

Practical 2

Execute various UNIX system calls for

- i. Process management
- ii. File management
- iii. Input/output Systems calls

System Calls -

The interface between a process and an operating system is provided by system calls. They are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

Types of system calls -

- 1. Process management
- 2. File management
- 3. Input/output calls
- 4. Device management
- 5. Communication

There are around 80 system calls in the Unix interface currently. Details about some of the important ones are given as follows -

System Call	Description
exec()	This system call runs when an executable file in the context of an already running process that replaces the older executable file.
wait()	This suspends the parent process and executes child process
access()	This checks if a calling process has access to the required file
chdir()	The chdir command changes the current directory of the system

chmod()	The mode of a file can be changed using this command
chown()	This changes the ownership of a particular file
kill()	This system call sends kill signal to one or more processes
link()	A new file name is linked to an existing file using link system call.
open()	This opens a file for the reading or writing process
pause()	The pause call suspends a file until a particular signal occurs.
stime()	This system call sets the correct time.
times()	Gets the parent and child process times
alarm()	The alarm system call sets the alarm clock of a process
fork()	A new process is created using this command
chroot()	This changes the root directory of a file.
exit()	The exit system call is used to exit a process.

Practical 3

Code-

```
#include<stdio.h>

int main()
{
    int n, bt[20], wt[20], tat[20], avwt = 0, avtat = 0, i, j;
    printf("Enter total number of processes:");
    scanf("%d", &n);

    printf("\nEnter Process Burst Time\n");
    for (i = 0; i < n; i++)
    {
        printf("P[%d]:", i + 1);
        scanf("%d", &bt[i]);
    }

    wt[0] = 0;

    //Calculating waiting time
    for (i = 1; i < n; i++)
    {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
    }

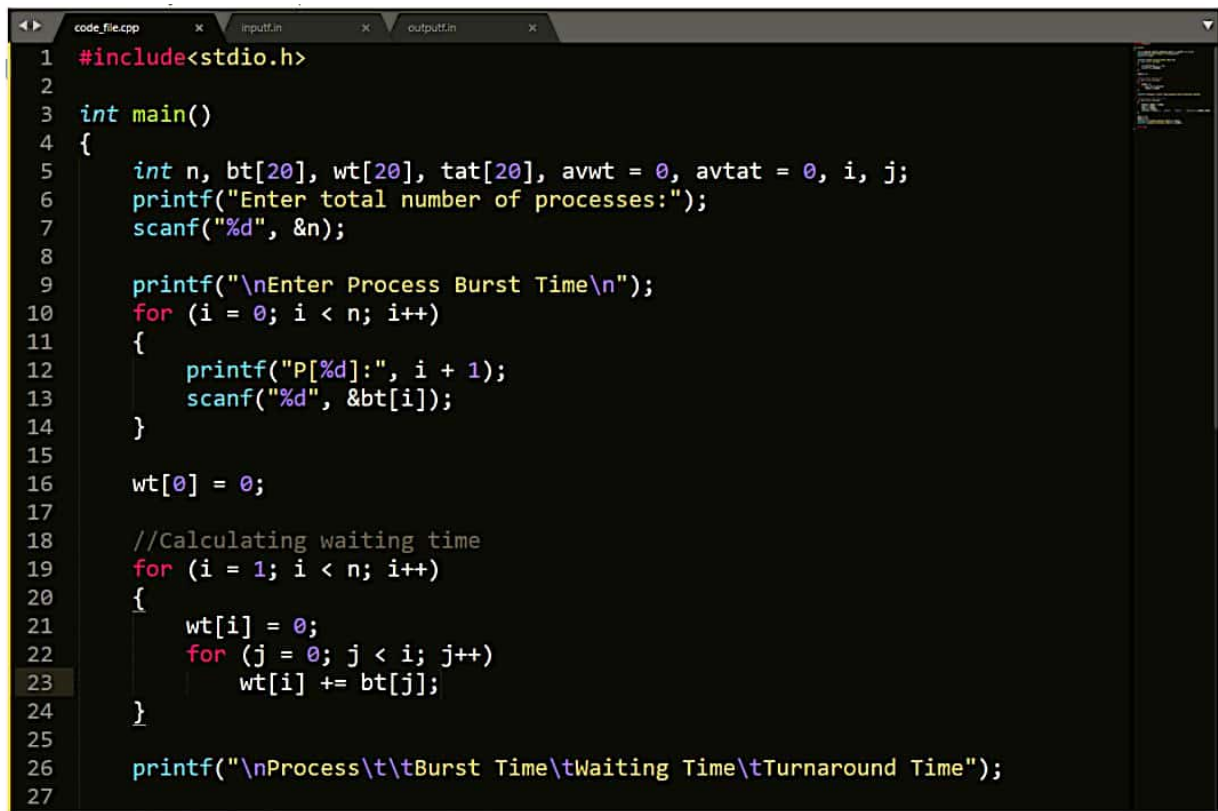
    printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

    //Calculating turnaround time
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
        avwt += wt[i];
        avtat += tat[i];
        printf("\n P[%d]\t\t\t %d\t\t\t %d\t\t\t %d", i + 1, bt[i], wt[i], tat[i]);
    }
}
```

```
    avwt /= i;
    avtat /= i;
    printf("\n\nAverage Waiting Time:%d", avwt);
    printf("\n\nAverage Turnaround Time:%d", avtat);

    return 0;
}
```

