Kulvir Singh

19BCE2074

CSE2005 – Operating Systems Lab

Assessment 2

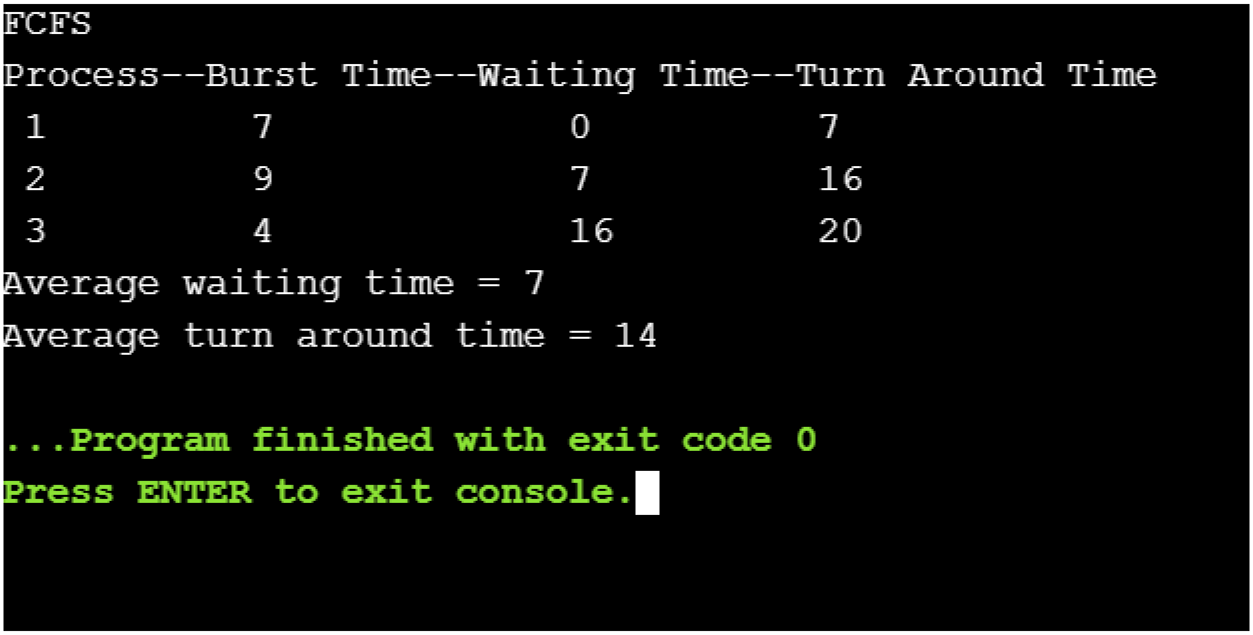
a) Implement the various process scheduling algorithms such as FCFS, SJF, Priority (Non Preemptive).

FCFS

CODE:

```
#include<stdio.h>
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 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 wt[n], tat[n], total_wt = 0, total_tat = 0;
        findWaitingTime(processes, n, bt, wt);
        findTurnAroundTime(processes, n, bt, wt, tat);
        printf("Process--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];
                printf(" %d ",(i+1));
                printf("           %d ", bt[i] );
                printf("       %d",wt[i] );
                printf("    %d\n",tat[i] );
        }
        int s=(float)total_wt / (float)n;
        int t=(float)total_tat / (float)n;
        printf("Average waiting time = %d",s);
        printf("\n");
        printf("Average turn around time = %d ",t);
}
int main()
{
   printf("FCFS\n");
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];
        int burst_time[] = {7, 9, 4};
        findavgTime(processes, n, burst_time);
        return 0;
}
```

OUTPUT SCREENSHOT



```
FCFS
Process--Burst Time--Waiting Time--Turn Around Time
 1         7             0          7
 2         9             7          16
 3         4             16         20
Average waiting time = 7
Average turn around time = 14

...Program finished with exit code 0
Press ENTER to exit console.
```

