OS Lab Assignment-2
Ripunjay Narula (19BCE0470)

CPU Scheduling

(a) Implement the various process scheduling algorithms such as FCFS, SJF, Priority (Non Preemptive). (Easy)

(b) Implement the various process scheduling algorithms such as Priority, Round Robin (preemptive). (Medium)

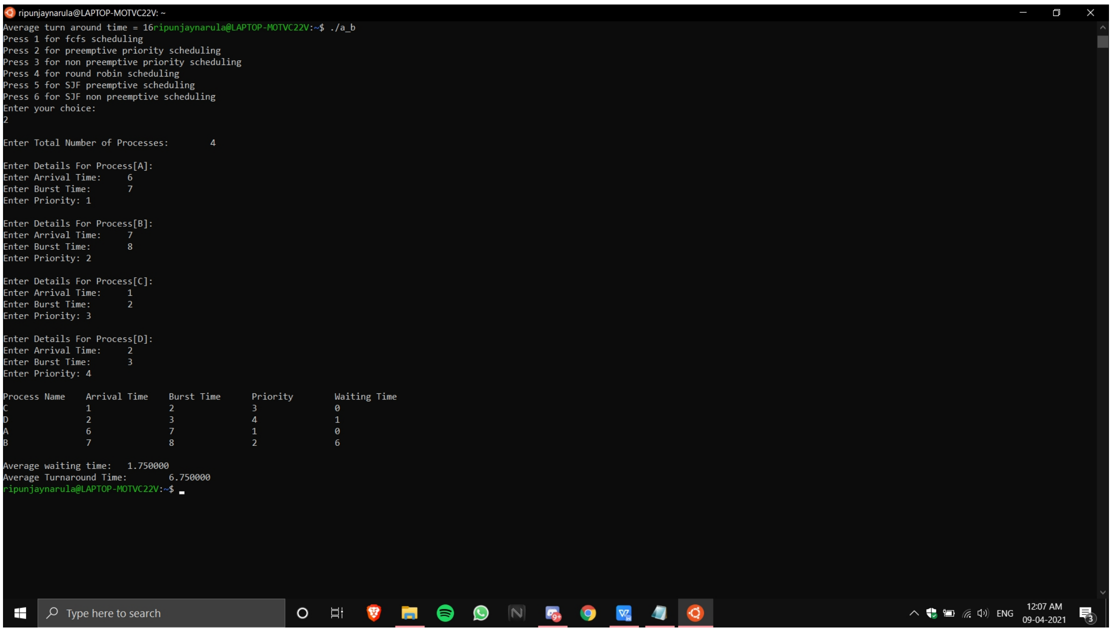
(d) Simulate with a program to provide deadlock avoidance of Banker's Algorithm including Safe state and additional resource request (High).

A&B:

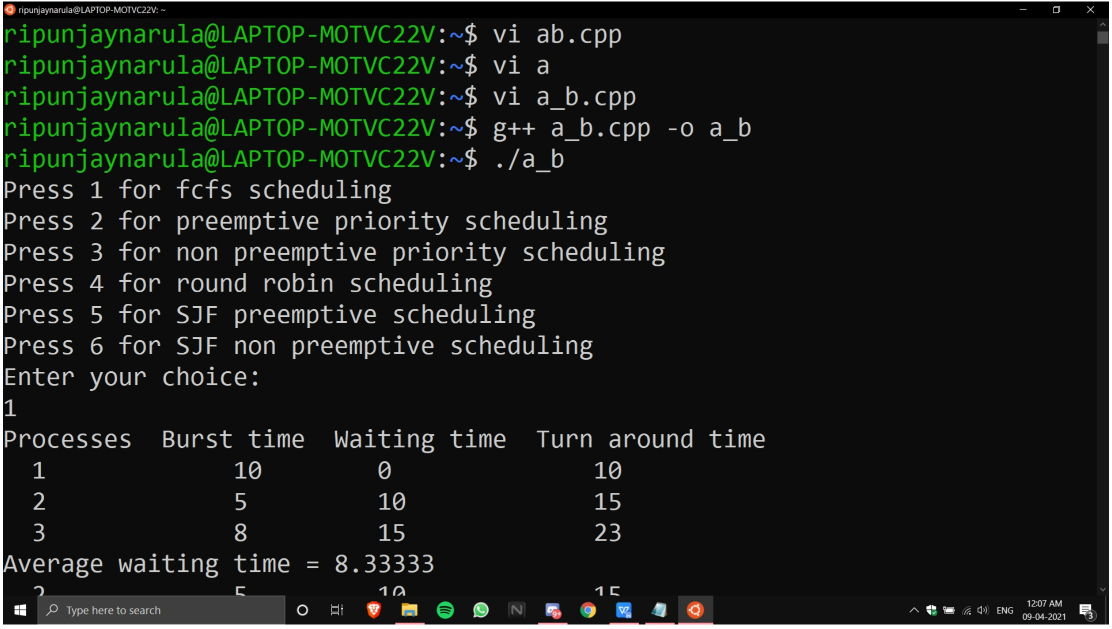
```
#include<iostream>
 #include<stdio.h>
 #include <sys/types.h>
#include<cstdlib>
using namespace ::std;
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
	wt[0] = 0;
	for (int i = 1; i < n ; i++ ) wt[i] = bt[i-1] + wt[i-1] ;
}
void fcfsfindTurnAroundTime( int processes[], int n, int 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 wt[1000], tat[1000], total_wt = 0, total_tat = 0; findWaitingTime(processes, n,
bt, wt);
	fcfsfindTurnAroundTime(processes, n, bt, wt, tat);
	cout << "Processes "<< " Burst time " << " Waiting time " << " Turn around
time\n";
	for (int i=0; i<n; i++)
	{
		total_wt = total_wt + wt[i]; total_tat = total_tat + tat[i];
		cout << "  " << i+1 << "\t\t" << bt[i] <<"\t  "<< wt[i] <<"\t\t " << tat[i] <<endl;
	}
```

```
        cout << "Average waiting time = " << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = " << (float)total_tat / (float)n;
}

int fcfs_sched()
{
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0]; int burst_time[] = {10, 5, 8};
findavgTime(processes, n, burst_time); return 0;
}


struct process
{
        char process_name;
        int arrival_time, burst_time, ct, waiting_time, turnaround_time, priority;
        int status;
}process_queue[10]; int limit;
void Arrival_Time_Sorting()
{
        struct process temp; int i, j;
        for(i = 0; i < limit - 1; i++)
        {
                for(j = i + 1; j < limit; j++)
                {
 if(process_queue[i].arrival_time > process_queue[j].arrival_time)
                        {
                                temp = process_queue[i]; process_queue[i] = process_queue[j];
process_queue[j] = temp;
                        }
                }

        }
}

int PrioPE()
{
        int i, time = 0, burst_time = 0, largest; char c;
        float wait_time = 0, turnaround_time = 0, average_waiting_time,
average_turnaround_time;
        printf("\nEnter Total Number of Processes:\t"); scanf("%d", &limit);
        for(i = 0, c = 'A'; i < limit; i++, c++)
        {
                process_queue[i].process_name = c;
                printf("\nEnter Details For
Process[%C]:\n",process_queue[i].process_name);
                printf("Enter Arrival Time:\t");
                scanf("%d", &process_queue[i].arrival_time );
                printf("Enter Burst Time:\t");
                scanf("%d", &process_queue[i].burst_time);
                printf("Enter Priority:\t");
                scanf("%d", &process_queue[i].priority);