Screenshots-



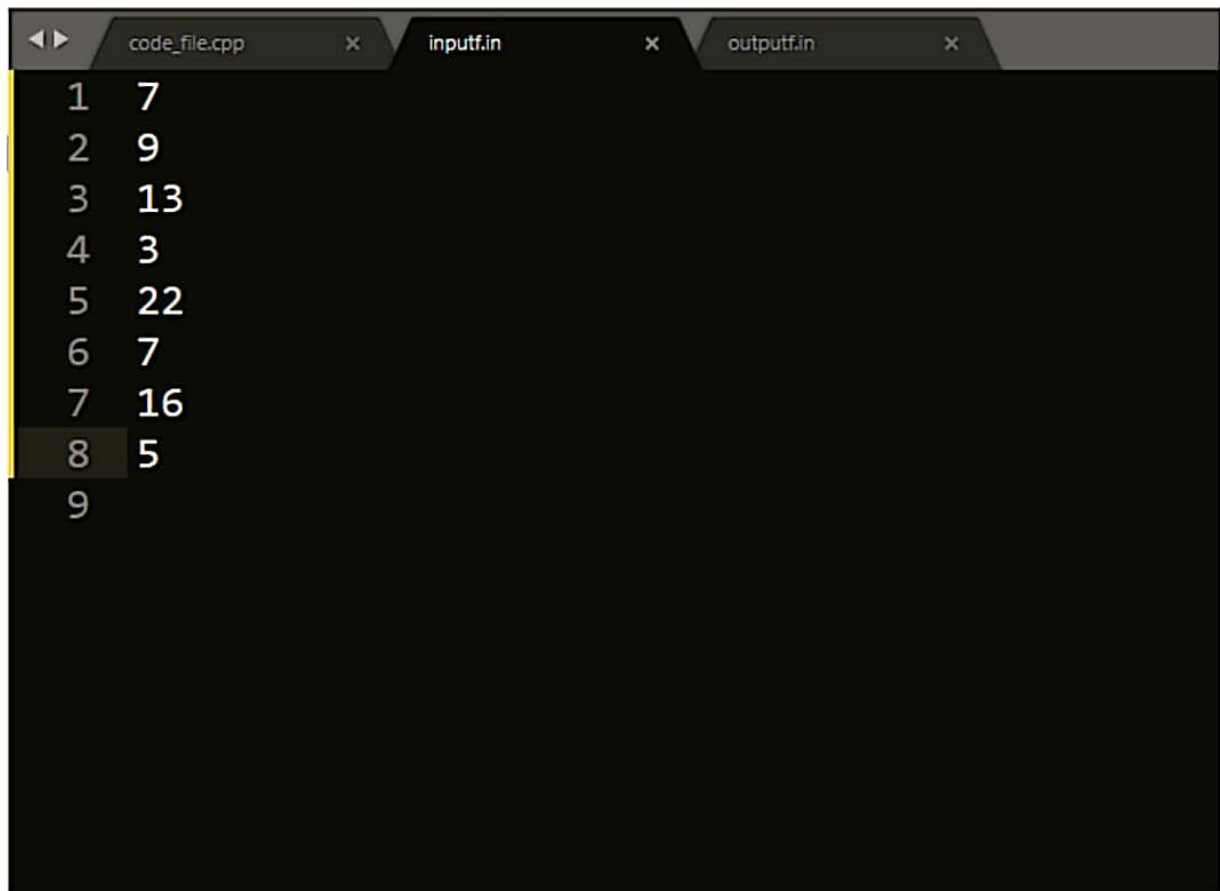
```
1  #include<stdio.h>
2
3  int main()
4  {
5      int n, bt[20], wt[20], tat[20], avwt = 0, avtat = 0, i, j;
6      printf("Enter total number of processes:");
7      scanf("%d", &n);
8
9      printf("\nEnter Process Burst Time\n");
10     for (i = 0; i < n; i++)
11     {
12         printf("P[%d]:", i + 1);
13         scanf("%d", &bt[i]);
14     }
15
16     wt[0] = 0;
17
18     //Calculating waiting time
19     for (i = 1; i < n; i++)
20     {
21         wt[i] = 0;
22         for (j = 0; j < i; j++)
23             wt[i] += bt[j];
24     }
25
26     printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
27
```

```

16     wt[0] = 0;
17
18     //Calculating waiting time
19     for (i = 1; i < n; i++)
20     {
21         wt[i] = 0;
22         for (j = 0; j < i; j++)
23             wt[i] += bt[j];
24     }
25
26     printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
27
28     //Calculating turnaround time
29     for (i = 0; i < n; i++)
30     {
31         tat[i] = bt[i] + wt[i];
32         avwt += wt[i];
33         avtat += tat[i];
34         printf("\n P[%d]\t\t\t %d\t\t\t %d\t\t\t %d", i + 1, bt[i], wt[i], tat[i]);
35     }
36
37     avwt /= i;
38     avtat /= i;
39     printf("\n\nAverage Waiting Time:%d", avwt);
40     printf("\n\nAverage Turnaround Time:%d", avtat);
41
42     return 0;
43 }

