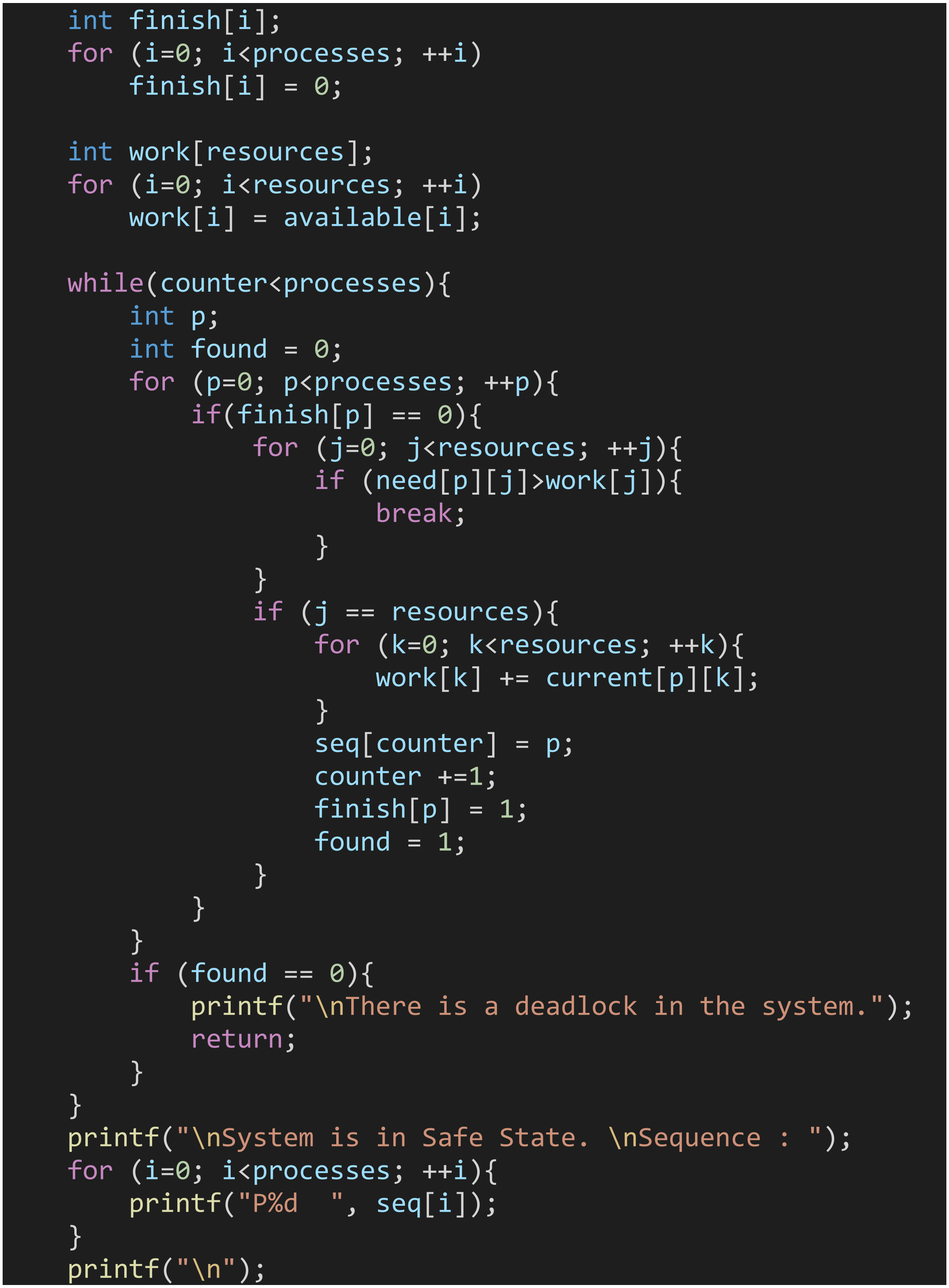
Aim: Write a program to implement Banker's Algorithm. The program should either print the safe sequence of execution of given processes (if any exists) or print "There is a deadlock in the system".