                process_queue[i].status = 0;
                burst_time = burst_time + process_queue[i].burst_time;
        }
        Arrival_Time_Sorting(); process_queue[9].priority = -9999;
        printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");
        for(time = process_queue[0].arrival_time; time < burst_time;)
        {
                largest = 9;
                for(i = 0; i < limit; i++)
                {

                        if(process_queue[i].arrival_time <= time && process_queue[i].status !=
1 && process_queue[i].priority > process_queue[largest].priority)
                        {
                                largest = i;
                        }
                }
                time = time + process_queue[largest].burst_time;
                process_queue[largest].ct = time;
                process_queue[largest].waiting_time =process_queue[largest].ct -
                process_queue[largest].arrival_time - process_queue[largest].burst_time;
                process_queue[largest].turnaround_time = process_queue[largest].ct -
                process_queue[largest].arrival_time;
                process_queue[largest].status = 1;
                wait_time = wait_time + process_queue[largest].waiting_time;
                turnaround_time = turnaround_time
+process_queue[largest].turnaround_time;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d",process_queue[largest].process_name,
process_queue[largest].arrival_time, process_queue[largest].burst_time,
process_queue[largest].priority, process_queue[largest].waiting_time);
        }
        average_waiting_time = wait_time / limit; average_turnaround_time =
turnaround_time / limit;
        printf("\n\nAverage waiting time:\t%f\n", average_waiting_time);
        printf("Average Turnaround Time:\t%f\n",
                        average_turnaround_time); return 0;
}


int PrioNPE()
{
        int burst_time[20], process[20], waiting_time[20], turnaround_time[20],
    priority[20];
        int i, j, limit, sum = 0, position, temp;
        float average_wait_time, average_turnaround_time;
        printf("Enter Total Number of Processes:\t");
        scanf("%d", &limit);
        printf("\nEnter Burst Time and Priority For %d Processes\n", limit);
        for(i = 0; i < limit; i++)
        {
                printf("\nProcess[%d]\n", i + 1); printf("Process Burst Time:\t");
                scanf("%d", &burst_time[i]); printf("Process Priority:\t");
                scanf("%d", &priority[i]); process[i] = i + 1;
        }
        for(i = 0; i < limit; i++)
        {
                position = i;
                for(j = i + 1; j < limit; j++)
                {
                        if(priority[j] < priority[position])
                        {
                            position = j;
                        }
                }
                temp = priority[i];
                priority[i] = priority[position];
                priority[position] = temp; temp = burst_time[i];