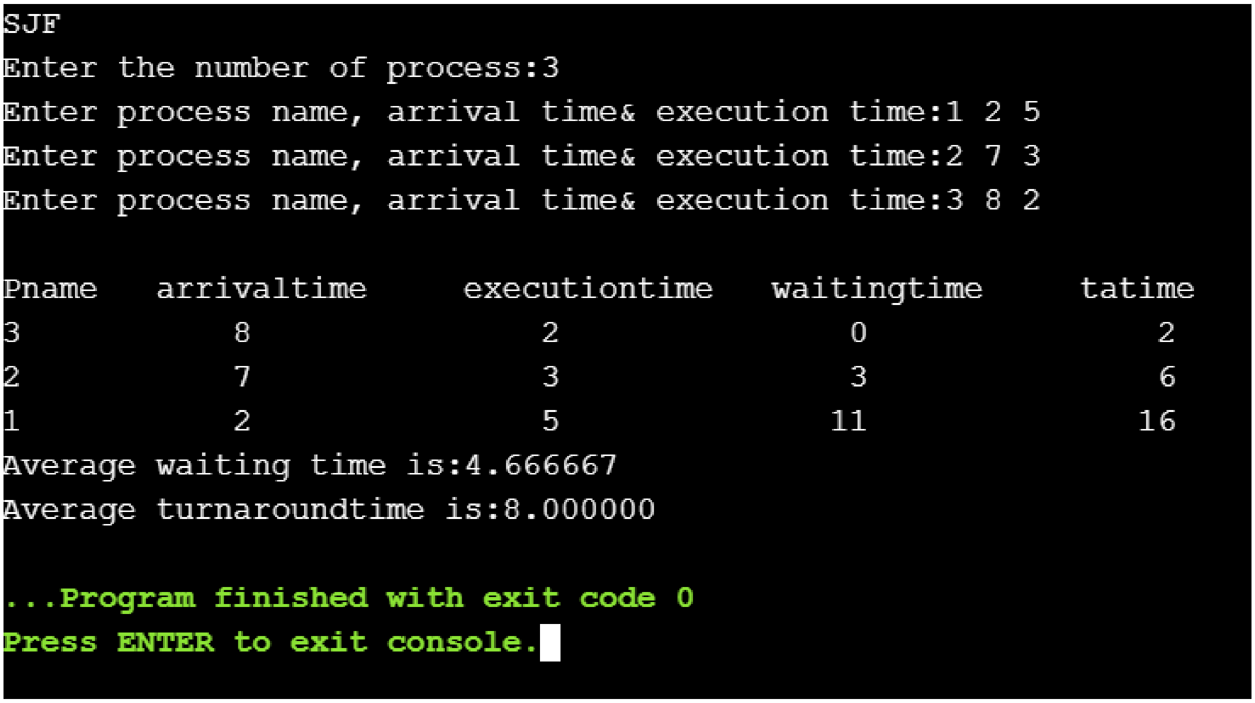
SJF

<u>CODE:</u>

```
#include<stdio.h>
#include<string.h>
void main()
{
   printf("SJF\n");
   int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
   int totwt=0,totta=0;
   float awt,ata;
   char pn[10][10],t[10];
   printf("Enter the number of process:");
   scanf("%d",&n);
   for(i=0; i<n; i++)
   {
      printf("Enter process name, arrival time& execution time:");
      scanf("%s%d%d",pn[i],&at[i],&et[i]);
   }
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
      {
         if(et[i]<et[j])
         {
            temp=at[i];
            at[i]=at[j];
            at[j]=temp;
            temp=et[i];
            et[i]=et[j];
            et[j]=temp;
            strcpy(t,pn[i]);
            strcpy(pn[i],pn[j]);
            strcpy(pn[j],t);
         }
      }
   for(i=0; i<n; i++)
   {
      if(i==0)
         st[i]=at[i];
      else
         st[i]=ft[i-1];
      wt[i]=st[i]-at[i];
      ft[i]=st[i]+et[i];
```

```
    ta[i]=ft[i]-at[i];
    totwt+=wt[i];
    totta+=ta[i];
  }
  awt=(float)totwt/n;
  ata=(float)totta/n;
  printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
  for(i=0; i<n; i++)
    printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
  printf("\nAverage waiting time is:%f",awt);
  printf("\nAverage turnaroundtime is:%f",ata);
  getch();
}
```