Example: consider the following snapshot of a system:

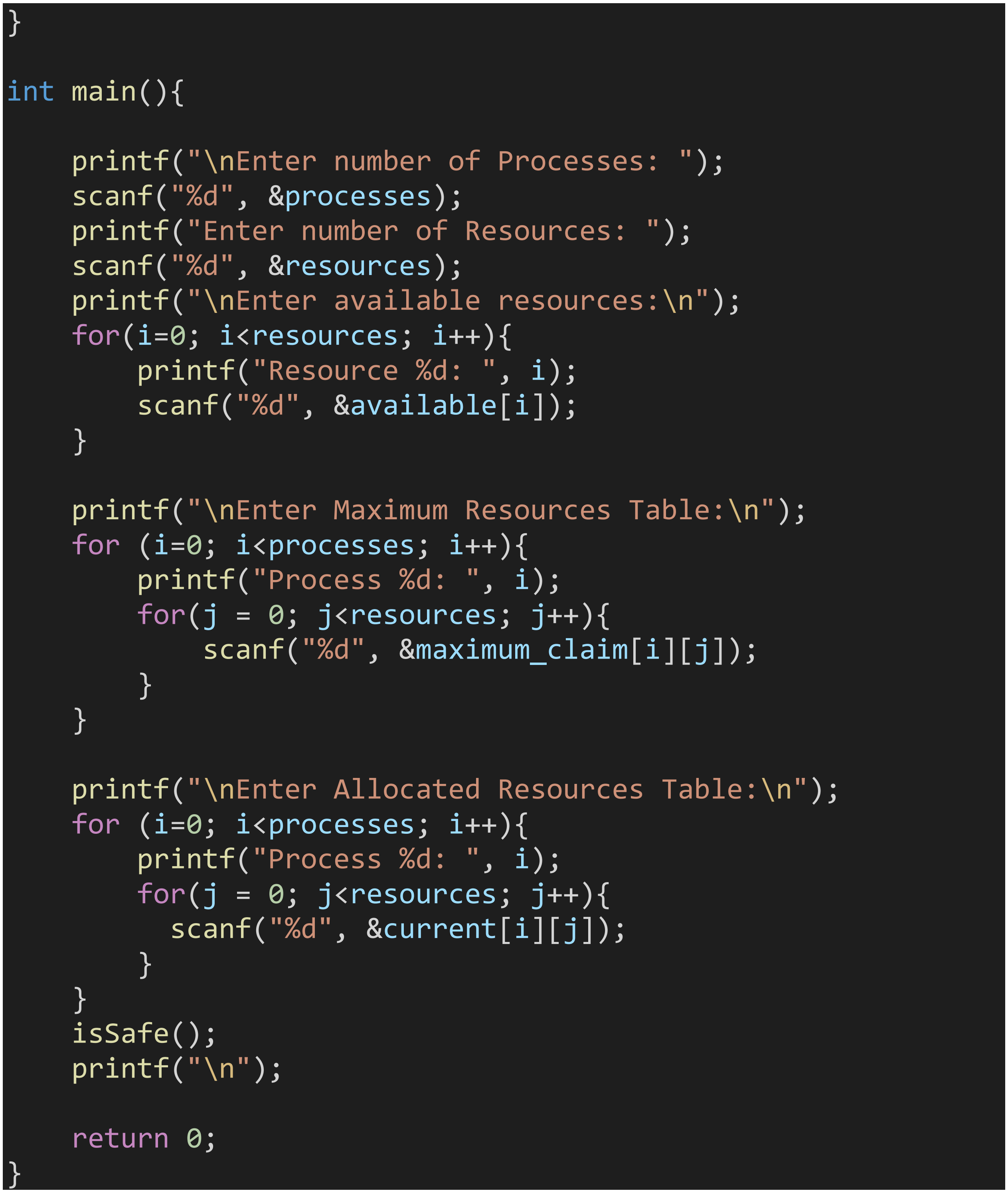
Processes	Allocation			Maximum			Available		
	A	B	C	A	B	C	A	B	C
P0	1	1	2	4	3	3	2	1	0
P1	2	1	2	3	2	2			
P2	4	0	1	9	0	2			
P3	0	2	0	7	5	3			
P4	1	1	2	1	1	2			

Code:

```c
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5];
int processes,resources;
int need[5][5];
int i, j, k, counter = 0;
int seq[5];

void isSafe(){

    for (i=0; i<processes; ++i){
       for (int j = 0; j < resources; ++j){
          need[i][j] = maximum_claim[i][j] - current[i][j];
       }
    }
```