                burst_time[i] = burst_time[position];
                burst_time[position] = temp;
                temp = process[i];
                process[i] = process[position];
                 process[position] = temp;
        }

        waiting_time[0] = 0; for(i = 1; i < limit; i++)
        {
                waiting_time[i] = 0; for(j = 0; j < i; j++)
               {
                        waiting_time[i] = waiting_time[i] + burst_time[j];
                }
                sum = sum + waiting_time[i];
        }
        average_wait_time = sum / limit; sum = 0;
        printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");
        for(i = 0; i < limit; i++)
        {
                turnaround_time[i] = burst_time[i] + waiting_time[i];
                sum = sum + turnaround_time[i];
                printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n", process[i],burst_time[i],
        waiting_time[i], turnaround_time[i]);
        }
        average_turnaround_time = sum / limit;
         printf("\nAverage Waiting Time:\t%f", average_wait_time);
         printf("\nAverage Turnaround Time:\t%f\n",average_turnaround_time);
         return 0;
        }


        void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum)
        {
        int rem_bt[1000];
        for (int i = 0 ; i < n ; i++) rem_bt[i] = bt[i];
        int t = 0;

        while (1)
        {
                bool done = true;
                for (int i = 0 ; i < n; i++)
                {
                        if (rem_bt[i] > 0)
                        {
                            done = false;
                            if (rem_bt[i] > quantum)
                            {
                                t += quantum; rem_bt[i] -= quantum;
                            }
                            else
                            {
                                t = t + rem_bt[i]; wt[i] = t - bt[i]; rem_bt[i] = 0;
                            }
                        }
                }
                if (done == true) break;
        }
        }
        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 quantum)
        {
        int wt[1000], tat[1000], total_wt = 0, total_tat = 0;
        findWaitingTime(processes, n, bt, wt, quantum);
        findTurnAroundTime(processes, n, bt, wt, tat);
        cout << "Processes "<< " Burst time "<< " Waiting time " << " Turn around
        time\n";
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << i+1 << "\t\t" << bt[i] <<"\t "<< wt[i] <<"\t\t " << tat[i] <<endl;
        }
        cout << "Average waiting time = "<< (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "<< (float)total_tat / (float)n;
        }
        int RR_sched()
        {
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];
        int burst_time[] = {10, 5, 8};
        int quantum = 2;
        findavgTime(processes, n, burst_time, quantum);
        return 0;
        }

        int SJF_PE()
        {