OUTPUT SCREENSHOT



```
SJF
Enter the number of process:3
Enter process name, arrival time& execution time:1 2 5
Enter process name, arrival time& execution time:2 7 3
Enter process name, arrival time& execution time:3 8 2

Pname   arrivaltime     executiontime   waitingtime     tatime
3           8               2               0               2
2           7               3               3               6
1           2               5              11              16
Average waiting time is:4.666667
Average turnaroundtime is:8.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

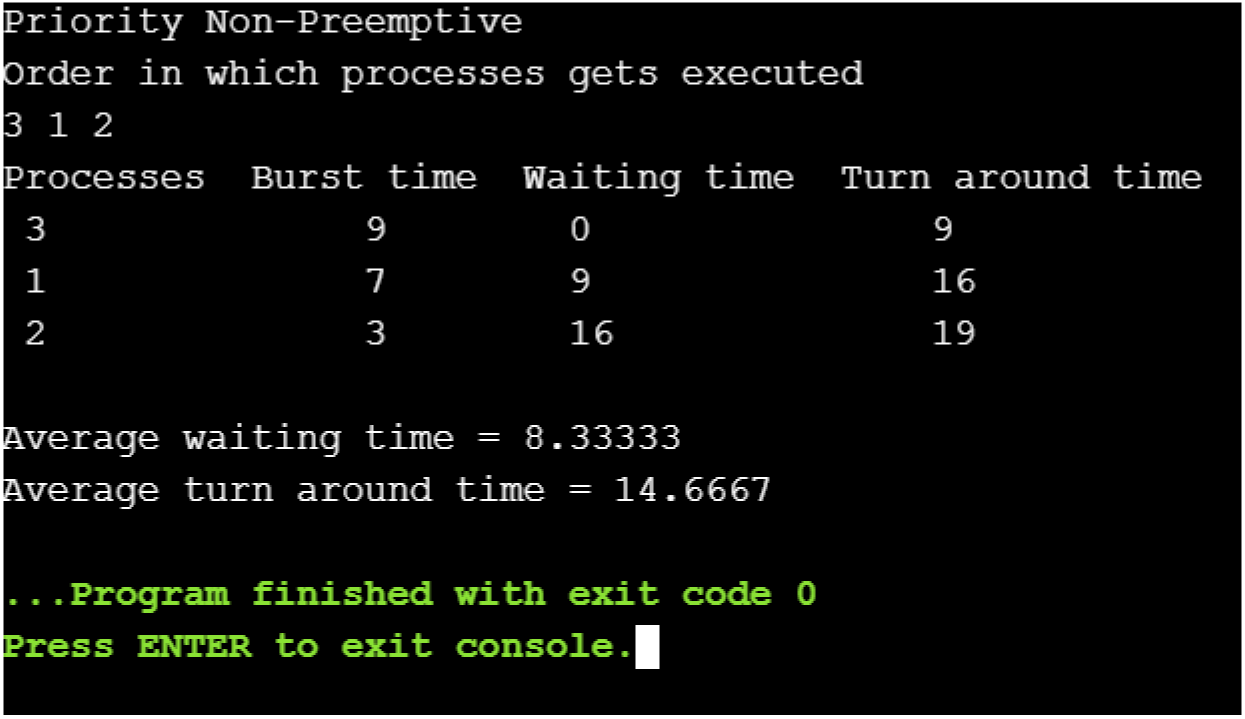
Priority

CODE:

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

struct Process
{
	int pid;
	int bt;
	int priority;
};
bool comparison(Process a, Process b)
{
	return (a.priority > b.priority);
}
void findWaitingTime(Process proc[], int n, int wt[])
{

	wt[0] = 0;
	for (int i = 1; i < n ; i++ )
		wt[i] = proc[i-1].bt + wt[i-1] ;
}
void findTurnAroundTime( Process proc[], int n,	int wt[], int tat[])
{
	for (int i = 0; i < n ; i++)
		tat[i] = proc[i].bt + wt[i];
}

void findavgTime(Process proc[], int n)
{
	int wt[n], tat[n], total_wt = 0, total_tat = 0;
	findWaitingTime(proc, n, wt);
	findTurnAroundTime(proc, n, wt, tat);
	cout << "\nProcesses "<< " 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 << " " << proc[i].pid << "\t\t"
			<< proc[i].bt << "\t " << wt[i]
			<< "\t\t " << tat[i] <<endl;
```

```
        }

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

void priorityScheduling(Process proc[], int n)
{
        sort(proc, proc + n, comparison);
        cout<< "Order in which processes gets executed \n";
        for (int i = 0 ; i < n; i++)
                cout << proc[i].pid <<" " ;

        findavgTime(proc, n);
}
int main()
{
  cout<<"Priority Non-Preemptive\n";
        Process proc[] = {{1, 7, 3}, {2, 3, 2}, {3, 9, 4}};
        int n = sizeof proc / sizeof proc[0];
        priorityScheduling(proc, n);
        return 0;
}
```

