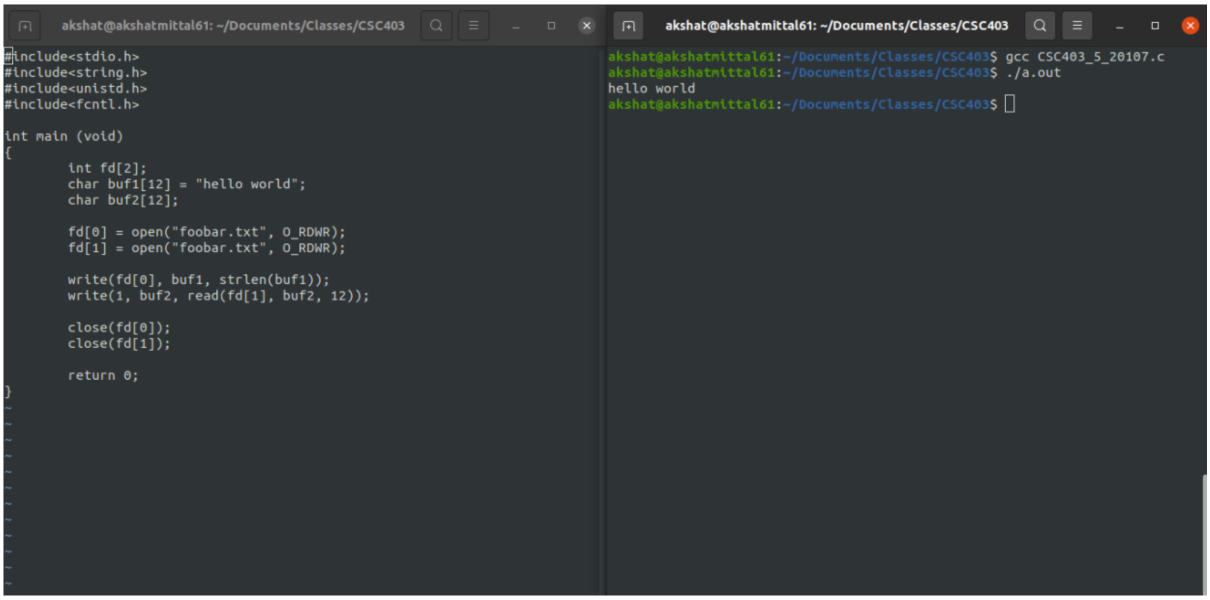
O_RDONLY: read only, O_WRONLY: write only, O_RDWR: read and write, O_CREAT: create file if it doesn't exist, O_EXCL: prevent creation if it already exists.

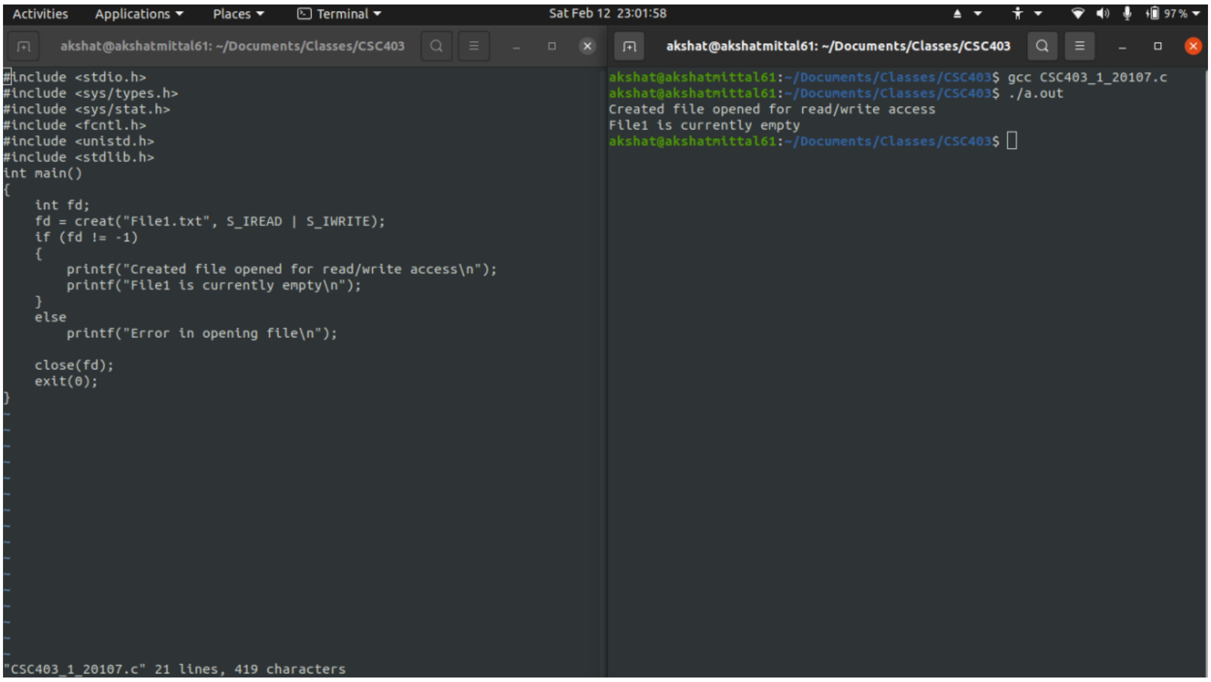
1. using O_RDONLY, O_CREAT, O_TR


akshat@akshatmittal61: ~/Documents/Classes/CSC403
akshat@akshatmittal61: ~/Documents/Classes/CSC403

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

int main (void)
{
        int fd[2];
        char buf1[12] = "hello world";
        char buf2[12];

        fd[0] = open("foobar.txt", O_RDWR);
        fd[1] = open("foobar.txt", O_RDWR);

        write(fd[0], buf1, strlen(buf1));
        write(1, buf2, read(fd[1], buf2, 12));

        close(fd[0]);
        close(fd[1]);

        return 0;
}
```

```
akshat@akshatmittal61:~/Documents/Classes/CSC403$ gcc CSC403_5_20107.c
akshat@akshatmittal61:~/Documents/Classes/CSC403$ ./a.out
hello world
akshat@akshatmittal61:~/Documents/Classes/CSC403$ 
```

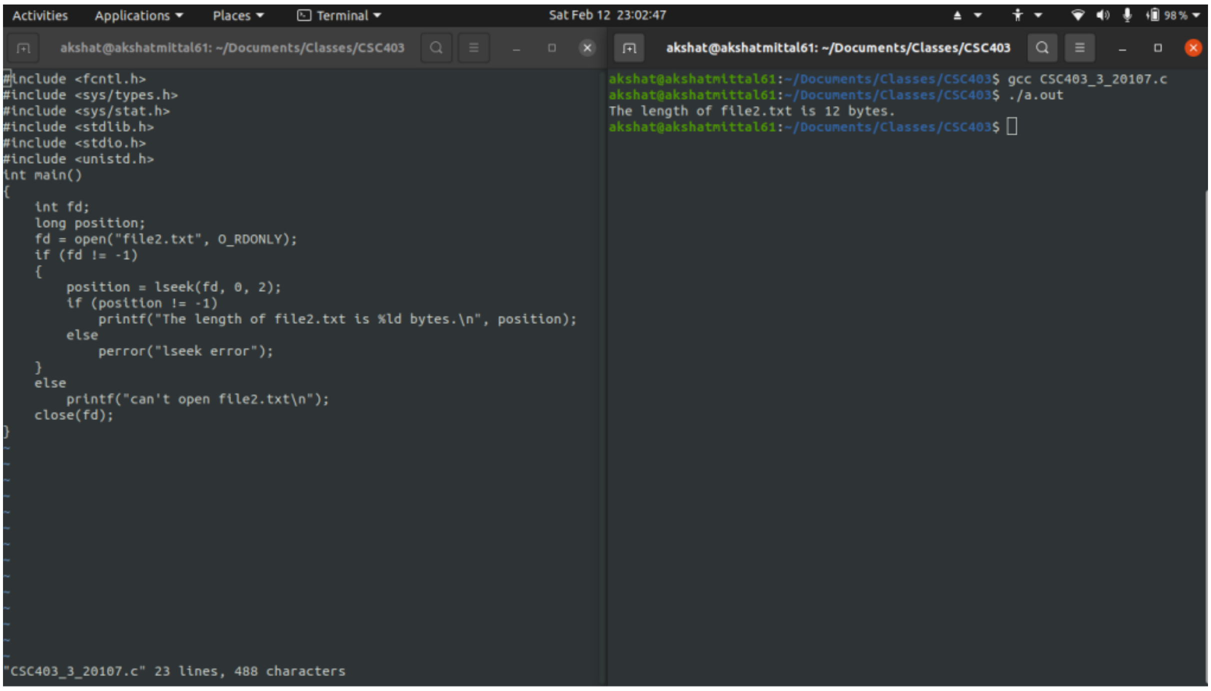
2. USING S_IREAD AND S_IWRITE


Activities
Applications ▾
Places ▾
Terminal ▾
Sat Feb 12 23:01:58
97 %
akshat@akshatmittal61: ~/Documents/Classes/CSC403
akshat@akshatmittal61: ~/Documents/Classes/CSC403

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    int fd;
    fd = creat("File1.txt", S_IREAD | S_IWRITE);
    if (fd != -1)
    {
        printf("Created file opened for read/write access\n");
        printf("File1 is currently empty\n");
    }
    else
        printf("Error in opening file\n");

    close(fd);
    exit(0);
}
"CSC403_1_20107.c" 21 lines, 419 characters
```

```
akshat@akshatmittal61:~/Documents/Classes/CSC403$ gcc CSC403_1_20107.c
akshat@akshatmittal61:~/Documents/Classes/CSC403$ ./a.out
Created file opened for read/write access
File1 is currently empty
akshat@akshatmittal61:~/Documents/Classes/CSC403$ 
```

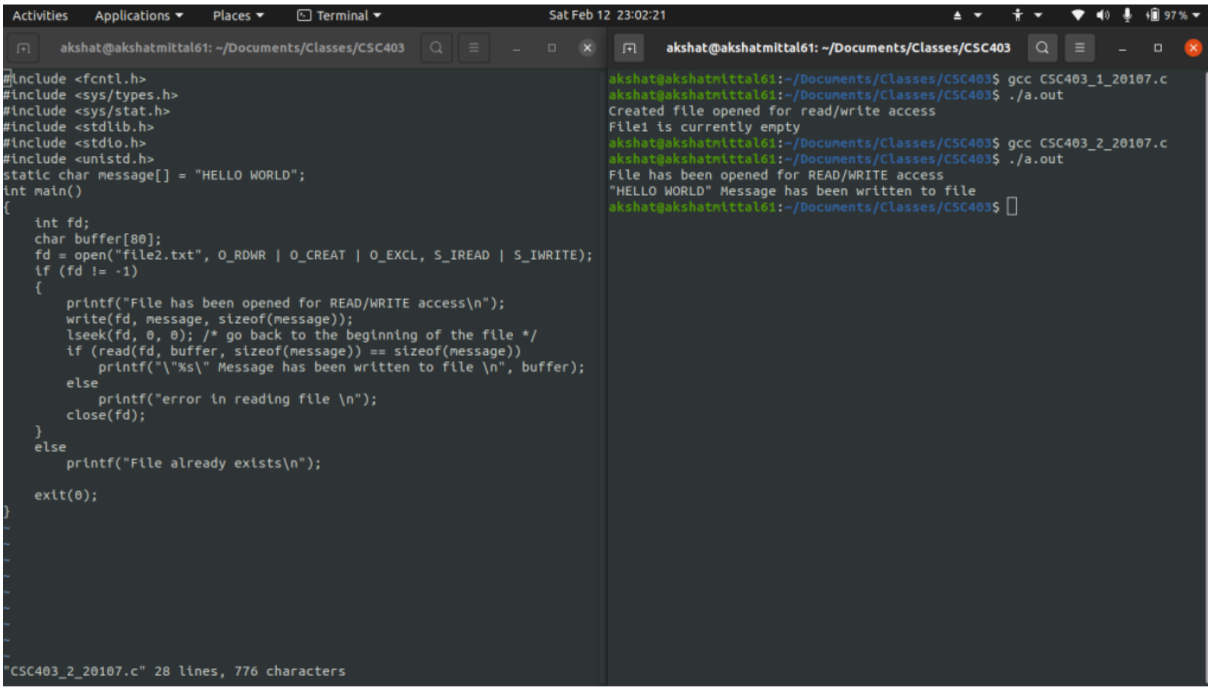
3. READING FROM FILE


Activities
Applications
Places
Terminal
Sat Feb 12 23:02:47
98 %
akshat@akshatmittal61: ~/Documents/Classes/CSC403
akshat@akshatmittal61: ~/Documents/Classes/CSC403

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    int fd;
    long position;
    fd = open("file2.txt", O_RDONLY);
    if (fd != -1)
    {
        position = lseek(fd, 0, 2);
        if (position != -1)
            printf("The length of file2.txt is %ld bytes.\n", position);
        else
            perror("lseek error");
    }
    else
        printf("can't open file2.txt\n");
    close(fd);
}
"CSC403_3_20107.c" 23 lines, 488 characters
```

```
akshat@akshatmittal61:~/Documents/Classes/CSC403$ gcc CSC403_3_20107.c
akshat@akshatmittal61:~/Documents/Classes/CSC403$ ./a.out
The length of file2.txt is 12 bytes.
akshat@akshatmittal61:~/Documents/Classes/CSC403$
```

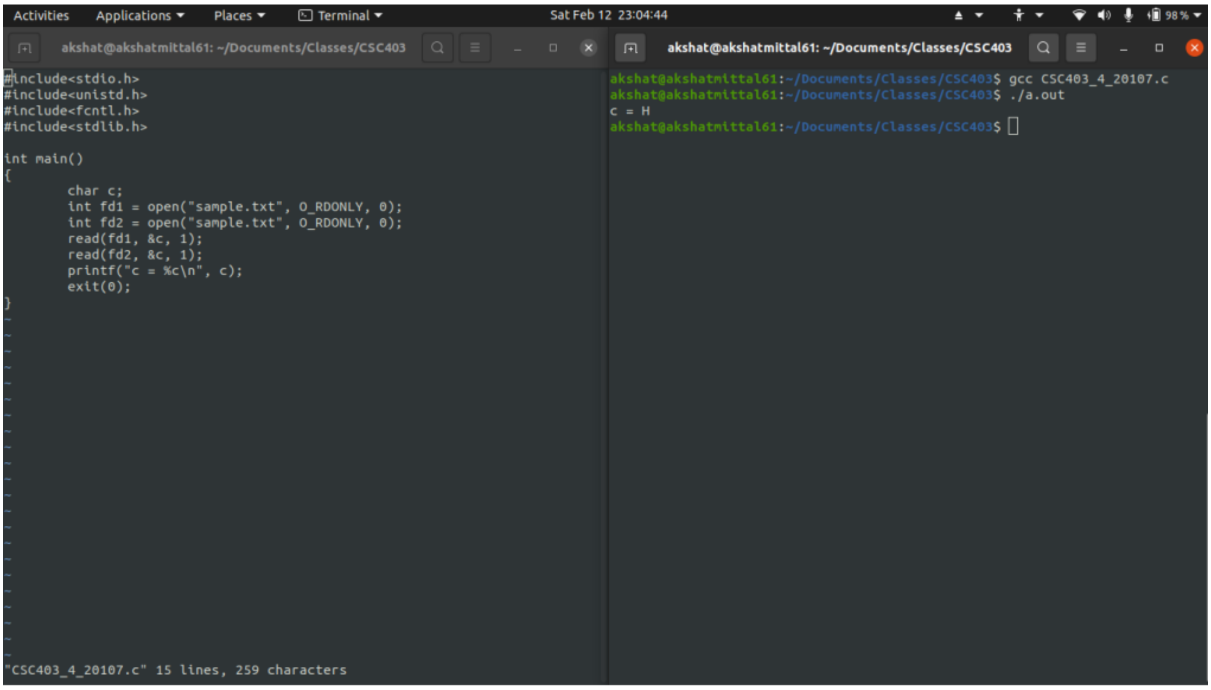
4. USING O_RDWR AND S_IREAD, S_IREAD, O_CREAT simultaneously


Activities
Applications
Places
Terminal
Sat Feb 12 23:02:21
97 %
akshat@akshatmittal61: ~/Documents/Classes/CSC403
akshat@akshatmittal61: ~/Documents/Classes/CSC403

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
static char message[] = "HELLO WORLD";
int main()
{
    int fd;
    char buffer[80];
    fd = open("file2.txt", O_RDWR | O_CREAT | O_EXCL, S_IREAD | S_IWRITE);
    if (fd != -1)
    {
        printf("File has been opened for READ/WRITE access\n");
        write(fd, message, sizeof(message));
        lseek(fd, 0, 0); /* go back to the beginning of the file */
        if (read(fd, buffer, sizeof(message)) == sizeof(message))
            printf("\"%s\" Message has been written to file \n", buffer);
        else
            printf("error in reading file \n");
        close(fd);
    }
    else
        printf("File already exists\n");

    exit(0);
}
"CSC403_2_20107.c" 28 lines, 776 characters
```

```
akshat@akshatmittal61:~/Documents/Classes/CSC403$ gcc CSC403_1_20107.c
akshat@akshatmittal61:~/Documents/Classes/CSC403$ ./a.out
Created file opened for read/write access
File1 is currently empty
akshat@akshatmittal61:~/Documents/Classes/CSC403$ gcc CSC403_2_20107.c
akshat@akshatmittal61:~/Documents/Classes/CSC403$ ./a.out
File has been opened for READ/WRITE access
"HELLO WORLD" Message has been written to file
akshat@akshatmittal61:~/Documents/Classes/CSC403$
```

5. read character from particular position in the file


Activities
Applications
Places
Terminal
Sat Feb 12 23:04:44
98 %
akshat@akshatmittal61: ~/Documents/Classes/CSC403
akshat@akshatmittal61: ~/Documents/Classes/CSC403

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

int main()
{
        char c;
        int fd1 = open("sample.txt", O_RDONLY, 0);
        int fd2 = open("sample.txt", O_RDONLY, 0);
        read(fd1, &c, 1);
        read(fd2, &c, 1);
        printf("c = %c\n", c);
        exit(0);
}
"CSC403_4_20107.c" 15 lines, 259 characters
```

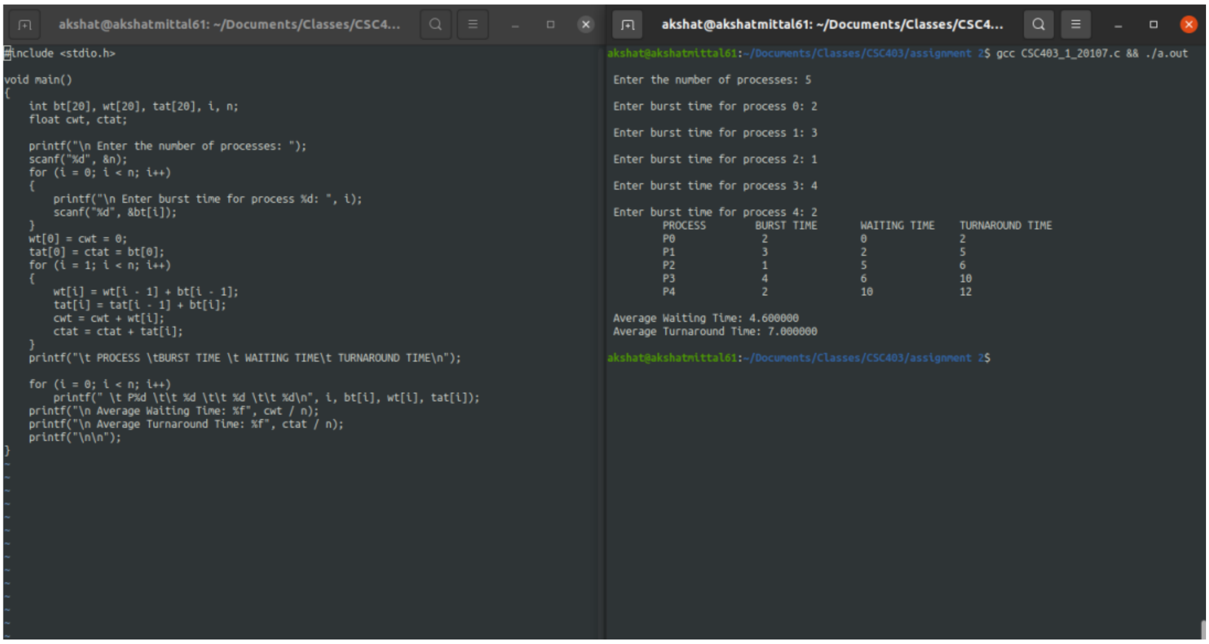
```
akshat@akshatmittal61:~/Documents/Classes/CSC403$ gcc CSC403_4_20107.c
akshat@akshatmittal61:~/Documents/Classes/CSC403$ ./a.out
c = H
akshat@akshatmittal61:~/Documents/Classes/CSC403$
```

Process Scheduling:

Process Scheduling is the activity of the process manager that handles the removal and selection of a process present in or waiting for the CPU.

This job is handles by the CPU schedulers: LTS(Long Term Schedulers), Medium Term Schedulers and Short term schedulers.

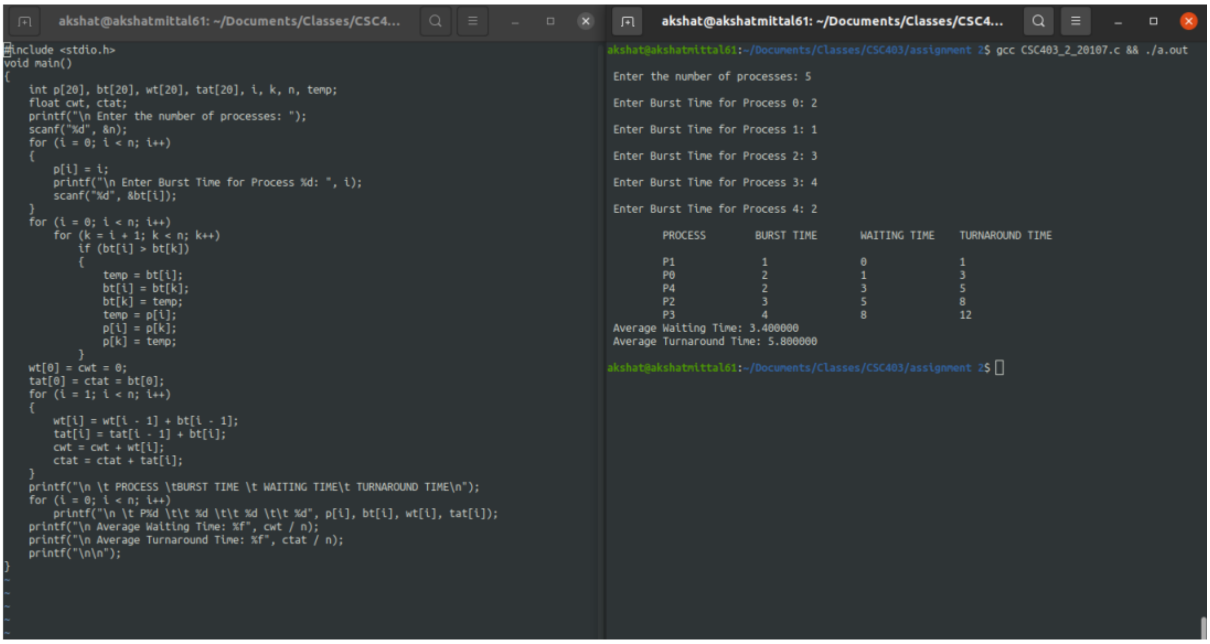
6. FCFS : same arrival time


akshat@akshatmittal61: ~/Documents/Classes/CSC4...
akshat@akshatmittal61: ~/Documents/Classes/CSC4...