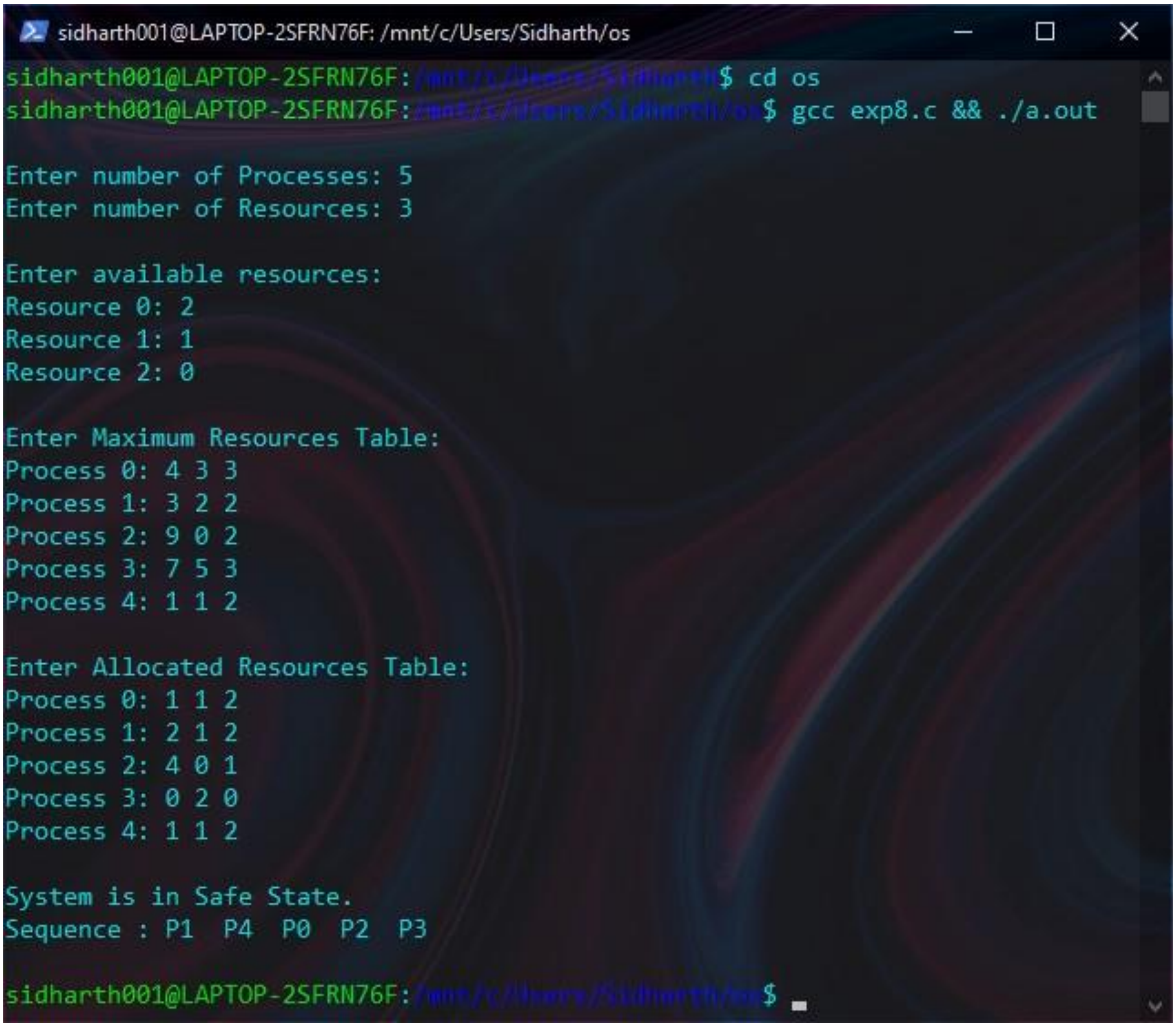


```
    int finish[i];
    for (i=0; i<processes; ++i)
        finish[i] = 0;

    int work[resources];
    for (i=0; i<resources; ++i)
        work[i] = available[i];

    while(counter<processes){
        int p;
        int found = 0;
        for (p=0; p<processes; ++p){
            if(finish[p] == 0){
                for (j=0; j<resources; ++j){
                    if (need[p][j]>work[j]){
                        break;
                    }
                }
                if (j == resources){
                    for (k=0; k<resources; ++k){
                        work[k] += current[p][k];
                    }
                    seq[counter] = p;
                    counter +=1;
                    finish[p] = 1;
                    found = 1;
                }
            }
        }
        if (found == 0){
            printf("\nThere is a deadlock in the system.");
            return;
        }
    }
    printf("\nSystem is in Safe State. \nSequence : ");
    for (i=0; i<processes; ++i){
        printf("P%d  ", seq[i]);
    }
    printf("\n");
```



```c
}

int main(){

    printf("\nEnter number of Processes: ");
    scanf("%d", &processes);
    printf("Enter number of Resources: ");
    scanf("%d", &resources);
    printf("\nEnter available resources:\n");
    for(i=0; i<resources; i++){
        printf("Resource %d: ", i);
        scanf("%d", &available[i]);
    }

    printf("\nEnter Maximum Resources Table:\n");
    for (i=0; i<processes; i++){
        printf("Process %d: ", i);
        for(j = 0; j<resources; j++){
            scanf("%d", &maximum_claim[i][j]);
        }
    }

    printf("\nEnter Allocated Resources Table:\n");
    for (i=0; i<processes; i++){
        printf("Process %d: ", i);
        for(j = 0; j<resources; j++){
          scanf("%d", &current[i][j]);
        }
    }
    isSafe();
    printf("\n");

    return 0;
}
```