        int arrival_time[10], burst_time[10], temp[10]; int i, smallest, count = 0, time,
        limit;
        double wait_time = 0, turnaround_time = 0, end;
        float average_waiting_time, average_turnaround_time;
        printf("\nEnter the Total Number of Processes:\t");
        scanf("%d", &limit);
        printf("\nEnter Details of %d Processes\n", limit);
        for(i = 0; i < limit; i++)
        {
                printf("\nEnter Arrival Time:\t");
                scanf("%d", &arrival_time[i]);
                printf("Enter Burst Time:\t");
                scanf("%d", &burst_time[i]); temp[i] = burst_time[i];
        }
        burst_time[9] = 9999;
        for(time = 0; count != limit; time++)
        {
                smallest = 9;
                for(i = 0; i < limit; i++)
        {
                        if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] &&
        burst_time[i] > 0)
                        {
                            smallest = i;
                        }
                }
                burst_time[smallest]--;
                 if(burst_time[smallest] == 0)
                 {
                        count++;
                        end = time + 1;
                        wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
                        turnaround_time = turnaround_time + end - arrival_time[smallest];
                 }
        }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
         printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
         printf("Average Turnaround Time:\t%lf\n",
                        average_turnaround_time);
         return 0;
        }

        int SJF_NPE()
        {
        int temp, i, j, limit, sum = 0, position;
        float average_wait_time, average_turnaround_time;
        int burst_time[20], process[20], waiting_time[20],turnaround_time[20];
        printf("\nEnter Total Number of Processes:\t");
        scanf("%d", &limit);
        for(i = 0; i < limit; i++)
        {
                printf("Enter Burst Time For Process[%d]:\t", i + 1);
                scanf("%d", &burst_time[i]);
                process[i] = i + 1;
        }
        for(i = 0; i < limit; i++)
        {
                position = i;
                for(j = i + 1; j < limit; j++)
                {
                        if(burst_time[j] < burst_time[position])
                        {
                            position = j;
                        }
                }
                temp = burst_time[i];
                burst_time[i] = burst_time[position];
         burst_time[position] = temp;
                temp = process[i];
                process[i] = process[position];
                process[position] = temp;
        }
        waiting_time[0] = 0;

        for(i = 1; i < limit; i++)
        {
                waiting_time[i] = 0;

                for(j = 0; j < i; j++)

                {
                        waiting_time[i] = waiting_time[i] + burst_time[j];