```c
#include <stdio.h>

void main()
{
    int bt[20], wt[20], tat[20], i, n;
    float cwt, ctat;

    printf("\n Enter the number of processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\n Enter burst time for process %d: ", i);
        scanf("%d", &bt[i]);
    }
    wt[0] = cwt = 0;
    tat[0] = ctat = bt[0];
    for (i = 1; i < n; i++)
    {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        cwt = cwt + wt[i];
        ctat = ctat + tat[i];
    }
    printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

    for (i = 0; i < n; i++)
        printf(" \t P%d \t\t %d \t\t %d \t\t %d\n", i, bt[i], wt[i], tat[i]);
    printf("\n Average Waiting Time: %f", cwt / n);
    printf("\n Average Turnaround Time: %f", ctat / n);
    printf("\n\n");
}
```

```
akshat@akshatmittal61:~/Documents/Classes/CSC403/assignment 2$ gcc CSC403_1_20107.c && ./a.out

 Enter the number of processes: 5

 Enter burst time for process 0: 2

 Enter burst time for process 1: 3

 Enter burst time for process 2: 1

 Enter burst time for process 3: 4

 Enter burst time for process 4: 2
         PROCESS        BURST TIME        WAITING TIME    TURNAROUND TIME
         P0              2                0               2
         P1              3                2               5
         P2              1                5               6
         P3              4                6               10
         P4              2                10              12

 Average Waiting Time: 4.600000
 Average Turnaround Time: 7.000000

akshat@akshatmittal61:~/Documents/Classes/CSC403/assignment 2$
```

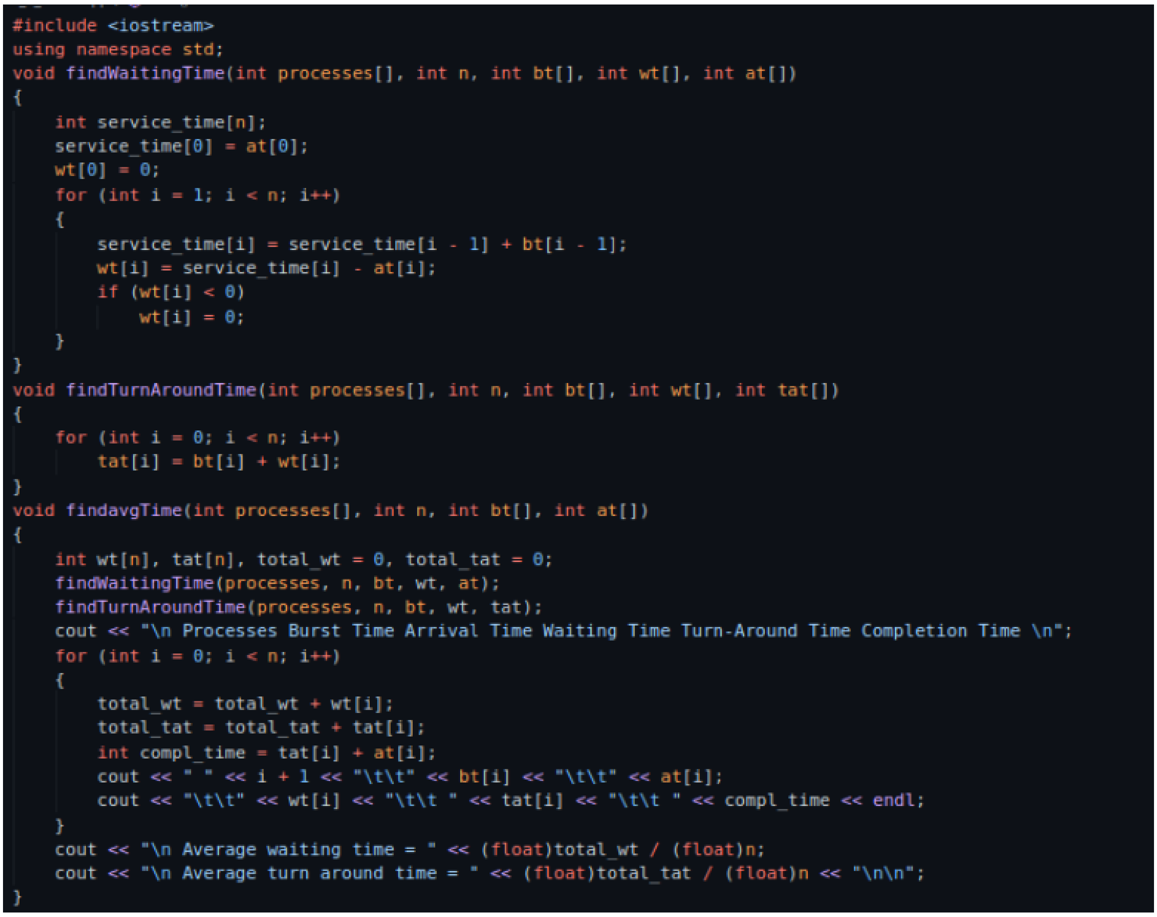
7. SJF : same arrival time


akshat@akshatmittal61: ~/Documents/Classes/CSC4...
akshat@akshatmittal61: ~/Documents/Classes/CSC4...

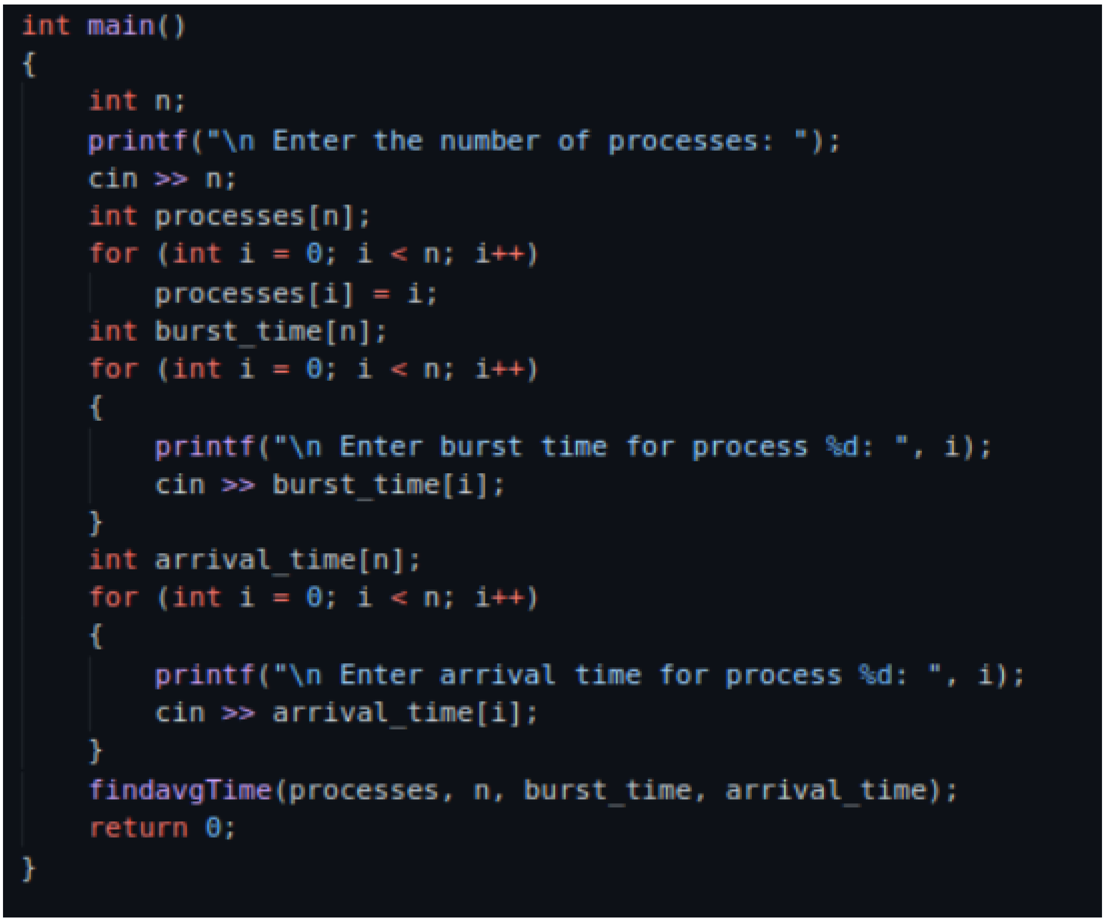
```c
#include <stdio.h>
void main()
{
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float cwt, ctat;
    printf("\n Enter the number of processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        p[i] = i;
        printf("\n Enter Burst Time for Process %d: ", i);
        scanf("%d", &bt[i]);
    }
    for (i = 0; i < n; i++)
        for (k = i + 1; k < n; k++)
            if (bt[i] > bt[k])
            {
                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;
                temp = p[i];
                p[i] = p[k];
                p[k] = temp;
            }
    wt[0] = cwt = 0;
    tat[0] = ctat = bt[0];
    for (i = 1; i < n; i++)
    {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        cwt = cwt + wt[i];
        ctat = ctat + tat[i];
    }
    printf("\n \t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
    for (i = 0; i < n; i++)
        printf("\n \t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
    printf("\n Average Waiting Time: %f", cwt / n);
    printf("\n Average Turnaround Time: %f", ctat / n);
    printf("\n\n");
}
~
~
~
~
~
```

```
akshat@akshatmittal61:~/Documents/Classes/CSC403/assignment 2$ gcc CSC403_2_20107.c && ./a.out

 Enter the number of processes: 5

 Enter Burst Time for Process 0: 2

 Enter Burst Time for Process 1: 1

 Enter Burst Time for Process 2: 3

 Enter Burst Time for Process 3: 4

 Enter Burst Time for Process 4: 2

         PROCESS        BURST TIME       WAITING TIME    TURNAROUND TIME

         P1              1               0               1
         P0              2               1               3
         P4              2               3               5
         P2              3               5               8
         P3              4               8               12
 Average Waiting Time: 3.400000
 Average Turnaround Time: 5.800000

akshat@akshatmittal61:~/Documents/Classes/CSC403/assignment 2$ 
```

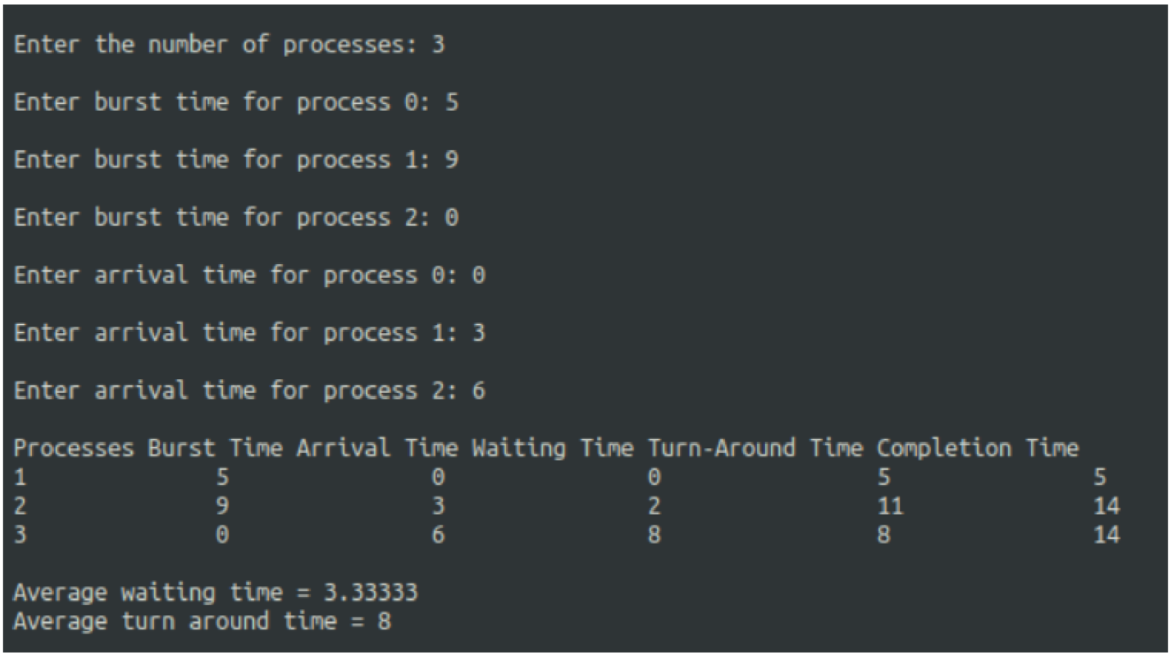
8. FCFS : different arrival time



```
#include <iostream>
using namespace std;
void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[])
{
    int service_time[n];
    service_time[0] = at[0];
    wt[0] = 0;
    for (int i = 1; i < n; i++)
    {
        service_time[i] = service_time[i - 1] + bt[i - 1];
        wt[i] = service_time[i] - at[i];
        if (wt[i] < 0)
            wt[i] = 0;
    }
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[], int at[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(processes, n, bt, wt, at);
    findTurnAroundTime(processes, n, bt, wt, tat);
    cout << "\n Processes Burst Time Arrival Time Waiting Time Turn-Around Time Completion Time \n";
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        int compl_time = tat[i] + at[i];
        cout << " " << i + 1 << "\t\t" << bt[i] << "\t\t" << at[i];
        cout << "\t\t" << wt[i] << "\t\t " << tat[i] << "\t\t " << compl_time << endl;
    }
    cout << "\n Average waiting time = " << (float)total_wt / (float)n;
    cout << "\n Average turn around time = " << (float)total_tat / (float)n << "\n\n";
}
```

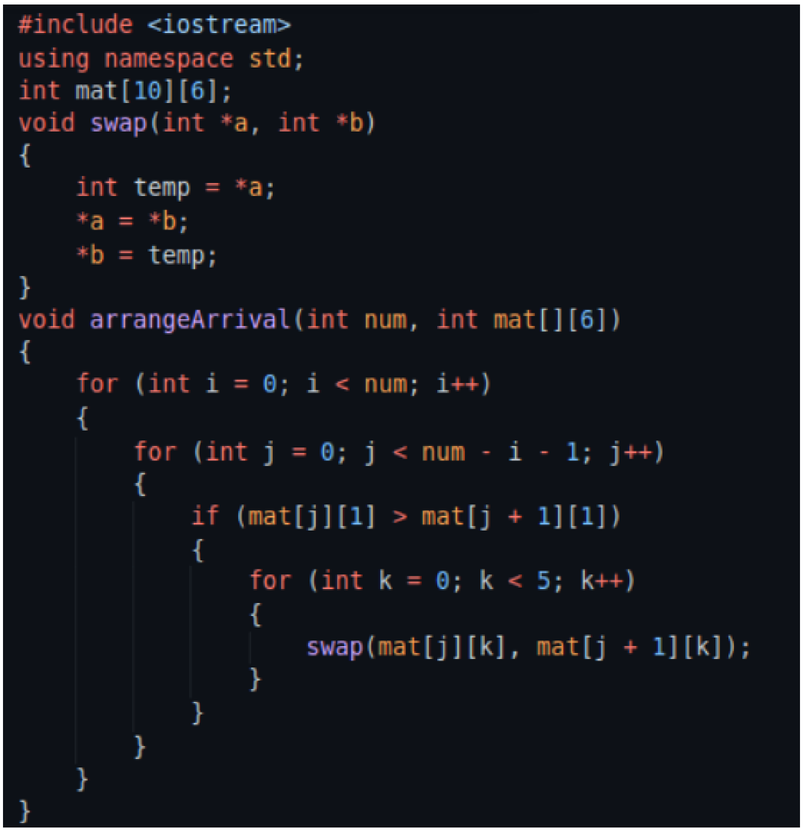


```
int main()
{
    int n;
    printf("\n Enter the number of processes: ");
    cin >> n;
    int processes[n];
    for (int i = 0; i < n; i++)
        processes[i] = i;
    int burst_time[n];
    for (int i = 0; i < n; i++)
    {
        printf("\n Enter burst time for process %d: ", i);
        cin >> burst_time[i];
    }
    int arrival_time[n];
    for (int i = 0; i < n; i++)
    {
        printf("\n Enter arrival time for process %d: ", i);
        cin >> arrival_time[i];
    }
    findavgTime(processes, n, burst_time, arrival_time);
    return 0;
}
```

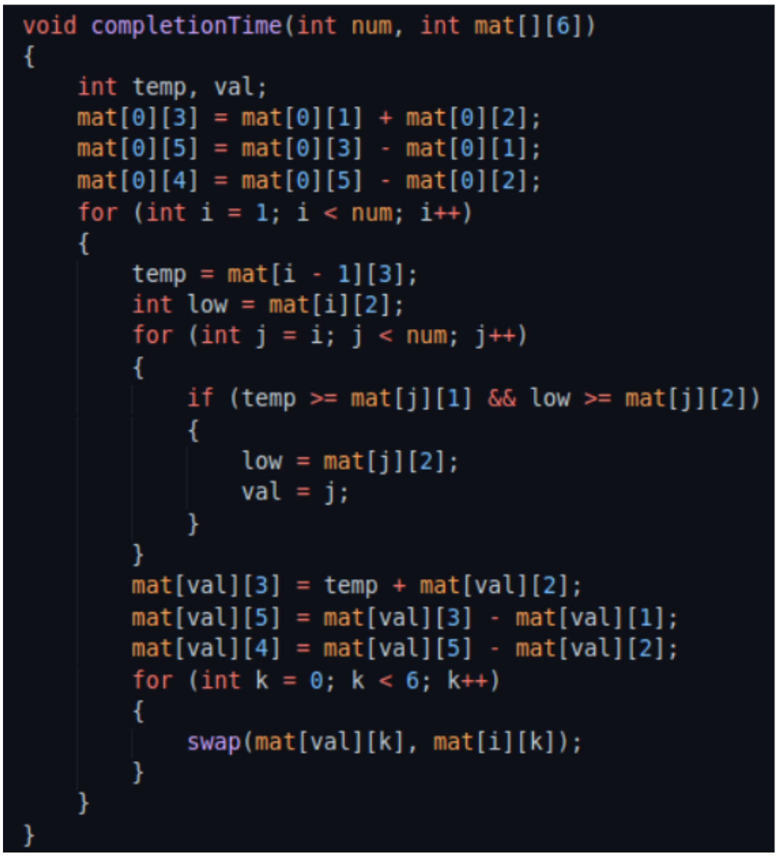


```
Enter the number of processes: 3

Enter burst time for process 0: 5

Enter burst time for process 1: 9

Enter burst time for process 2: 0

Enter arrival time for process 0: 0

Enter arrival time for process 1: 3

Enter arrival time for process 2: 6

Processes Burst Time Arrival Time Waiting Time Turn-Around Time Completion Time
1              5               0               0                5               5
2              9               3               2                11              14
3              0               6               8                8               14