```

Input-



The image shows a screenshot of a code editor with three tabs: 'code_file.cpp', 'inputf.in', and 'outputf.in'. The 'inputf.in' tab is active, displaying a list of numbers. The numbers are arranged in two columns: the first column contains the numbers 1 through 9, and the second column contains the numbers 7, 9, 13, 3, 22, 7, 16, 5, and an empty space. The number 5 in the second column is highlighted with a dark background.

1	7
2	9
3	13
4	3
5	22
6	7
7	16
8	5
9	

Output-

```
code_file.cpp x input.in x output.in
1 Enter total number of processes: 7
2
3 Enter Process Burst Time
4 P[1]:9
5 P[2]:13
6 P[3]:3
7 P[4]:22
8 P[5]:7
9 P[6]:16
10 P[7]:5
11
12 Process          Burst Time      Waiting Time    Turnaround Time
13 P[1]              9              0              9
14 P[2]              13             9              22
15 P[3]              3              22             25
16 P[4]              22             25             47
17 P[5]              7              47             54
18 P[6]              16             54             70
19 P[7]              5              70             75
20
21 Average Waiting Time:32
22 Average Turnaround Time:43
```


Practical 4

Code-

```
#include<stdio.h>

int main() {

    int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of process:");
    scanf("%d", &n);

    printf("\nEnter Burst Time:\n");
    for (i = 0; i < n; i++) {
        printf("\np%d:", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1;    //contains process number
    }

    //sorting burst time in ascending order using selection sort
    for (i = 0; i < n; i++) {
        pos = i;
        for (j = i + 1; j < n; j++) {
            if (bt[j] < bt[pos])
                pos = j;
        }

        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;

        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
    }

    wt[0] = 0;

    for (i = 1; i < n; i++)
    {
```

```

        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];

        total += wt[i];
    }

    avg_wt = (float)total / n; //average waiting time
    total = 0;

    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i]; //calculate turnaround time
        total += tat[i];
        printf("\n p%d\t\t %d\t\t %d\t\t\t %d",
            p[i], bt[i], wt[i], tat[i]);
    }

    avg_tat = (float)total / n; //average turnaround time
    printf("\n\nAverage Waiting Time=%f", avg_wt);
    printf("\n\nAverage Turnaround Time=%f\n", avg_tat);
}

