DELHI TECHNOLOGICAL UNIVERSITY

OPERATING SYSTEMS PRACTICAL FILE

Submitted By :

Dhananjay Negi

2k17/CO/108

Index

S. No.	Experiments	Date	Sign	Remarks

PROGRAM NO. 1

AIM:

Write a program to find the waiting time and turn around time of n processes given their burst time using FCFS(First Come First Serve) Scheduling Algorithm

INTRODUCTION:

First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.
In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

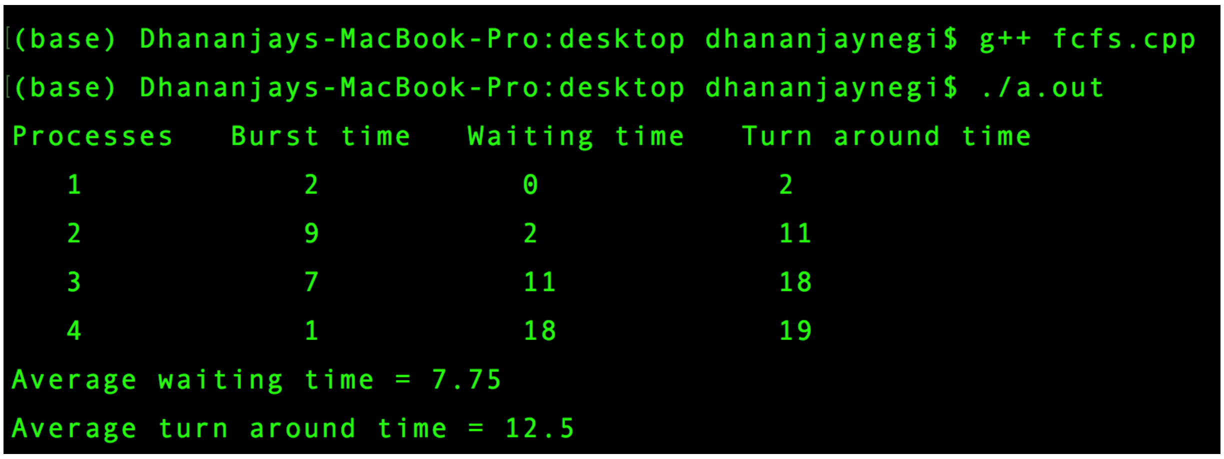
ALGORITHM:

1. Start.
2. Input the processes along with their burst time (bt).
3. Find waiting time (wt) for all processes.
4. As first process that comes need not to wait so waiting time for process no. 1 is 0.
5. Find waiting time for all other processes i.e. for process i -> wt[i] = bt[i-1] + wt[i-1] .
6. Find turnaround time = waiting_time + burst_time for all processes.
7. Find average waiting time = total_waiting_time / no_of_processes.
8. Similarly, find average turnaround time = total_turn_around_time / no_of_processes.
9. Stop.

CODE:

```
#include<iostream>
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 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);
    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;
    cout<<endl;
}
int main()
{
    int processes[] = { 1, 2, 3, 4};
    int n = sizeof(processes) / sizeof(processes[0]);
    int  burst_time[] = {2, 9, 7, 1};
    findavgTime(processes, n,  burst_time);
    return 0;
}
```

OUTPUT:



```
[(base) Dhananjays-MacBook-Pro:desktop dhananjaynegi$ g++ fcfs.cpp
[(base) Dhananjays-MacBook-Pro:desktop dhananjaynegi$ ./a.out
 Processes    Burst time    Waiting time    Turn around time
    1            2            0               2
    2            9            2               11
    3            7            11              18
    4            1            18              19
 Average waiting time = 7.75
 Average turn around time = 12.5
```

RESULT:

We have successfully found the waiting time and turn around time of n processes given their burst time using FCFS(First Come First Serve) Scheduling Algorithm

CONCLUSION:

In this program we implemented FCFS Scheduling Algorithm and with the help of that successfully evaluated the waiting time and turn around time of certain processes.

PROGRAM NO. 2

AIM:

Write a program to find the waiting time and turn around time of n processes given their burst time using SJF(Shortest Job First) Scheduling Algorithm.