Average waiting time = 3.33333
Average turn around time = 8
```

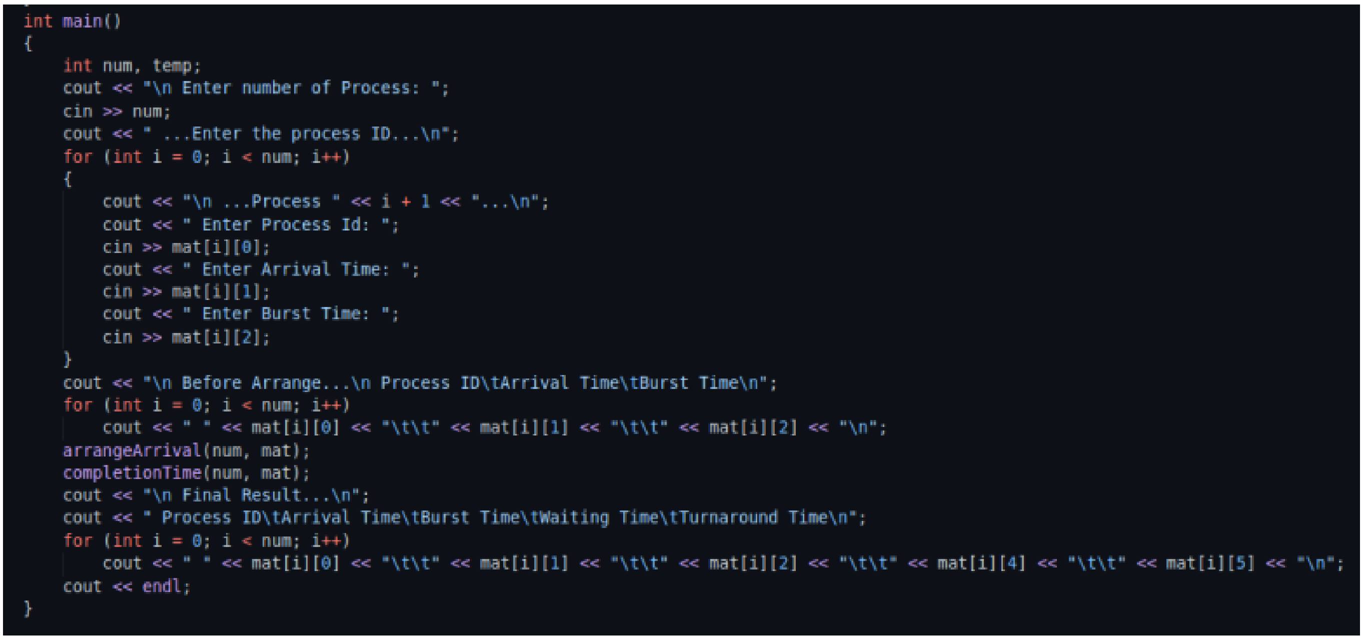
9. SJF : different arrival time



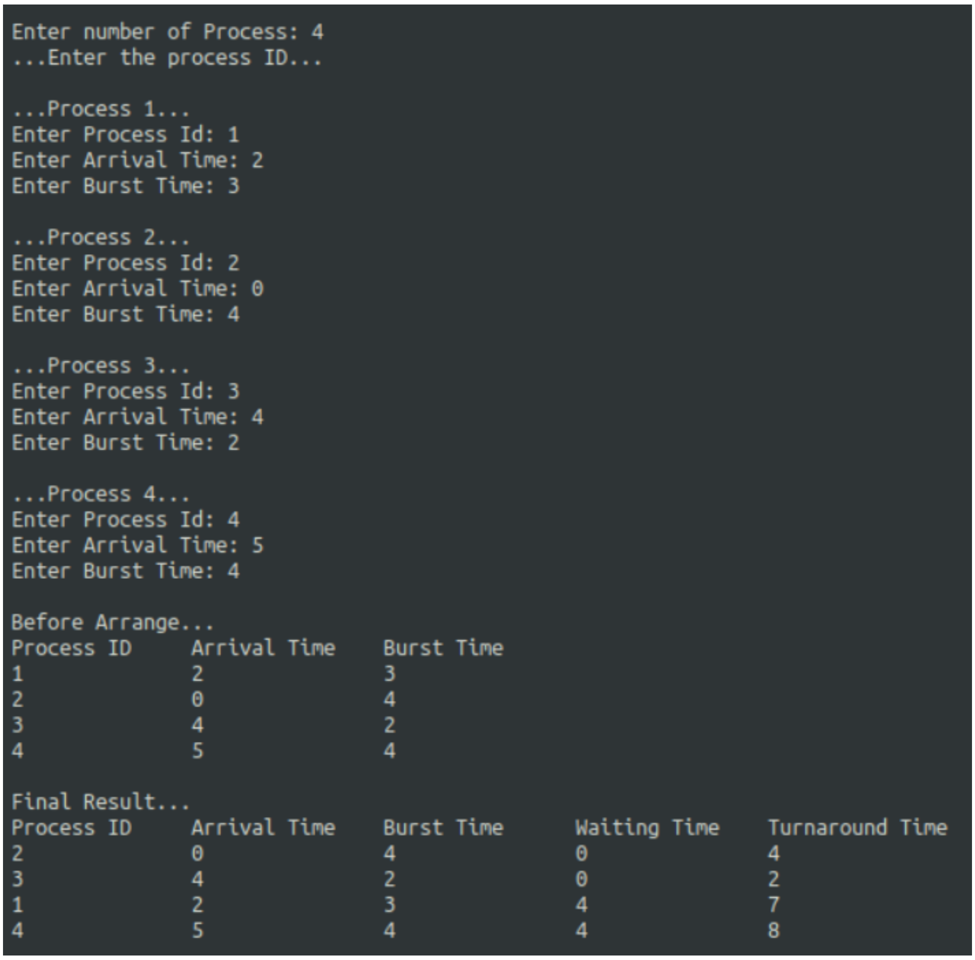
```
#include <iostream>
using namespace std;
int mat[10][6];
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void arrangeArrival(int num, int mat[][6])
{
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num - i - 1; j++)
        {
            if (mat[j][1] > mat[j + 1][1])
            {
                for (int k = 0; k < 5; k++)
                {
                    swap(mat[j][k], mat[j + 1][k]);
                }
            }
        }
    }
}
```



```
void completionTime(int num, int mat[][6])
{
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];
    for (int i = 1; i < num; i++)
    {
        temp = mat[i - 1][3];
        int low = mat[i][2];
        for (int j = i; j < num; j++)
        {
            if (temp >= mat[j][1] && low >= mat[j][2])
            {
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for (int k = 0; k < 6; k++)
        {
            swap(mat[val][k], mat[i][k]);
        }
    }
}
```

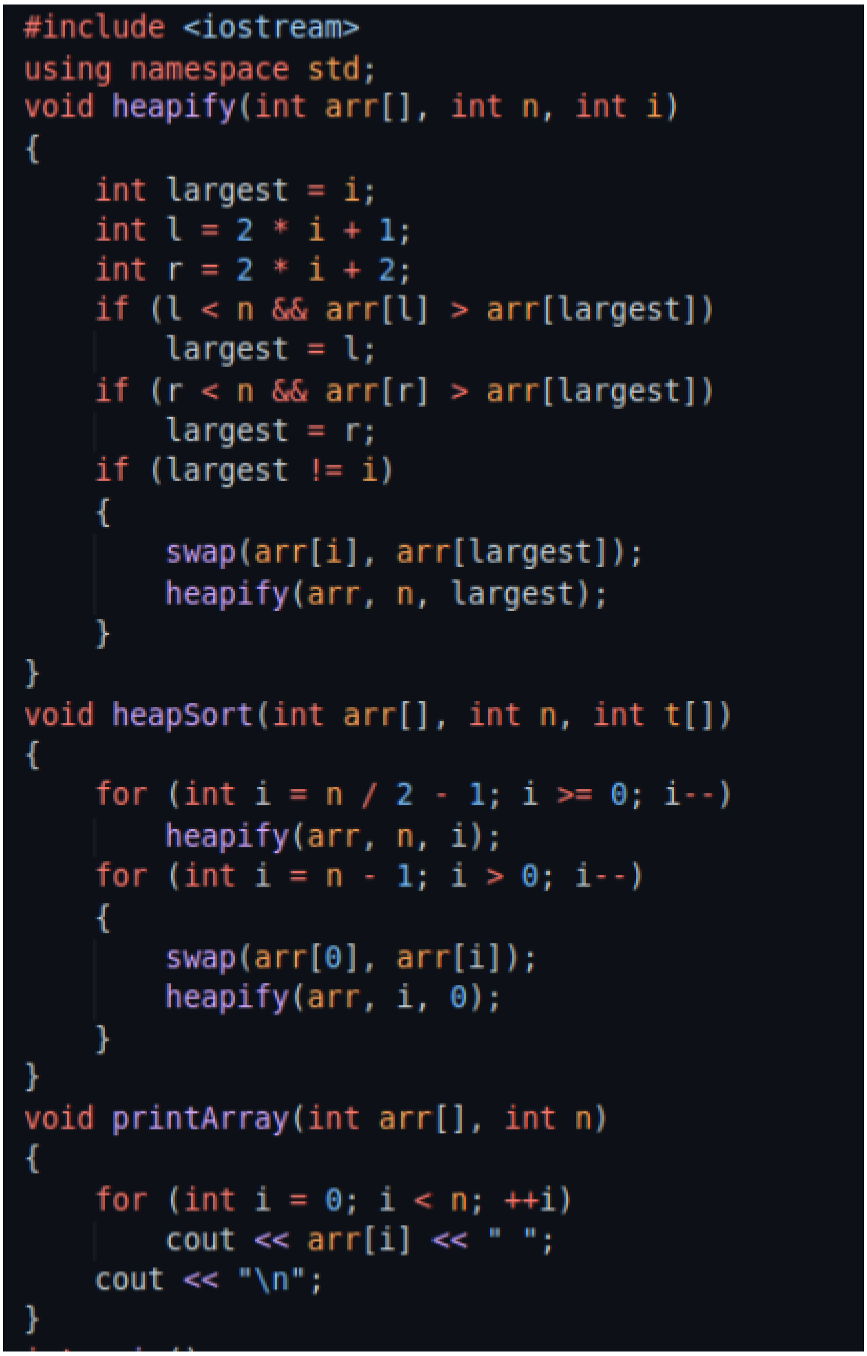


```
int main()
{
    int num, temp;
    cout << "\n Enter number of Process: ";
    cin >> num;
    cout << " ...Enter the process ID...\n";
    for (int i = 0; i < num; i++)
    {
        cout << "\n ...Process " << i + 1 << "...\n";
        cout << " Enter Process Id: ";
        cin >> mat[i][0];
        cout << " Enter Arrival Time: ";
        cin >> mat[i][1];
        cout << " Enter Burst Time: ";
        cin >> mat[i][2];
    }
    cout << "\n Before Arrange...\n Process ID\tArrival Time\tBurst Time\n";
    for (int i = 0; i < num; i++)
        cout << " " << mat[i][0] << "\t\t" << mat[i][1] << "\t\t" << mat[i][2] << "\n";
    arrangeArrival(num, mat);
    completionTime(num, mat);
    cout << "\n Final Result...\n";
    cout << " Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
    for (int i = 0; i < num; i++)
        cout << " " << mat[i][0] << "\t\t" << mat[i][1] << "\t\t" << mat[i][2] << "\t\t" << mat[i][4] << "\t\t" << mat[i][5] << "\n";
    cout << endl;
}
```

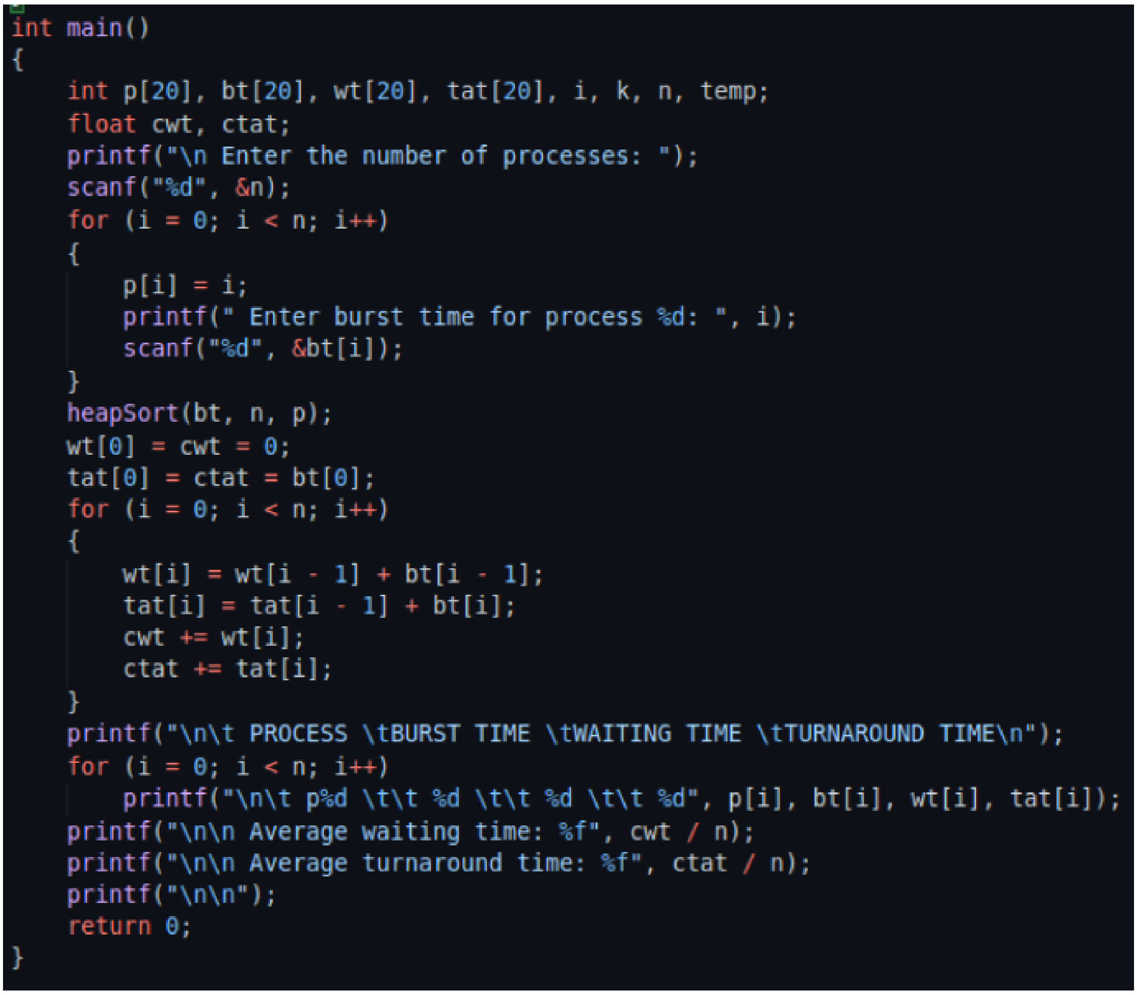


```
Enter number of Process: 4
...Enter the process ID...

...Process 1...
Enter Process Id: 1
Enter Arrival Time: 2
Enter Burst Time: 3

...Process 2...
Enter Process Id: 2
Enter Arrival Time: 0
Enter Burst Time: 4

...Process 3...
Enter Process Id: 3
Enter Arrival Time: 4
Enter Burst Time: 2

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

Before Arrange...
Process ID     Arrival Time    Burst Time
1              2               3
2              0               4
3              4               2
4              5               4

Final Result...
Process ID     Arrival Time    Burst Time      Waiting Time    Turnaround Time
2              0               4               0               4
3              4               2               0               2
1              2               3               4               7
4              5               4               4               8
```

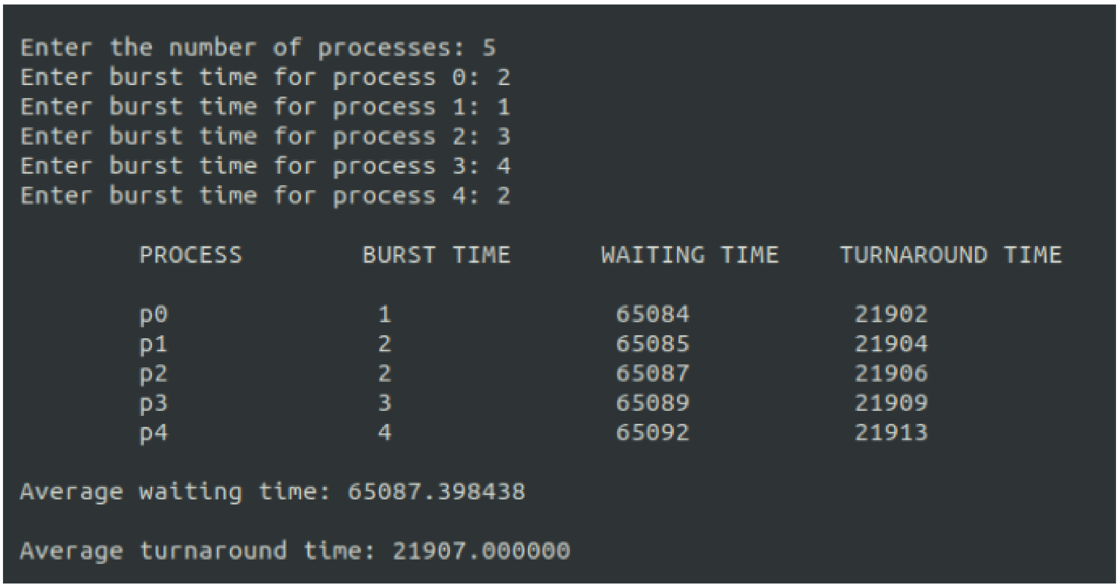
10. SJF: using heap



```
#include <iostream>
using namespace std;
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n, int t[])
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--)
    {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}
```



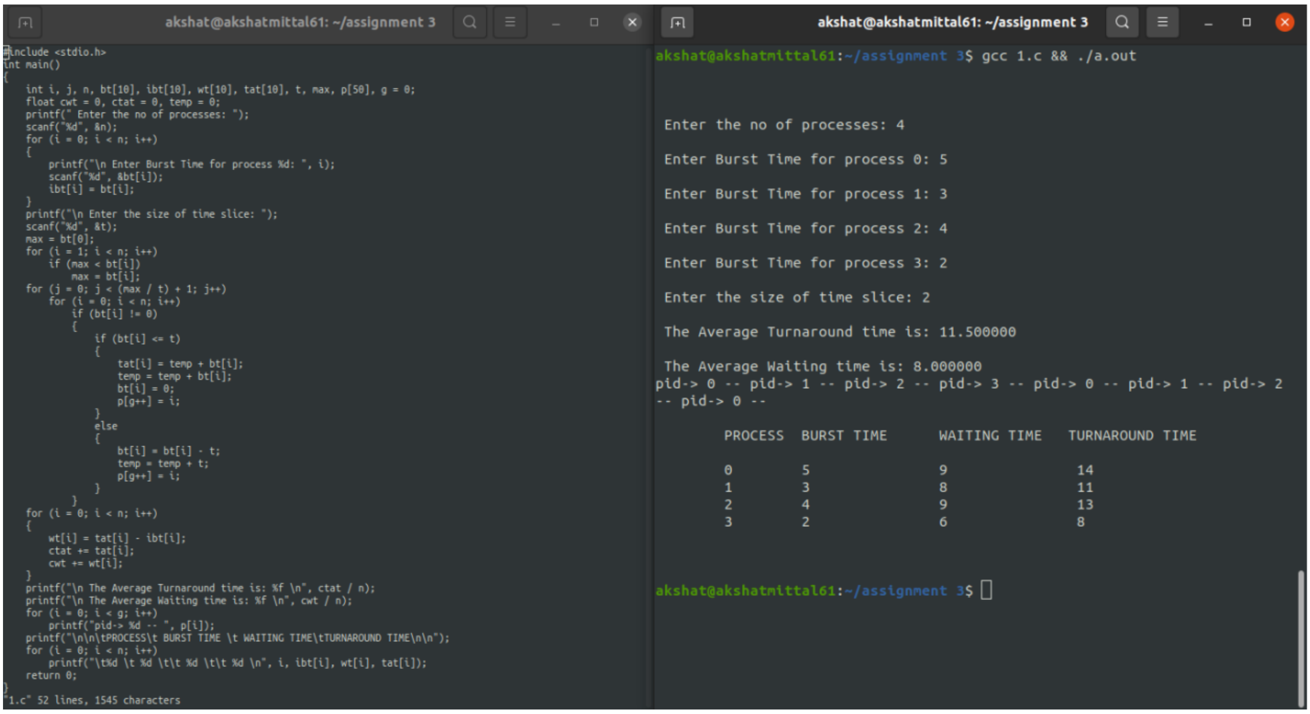
```
int main()
{
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float cwt, ctat;
    printf("\n Enter the number of processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        p[i] = i;
        printf(" Enter burst time for process %d: ", i);
        scanf("%d", &bt[i]);
    }
    heapSort(bt, n, p);
    wt[0] = cwt = 0;
    tat[0] = ctat = bt[0];
    for (i = 0; i < n; i++)
    {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        cwt += wt[i];
        ctat += tat[i];
    }
    printf("\n\t PROCESS \tBURST TIME \tWAITING TIME \tTURNAROUND TIME\n");
    for (i = 0; i < n; i++)
        printf("\n\t p%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
    printf("\n\n Average waiting time: %f", cwt / n);
    printf("\n\n Average turnaround time: %f", ctat / n);
    printf("\n\n");
    return 0;
}
```



```
Enter the number of processes: 5
Enter burst time for process 0: 2
Enter burst time for process 1: 1
Enter burst time for process 2: 3
Enter burst time for process 3: 4
Enter burst time for process 4: 2

        PROCESS        BURST TIME      WAITING TIME    TURNAROUND TIME

        p0              1               65084           21902
        p1              2               65085           21904
        p2              2               65087           21906
        p3              3               65089           21909
        p4              4               65092           21913

Average waiting time: 65087.398438

Average turnaround time: 21907.000000
```

11. Round Robin with same arrival time

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to implement, and starvation-free as all processes get fair share of CPU. One of the most commonly used technique in CPU scheduling as a core. It is preemptive as processes are assigned CPU only for a fixed slice of time at most.


akshat@akshatmittal61: ~/assignment 3
akshat@akshatmittal61: ~/assignment 3