                }
                sum = sum + waiting_time[i];
        }
        average_wait_time = (float)sum / limit; sum = 0;
        printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");
        for(i = 0; i < limit; i++)
        {
                turnaround_time[i] = burst_time[i] + waiting_time[i];

                sum = sum + turnaround_time[i];
                printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n", process[i],burst_time[i],
        waiting_time[i], turnaround_time[i]);
        }
        average_turnaround_time = (float)sum / limit;
        printf("\nAverage Waiting Time:\t%f\n", average_wait_time);
        printf("\nAverage Turnaround Time:\t%f\n", average_turnaround_time); return
        0;
        }


        int main()
        {
        float result;
        int choice, num;
        printf("Press 1 for fcfs scheduling\n");
        printf("Press 2 for preemptive priority scheduling\n");
        printf("Press 3 for non preemptive priority scheduling\n");
        printf("Press 4 for round robin scheduling\n");
        printf("Press 5 for SJF preemptive scheduling\n");
        printf("Press 6 for SJF non preemptive scheduling\n");
        printf("Enter your choice:\n");

        cin >> choice;

        switch (choice) {
```

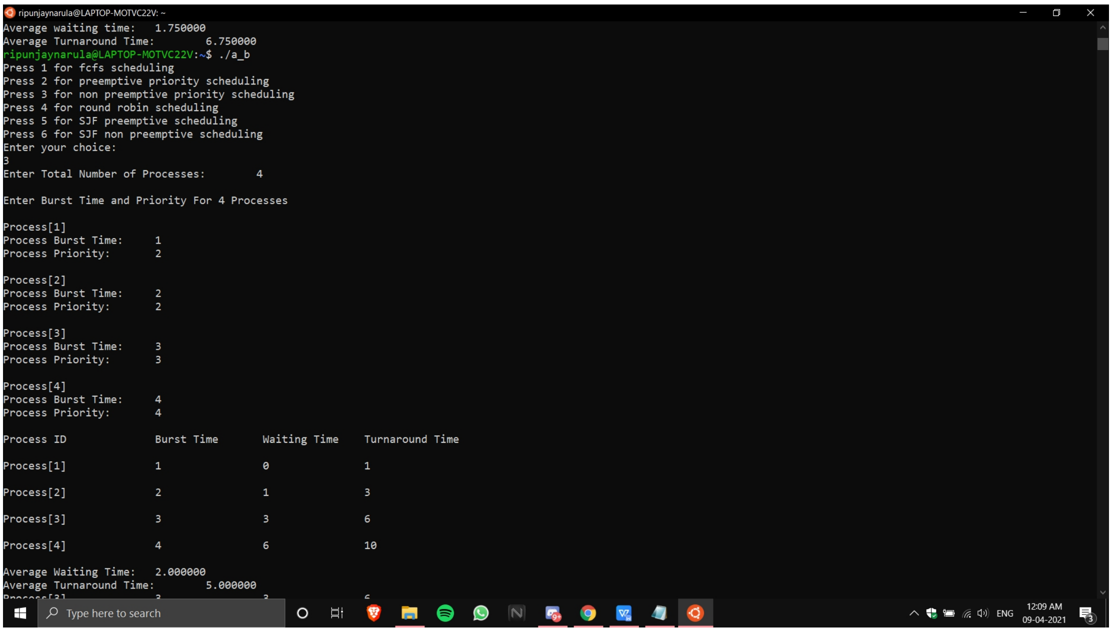
```
case 1: {
                fcfs_sched();
                break;
            }
        case 2: { PrioPE();
                 break;
            }
        case 3: { PrioNPE();
                break;
            }
        case 4: { RR_sched(); break;
            }
        case 5: { SJF_PE();
                break;
            }
        case 6: { SJF_NPE();
                break;
            }
        default:
            printf("wrong Input\n");
    }
    return 0;
}
```



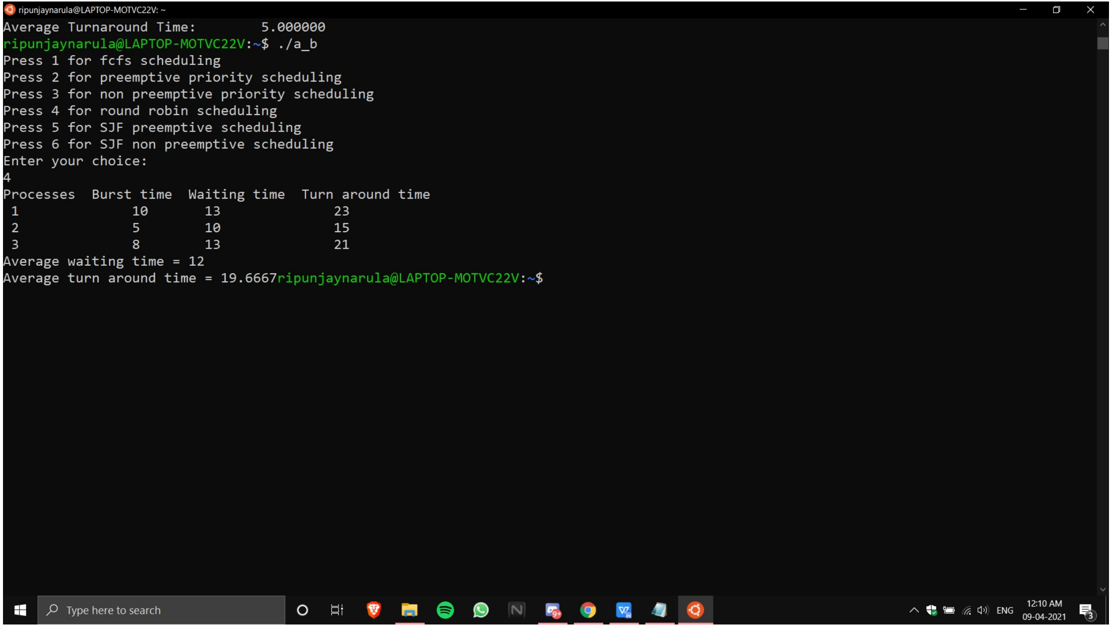
```
Average turn around time = 16ripunjaynarula@LAPTOP-MOTVC22V:~$ ./a_b
Press 1 for fcfs scheduling
Press 2 for preemptive priority scheduling
Press 3 for non preemptive priority scheduling
Press 4 for round robin scheduling
Press 5 for SJF preemptive scheduling
Press 6 for SJF non preemptive scheduling
Enter your choice:
2

Enter Total Number of Processes:        4

Enter Details For Process[A]:
Enter Arrival Time:     6
Enter Burst Time:       7
Enter Priority: 1

Enter Details For Process[B]:
Enter Arrival Time:     7
Enter Burst Time:       8
Enter Priority: 2

Enter Details For Process[C]:
Enter Arrival Time:     1
Enter Burst Time:       2
Enter Priority: 3

Enter Details For Process[D]:
Enter Arrival Time:     2
Enter Burst Time:       3
Enter Priority: 4

Process Name    Arrival Time    Burst Time      Priority        Waiting Time
C               1               2               3               0
D               2               3               4               1
A               6               7               1               0
B               7               8               2               6