```

Screenshots-

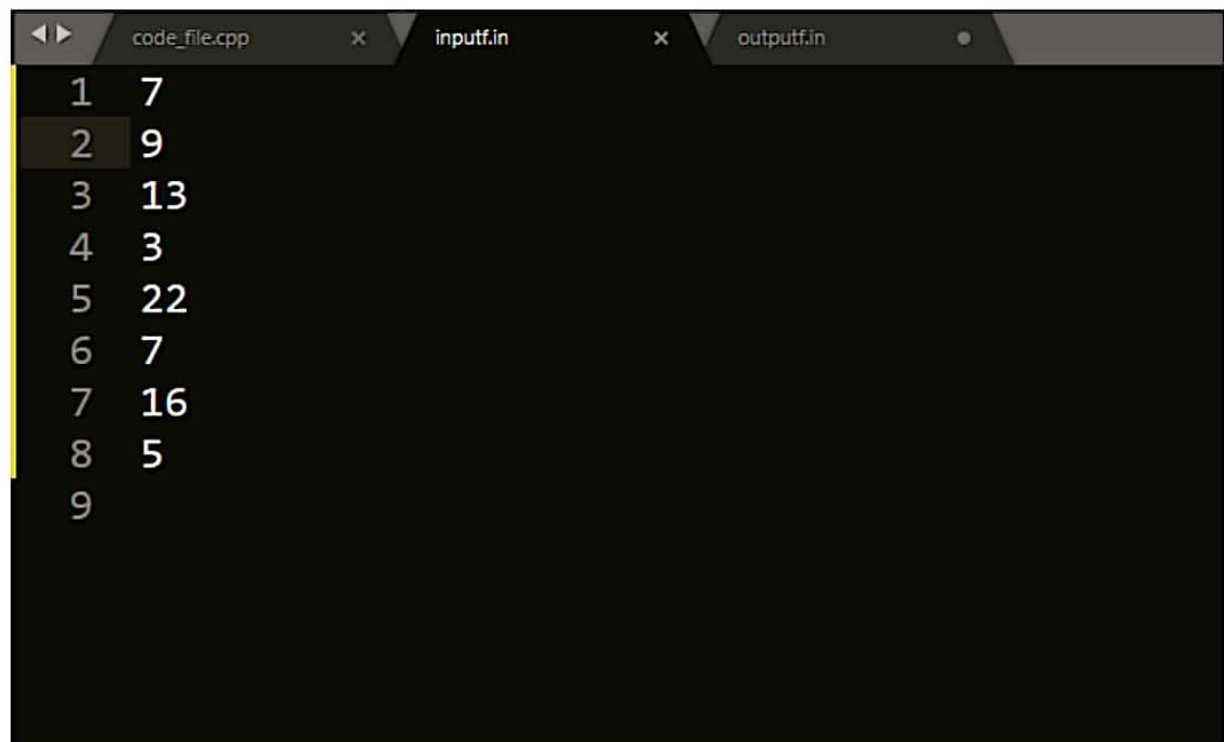
```
code_file.cpp  input.in  output.in
1  #include<stdio.h>
2
3  int main(){
4
5      int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
6      float avg_wt, avg_tat;
7      printf("Enter number of process:");
8      scanf("%d", &n);
9
10     printf("\nEnter Burst Time:\n");
11     for (i = 0; i < n; i++){
12         printf("\np%d:", i + 1);
13         scanf("%d", &bt[i]);
14         p[i] = i + 1; //contains process number
15     }
16
17     //sorting burst time in ascending order using selection sort
18     for (i = 0; i < n; i++){
19         pos = i;
20         for (j = i + 1; j < n; j++){
21             if (bt[j] < bt[pos])
22                 pos = j;
23         }
24
25         temp = bt[i];
26         bt[i] = bt[pos];
27         bt[pos] = temp;
28
29         temp = p[i];
30         p[i] = p[pos];
31         p[pos] = temp;
32     }
```

```

31     p[pos] = temp;
32 }
33
34 wt[0] = 0;
35
36 for (i = 1; i < n; i++)
37 {
38     wt[i] = 0;
39     for (j = 0; j < i; j++)
40         wt[i] += bt[j];
41
42     total += wt[i];
43 }
44
45 avg_wt = (float)total / n; //average waiting time
46 total = 0;
47
48 printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
49 for (i = 0; i < n; i++)
50 {
51     tat[i] = bt[i] + wt[i]; //calculate turnaround time
52     total += tat[i];
53     printf("\n p%d\t\t    %d\t\t    %d\t\t\t    %d",
54           p[i], bt[i], wt[i], tat[i]);
55 }
56
57 avg_tat = (float)total / n; //average turnaround time
58 printf("\n\nAverage Waiting Time=%f", avg_wt);
59 printf("\nAverage Turnaround Time=%f\n", avg_tat);
60 }