OUTPUT SCREENSHOT



```
Priority Non-Preemptive
Order in which processes gets executed
3 1 2
Processes  Burst time  Waiting time  Turn around time
 3              9        0                9
 1              7        9                16
 2              3        16               19

Average waiting time = 8.33333
Average turn around time = 14.6667

...Program finished with exit code 0
Press ENTER to exit console.
```

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

Priority Preemptive

CODE:

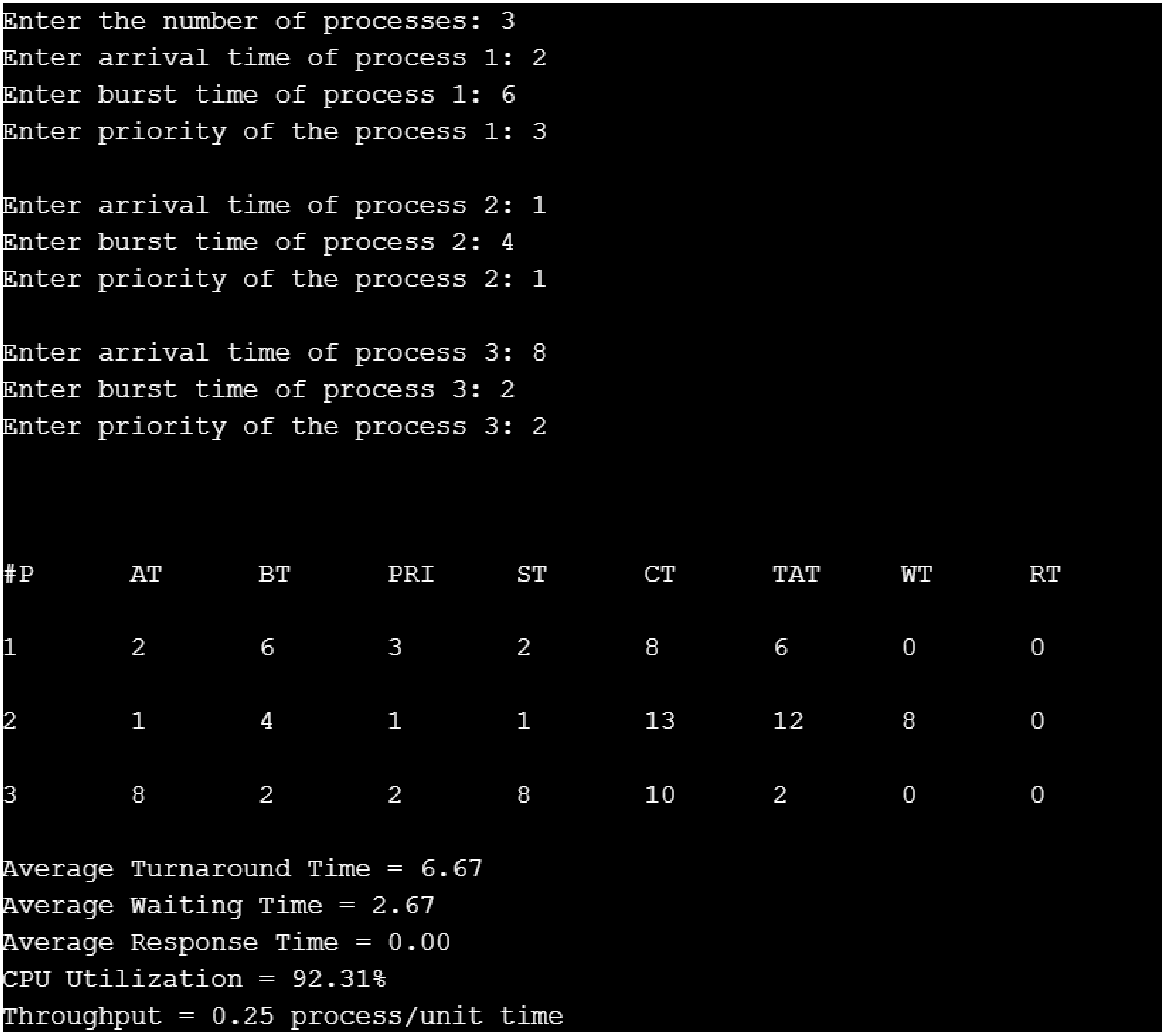
```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

struct process {
   int pid;
   int arrival_time;
   int burst_time;
   int priority;
   int start_time;
   int completion_time;
   int turnaround_time;
   int waiting_time;
   int response_time;
};

int main() {

   int n;
   struct process p[100];
   float avg_turnaround_time;
   float avg_waiting_time;
   float avg_response_time;
   float cpu_utilisation;
   int total_turnaround_time = 0;
   int total_waiting_time = 0;
   int total_response_time = 0;
   int total_idle_time = 0;
   float throughput;
   int burst_remaining[100];
   int is_completed[100];
   memset(is_completed,0,sizeof(is_completed));

   cout << setprecision(2) << fixed;
```

```
cout<<"Enter the number of processes: ";
cin>>n;

for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    cout<<"Enter priority of the process "<<i+1<<": ";
    cin>>p[i].priority;
    p[i].pid = i+1;
    burst_remaining[i] = p[i].burst_time;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mx = -1;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(p[i].priority > mx) {
                mx = p[i].priority;
                idx = i;
            }
            if(p[i].priority == mx) {
                if(p[i].arrival_time < p[idx].arrival_time) {
                    mx = p[i].priority;
                    idx = i;
                }
            }
        }
    }

    if(idx != -1) {
        if(burst_remaining[idx] == p[idx].burst_time) {
            p[idx].start_time = current_time;
            total_idle_time += p[idx].start_time - prev;
        }
        burst_remaining[idx] -= 1;
```

```
            current_time++;
            prev = current_time;

            if(burst_remaining[idx] == 0) {
                p[idx].completion_time = current_time;
                p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
                p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
                p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

                total_turnaround_time += p[idx].turnaround_time;
                total_waiting_time += p[idx].waiting_time;
                total_response_time += p[idx].response_time;

                is_completed[idx] = 1;
                completed++;
            }
        }
        else {
             current_time++;
        }
    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for(int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
        max_completion_time = max(max_completion_time,p[i].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time
)*100;
    throughput = float(n) / (max_completion_time - min_arrival_time);

    cout<<endl<<endl;


cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<
endl;

    for(int i = 0; i < n; i++) {
```

```
cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].priority<<"\t"<<p[i].
start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time
<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
   }
   cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
   cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
   cout<<"Average Response Time = "<<avg_response_time<<endl;
   cout<<"CPU Utilization = "<<cpu_utilisation<<"%"<<endl;
   cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;

}
```