```c
#include <stdio.h>
int main()
{
    int i, j, n, bt[10], ibt[10], wt[10], tat[10], t, max, p[50], g = 0;
    float cwt = 0, ctat = 0, temp = 0;
    printf(" Enter the no of processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\n Enter Burst Time for process %d: ", i);
        scanf("%d", &bt[i]);
        ibt[i] = bt[i];
    }
    printf("\n Enter the size of time slice: ");
    scanf("%d", &t);
    max = bt[0];
    for (i = 1; i < n; i++)
        if (max < bt[i])
            max = bt[i];
    for (j = 0; j < (max / t) + 1; j++)
        for (i = 0; i < n; i++)
            if (bt[i] != 0)
            {
                if (bt[i] <= t)
                {
                    tat[i] = temp + bt[i];
                    temp = temp + bt[i];
                    bt[i] = 0;
                    p[g++] = i;
                }
                else
                {
                    bt[i] = bt[i] - t;
                    temp = temp + t;
                    p[g++] = i;
                }
            }
    for (i = 0; i < n; i++)
    {
        wt[i] = tat[i] - ibt[i];
        ctat += tat[i];
        cwt += wt[i];
    }
    printf("\n The Average Turnaround time is: %f \n", ctat / n);
    printf("\n The Average Waiting time is: %f \n", cwt / n);
    for (i = 0; i < g; i++)
        printf("pid-> %d -- ", p[i]);
    printf("\n\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n\n");
    for (i = 0; i < n; i++)
        printf("\t%d \t %d \t\t %d \t\t %d \n", i, ibt[i], wt[i], tat[i]);
    return 0;
}
"1.c" 52 lines, 1545 characters
```

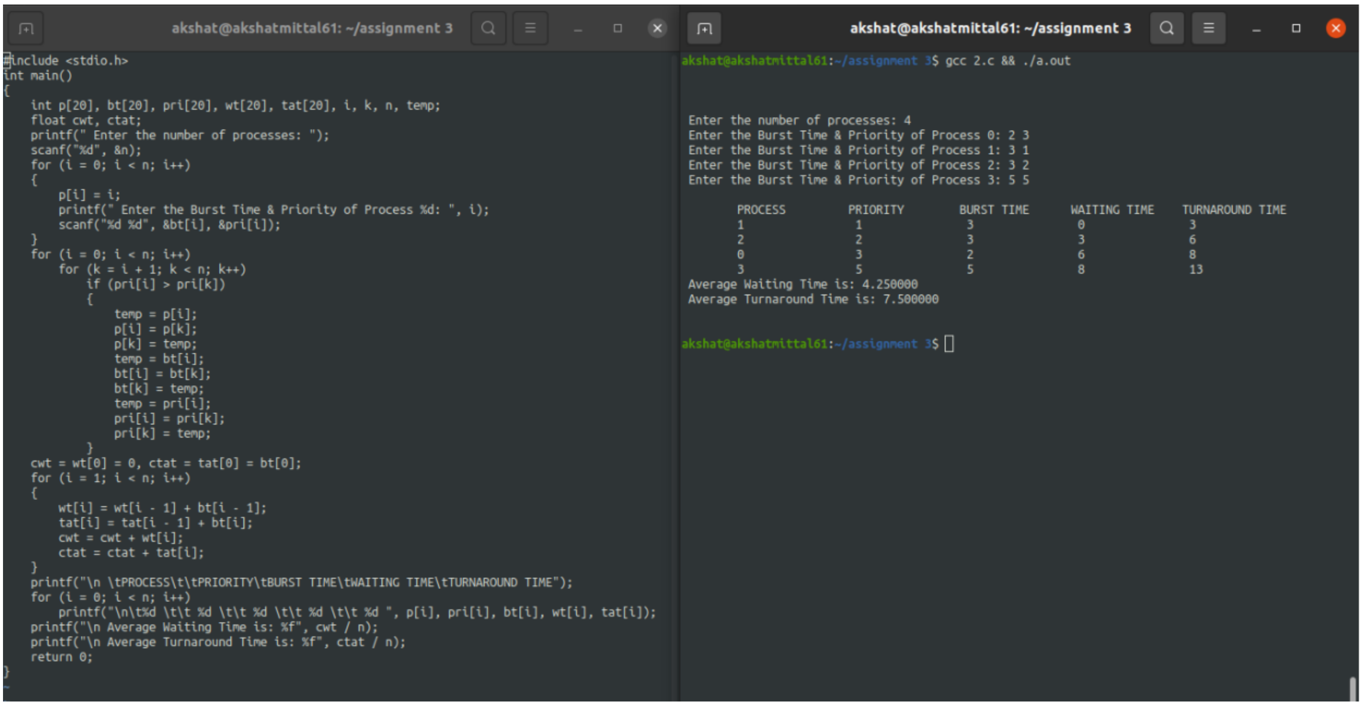
```
akshat@akshatmittal61:~/assignment 3$ gcc 1.c && ./a.out


 Enter the no of processes: 4

 Enter Burst Time for process 0: 5

 Enter Burst Time for process 1: 3

 Enter Burst Time for process 2: 4

 Enter Burst Time for process 3: 2

 Enter the size of time slice: 2

 The Average Turnaround time is: 11.500000

 The Average Waiting time is: 8.000000
pid-> 0 -- pid-> 1 -- pid-> 2 -- pid-> 3 -- pid-> 0 -- pid-> 1 -- pid-> 2
-- pid-> 0 --

        PROCESS  BURST TIME       WAITING TIME   TURNAROUND TIME

        0        5                9               14
        1        3                8               11
        2        4                9               13
        3        2                6               8


akshat@akshatmittal61:~/assignment 3$ 
```

12. Priority scheduling: same arrival time

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis.

Implementation:

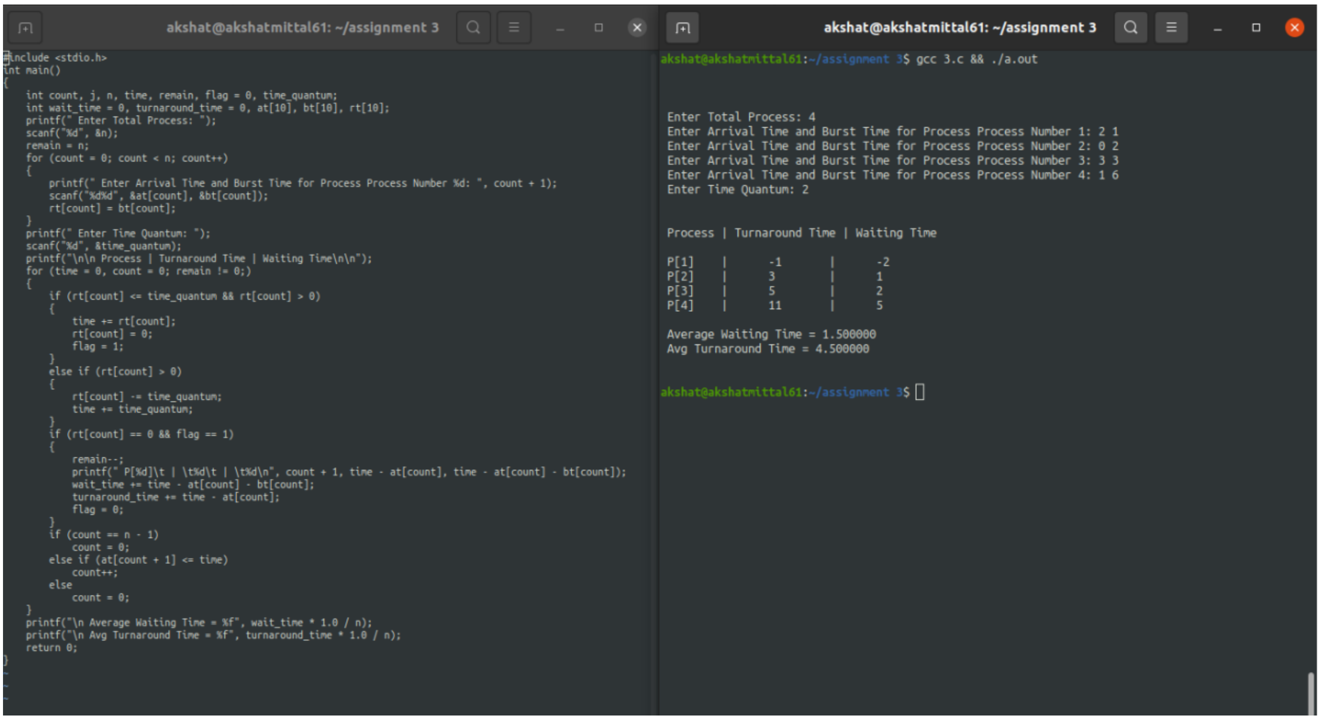
a) First input the processes with their arrival time, burst time and priority.
b) First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.
c) Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority).
d) If two process priority are same then sort according to process number.
e) They will clearly mention, which number will have higher priority and which number will have lower priority. Once all the processes have been arrived.


akshat@akshatmittal61: ~/assignment 3
akshat@akshatmittal61: ~/assignment 3

```c
#include <stdio.h>
int main()
{
    int p[20], bt[20], pri[20], wt[20], tat[20], i, k, n, temp;
    float cwt, ctat;
    printf(" Enter the number of processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        p[i] = i;
        printf(" Enter the Burst Time & Priority of Process %d: ", i);
        scanf("%d %d", &bt[i], &pri[i]);
    }
    for (i = 0; i < n; i++)
        for (k = i + 1; k < n; k++)
            if (pri[i] > pri[k])
            {
                temp = p[i];
                p[i] = p[k];
                p[k] = temp;
                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;
                temp = pri[i];
                pri[i] = pri[k];
                pri[k] = temp;
            }
    cwt = wt[0] = 0, ctat = tat[0] = bt[0];
    for (i = 1; i < n; i++)
    {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        cwt = cwt + wt[i];
        ctat = ctat + tat[i];
    }
    printf("\n \tPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
    for (i = 0; i < n; i++)
        printf("\n\t%d \t\t %d \t\t %d \t\t %d \t\t %d ", p[i], pri[i], bt[i], wt[i], tat[i]);
    printf("\n Average Waiting Time is: %f", cwt / n);
    printf("\n Average Turnaround Time is: %f", ctat / n);
    return 0;
}
~
```

```
akshat@akshatmittal61:~/assignment 3$ gcc 2.c && ./a.out



 Enter the number of processes: 4
 Enter the Burst Time & Priority of Process 0: 2 3
 Enter the Burst Time & Priority of Process 1: 3 1
 Enter the Burst Time & Priority of Process 2: 3 2
 Enter the Burst Time & Priority of Process 3: 5 5

        PROCESS         PRIORITY        BURST TIME      WAITING TIME    TURNAROUND TIME
        1                1               3               0               3
        2                2               3               3               6
        0                3               2               6               8
        3                5               5               8               13
 Average Waiting Time is: 4.250000
 Average Turnaround Time is: 7.500000


akshat@akshatmittal61:~/assignment 3$ 
```

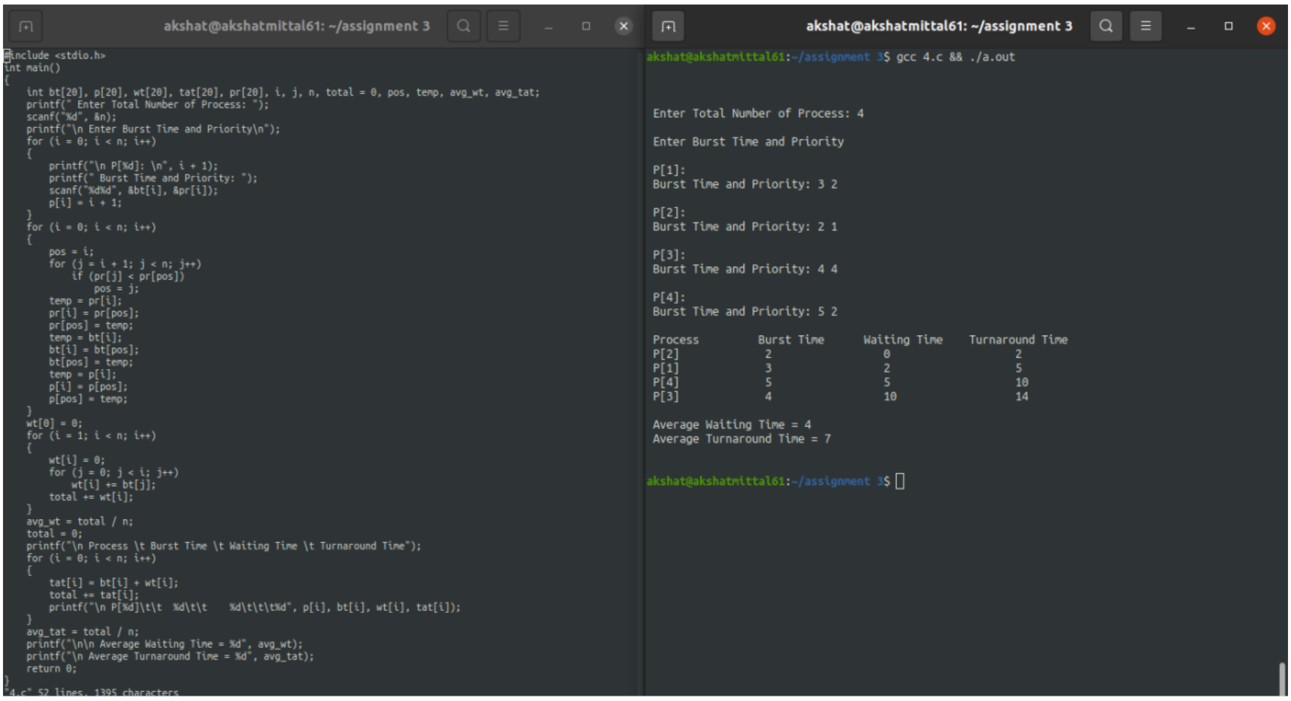
13. Round Robin: Different Arrival Time


akshat@akshatmittal61: ~/assignment 3
akshat@akshatmittal61: ~/assignment 3

```c
#include <stdio.h>
int main()
{
    int count, j, n, time, remain, flag = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, at[10], bt[10], rt[10];
    printf(" Enter Total Process: ");
    scanf("%d", &n);
    remain = n;
    for (count = 0; count < n; count++)
    {
        printf(" Enter Arrival Time and Burst Time for Process Process Number %d: ", count + 1);
        scanf("%d%d", &at[count], &bt[count]);
        rt[count] = bt[count];
    }
    printf(" Enter Time Quantum: ");
    scanf("%d", &time_quantum);
    printf("\n\n Process | Turnaround Time | Waiting Time\n\n");
    for (time = 0, count = 0; remain != 0;)
    {
        if (rt[count] <= time_quantum && rt[count] > 0)
        {
            time += rt[count];
            rt[count] = 0;
            flag = 1;
        }
        else if (rt[count] > 0)
        {
            rt[count] -= time_quantum;
            time += time_quantum;
        }
        if (rt[count] == 0 && flag == 1)
        {
            remain--;
            printf(" P[%d]\t | \t%d\t | \t%d\n", count + 1, time - at[count], time - at[count] - bt[count]);
            wait_time += time - at[count] - bt[count];
            turnaround_time += time - at[count];
            flag = 0;
        }
        if (count == n - 1)
            count = 0;
        else if (at[count + 1] <= time)
            count++;
        else
            count = 0;
    }
    printf("\n Average Waiting Time = %f", wait_time * 1.0 / n);
    printf("\n Avg Turnaround Time = %f", turnaround_time * 1.0 / n);
    return 0;
}
~
~
~
```

```
akshat@akshatmittal61:~/assignment 3$ gcc 3.c && ./a.out


 Enter Total Process: 4
 Enter Arrival Time and Burst Time for Process Process Number 1: 2 1
 Enter Arrival Time and Burst Time for Process Process Number 2: 0 2
 Enter Arrival Time and Burst Time for Process Process Number 3: 3 3
 Enter Arrival Time and Burst Time for Process Process Number 4: 1 6
 Enter Time Quantum: 2


 Process | Turnaround Time | Waiting Time

 P[1]    |       -1        |       -2
 P[2]    |       3         |       1
 P[3]    |       5         |       2
 P[4]    |       11        |       5

 Average Waiting Time = 1.500000
 Avg Turnaround Time = 4.500000

akshat@akshatmittal61:~/assignment 3$ ▯
```

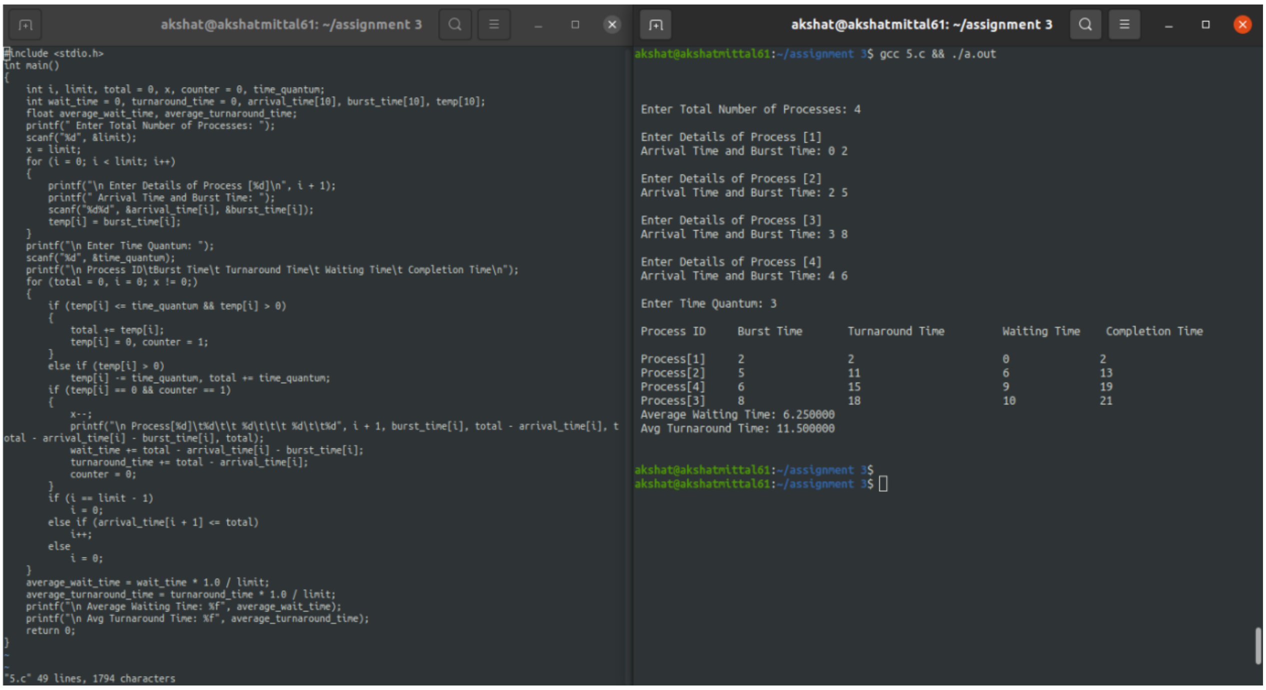
14. Priority scheduling: different arrival time


akshat@akshatmittal61: ~/assignment 3
akshat@akshatmittal61: ~/assignment 3

```c
#include <stdio.h>
int main()
{
    int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total = 0, pos, temp, avg_wt, avg_tat;
    printf(" Enter Total Number of Process: ");
    scanf("%d", &n);
    printf("\n Enter Burst Time and Priority\n");
    for (i = 0; i < n; i++)
    {
        printf("\n P[%d]: \n", i + 1);
        printf(" Burst Time and Priority: ");
        scanf("%d%d", &bt[i], &pr[i]);
        p[i] = i + 1;
    }
    for (i = 0; i < n; i++)
    {
        pos = i;
        for (j = i + 1; j < n; j++)
            if (pr[j] < pr[pos])
                pos = j;
        temp = pr[i];
        pr[i] = pr[pos];
        pr[pos] = temp;
        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 = total / n;
    total = 0;
    printf("\n Process \t Burst Time \t Waiting Time \t Turnaround Time");
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
        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 = total / n;
    printf("\n\n Average Waiting Time = %d", avg_wt);
    printf("\n Average Turnaround Time = %d", avg_tat);
    return 0;
}
"4.c" 52 lines, 1395 characters
```

```
akshat@akshatmittal61:~/assignment 3$ gcc 4.c && ./a.out


 Enter Total Number of Process: 4

 Enter Burst Time and Priority

 P[1]:
 Burst Time and Priority: 3 2

 P[2]:
 Burst Time and Priority: 2 1

 P[3]:
 Burst Time and Priority: 4 4

 P[4]:
 Burst Time and Priority: 5 2

 Process         Burst Time      Waiting Time    Turnaround Time
 P[2]             2               0                      2
 P[1]             3               2                      5
 P[4]             5               5                      10
 P[3]             4               10                     14