```

Input-



A screenshot of a code editor window with three tabs: 'code_file.cpp', 'inputf.in', and 'outputf.in'. The 'inputf.in' tab is active and displays a list of nine pairs of numbers. The first column contains integers from 1 to 9, and the second column contains the values 7, 9, 13, 3, 22, 7, 16, 5, and an empty space for the final row. The second row, containing '2' and '9', is highlighted with a dark background.

1	7
2	9
3	13
4	3
5	22
6	7
7	16
8	5
9	

Output-

```
code_file.cpp x input.in x output.in
1 Enter number of process:7
2 Enter Burst Time:
3 p1:9
4 p2:13
5 p3:3
6 p4:22
7 p5:7
8 p6:16
9 p7:5
10
11 Process      Burst Time      Waiting Time      Turnaround Time
12 p3           3              0                3
13 p7           5              3                8
14 p5           7              8               15
15 p1           9             15               24
16 p2          13             24               37
17 p6          16             37               53
18 p4          22             53               75
19
20 Average Waiting Time=20.000000
21 Average Turnaround Time=30.714285
22
```

Practical 5

Shortest Remaining Time First (Preemptive)

Code →

```
#include <bits/stdc++.h>
using namespace std;
//structure for every process
struct Process {
    int pid; // Process ID
    int bt; // Burst Time
    int art; // Arrival Time
};
void findTurnAroundTime(Process proc[], int n, int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
//waiting time of all process
void findWaitingTime(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 shortest = 0, finish_time;
    bool check = false;
    while (complete != n) {
        for (int j = 0; j < n; j++) {
            if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
            }
        }
        t = t + minm;
        wt[shortest] = t - proc[shortest].art;
        rt[shortest] = 0;
        complete++;
    }
}
```

```

        check = true;
    }
}
if (check == false) {
    t++;
    continue;
}
// decrementing the remaining time
rt[shortest]--;
minm = rt[shortest];
if (minm == 0)
    minm = INT_MAX;
// If a process gets completely
// executed
if (rt[shortest] == 0) {
    complete++;
    check = false;
    finish_time = t + 1;
    // Calculate waiting time
    wt[shortest] = finish_time -
        proc[shortest].bt -
        proc[shortest].art;
    if (wt[shortest] < 0)
        wt[shortest] = 0;
}
// Increment time
t++;
}
}
// Function to calculate average time
void findavgTime(Process proc[], int n) {
    int wt[n], tat[n], total_wt = 0,

```



```

total_tat = 0;
// Function to find waiting time of all
// processes
findWaitingTime(proc, n, wt);
// Function to find turn around time for
// all processes
findTurnAroundTime(proc, n, wt, tat);
cout << "Processes " << " Burst time " << " Waiting time " << "
Turnaround time\n";
for (int i = 0; i < n; i++) {
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    cout << " " << proc[i].pid << "\t\t" << proc[i].bt << "\t\t " << wt[i] << "\t\t "
<< tat[i] << endl;
}
cout << "\nAverage waiting time = " << (float)total_wt / (float)n; cout <<
"\nAverage turnaround time = " << (float)total_tat / (float)n;
}
// main function
int main() {
    Process proc[] = { { 1, 5, 1 }, { 2, 3, 1 }, { 3, 6, 2 }, { 4, 5, 3 } };
    int n = sizeof(proc) / sizeof(proc[0]);
    findavgTime(proc, n);
    return 0;
}