INTRODUCTION:

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm. Shortest Job first has the advantage of having minimum average waiting time among all scheduling algorithms. It is a Greedy Algorithm. It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.

ALGORITHM:

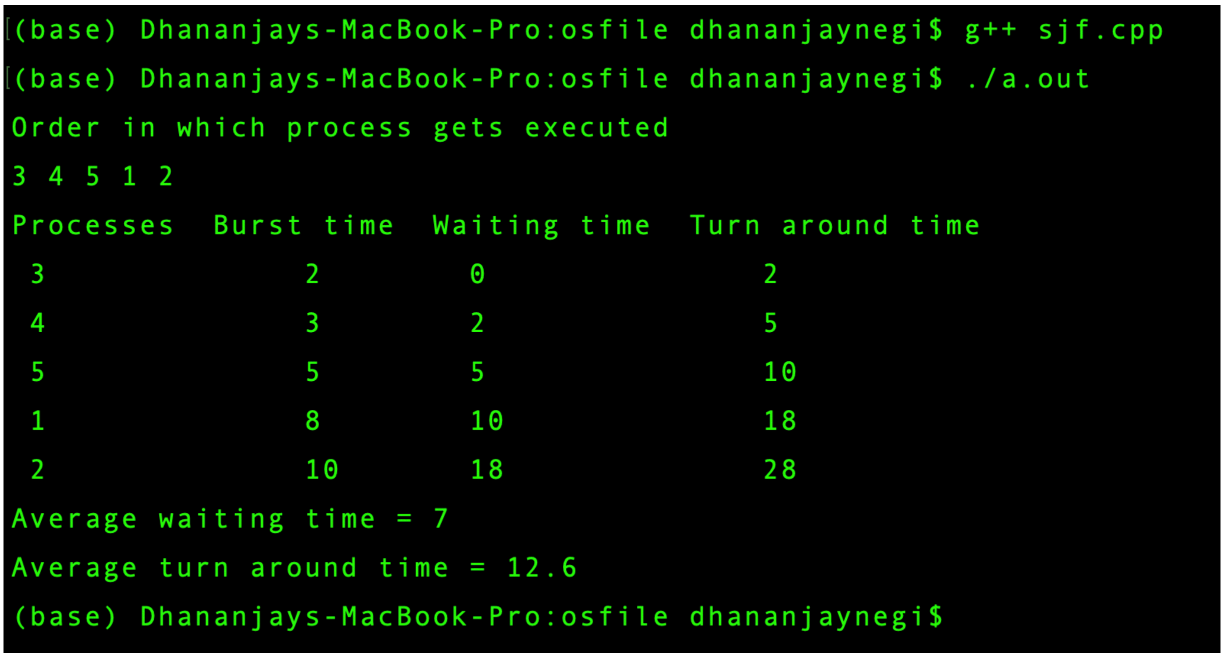
1. Start
2. Input the processes along with their burst time (bt).
3. Find waiting time (wt) for all processes.
4. Sort all processes in increasing order according to their burst time.
5. Now the first process that comes need not to wait as it will be the shortest job.
6. Find waiting time for all other processes i.e. for process i -> wt[i] = bt[i-1] + wt[i-1] .
7. Find turnaround time = waiting_time + burst_time for all processes.
8. Find average waiting time = total_waiting_time / no_of_processes.
9. Similarly, find average turnaround time = total_turn_around_time / no_of_processes.
10. Stop

CODE:

```
#include<bits/stdc++.h>
using namespace std;
struct Process
{
int pid;
int bt;
};
bool comparison(Process a, Process b)
{
	return (a.bt < b.bt);
}
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 << "Average waiting time = "<< (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "<< (float)total_tat / (float)n;
}

int main()
{
        Process proc[] = {{1, 8}, {2, 10}, {3, 2}, {4, 3}, {5,5}};
        int n = sizeof proc / sizeof proc[0];
        sort(proc, proc + n, comparison);
        cout << "Order in which process gets executed\n";
        for (int i = 0 ; i < n; i++)
                cout << proc[i].pid <<" ";
        findavgTime(proc, n);
        cout<<endl;
        return 0;
}
```