 Average Waiting Time = 4
 Average Turnaround Time = 7

akshat@akshatmittal61:~/assignment 3$ ▯
```

15. Premptive scheduling using min heap


akshat@akshatmittal61: ~/assignment 3
akshat@akshatmittal61: ~/assignment 3

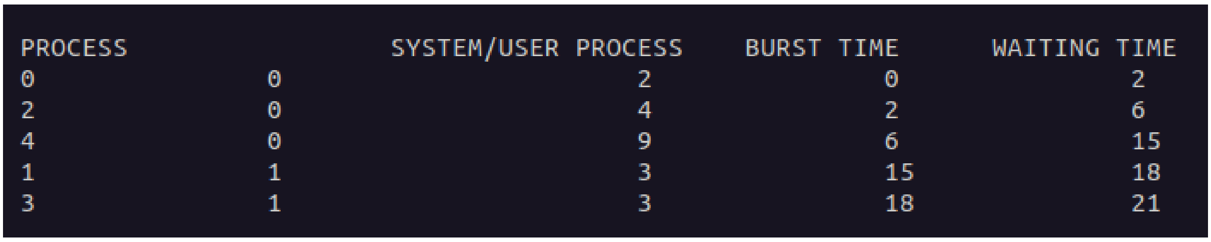
```
#include <stdio.h>
int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
    float average_wait_time, average_turnaround_time;
    printf(" Enter Total Number of Processes: ");
    scanf("%d", &limit);
    x = limit;
    for (i = 0; i < limit; i++)
    {
        printf("\n Enter Details of Process [%d]\n", i + 1);
        printf(" Arrival Time and Burst Time: ");
        scanf("%d%d", &arrival_time[i], &burst_time[i]);
        temp[i] = burst_time[i];
    }
    printf("\n Enter Time Quantum: ");
    scanf("%d", &time_quantum);
    printf("\n Process ID\tBurst Time\t Turnaround Time\t Waiting Time\t Completion Time\n");
    for (total = 0, i = 0; x != 0;)
    {
        if (temp[i] <= time_quantum && temp[i] > 0)
        {
            total += temp[i];
            temp[i] = 0, counter = 1;
        }
        else if (temp[i] > 0)
            temp[i] -= time_quantum, total += time_quantum;
        if (temp[i] == 0 && counter == 1)
        {
            x--;
            printf("\n Process[%d]\t%d\t\t %d\t\t\t %d\t\t%d", i + 1, burst_time[i], total - arrival_time[i], t
otal - arrival_time[i] - burst_time[i], total);
            wait_time += total - arrival_time[i] - burst_time[i];
            turnaround_time += total - arrival_time[i];
            counter = 0;
        }
        if (i == limit - 1)
            i = 0;
        else if (arrival_time[i + 1] <= total)
            i++;
        else
            i = 0;
    }
    average_wait_time = wait_time * 1.0 / limit;
    average_turnaround_time = turnaround_time * 1.0 / limit;
    printf("\n Average Waiting Time: %f", average_wait_time);
    printf("\n Avg Turnaround Time: %f", average_turnaround_time);
    return 0;
}
~
~
"5.c" 49 lines, 1794 characters
```

```
akshat@akshatmittal61:~/assignment 3$ gcc 5.c && ./a.out



 Enter Total Number of Processes: 4

 Enter Details of Process [1]
 Arrival Time and Burst Time: 0 2

 Enter Details of Process [2]
 Arrival Time and Burst Time: 2 5

 Enter Details of Process [3]
 Arrival Time and Burst Time: 3 8

 Enter Details of Process [4]
 Arrival Time and Burst Time: 4 6

 Enter Time Quantum: 3

 Process ID     Burst Time       Turnaround Time         Waiting Time    Completion Time

 Process[1]     2                2                       0               2
 Process[2]     5                11                      6               13
 Process[4]     6                15                      9               19
 Process[3]     8                18                      10              21
 Average Waiting Time: 6.250000
 Avg Turnaround Time: 11.500000

akshat@akshatmittal61:~/assignment 3$
akshat@akshatmittal61:~/assignment 3$ 
```

Course: Operating Systems	**Course Code:** CSC403
Roll No: 20107	**Teacher Name:** Dr. Sahil
Student Name: Akshat Mittal	**E-mail:** 20107@iiitu.ac.in

TASK 1 : i:- Multilevel queue where every process have same arrival time and user process have less priority then system process. Implement FCFS for user process and SJF for system process.

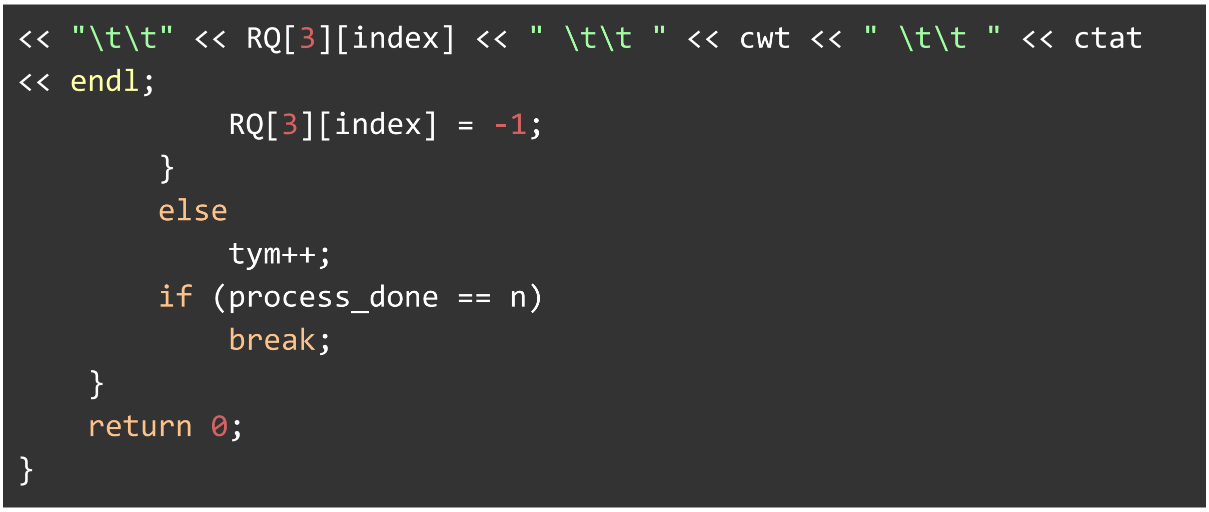
```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int n;
    float wtavg, tatavg;
    printf(" Enter the number of processes: ");
    scanf("%d", &n);
    int p[n], bt[n], su[n], wt[n], ct[n], i, k, temp;
    for (i = 0; i < n; i++)
    {
        p[i] = i;
        printf(" Enter the Burst Time of Process %d: ", i);
        scanf("%d", &bt[i]);
        printf(" System/User Process (0/1) ? ");
        scanf("%d", &su[i]);
    }
    for (i = 0; i < n; i++)
        for (k = i + 1; k < n; k++)
            if (su[i] > su[k])
            {
                swap(&p[i], &p[k]);
                swap(&bt[i], &bt[k]);
                swap(&su[i], &su[k]);
            }
    wtavg = wt[0] = 0;
    int sy = 0, us = 0;
    for (i = 0; i < n; i++)
    {
        if (su[i] == 1)
            us++;
        else
            sy++;
    }
    int stwt = 0, scpt = 0;
    for (i = 0; i < sy; i++)
        for (k = i + 1; k < sy; k++)
            if (bt[i] > bt[k])
            {
                swap(&p[i], &p[k]);
                swap(&bt[i], &bt[k]);
                swap(&su[i], &su[k]);
            }
    for (i = sy; i < n; i++)
        for (k = i + 1; k < n; k++)
            if (p[i] > p[k])
            {
                swap(&p[i], &p[k]);
                swap(&bt[i], &bt[k]);
                swap(&su[i], &su[k]);
            }
    for (i = 0; i < n; i++)
    {
        wt[i] = stwt;
        ct[i] = stwt + bt[i];
        stwt += bt[i];
    }
    printf(" \n PROCESS\t\t SYSTEM/USER PROCESS \tBURST
TIME\tWAITING TIME\tCOMPLETION TIME");
    for (i = 0; i < n; i++)
        printf(" \n %d \t\t %d \t\t\t %d \t\t %d \t\t %d", p[i],
su[i], bt[i], wt[i], ct[i]);
    return 0;
}
```



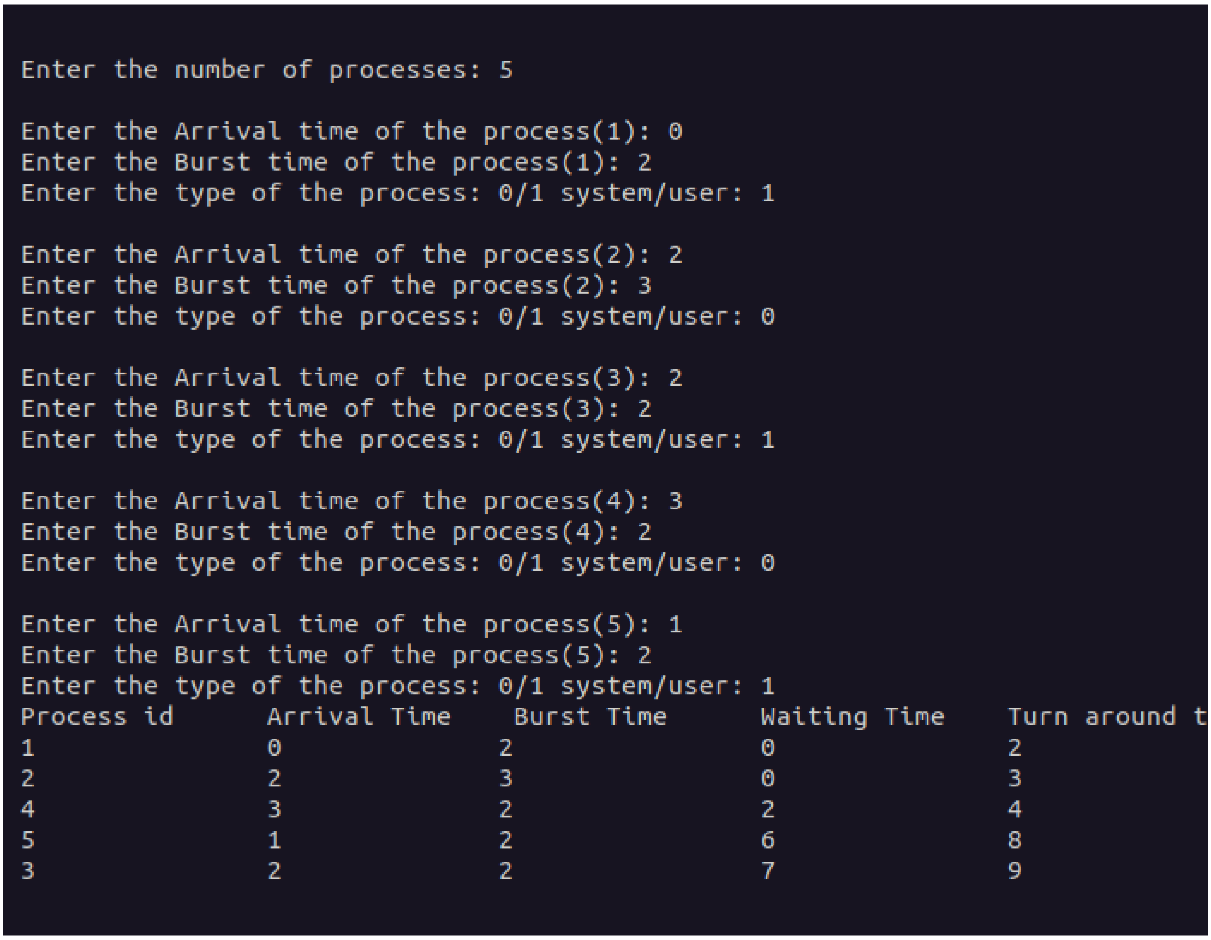
```
PROCESS                    SYSTEM/USER PROCESS    BURST TIME       WAITING TIME
0             0                      2                0                 2
2             0                      4                2                 6
4             0                      9                6                 15
1             1                      3                15                18
3             1                      3                18                21
```

ii. Multilevel queue where every process have different arrival time and user process have less priority then system process. Implement FCFS for user process and SJF for system process.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cout << " Enter the number of processes: ";
    cin >> n;
    int AT[n], type[n], BT[n];
    for (int i = 0; i < n; i++)
    {
        cout << "\n Enter the Arrival time of the process(" << i +
1 << "): ";
        cin >> AT[i];
        cout << " Enter the Burst time of the process(" << i + 1
<< "): ";
        cin >> BT[i];
        cout << " Enter the type of the process: 0/1 system/user:
";
        cin >> type[i];
    }
    int o_tym = -1, tym = 0, RQ[4][n], count = 0, process_done =
0;
    float cwt = 0, ctat = 0;
    cout << " Process id\t Arrival Time\t Burst Time\t Waiting
Time\t Turn around time\n";
    while (1)
    {
        for (int i = 0; i < n; i++)
        {
            if (AT[i] > o_tym && AT[i] <= tym)
            {
                RQ[0][count] = AT[i];
                RQ[1][count] = type[i];
                RQ[2][count] = i + 1;
                RQ[3][count] = BT[i];
                count++;
            }
        }
        o_tym = tym;
        int min = INT_MAX, index = -1;
        for (int i = 0; i < count; i++)
            if (RQ[3][i] < min && RQ[1][i] == 0 && RQ[3][i] > 0)
                min = RQ[3][i], index = i;
        if (index != -1)
        {
            process_done++;
            tym += RQ[3][index];
            ctat = tym - RQ[0][index];
            cwt = ctat - RQ[3][index];
            cout << " " << RQ[2][index] << "\t\t " << RQ[0][index]
<< "\t\t" << min << "\t\t " << cwt << "\t\t " << ctat << endl;
            RQ[3][index] = -1;
        }
        else if (process_done < count)
        {
            int min = INT_MAX, index = -1;
            for (int i = 0; i < count; i++)
                if (RQ[3][i] > 0 && RQ[1][i] == 1 && RQ[0][i] <
min)
                    min = RQ[0][i], index = i;
            process_done++;
            tym += RQ[3][index];
            ctat = tym - RQ[0][index];
            cwt = ctat - RQ[3][index];
            cout << " " << RQ[2][index] << "\t\t " << RQ[0][index]
```



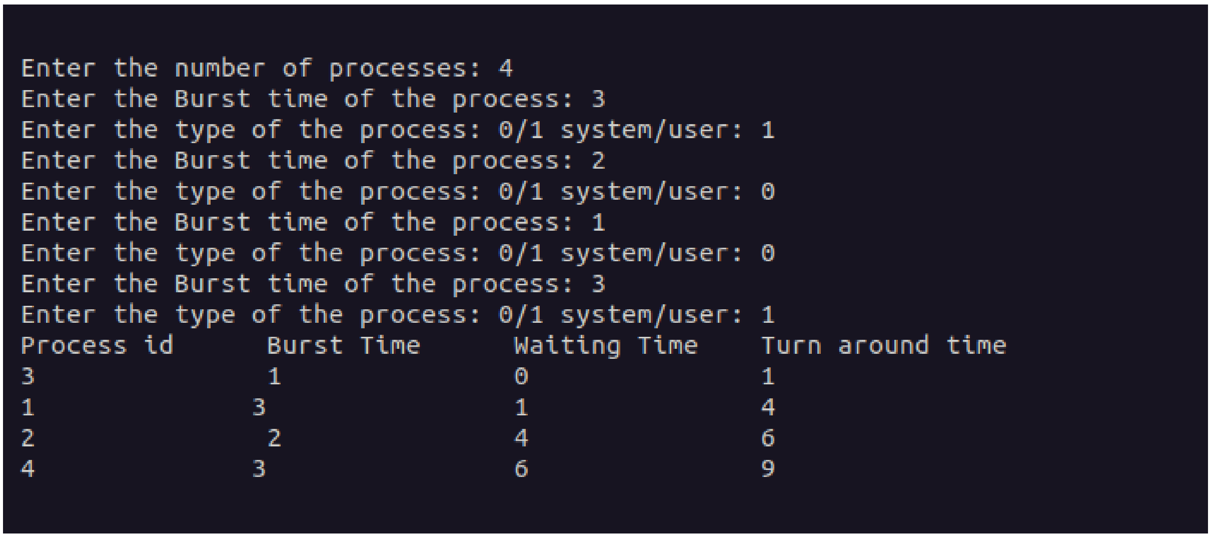
```
<< "\t\t" << RQ[3][index] << " \t\t " << cwt << " \t\t " << ctat
<< endl;
                RQ[3][index] = -1;
            }
            else
                tym++;
            if (process_done == n)
                break;
        }
        return 0;
}
```



```
Enter the number of processes: 5

Enter the Arrival time of the process(1): 0
Enter the Burst time of the process(1): 2
Enter the type of the process: 0/1 system/user: 1

Enter the Arrival time of the process(2): 2
Enter the Burst time of the process(2): 3
Enter the type of the process: 0/1 system/user: 0

Enter the Arrival time of the process(3): 2
Enter the Burst time of the process(3): 2
Enter the type of the process: 0/1 system/user: 1

Enter the Arrival time of the process(4): 3
Enter the Burst time of the process(4): 2
Enter the type of the process: 0/1 system/user: 0

Enter the Arrival time of the process(5): 1
Enter the Burst time of the process(5): 2
Enter the type of the process: 0/1 system/user: 1
Process id      Arrival Time    Burst Time      Waiting Time    Turn around t
1               0              2               0               2
2               2              3               0               3
4               3              2               2               4
5               1              2               6               8
3               2              2               7               9
```

Task 2. Multilevel Feedback queue with time counter for each queue

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cout << " Enter the number of processes: ";
    cin >> n;
    int type[n], BT[n];
    for (int i = 0; i < n; i++)
    {
        cout << " Enter the Burst time of the process: ";
        cin >> BT[i];
        cout << " Enter the type of the process: 0/1 system/user: ";
        cin >> type[i];
    }
    int RQ[3][n], count = 0, process_done = 0, s_count = 0, u_count = 0;
    float cwt = 0, ctat = 0;
    cout << " Process id\t Burst Time\t Waiting Time\t Turn around time\n";
    for (int i = 0; i < n; i++)
    {
        if (BT[i] <= 6 && type[i] == 0)
        {
            RQ[0][count] = BT[i];
            RQ[1][count] = type[i];
            RQ[2][count] = i + 1;
            count++;
            s_count++;
        }
        else if (BT[i] <= 10 && type[i] == 1)
        {
            RQ[0][count] = BT[i];
            RQ[1][count] = type[i];
            RQ[2][count] = i + 1;
            count++;
            u_count++;
        }
    }
    int p = 0;
    while (1)
    {
        int min = INT_MAX, index = -1;
        if (p == 0)
        {
            if (p == 0 && s_count > 0)
            {
                for (int i = 0; i < count; i++)
                    if (RQ[0][i] < min && RQ[1][i] == 0 &&
RQ[0][i] > 0)
                        min = RQ[0][i], index = i;
                if (index != -1)
                {
                    ctat += min;
                    process_done++;
                    cout << " " << RQ[2][index] << "\t\t " << min
<< "\t\t " << cwt << "\t\t " << ctat << endl;
                    cwt += RQ[0][index];
                    RQ[0][index] = -1;
                    p = 1;
                }
                s_count--;
            }
            else
                p = 1;
        }
        else if (p == 1 && u_count > 0)
        {
            {
                u_count--;
                p = 0;
                for (int i = 0; i < count; i++)
                {
                    if (RQ[0][i] > 0 && RQ[1][i] == 1)
                    {
                        min = RQ[0][i];
                        index = i;
                        break;
                    }
                }
                ctat += RQ[0][index];
                process_done++;
                cout << " " << RQ[2][index] << "\t\t" <<
RQ[0][index] << " \t\t " << cwt << " \t\t " << ctat << endl;
                cwt += RQ[0][index];
                RQ[0][index] = -1;
            }
        }
        else
        {
            if (p == 0)
                p = 1;
            else
                p = 0;
        }
        if (process_done == count)
            break;
    }
    return 0;
}
```



```
Enter the number of processes: 4
Enter the Burst time of the process: 3
Enter the type of the process: 0/1 system/user: 1
Enter the Burst time of the process: 2
Enter the type of the process: 0/1 system/user: 0
Enter the Burst time of the process: 1
Enter the type of the process: 0/1 system/user: 0
Enter the Burst time of the process: 3
Enter the type of the process: 0/1 system/user: 1
Process id       Burst Time       Waiting Time     Turn around time
3                 1               0                1
1                3                1                4
2                 2               4                6
4                3                6                9
```

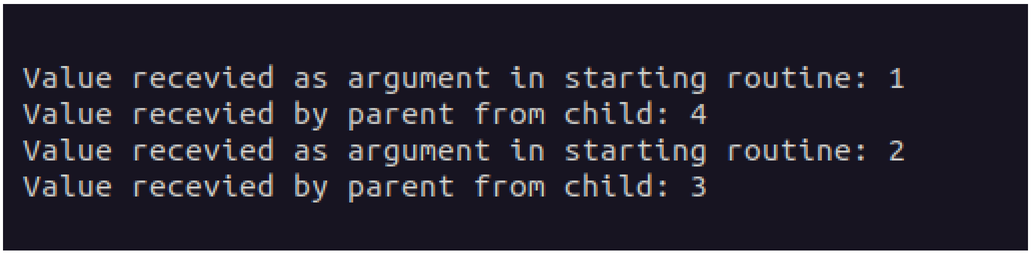
Threads

A thread of a process can be defined as a execution unit that can be a part of a process. Any process can have multiple threads.

Execution of a process by threads has multiple advantages like better responseniveness, faster context switching, resource sharing, enhanced throughput etc.

1. Creation of Thread

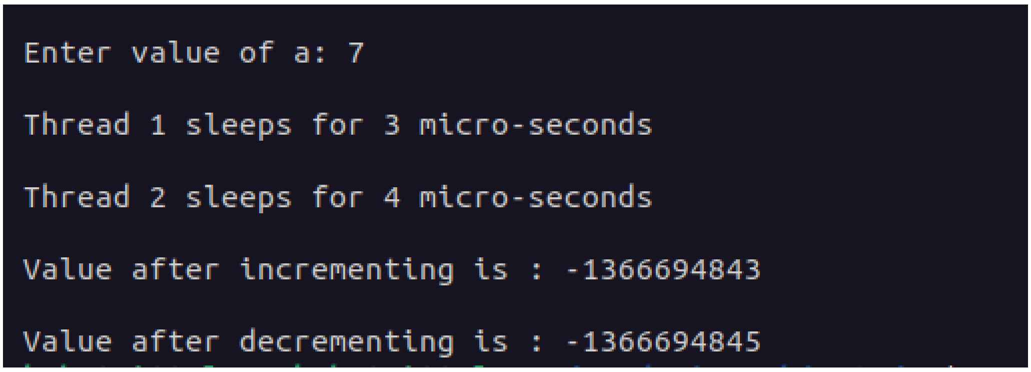
```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
int i = 3;
void *foo(void *p)
{
    printf(" Value received as argument in starting routine: ");
    printf("%i\n", *(int *)p);
    i++;
    pthread_exit(&i);
}
void *hoo(void *p)
{
    printf(" Value received as argument in starting routine: 
%i\n", *(int *)p);
    i--;
    pthread_exit(&i);
}
int main(void)
{
    pthread_t id1 = 1, id2 = 2;
    int j = 1, k = 2, *ptr1, *ptr2;
    pthread_create(&id1, NULL, foo, &j);
    pthread_join(id1, (void **)&ptr1);
    printf(" Value received by parent from child: %i\n", *ptr1);
    pthread_create(&id2, NULL, hoo, &k);
    pthread_join(id2, (void **)&ptr2);
    printf(" Value received by parent from child: %i\n", *ptr2);
    return 0;
}
```