Output:



```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ gcc exp8.c && ./a.out

Enter number of Processes: 5
Enter number of Resources: 3

Enter available resources:
Resource 0: 2
Resource 1: 1
Resource 2: 0

Enter Maximum Resources Table:
Process 0: 4 3 3
Process 1: 3 2 2
Process 2: 9 0 2
Process 3: 7 5 3
Process 4: 1 1 2

Enter Allocated Resources Table:
Process 0: 1 1 2
Process 1: 2 1 2
Process 2: 4 0 1
Process 3: 0 2 0
Process 4: 1 1 2

System is in Safe State.
Sequence : P1  P4  P0  P2  P3

sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```

Sidharth

2K18/MC/114

Experiment 9

Aim: Implement the reader writer problem and record your observations. Simulate two children process that try to read/write the file simultaneously.

Code:

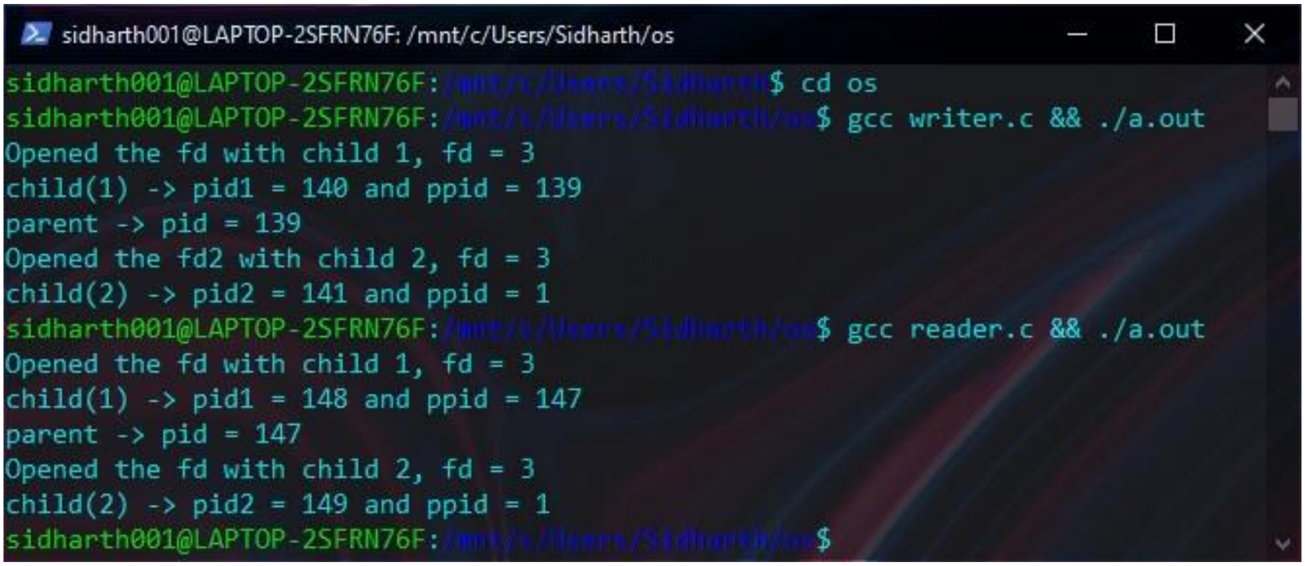
writer.c

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

int main(){
    int pid1, pid2;
    pid1 = fork();
    if(pid1 == 0){
        int fd = open("sample.txt", O_WRONLY | O_CREAT| O_TRUNC, 0644);
        printf("Opened the fd with child 1, fd = %d\n", fd);
        if(fd == -1){
            perror("Error: unable to open!");
        }
        printf("child(1) -
> pid1 = %d and ppid = %d\n", getpid(), getppid());
        return 0;
    }else{
        pid2 = fork();
        if(pid2 == 0){
            int fd2 = open("dummy.txt", O_WRONLY | O_CREAT| O_TRUNC, 0644);
            if(fd2 == -1){
                perror("Error: unable to open!");
            }
            printf("Opened the fd2 with child 2, fd = %d\n", fd2);
            printf("child(2) -
> pid2 = %d and ppid = %d\n", getpid(), getppid());
        }else{
            printf("parent -> pid = %d\n", getpid());
        }
    }
    return 0;
}
```