Average waiting time:   1.750000
Average Turnaround Time:        6.750000
ripunjaynarula@LAPTOP-MOTVC22V:~$
```



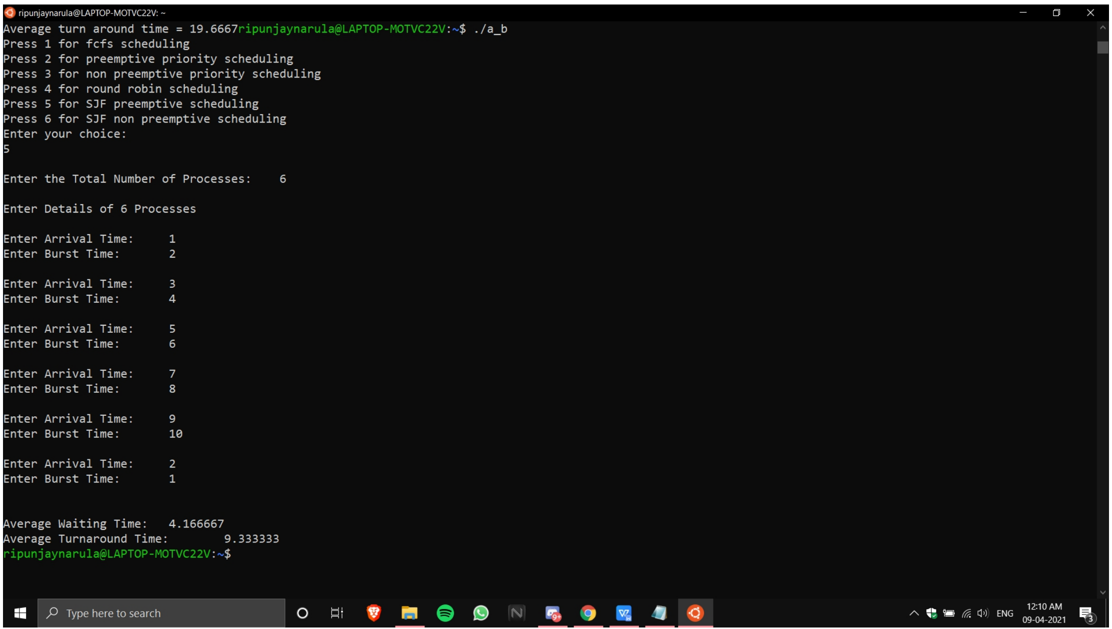
```
ripunjaynarula@LAPTOP-MOTVC22V:~$ vi ab.cpp
ripunjaynarula@LAPTOP-MOTVC22V:~$ vi a
ripunjaynarula@LAPTOP-MOTVC22V:~$ vi a_b.cpp
ripunjaynarula@LAPTOP-MOTVC22V:~$ g++ a_b.cpp -o a_b
ripunjaynarula@LAPTOP-MOTVC22V:~$ ./a_b
Press 1 for fcfs scheduling
Press 2 for preemptive priority scheduling
Press 3 for non preemptive priority scheduling
Press 4 for round robin scheduling
Press 5 for SJF preemptive scheduling
Press 6 for SJF non preemptive scheduling
Enter your choice:
1
Processes  Burst time  Waiting time  Turn around time
  1             10         0              10
  2             5          10             15
  3             8          15             23
Average waiting time = 8.33333
  2             5          10             15
```



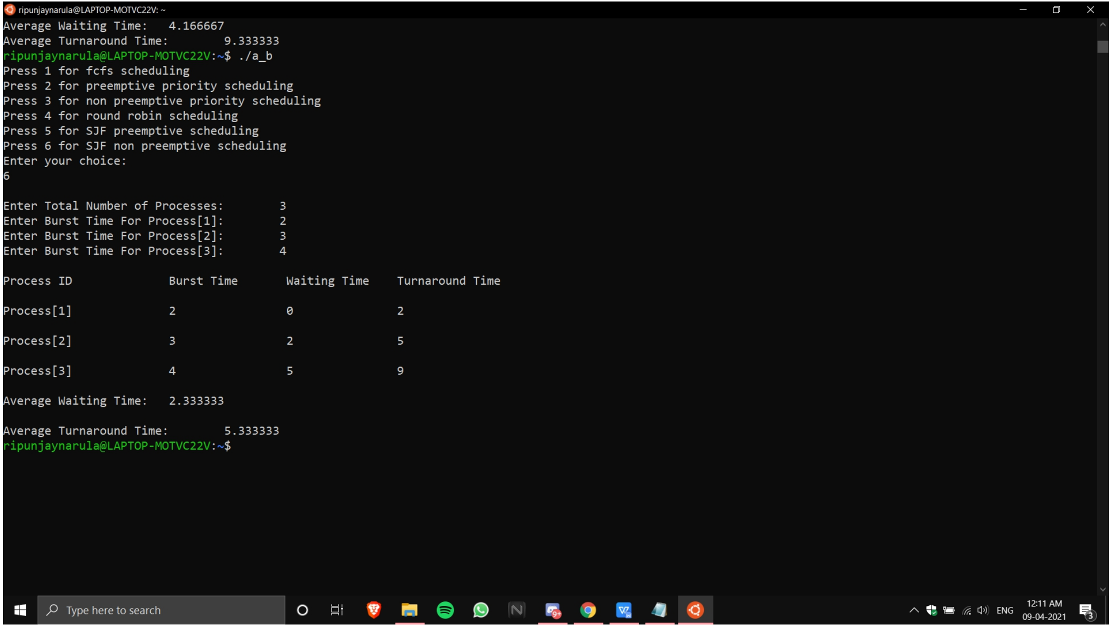
```
Average waiting time:   1.750000
Average Turnaround Time:        6.750000
ripunjaynarula@LAPTOP-MOTVC22V:~$ ./a_b
Press 1 for fcfs scheduling
Press 2 for preemptive priority scheduling
Press 3 for non preemptive priority scheduling
Press 4 for round robin scheduling
Press 5 for SJF preemptive scheduling
Press 6 for SJF non preemptive scheduling
Enter your choice:
3
Enter Total Number of Processes:        4

Enter Burst Time and Priority For 4 Processes

Process[1]
Process Burst Time:     1
Process Priority:       2

Process[2]
Process Burst Time:     2
Process Priority:       2

Process[3]
Process Burst Time:     3
Process Priority:       3

Process[4]
Process Burst Time:     4
Process Priority:       4

Process ID              Burst Time      Waiting Time    Turnaround Time

Process[1]              1               0               1

Process[2]              2               1               3

Process[3]              3               3               6

Process[4]              4               6               10

Average Waiting Time:   2.000000
Average Turnaround Time:        5.000000
Process[3]              3               3               6
```



```
Average Turnaround Time:        5.000000
ripunjaynarula@LAPTOP-MOTVC22V:~$ ./a_b
Press 1 for fcfs scheduling
Press 2 for preemptive priority scheduling
Press 3 for non preemptive priority scheduling
Press 4 for round robin scheduling
Press 5 for SJF preemptive scheduling
Press 6 for SJF non preemptive scheduling
Enter your choice:
4
Processes  Burst time  Waiting time  Turn around time
 1              10       13               23
 2              5        10               15
 3              8        13               21
Average waiting time = 12
Average turn around time = 19.6667ripunjaynarula@LAPTOP-MOTVC22V:~$
```



```
Average turn around time = 19.6667ripunjaynarula@LAPTOP-MOTVC22V:~$ ./a_b
Press 1 for fcfs scheduling
Press 2 for preemptive priority scheduling
Press 3 for non preemptive priority scheduling
Press 4 for round robin scheduling