```
Value recevied as argument in starting routine: 1
Value recevied by parent from child: 4
Value recevied as argument in starting routine: 2
Value recevied by parent from child: 3
```

2. Increment and Decrement using thread call

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
sem_t semaphore1;
sem_t semaphore2;
void *inc(void *a)
{
    sem_init(&semaphore1, 0, 1);
    sem_wait(&semaphore1);
    printf("\n Thread 1 sleeps for 3 micro-seconds\n");
    sleep(3);
    int temp = (int *)a;
    temp++;
    printf("\n Value after incrementing is : %d\n", temp);
    sem_post(&semaphore1);
}
void *dec(void *a)
{
    sem_init(&semaphore2, 0, 1);
    sem_wait(&semaphore2);
    printf("\n Thread 2 sleeps for 4 micro-seconds\n");
    sleep(4);
    int temp = (int *)a;
    temp--;
    printf("\n Value after decrementing is : %d\n", temp);
    sem_post(&semaphore2);
}
int main()
{
    int a;
    printf(" Enter value of a: ");
    scanf("%d", &a);
    pthread_t increase;
    pthread_create(&increase, NULL, &inc, &a);
    pthread_t decrease;
    pthread_create(&decrease, NULL, &dec, &a);
    pthread_exit(NULL);
    return 0;
}
```



```
Enter value of a: 7

Thread 1 sleeps for 3 micro-seconds

Thread 2 sleeps for 4 micro-seconds

Value after incrementing is : -1366694843

Value after decrementing is : -1366694845
```

Semaphore

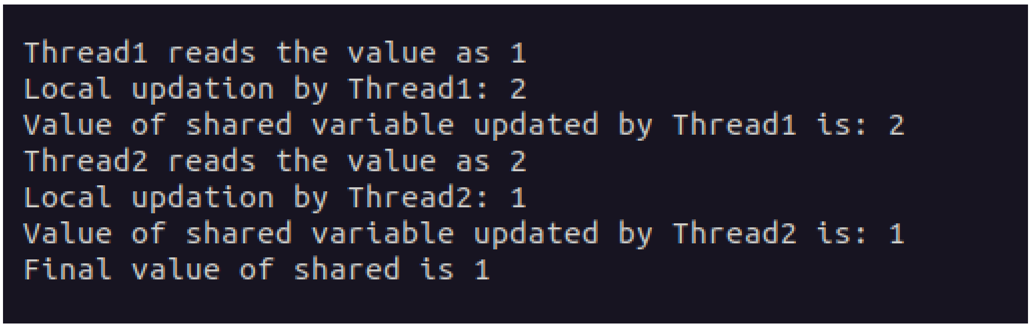
Semaphore is aninteger variable that is shared between threads for process synchronization. This variable is used to ensure that no other process enters the critical section until it is freed from the former process.

There are 2 types of semaphore:

a) Counting semaphore: It is an integer variable that can hold any value (unrestricted domain).
b) Binary Semaphore(Mutex): It can only hold 2 values: 0 and 1.

3. Synchronization in threads using semaphore

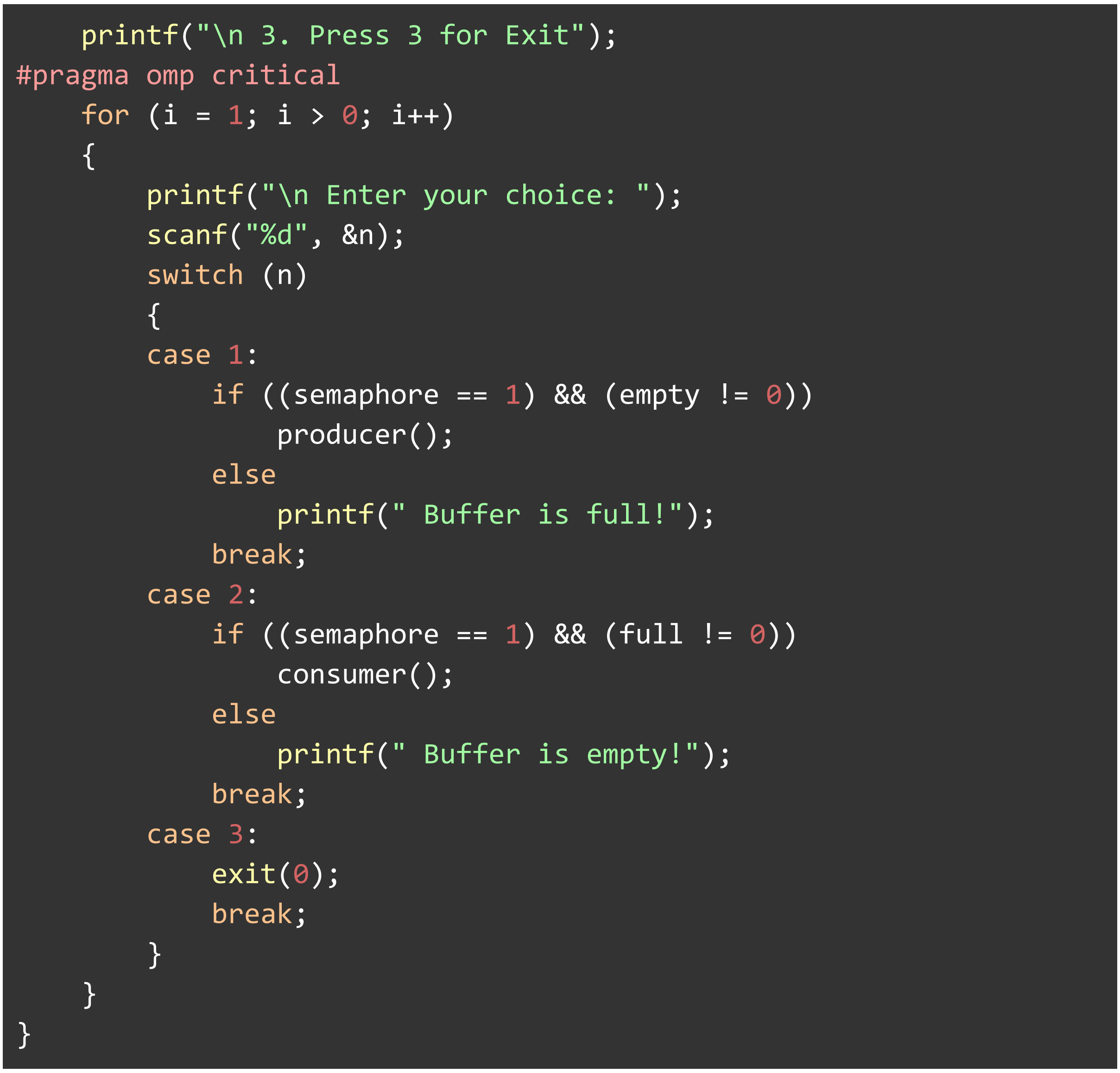
```
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
void *fun1();
void *fun2();
int shared = 1;
sem_t s;
int main()
{
    sem_init(&s, 0, 1);
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, fun1, NULL);
    pthread_create(&thread2, NULL, fun2, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf(" Final value of shared is %d\n", shared);
}
void *fun1()
{
    int x;
    sem_wait(&s);
    x = shared;
    printf(" Thread1 reads the value as %d\n", x);
    x++;
    printf(" Local updation by Thread1: %d\n", x);
    sleep(1);
    shared = x;
    printf(" Value of shared variable updated by Thread1 is:
%d\n", shared);
    sem_post(&s);
}
void *fun2()
{
    int y;
    sem_wait(&s);
    y = shared;
    printf(" Thread2 reads the value as %d\n", y);
    y--;
    printf(" Local updation by Thread2: %d\n", y);
    sleep(1);
    shared = y;
    printf(" Value of shared variable updated by Thread2 is:
%d\n", shared);
    sem_post(&s);
}
```



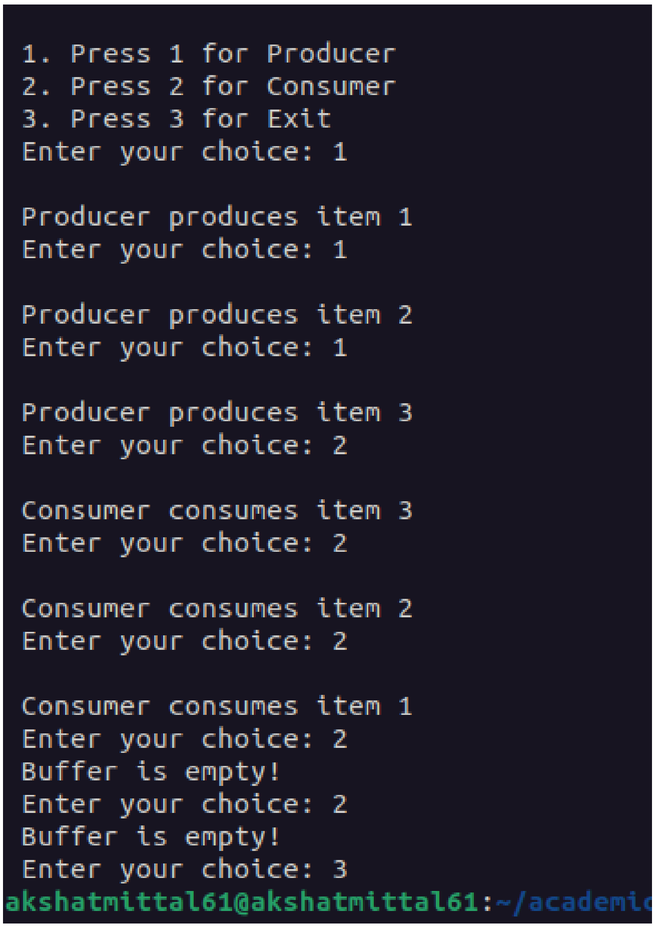
```
Thread1 reads the value as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 2
Local updation by Thread2: 1
Value of shared variable updated by Thread2 is: 1
Final value of shared is 1
```

4. Producer, Consumer Concept using semaphore

```c
#include <stdio.h>
#include <stdlib.h>
int semaphore = 1;
int full = 0;
int empty = 10, x = 0;
void producer()
{
    --semaphore;
    ++full;
    --empty;
    x++;
    printf("\n Producer produces item %d", x);
    ++semaphore;
}
void consumer()
{
    --semaphore;
    --full;
    ++empty;
    printf("\n Consumer consumes item %d", x);
    x--;
    ++semaphore;
}
int main()
{
    int n, i;
    printf("\n 1. Press 1 for Producer");
    printf("\n 2. Press 2 for Consumer");
```



```
    printf("\n 3. Press 3 for Exit");
#pragma omp critical
    for (i = 1; i > 0; i++)
    {
        printf("\n Enter your choice: ");
        scanf("%d", &n);
        switch (n)
        {
        case 1:
            if ((semaphore == 1) && (empty != 0))
                producer();
            else
                printf(" Buffer is full!");
            break;
        case 2:
            if ((semaphore == 1) && (full != 0))
                consumer();
            else
                printf(" Buffer is empty!");
            break;
        case 3:
            exit(0);
            break;
        }
    }
}
```



```
1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice: 1

Producer produces item 1
Enter your choice: 1

Producer produces item 2
Enter your choice: 1

Producer produces item 3
Enter your choice: 2

Consumer consumes item 3
Enter your choice: 2

Consumer consumes item 2
Enter your choice: 2

Consumer consumes item 1
Enter your choice: 2
Buffer is empty!
Enter your choice: 2
Buffer is empty!
Enter your choice: 3
akshatmittal61@akshatmittal61:~/academi
```

Fork

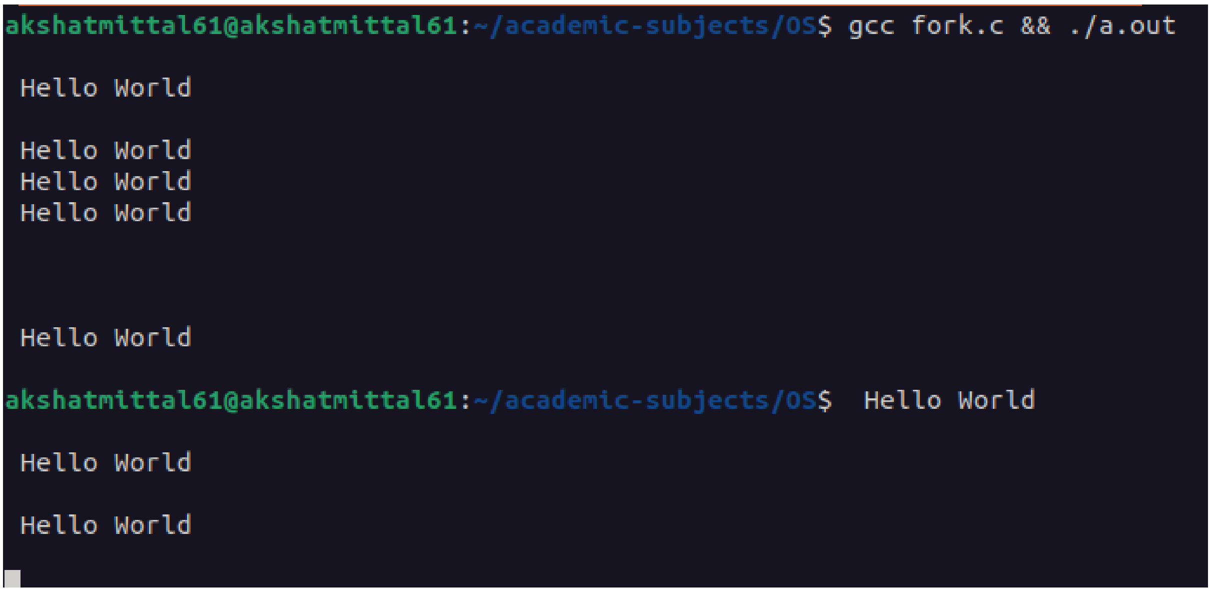
A fork is a system call that is used to create a new child process from an existing parent process. It runs concurrently with the parent process while using the same PC and same registers.

In C programming, a fork call can produce 3 output:

a) -1: if the creation of child process was unsuccessful.
b) 0: If the child process is created successfully, then the child process returns the value 0.
c) 1: If the child process is created successfully, then he parent process returns the value 1.

1. Using fork() demonstrate no. of child process executed will be 2 power (n-1) if n is no. of fork calls.

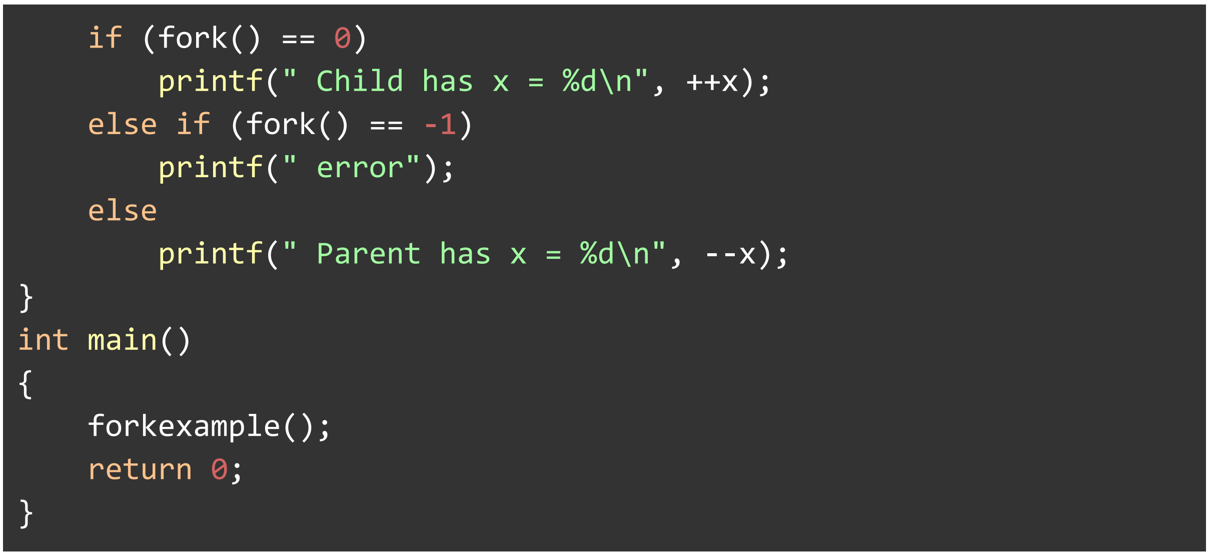
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    printf(" Hello World\n");
    return 0;
}
```



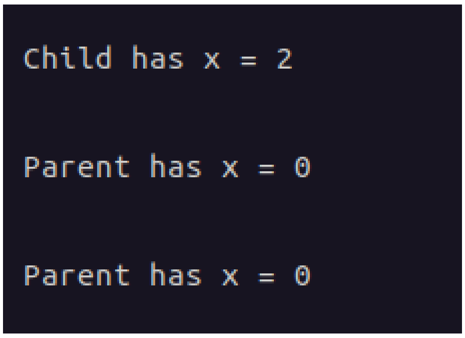
```
akshatmittal61@akshatmittal61:~/academic-subjects/OS$ gcc fork.c && ./a.out

 Hello World

 Hello World
 Hello World
 Hello World



 Hello World

akshatmittal61@akshatmittal61:~/academic-subjects/OS$  Hello World

 Hello World

 Hello World

```

2. Create a process using fork(). Trace execution of parent and child process.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
    int x = 1;
```

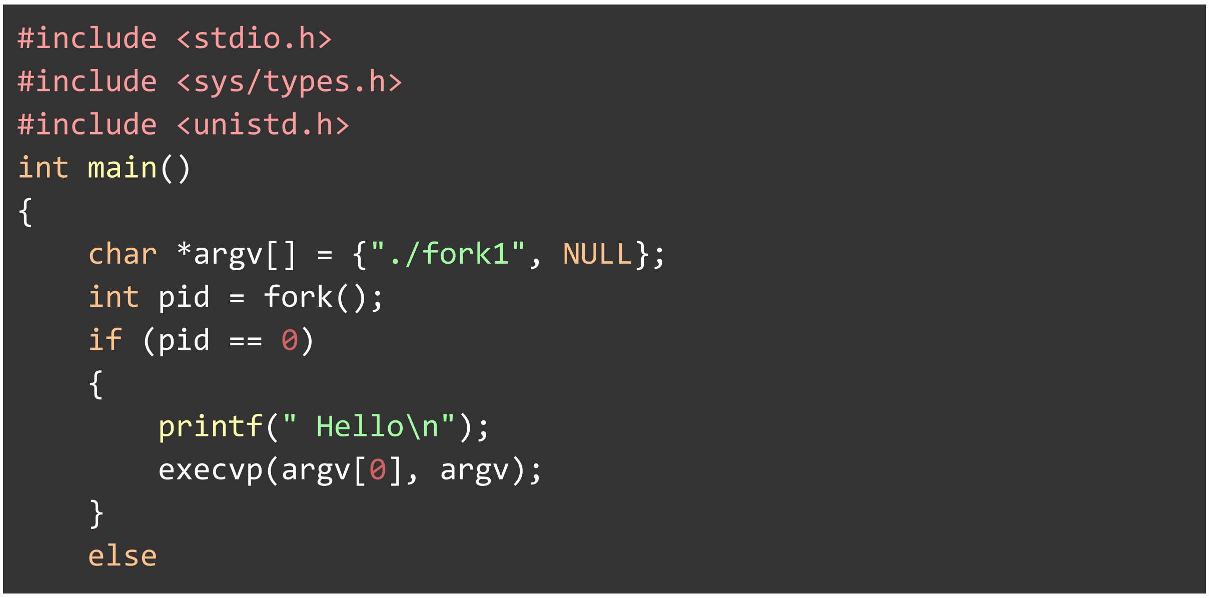


```
    if (fork() == 0)
        printf(" Child has x = %d\n", ++x);
    else if (fork() == -1)
        printf(" error");
    else
        printf(" Parent has x = %d\n", --x);
}
int main()
{
    forkexample();
    return 0;
}
```

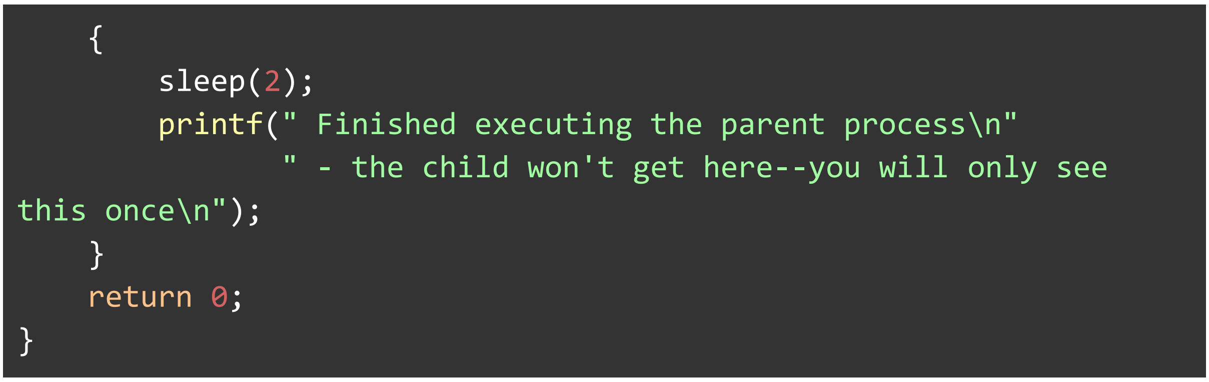


```
Child has x = 2

Parent has x = 0

Parent has x = 0
```

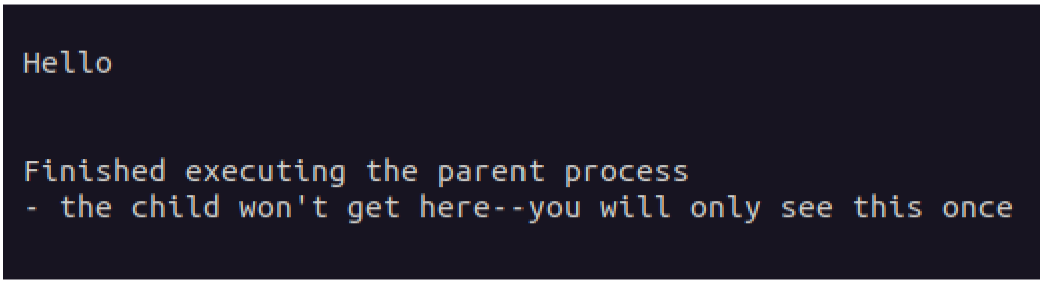
3. Replace code image of parent and child using exec() system call.