```

Output →

```
"C:\Users\hpw\Documents\codeblocks program\sjf_preemptive.exe"
Processes  Burst time  Waiting time  Turn around time
1          5          3          8
2          3          0          3
3          6          12         18
4          5          6          11

Average waiting time = 5.25
Average turn around time = 10
Process returned 0 (0x0)   execution time : 0.189 s
Press any key to continue.
```

Practical 6

Priority Scheduling (Non-Preemptive)

Code →

```
#include<bits/stdc++.h>
using namespace std;
struct Process {
    int pid; // Process ID
    int bt; // CPU Burst time required
    int priority; // Priority of this process
};
// sorting the Process acc. to the priority
bool compare(Process a, Process b) {
    return (a.priority > b.priority);
}
void waitingtime(Process pro[], int n, int wt[]) {
    // Initial waiting time for a process is 0
    wt[0] = 0;
    // calculating waiting time
    for (int i = 1; i < n ; i++)
        wt[i] = pro[i-1].bt + wt[i-1] ;
}
// Function to calculate turn around time
void turnaround( Process pro[], int n, int wt[], int tat[]) {
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = pro[i].bt + wt[i];
}
```

```

}
//Function to calculate average time
void avgtime(Process pro[], int n) {
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    //Function to find waiting time of all processes
    waitingtime(pro, n, wt);
    //Function to find turn around time for all processes
    turnaround(pro, n, wt, tat);
    //Display processes along with all details
    cout << "\nProcesses " << " Burst time " << " Waiting time " << "
Turn around time\n";
    // Calculate total waiting time and total turn
    // around time
    for (int i=0; i<n; i++) {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << pro[i].pid << "\t\t" << pro[i].bt << "\t " << wt[i] <<
"\t\t " << tat[i] << endl;
    }
    cout << "\nAverage waiting time = " << (float)total_wt / (float)n;
    cout << "\nAverage turnaround time = " << (float)total_tat /
(float)n;
}
void scheduling(Process pro[], int n) {
    // Sort processes by priority
    sort(pro, pro + n, compare);
    cout<< "Order in which processes gets executed \n";
    for (int i = 0 ; i < n; i++)
        cout << pro[i].pid <<" ";
    avg time(pro, n);
}

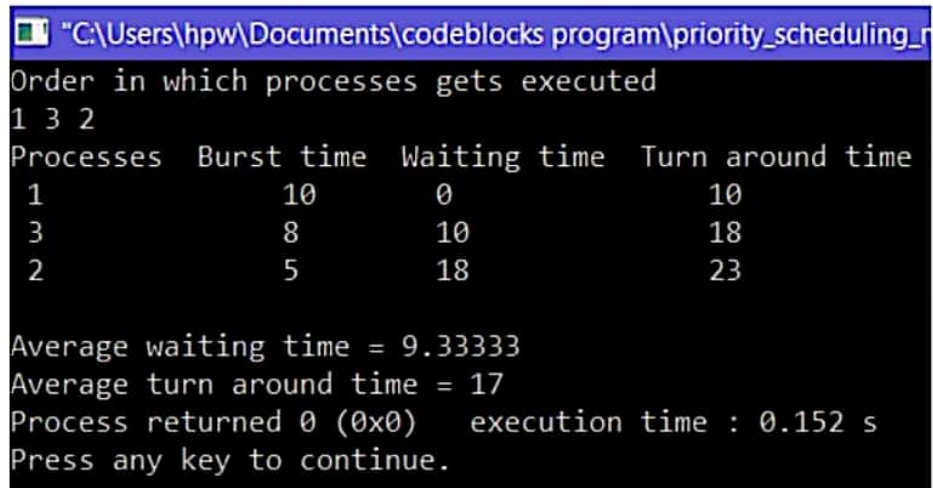
```

```

}
// main function
int main() {
    Process pro[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};
    int n = sizeof pro / sizeof pro[0];
    scheduling(pro, n);
    return 0;
}

```

Output →



```

"C:\Users\hpw\Documents\codeblocks program\priority_scheduling_r
Order in which processes gets executed
1 3 2
Processes  Burst time  Waiting time  Turn around time
1           10         0              10
3           8         10              18
2           5         18              23

Average waiting time = 9.33333
Average turn around time = 17
Process returned 0 (0x0)   execution time : 0.152 s
Press any key to continue.