reader.c

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

int main(){
    int pid1, pid2;
    pid1 = fork();
    if(pid1 == 0){
        int fd = open("sample.txt", O_RDONLY);
        printf("Opened the fd with child 1, fd = %d\n", fd);
        if(fd == -1){
            perror("Error: unable to open!");
        }
        printf("child(1) -
> pid1 = %d and ppid = %d\n", getpid(), getppid());
        return 0;
    }else{
        pid2 = fork();
        if(pid2 == 0){
            int fd = open("dummy.txt", O_RDONLY);
            if(fd == -1){
                perror("Error: unable to open!");
            }
            printf("Opened the fd with child 2, fd = %d\n", fd);
            printf("child(2) -
> pid2 = %d and ppid = %d\n", getpid(), getppid());
        }else{
            printf("parent -> pid = %d\n", getpid());
        }
    }
    return 0;
}
```

Output:



```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ gcc writer.c && ./a.out
Opened the fd with child 1, fd = 3
child(1) -> pid1 = 140 and ppid = 139
parent -> pid = 139
Opened the fd2 with child 2, fd = 3
child(2) -> pid2 = 141 and ppid = 1
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ gcc reader.c && ./a.out
Opened the fd with child 1, fd = 3
child(1) -> pid1 = 148 and ppid = 147
parent -> pid = 147
Opened the fd with child 2, fd = 3
child(2) -> pid2 = 149 and ppid = 1
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```

Sidharth

2K18/MC/114

Experiment 10

Aim: Write a program to implement Least Recently Used algorithm for page replacement.

Input:

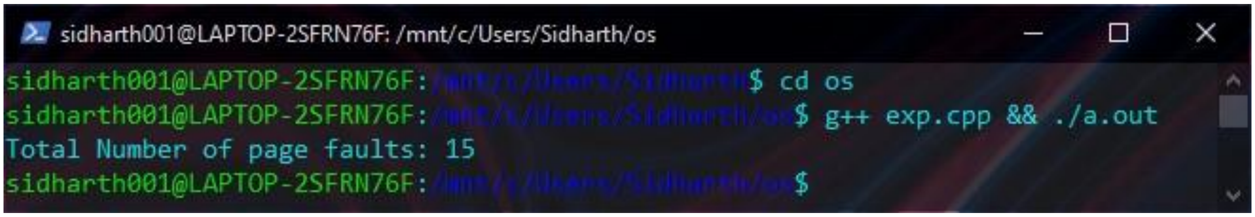
Reference String - 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1.

Number of frames - 3.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int frame_size=3;
    int page_faults=0;
    vector<int> ref_string={7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 
2, 0, 1, 7, 0, 1};
    queue<int> q;
    unordered_set<int> check;
    for(auto it:ref_string){
        if(check.find(it)==check.end()){
            if(q.size()>=frame_size){
                check.erase(q.front());
                q.pop();
            }
            q.push(it);
            check.insert(it);
            page_faults++;
        }
    }
    cout<<"Total Number of page faults: "<<page_faults<<"\n";
}
```

Output:



sidharth001@LAPTOP-2SFRN76F: /mnt/c/Users/Sidharth/os

```
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth$ cd os
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$ g++ exp.cpp && ./a.out
Total Number of page faults: 15
sidharth001@LAPTOP-2SFRN76F:/mnt/c/Users/Sidharth/os$
```