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    char *argv[] = {"./fork1", NULL};
    int pid = fork();
    if (pid == 0)
    {
        printf(" Hello\n");
        execvp(argv[0], argv);
    }
    else
```



```
    {
        sleep(2);
        printf(" Finished executing the parent process\n"
               " - the child won't get here--you will only see
this once\n");
    }
    return 0;
}
```



```
Hello


Finished executing the parent process
- the child won't get here--you will only see this once
```

Deadlock

A deadlock is a situation in which 2 process sharing the same resource are effectively preventing each other from accessing the resource, resulting in a situation in which no single process can be executed successfully.

Necessary:

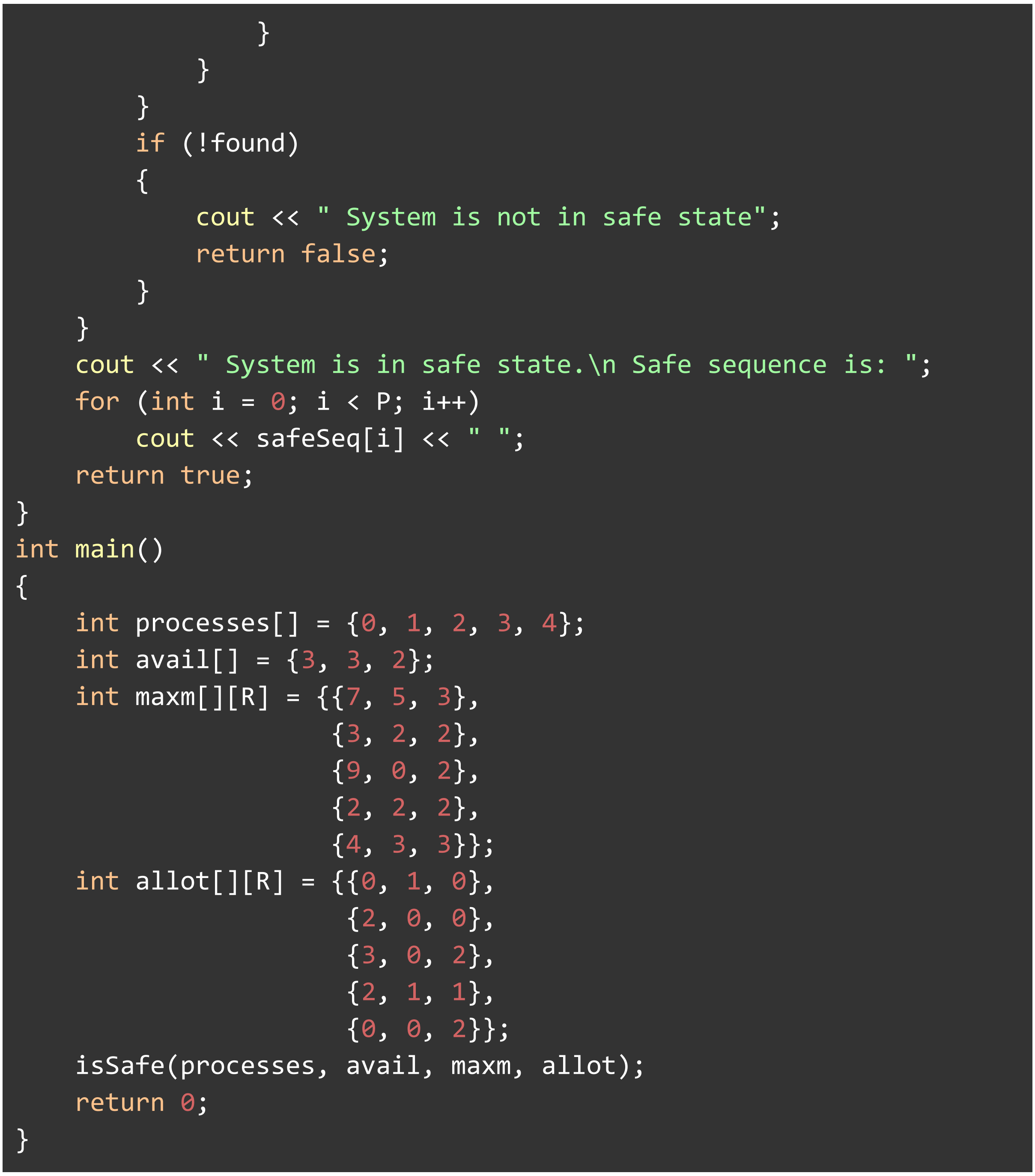
a) Circular wait
b) No preemption
c) Hold and wait
d) Mutual Exclusion

Bankers Algorithm

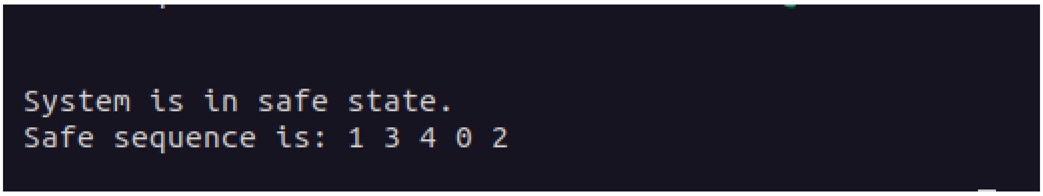
Bankers algorithm is a deadlock avoidance algorithm commonly used by operating systems to avoid any kind of deadlock situation if possible.

4. Banker's Algorithm Implementation

```
#include <iostream>
using namespace std;
const int P = 5, R = 3;
void calculateNeed(int need[P][R], int max[P][R], int allot[P][R])
{
    for (int i = 0; i < P; i++)
        for (int j = 0; j < R; j++)
            need[i][j] = max[i][j] - allot[i][j];
}
bool isSafe(int processes[], int avail[], int maxm[][R], int
allot[][R])
{
    int need[P][R];
    calculateNeed(need, maxm, allot);
    bool finish[P] = {0};
    int safeSeq[P], work[R], count = 0;
    for (int i = 0; i < R; i++)
        work[i] = avail[i];
    while (count < P)
    {
        bool found = false;
        for (int p = 0; p < P; p++)
        {
            if (finish[p] == 0)
            {
                int j;
                for (j = 0; j < R; j++)
                    if (need[p][j] > work[j])
                        break;
                if (j == R)
                {
                    for (int k = 0; k < R; k++)
                        work[k] += allot[p][k];
                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = true;
```



```
                }
            }
        }
        if (!found)
        {
            cout << " System is not in safe state";
            return false;
        }
    }
    cout << " System is in safe state.\n Safe sequence is: ";
    for (int i = 0; i < P; i++)
        cout << safeSeq[i] << " ";
    return true;
}
int main()
{
    int processes[] = {0, 1, 2, 3, 4};
    int avail[] = {3, 3, 2};
    int maxm[][R] = {{7, 5, 3},
                     {3, 2, 2},
                     {9, 0, 2},
                     {2, 2, 2},
                     {4, 3, 3}};
    int allot[][R] = {{0, 1, 0},
                      {2, 0, 0},
                      {3, 0, 2},
                      {2, 1, 1},
                      {0, 0, 2}};
    isSafe(processes, avail, maxm, allot);
    return 0;
}
```



```
System is in safe state.
Safe sequence is: 1 3 4 0 2
```

Course: Operating Systems **Course Code:** CSC403

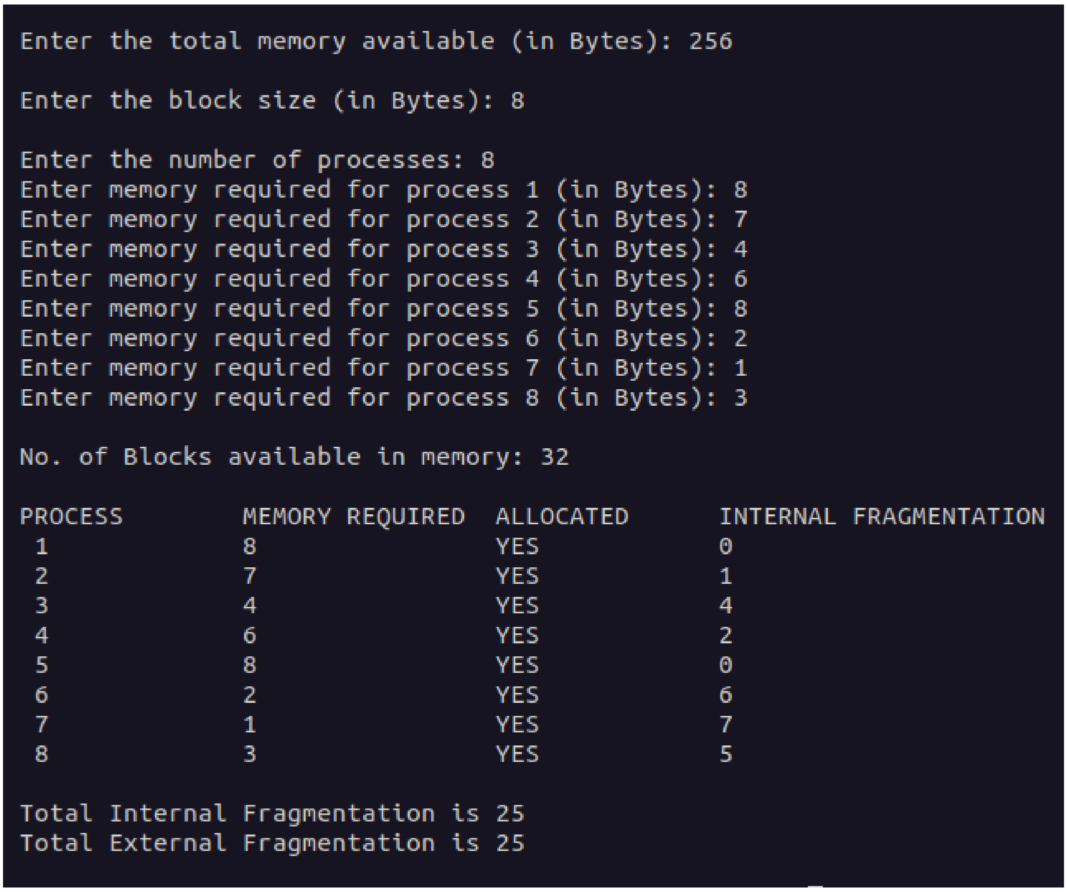
Roll No: 20107 **Teacher Name:** Dr. Sahil

Student Name: Akshat Mittal **E-mail:** 20107@iiitu.ac.in

1. MFT

```c
#include <stdio.h>
int main()
{
    int ms, bs, nob, ef, n, mp[10], tif = 0, oop;
    int i, p = 0;
    printf("\n Enter the total memory available (in Bytes): ");
    scanf("%d", &ms);
    printf("\n Enter the block size (in Bytes): ");
    scanf("%d", &bs);
    nob = ms / bs;
    oop = ms - nob * bs;
    printf("\n Enter the number of processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf(" Enter memory required for process %d (in Bytes):
", i + 1);
        scanf("%d", &mp[i]);
    }
    printf("\n No. of Blocks available in memory: %d", nob);
    printf("\n\n PROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");
    for (i = 0; i < n && p < nob; i++)
    {
        printf("\n  %d\t\t%d", i + 1, mp[i]);
        if (mp[i] > bs)
            printf("\t\t NO\t\t\t: ");
        else
        {
            printf("\t\t YES\t\t%d", bs - mp[i]);
            tif = tif + bs - mp[i];
            p++;
        }
    }
    ef = oop + tif;
    if (i < n)
```

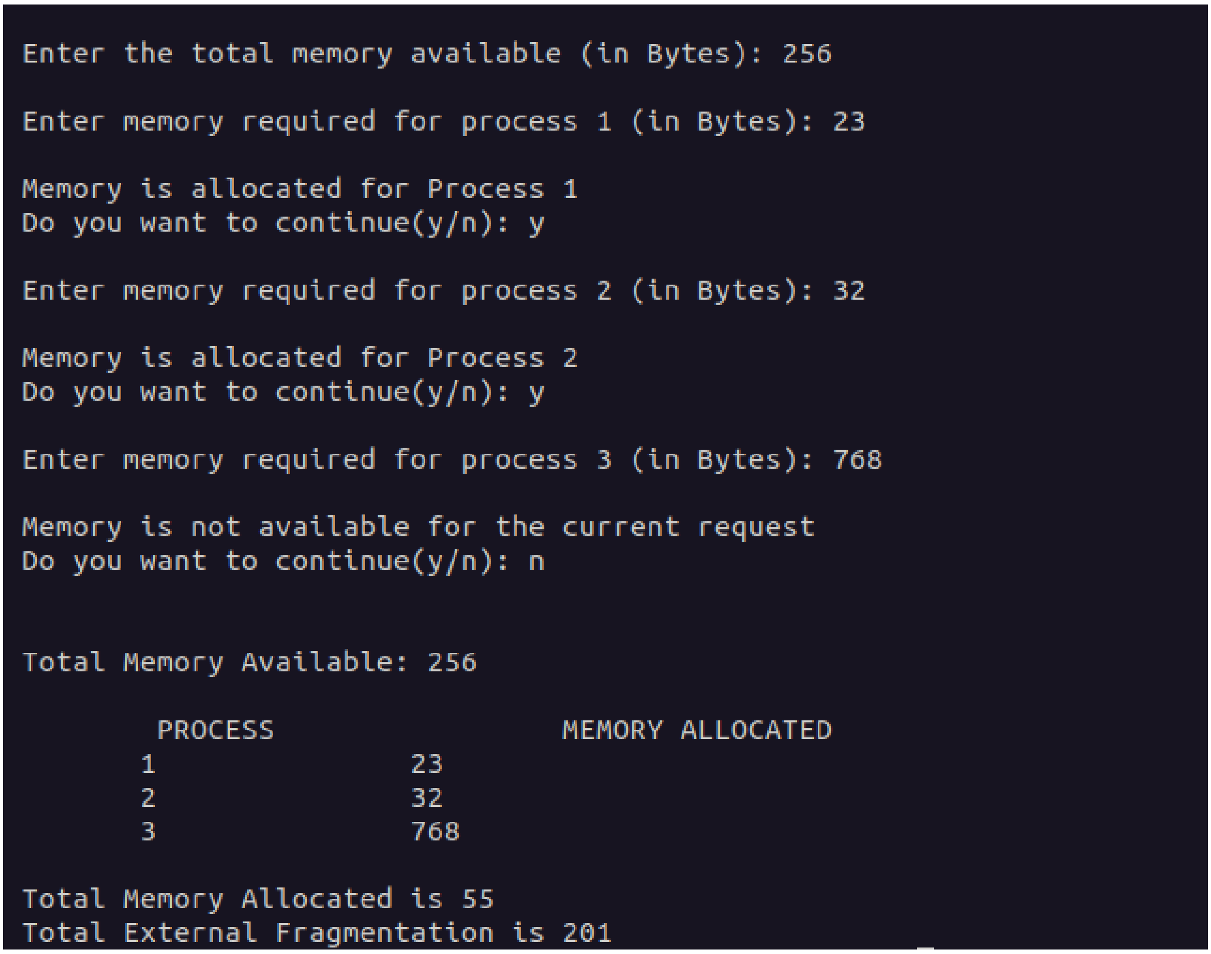
```
        printf("\n Memory is Full, Remaining Processes cannot be
accomodated");
    printf("\n\n Total Internal Fragmentation is %d", tif);
    printf("\n Total External Fragmentation is %d", ef);
    return 0;
}
```



```
Enter the total memory available (in Bytes): 256

Enter the block size (in Bytes): 8

Enter the number of processes: 8
Enter memory required for process 1 (in Bytes): 8
Enter memory required for process 2 (in Bytes): 7
Enter memory required for process 3 (in Bytes): 4
Enter memory required for process 4 (in Bytes): 6
Enter memory required for process 5 (in Bytes): 8
Enter memory required for process 6 (in Bytes): 2
Enter memory required for process 7 (in Bytes): 1
Enter memory required for process 8 (in Bytes): 3

No. of Blocks available in memory: 32

PROCESS        MEMORY REQUIRED  ALLOCATED      INTERNAL FRAGMENTATION
 1             8                YES            0
 2             7                YES            1
 3             4                YES            4
 4             6                YES            2
 5             8                YES            0
 6             2                YES            6
 7             1                YES            7
 8             3                YES            5

Total Internal Fragmentation is 25
Total External Fragmentation is 25
```

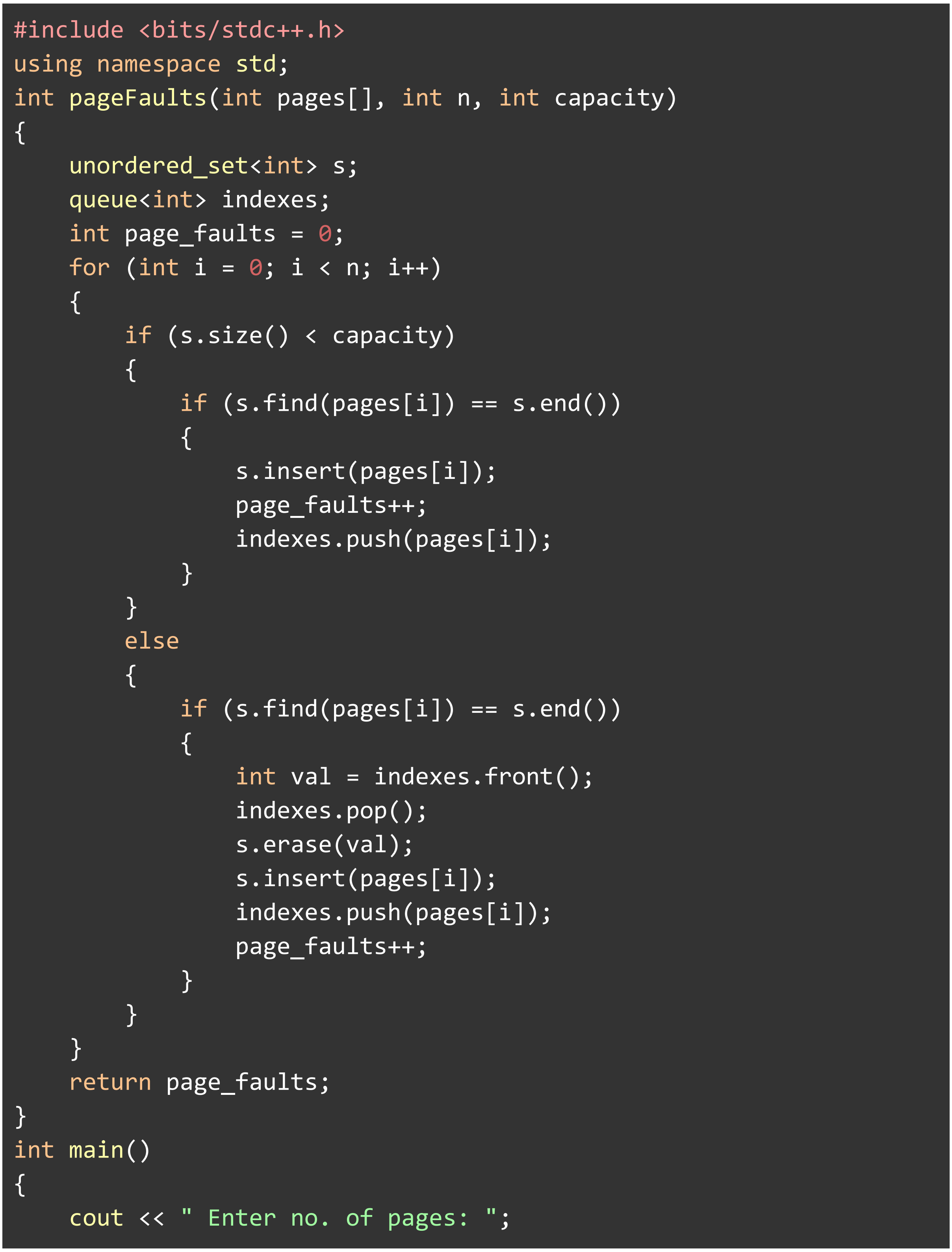
2. MVT