<u>OUTPUT SCREENSHOT</u>



```
Enter the number of processes: 3
Enter arrival time of process 1: 2
Enter burst time of process 1: 6
Enter priority of the process 1: 3

Enter arrival time of process 2: 1
Enter burst time of process 2: 4
Enter priority of the process 2: 1

Enter arrival time of process 3: 8
Enter burst time of process 3: 2
Enter priority of the process 3: 2



#P      AT      BT      PRI     ST      CT      TAT     WT      RT

1       2       6       3       2       8       6       0       0

2       1       4       1       1       13      12      8       0

3       8       2       2       8       10      2       0       0

Average Turnaround Time = 6.67
Average Waiting Time = 2.67
Average Response Time = 0.00
CPU Utilization = 92.31%
Throughput = 0.25 process/unit time
```

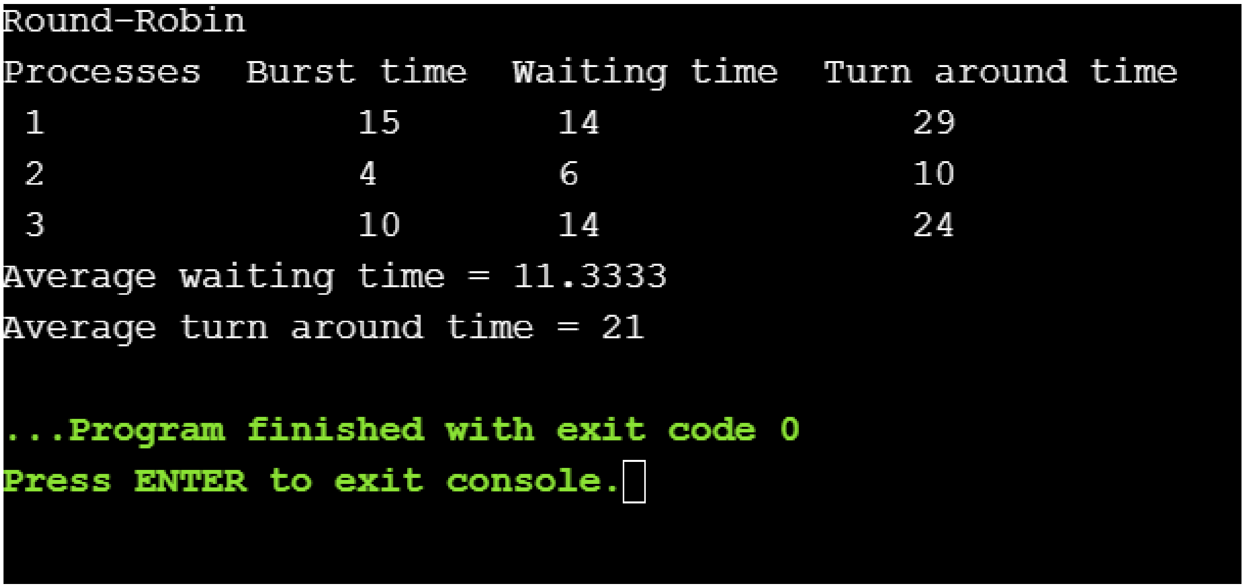
Round robin

CODE:

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

void findWaitingTime(int processes[], int n,
            int bt[], int wt[], int quantum)
{
    int rem_bt[n];
    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[n], tat[n], 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 main()
{
  cout<<"Round-Robin\n";
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];
    int burst_time[] = {15, 4, 10};
    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}
```

__OUTPUT SCREENSHOT__



```
Round-Robin
Processes  Burst time  Waiting time  Turn around time
 1              15        14              29
 2              4         6               10
 3              10        14              24
Average waiting time = 11.3333
Average turn around time = 21

...Program finished with exit code 0
Press ENTER to exit console.
```

c) Consider a corporate hospital where we have n number of patients waiting for consultation. The amount of time required to serve a patient may vary, say 10 to 30 minutes. If a patient arrives with an emergency,he /she should be attended immediately before other patients, which may increase the waiting time of other patients. If you are given this problem with the following algorithms how would our devise an effective scheduling so that it optimizes the overall performance such as minimizing the waiting time of all patients. [Single queue or multi-level queue can be used].

Consider the availability of single and multiple doctors • Assign top priority for patients with emergency case, women, children, elders, and youngsters. • Patients coming for review may take less time than others. This can be taken into account while using SJF.

1. FCFS

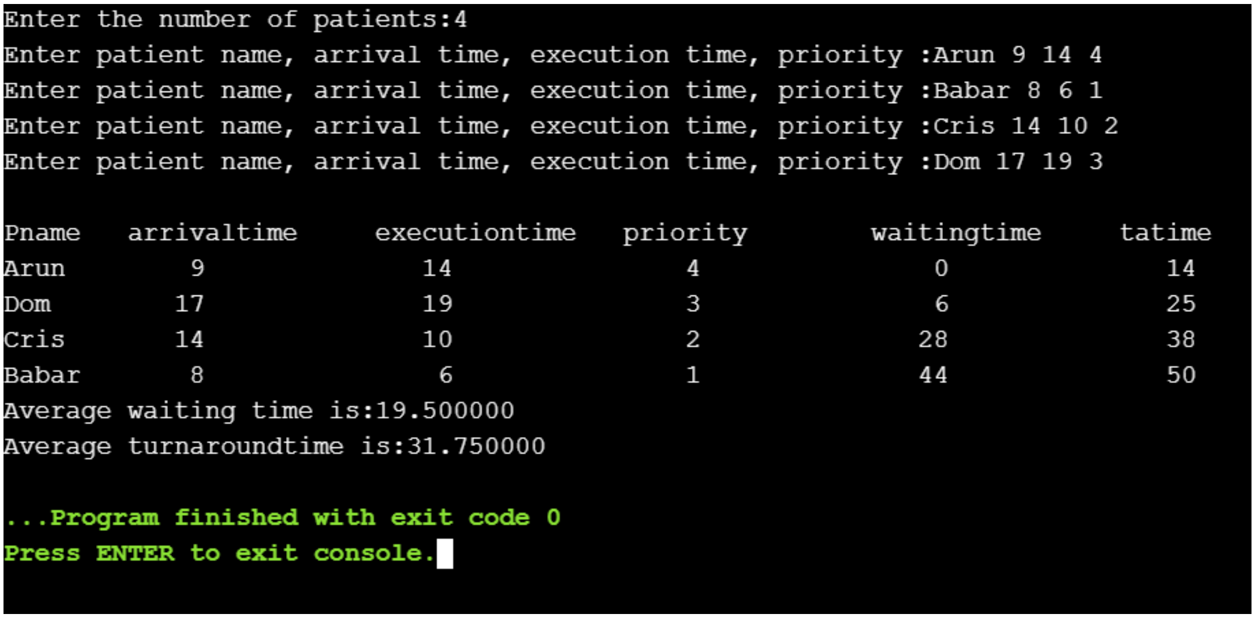
2. SJF (primitive and non-pre-emptive)

CODE:

```
#include<stdio.h>
#include<string.h>
void main(){
   int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
   char ch;int totwt=0,totta=0;
   float awt,ata;
   char pn[10][10],t[10];
   printf("Enter the number of patients:");
   scanf("%d",&n);
   for(i=0; i<n; i++)
   {
      printf("Enter patient name, arrival time, execution time, priority :");
      scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);}
      for(i=0; i<n; i++)
      for(j=0; j<n; j++)
      {
         if(p[i]>p[j])
         {
            temp=p[i];
            p[i]=p[j];
            p[j]=temp;
            temp=at[i];
            at[i]=at[j];
            at[j]=temp;
            temp=et[i];
            et[i]=et[j];
            et[j]=temp;
            strcpy(t,pn[i]);
            strcpy(pn[i],pn[j]);
            strcpy(pn[j],t);

         }

      }
      for(i=0; i<n; i++)
      {
         if(i==0)
         {
            st[i]=at[i];
            wt[i]=st[i]-at[i];
            ft[i]=st[i]+et[i];
```

```
        ta[i]=ft[i]-at[i];

      }
      else
      {
        st[i]=ft[i-1];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];

      }
      totwt+=wt[i];
      totta+=ta[i];

    }
    awt=(float)totwt/n;
    ata=(float)totta/n;
    printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
    for(i=0; i<n; i++)
    printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
    printf("\nAverage waiting time is:%f",awt);
    printf("\nAverage turnaroundtime is:%f",ata);

}
```

OUTPUT SCREENSHOT



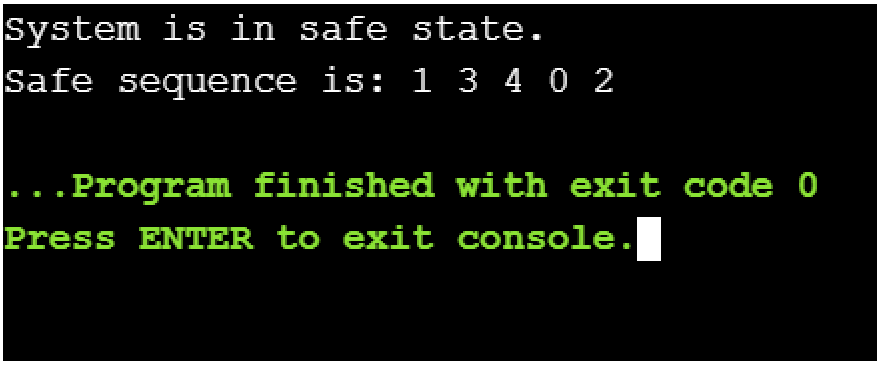
```
Enter the number of patients:4
Enter patient name, arrival time, execution time, priority :Arun 9 14 4
Enter patient name, arrival time, execution time, priority :Babar 8 6 1
Enter patient name, arrival time, execution time, priority :Cris 14 10 2
Enter patient name, arrival time, execution time, priority :Dom 17 19 3

Pname   arrivaltime     executiontime   priority        waitingtime     tatime
Arun        9               14              4               0              14
Dom        17               19              3               6              25
Cris       14               10              2              28              38
Babar       8                6              1              44              50
Average waiting time is:19.500000
Average turnaroundtime is:31.750000

...Program finished with exit code 0
Press ENTER to exit console.
```

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

CODE:

```
#include<iostream>
using namespace std;
const int P = 5;
const int R = 3;
void calculateNeed(int need[P][R], int maxm[P][R],
                                        int allot[P][R])
{
          for (int i = 0 ; i < P ; i++)
                    for (int j = 0 ; j < R ; j++)
                              need[i][j] = maxm[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];
          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] += allot[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 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;
}
```

OUTPUT SCREENSHOT



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

...Program finished with exit code 0
Press ENTER to exit console.
```