```

Practical 7

Round Robin Scheduling

Code →

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

void main()
{
    // initialize the variable name
    int i, NOP, sum=0, count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
    float avg_wt, avg_tat;
    printf(" Total number of process in the system: ");
    scanf("%d", &NOP);
    y = NOP; // Assign the number of process to variable y

    // Use for loop to enter the details of the process like Arrival time and the
    Burst Time
    for(i=0; i<NOP; i++)
    {
        printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
        printf(" Arrival time is: \t"); // Accept arrival time
        scanf("%d", &at[i]);
        printf(" \nBurst time is: \t"); // Accept the Burst time
        scanf("%d", &bt[i]);
        temp[i] = bt[i]; // store the burst time in temp array
    }
    // Accept the Time quantum
    printf("Enter the Time Quantum for the process: \t");
    scanf("%d", &quant);
```

```

// Display the process No, burst time, Turn Around Time and the waiting
time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
    sum = sum + temp[i];
    temp[i] = 0;
    count=1;
}
else if(temp[i] > 0)
{
    temp[i] = temp[i] - quant;
    sum = sum + quant;
}
if(temp[i]==0 && count==1)
{
    y--; //decrement the process no.
    printf("\nProcess No[%d] \t\t %d\t\t\t %d\t\t\t %d", i+1, bt[i],
sum-at[i], sum-at[i]-bt[i]);
    wt = wt+sum-at[i]-bt[i];
    tat = tat+sum-at[i];
    count =0;
}
if(i==NOP-1)
{
    i=0;
}
else if(at[i+1]<=sum)

```

```

    {
        i++;
    }
    else
    {
        i=0;
    }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turnaround Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}

```

Output→


```
"C:\Users\hpw\Documents\codeblocks program\round_robin.exe"
Total number of process in the system: 4

Enter the Arrival and Burst time of the Process[1]
Arrival time is:      0

Burst time is:  8

Enter the Arrival and Burst time of the Process[2]
Arrival time is:      1

Burst time is:  5

Enter the Arrival and Burst time of the Process[3]
Arrival time is:      2

Burst time is: 10

Enter the Arrival and Burst time of the Process[4]
Arrival time is:      3

Burst time is: 11
Enter the Time Quantum for the process:      6

Process No      Burst Time      TAT      Waiting Time
Process No[2]      5      10      5
Process No[1]      8      25      17
Process No[3]      10      27      17
Process No[4]      11      31      20
Average Turn Around Time:      14.750000
Average Waiting Time: 23.250000
```

Banker's Algorithm

Code →

```
// Banker's Algorithm
#include <iostream>
using namespace std;

int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

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

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

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

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
```

```

        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {

                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]){
                        flag = 1;
                        break;
                    }
                }

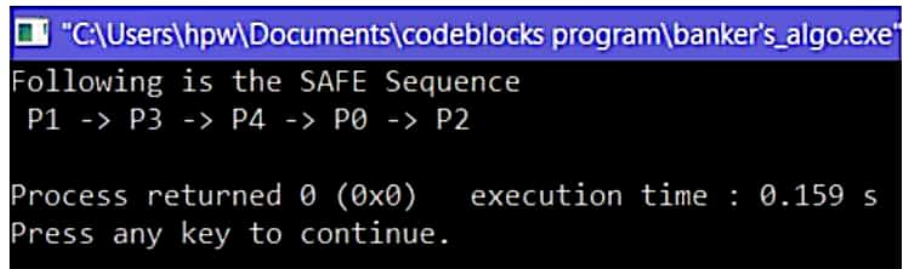
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    cout << "Following is the SAFE Sequence" << endl;
    for (i = 0; i < n - 1; i++)
        cout << " P" << ans[i] << " ->";
    cout << " P" << ans[n - 1] << endl;

    return (0);
}

```

Output →



```
"C:\Users\hpw\Documents\codeblocks program\banker's_algo.exe"  
Following is the SAFE Sequence  
P1 -> P3 -> P4 -> P0 -> P2  
  
Process returned 0 (0x0)   execution time : 0.159 s  
Press any key to continue.
```