Press 5 for SJF preemptive scheduling
Press 6 for SJF non preemptive scheduling
Enter your choice:
5

Enter the Total Number of Processes:    6

Enter Details of 6 Processes

Enter Arrival Time:     1
Enter Burst Time:       2

Enter Arrival Time:     3
Enter Burst Time:       4

Enter Arrival Time:     5
Enter Burst Time:       6

Enter Arrival Time:     7
Enter Burst Time:       8

Enter Arrival Time:     9
Enter Burst Time:       10

Enter Arrival Time:     2
Enter Burst Time:       1


Average Waiting Time:   4.166667
Average Turnaround Time:        9.333333
ripunjaynarula@LAPTOP-MOTVC22V:~$
```



```
Average Waiting Time:   4.166667
Average Turnaround Time:        9.333333
ripunjaynarula@LAPTOP-MOTVC22V:~$ ./a_b
Press 1 for fcfs scheduling
Press 2 for preemptive priority scheduling
Press 3 for non preemptive priority scheduling
Press 4 for round robin scheduling
Press 5 for SJF preemptive scheduling
Press 6 for SJF non preemptive scheduling
Enter your choice:
6

Enter Total Number of Processes:        3
Enter Burst Time For Process[1]:        2
Enter Burst Time For Process[2]:        3
Enter Burst Time For Process[3]:        4

Process ID              Burst Time       Waiting Time     Turnaround Time

Process[1]              2                0                2

Process[2]              3                2                5

Process[3]              4                5                9

Average Waiting Time:   2.333333

Average Turnaround Time:        5.333333
ripunjaynarula@LAPTOP-MOTVC22V:~$
```

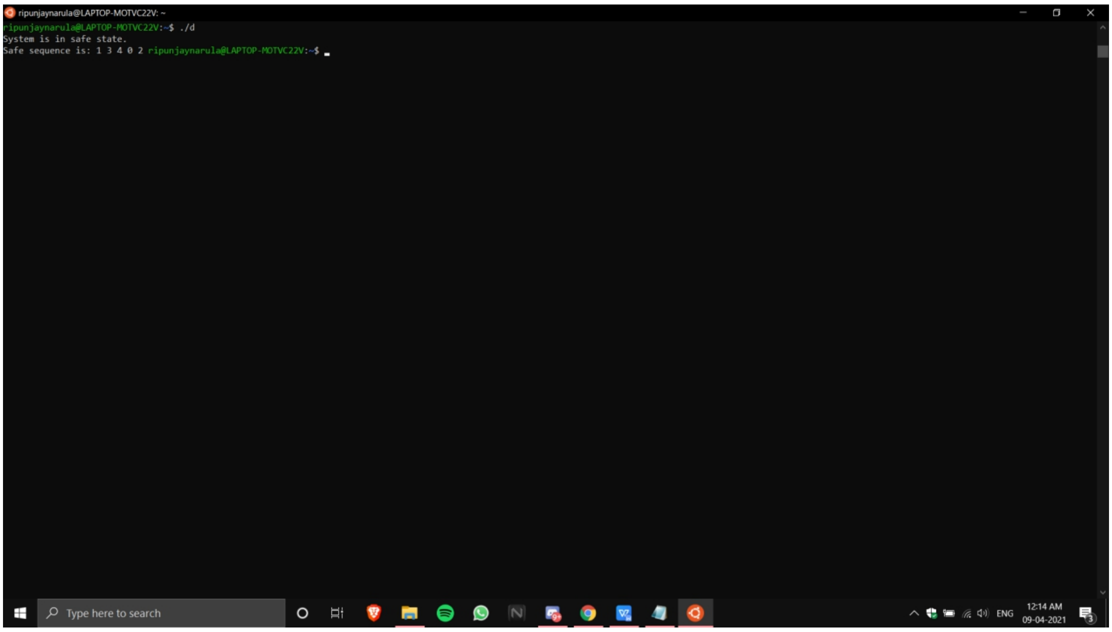
D:

Safe State:

```
#include<iostream>
using namespace std;
const int P = 5;
const int R = 3;
void Need(int need[P][R], int max[P][R], int alloc[P][R])
```

```
{
    for (int i = 0 ; i < P ; i++)
        for (int j = 0 ; j < R ; j++)
            need[i][j] = max[i][j] - alloc[i][j];
}
bool Safe(int processes[], int avail[], int max[][R], int alloc[][R])
{
    int need[P][R]; Need(need, max, alloc); bool finish[P] = {0};
    int safeSeq[P]; int work[R];
    for (int i = 0; i < R ; i++) work[i] = avail[i];
    int count = 0; 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] += alloc[p][k];
                    safeSeq[count++] = p; finish[p] = 1;
                    found = true;
                }
            }
        }
        if (found == false)
        {
            cout << "System is not in safe state"; return false;
        }
    }
    cout << "System is in safe state.\nSafe" " sequence is: ";
    for (int i = 0; i < P ; i++) cout << safeSeq[i] << " ";
    return true;
}
int main()
{
    int pro[] = {0, 1, 2, 3, 4};
    int avail[] = {3, 3, 2};
    int max[][R] = {{7, 5, 3},
        {3, 2, 2},
       {9, 0, 2},
        {2, 2, 2},
        {4, 3, 3}};
```

```
	int alloc[][R] = {{0, 1, 0},
		{2, 0, 0},
		{3, 0, 2},
		{2, 1, 1},
		{0, 0, 2}};

	Safe(pro, avail, max, alloc);
	return 0;
```

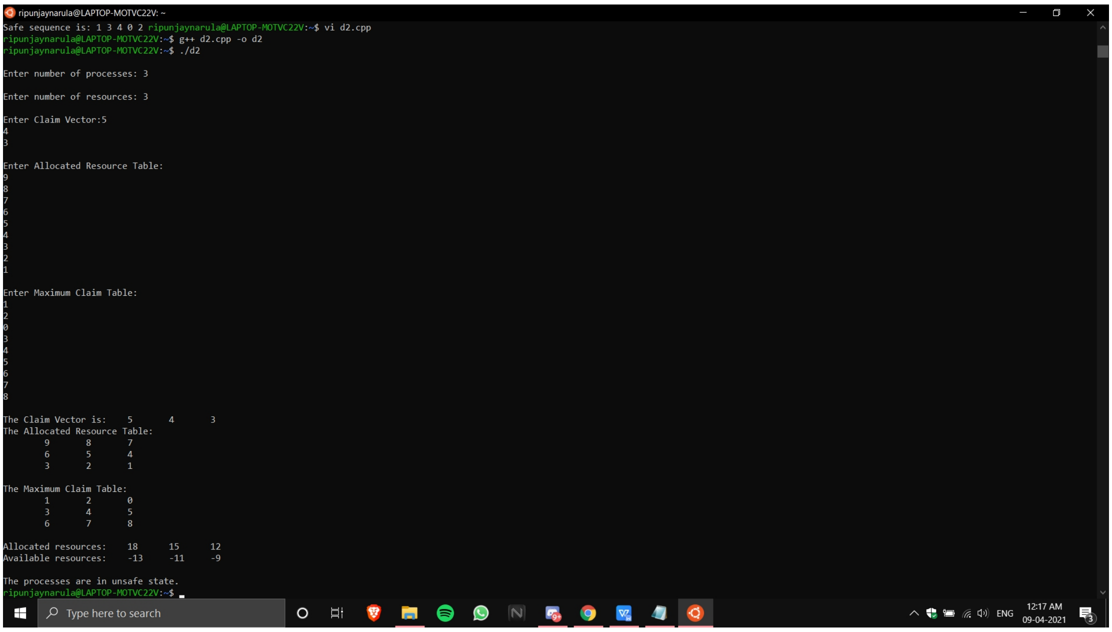


```
ripunjaynarula@LAPTOP-MOTVC22V:~$ ./d
System is in safe state.