```
#include <stdio.h>
int main()
{
    int ms, mp[10], i = 0, temp, n = 0;
    char ch = 'y';
    printf("\n Enter the total memory available (in Bytes): ");
    scanf("%d", &ms);
    temp = ms;
    while (ch == 'y')
    {
        printf("\n Enter memory required for process %d (in
Bytes): ", i + 1);
        scanf("%d", &mp[i]);
        if (mp[i] <= temp)
        {
            printf("\n Memory is allocated for Process %d ", i +
1);
            temp = temp - mp[i];
        }
        else
            printf("\n Memory unavailable for the current
request");
        printf("\n Do you want to continue(y/n): ");
        scanf("%d", &temp);
        scanf("%c", &ch);
        i++;
        n++;
    }
    printf("\n\n Total Memory Available: %d", ms);
    printf("\n\n\t PROCESS\t\t MEMORY ALLOCATED ");
    for (i = 0; i < n; i++)
        printf("\n\t%d\t\t%d", i + 1, mp[i]);
    printf("\n\n Total Memory Allocated is %d", ms - temp);
    printf("\n Total External Fragmentation is %d\n", temp);
    return 0;
}
```

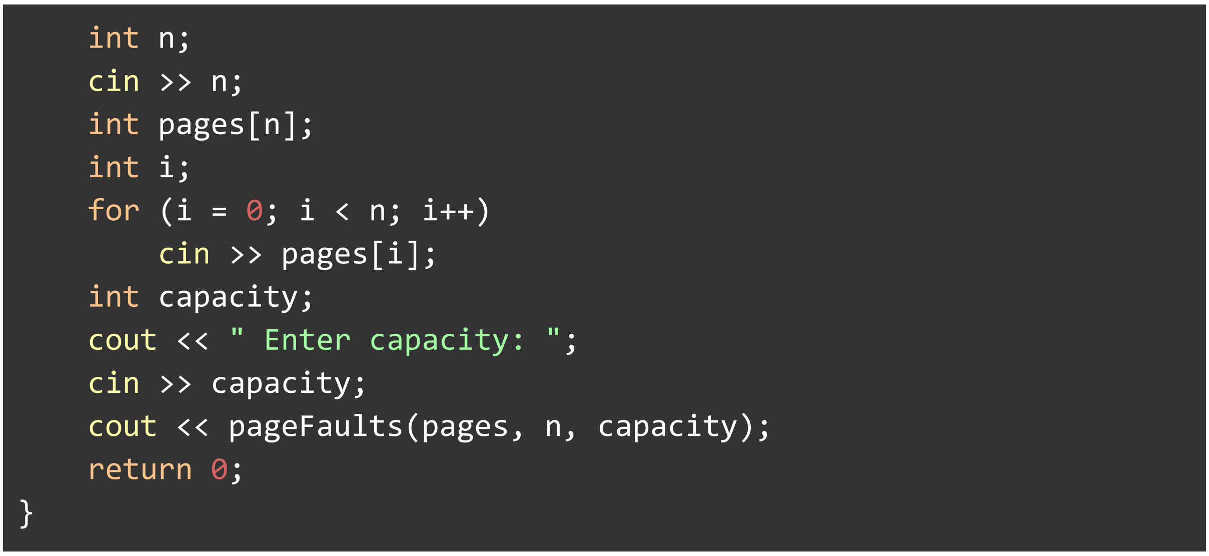


```
Enter the total memory available (in Bytes): 256

Enter memory required for process 1 (in Bytes): 23

Memory is allocated for Process 1
Do you want to continue(y/n): y

Enter memory required for process 2 (in Bytes): 32

Memory is allocated for Process 2
Do you want to continue(y/n): y

Enter memory required for process 3 (in Bytes): 768

Memory is not available for the current request
Do you want to continue(y/n): n


Total Memory Available: 256

        PROCESS                 MEMORY ALLOCATED
       1               23
       2               32
       3               768

Total Memory Allocated is 55
Total External Fragmentation is 201
```

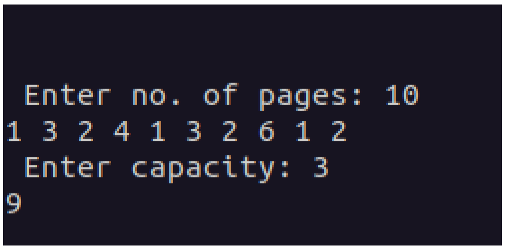
1. FIFO page replacement policy



```
#include <bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int capacity)
{
    unordered_set<int> s;
    queue<int> indexes;
    int page_faults = 0;
    for (int i = 0; i < n; i++)
    {
        if (s.size() < capacity)
        {
            if (s.find(pages[i]) == s.end())
            {
                s.insert(pages[i]);
                page_faults++;
                indexes.push(pages[i]);
            }
        }
        else
        {
            if (s.find(pages[i]) == s.end())
            {
                int val = indexes.front();
                indexes.pop();
                s.erase(val);
                s.insert(pages[i]);
                indexes.push(pages[i]);
                page_faults++;
            }
        }
    }
    return page_faults;
}
int main()
{
    cout << " Enter no. of pages: ";
```

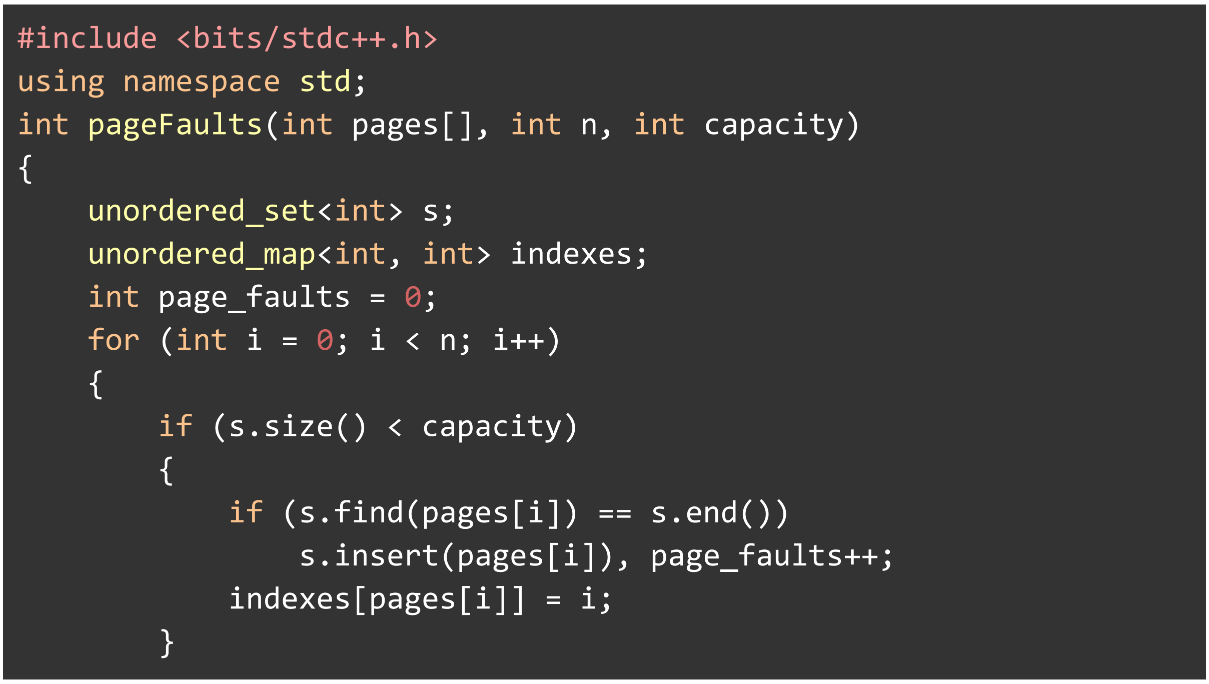


```
    int n;
    cin >> n;
    int pages[n];
    int i;
    for (i = 0; i < n; i++)
        cin >> pages[i];
    int capacity;
    cout << " Enter capacity: ";
    cin >> capacity;
    cout << pageFaults(pages, n, capacity);
    return 0;
}
```

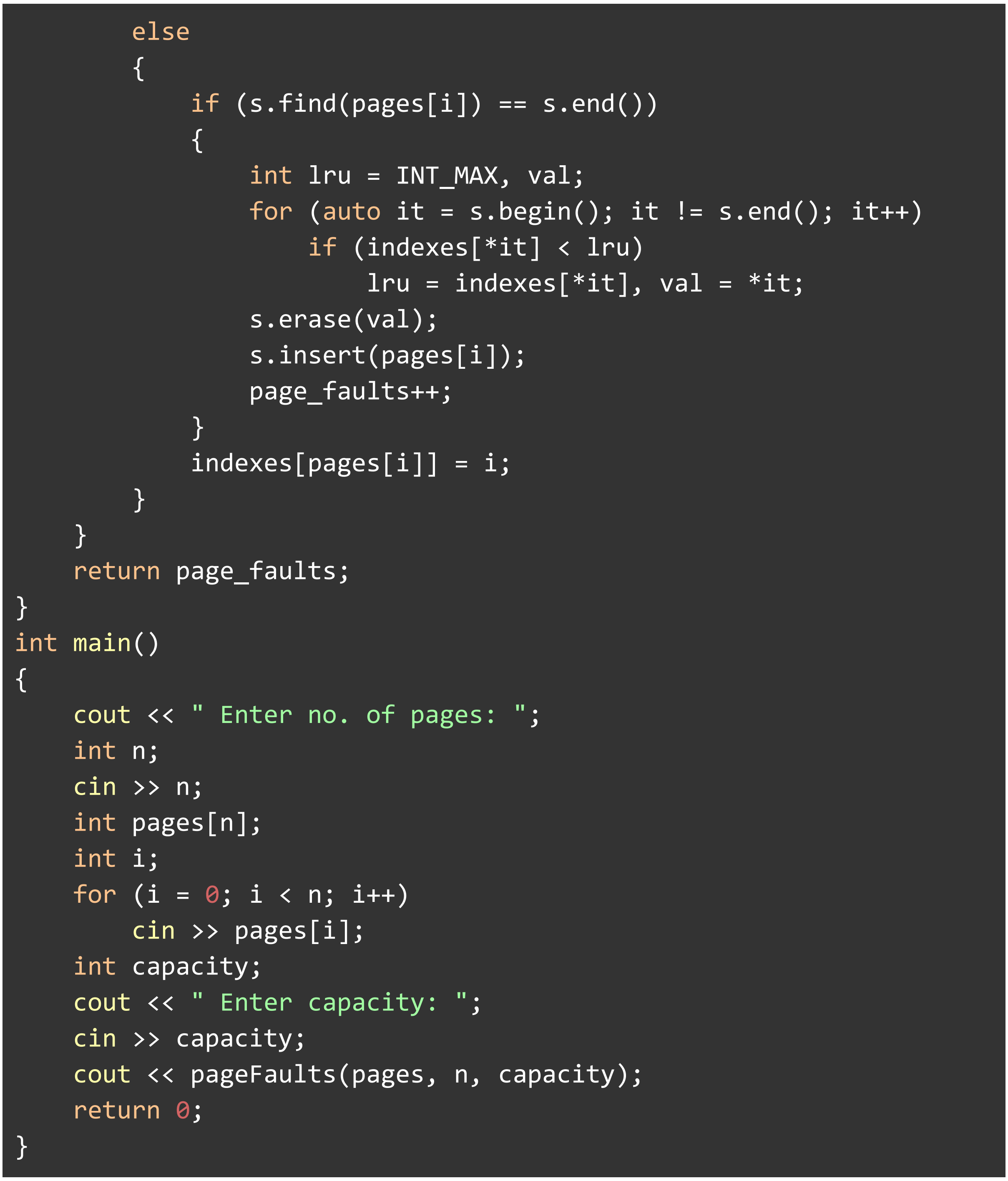


```
 Enter no. of pages: 10
1 3 2 4 1 3 2 6 1 2
 Enter capacity: 3
9
```

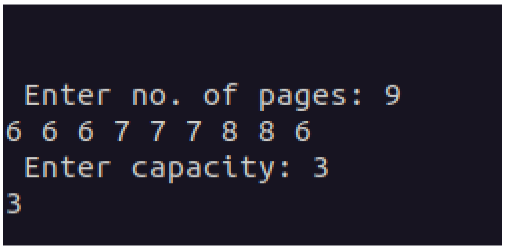
2. Least Recently Used Page Replacement Policy



```
#include <bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int capacity)
{
    unordered_set<int> s;
    unordered_map<int, int> indexes;
    int page_faults = 0;
    for (int i = 0; i < n; i++)
    {
        if (s.size() < capacity)
        {
            if (s.find(pages[i]) == s.end())
                s.insert(pages[i]), page_faults++;
            indexes[pages[i]] = i;
        }
```



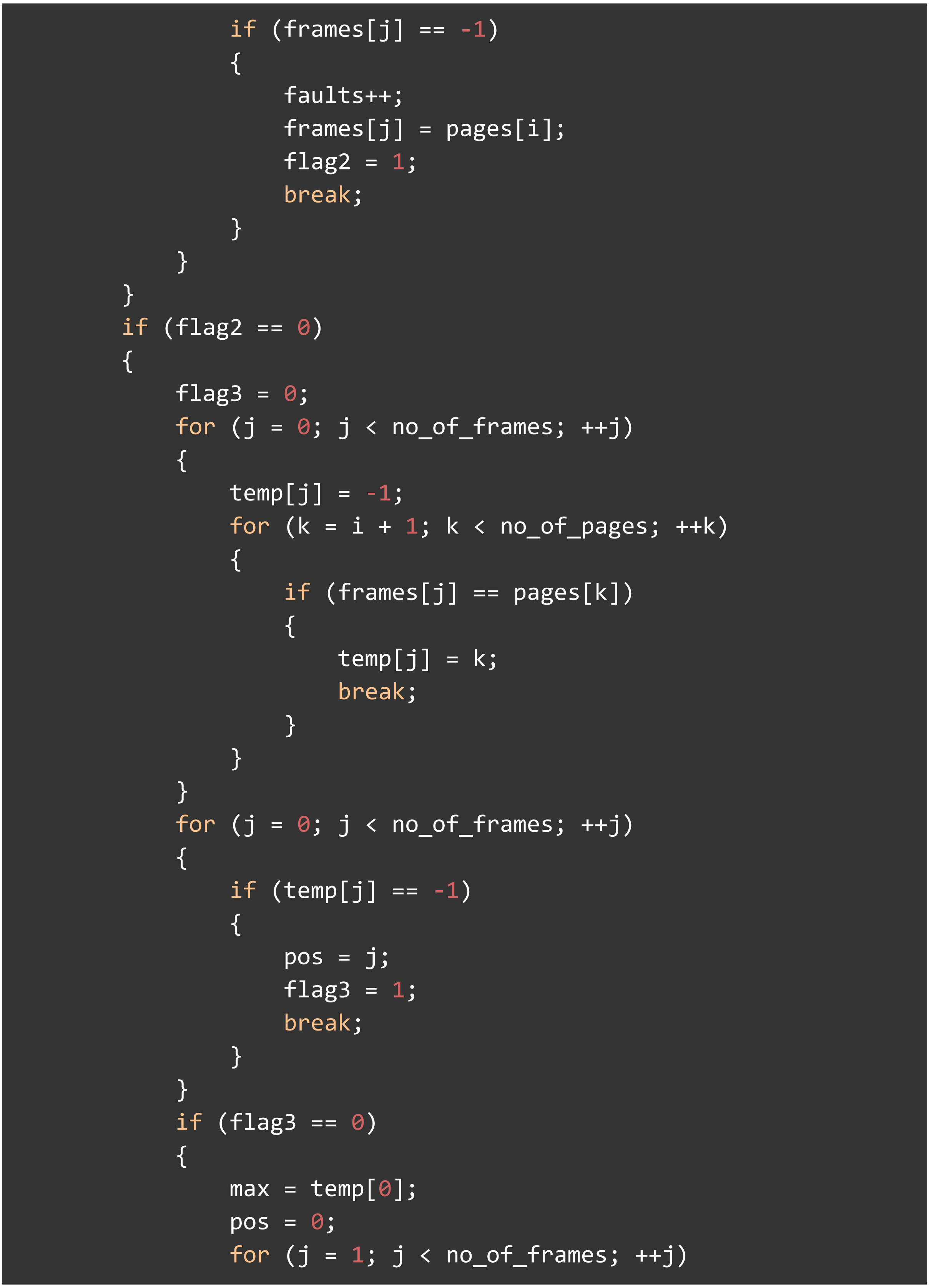
```
        else
        {
            if (s.find(pages[i]) == s.end())
            {
                int lru = INT_MAX, val;
                for (auto it = s.begin(); it != s.end(); it++)
                    if (indexes[*it] < lru)
                        lru = indexes[*it], val = *it;
                s.erase(val);
                s.insert(pages[i]);
                page_faults++;
            }
            indexes[pages[i]] = i;
        }
    }
    return page_faults;
}
int main()
{
    cout << " Enter no. of pages: ";
    int n;
    cin >> n;
    int pages[n];
    int i;
    for (i = 0; i < n; i++)
        cin >> pages[i];
    int capacity;
    cout << " Enter capacity: ";
    cin >> capacity;
    cout << pageFaults(pages, n, capacity);
    return 0;
}
```



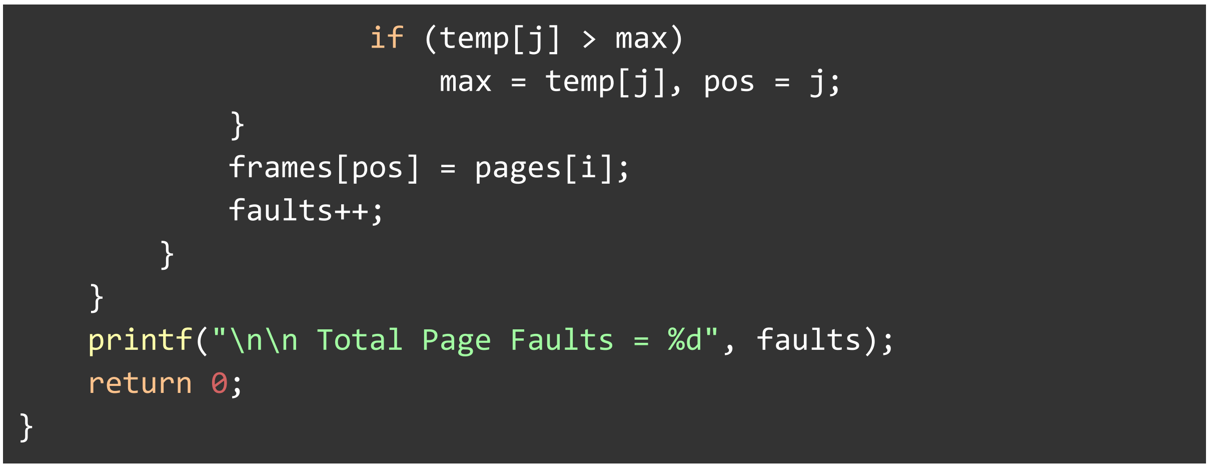
```
 Enter no. of pages: 9
6 6 6 7 7 7 8 8 6
 Enter capacity: 3
3
```

3. Optimal Page Replacement Policy

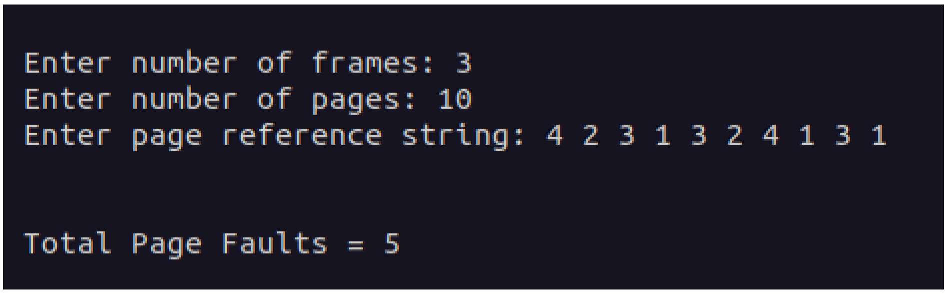
```
#include <stdio.h>
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30],
temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
    printf(" Enter number of frames: ");
    scanf("%d", &no_of_frames);
    printf(" Enter number of pages: ");
    scanf("%d", &no_of_pages);
    printf(" Enter page reference string: ");
    for (i = 0; i < no_of_pages; ++i)
        scanf("%d", &pages[i]);
    for (i = 0; i < no_of_frames; ++i)
        frames[i] = -1;
    for (i = 0; i < no_of_pages; ++i)
    {
        flag1 = flag2 = 0;
        for (j = 0; j < no_of_frames; ++j)
        {
            if (frames[j] == pages[i])
            {
                flag1 = flag2 = 1;
                break;
            }
        }
        if (flag1 == 0)
        {
            for (j = 0; j < no_of_frames; ++j)
            {
```



```
                if (frames[j] == -1)
                {
                    faults++;
                    frames[j] = pages[i];
                    flag2 = 1;
                    break;
                }
            }
        }
        if (flag2 == 0)
        {
            flag3 = 0;
            for (j = 0; j < no_of_frames; ++j)
            {
                temp[j] = -1;
                for (k = i + 1; k < no_of_pages; ++k)
                {
                    if (frames[j] == pages[k])
                    {
                        temp[j] = k;
                        break;
                    }
                }
            }
            for (j = 0; j < no_of_frames; ++j)
            {
                if (temp[j] == -1)
                {
                    pos = j;
                    flag3 = 1;
                    break;
                }
            }
            if (flag3 == 0)
            {
                max = temp[0];
                pos = 0;
                for (j = 1; j < no_of_frames; ++j)
```



```
                    if (temp[j] > max)
                        max = temp[j], pos = j;
            }
            frames[pos] = pages[i];
            faults++;
        }
    }
    printf("\n\n Total Page Faults = %d", faults);
    return 0;
}
```



```
Enter number of frames: 3
Enter number of pages: 10
Enter page reference string: 4 2 3 1 3 2 4 1 3 1


Total Page Faults = 5
```