Safe sequence is: 1 3 4 0 2 ripunjaynarula@LAPTOP-MOTVC22V:~$ _
```

Additional Resource Request:

```
#include <stdio.h>
int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;

int main()
{
	printf("\nEnter number of processes: ");
	scanf("%d", &processes);

	for (i = 0; i < processes; i++)
	{
		running[i] = 1; counter++;
	}

	printf("\nEnter number of resources: ");
	scanf("%d", &resources);
```

```
    printf("\nEnter Claim Vector:"); for (i = 0; i < resources; i++)
    {

        scanf("%d", &maxres[i]);
    }

    printf("\nEnter Allocated Resource Table:\n");

    for (i = 0; i < processes; i++)
    {
        for(j = 0; j < resources; j++)
        {
            scanf("%d", &current[i][j]);
        }
    }

    printf("\nEnter Maximum Claim Table:\n");

    for (i = 0; i < processes; i++)
    {
        for(j = 0; j < resources; j++)
        {
            scanf("%d", &maximum_claim[i][j]);
        }
    }

    printf("\nThe Claim Vector is: ");

    for (i = 0; i < resources; i++)
    {
{
        printf("\t%d", maxres[i]);
    }

    printf("\nThe Allocated Resource Table:\n");

    for (i = 0; i < processes; i++)
    {
        for (j = 0; j < resources; j++)
        {
            printf("\t%d", current[i][j]);
        }
        printf("\n");
    }
    printf("\nThe Maximum Claim Table:\n");

    for (i = 0; i < processes; i++)
    {
        for (j = 0; j < resources; j++)
        {
            printf("\t%d", maximum_claim[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < processes; i++)
    {
        for (j = 0; j < resources; j++)
        {
            allocation[j] += current[i][j];
        }
    }

    printf("\nAllocated resources:");

    for (i = 0; i < resources; i++)
    {
        printf("\t%d", allocation[i]);
    }

    for (i = 0; i < resources; i++)
    {
        available[i] = maxres[i] - allocation[i];
    }

    printf("\nAvailable resources:");

for (i = 0; i < resources; i++)
    {
        printf("\t%d", available[i]);
    }
    printf("\n");


    while (counter != 0)
    {
        safe = 0;
        for (i = 0; i < processes; i++)
        {
            if (running[i])
            {
                exec = 1;
                for (j = 0; j < resources; j++)
                {
                    if (maximum_claim[i][j] - current[i][j] >available[j])


                    {
                        exec = 0; break;
                    }
                }

                if (exec)
                {
                    printf("\nProcess%d is executing\n", i + 1); running[i] = 0;
                    counter--;

                    safe = 1;

                    for (j = 0; j < resources; j++)
                    {
                        available[j] += current[i][j];
                    }
                    break;
                }
            }
        }
        if (!safe)
        {
            printf("\nThe processes are in unsafe state.\n");

            break;
        }
 else
        {
            printf("\nThe process is in safe state");
            printf("\nAvailable vector:");
            for (i = 0; i < resources; i++)
            {
                printf("\t%d", available[i]);
            }
            printf("\n");
        }
    }
    return 0;
}
```



```
Safe sequence is: 1 3 4 0 2 ripunjaynarula@LAPTOP-MOTVC22V:~$ vi d2.cpp
ripunjaynarula@LAPTOP-MOTVC22V:~$ g++ d2.cpp -o d2
ripunjaynarula@LAPTOP-MOTVC22V:~$ ./d2

Enter number of processes: 3

Enter number of resources: 3

Enter Claim Vector:5
4
3

Enter Allocated Resource Table:
9
8
7
6
5
4
3
2
1

Enter Maximum Claim Table:
1
2
0
3
4
5
6
7
8

The Claim Vector is:    5       4       3
The Allocated Resource Table:
        9       8       7
        6       5       4
        3       2       1

The Maximum Claim Table:
        1       2       0
        3       4       5
        6       7       8

Allocated resources:    18      15      12
Available resources:    -13     -11     -9

The processes are in unsafe state.
ripunjaynarula@LAPTOP-MOTVC22V:~$
```