OUTPUT:



```
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ g++ sjf.cpp
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ ./a.out
Order in which process gets executed
3 4 5 1 2
Processes  Burst time  Waiting time  Turn around time
 3             2        0               2
 4             3        2               5
 5             5        5               10
 1             8        10              18
 2             10       18              28
Average waiting time = 7
Average turn around time = 12.6
(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$
```

RESULT:

We have successfully found the waiting time and turn around time of n processes given their burst time using SJF(Shortest Job First) Scheduling Algorithm

CONCLUSION:

In this program we implemented SJF(Shortest Job First) Scheduling Algorithm and with the help of that successfully evaluated the waiting time and turn around time of certain processes.

PROGRAM NO 3

AIM:

Write a program to find the waiting time and turn around time of n processes given their burst time using SRTF(Shortest Remaining Time First) Scheduling Algorithm

INTRODUCTION:

In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

ALGORITHM:

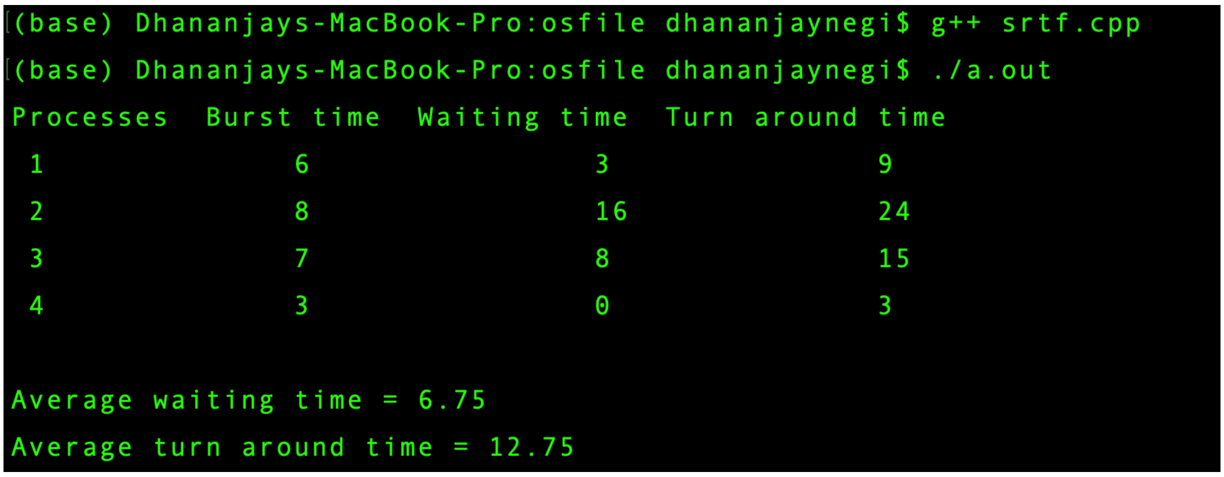
1. Start
2. Traverse until all process gets completely executed.
3. Find process with minimum remaining time at every single time lap.
4. Reduce its time by 1.
5. Check if its remaining time becomes 0.
6. Increment the counter of process completion.
7. Completion time of current process = current_time +1;
8. Calculate waiting time for each completed process.

   wt[i]= Completion time - arrival_time-burst_time
9. Increment time lap by one.
10. Find turnaround time (waiting_time+burst_time).
11. Stop

CODE:

```
#include <bits/stdc++.h>
using namespace std;
struct Process {
        int pid;
        int bt;
        int art;
};
void findWaitingTime(Process proc[], int n,int wt[])
{
        int rt[n];
        for (int i = 0; i < n; i++)
                rt[i] = proc[i].bt;
        int complete = 0, t = 0, minm = INT_MAX;
        int shortest = 0, finish_time;
        bool check = false;
        while (complete != n) {
                for (int j = 0; j < n; j++) {
                        if ((proc[j].art <= t) &&
                        (rt[j] < minm) && rt[j] > 0) {
                                minm = rt[j];
                                shortest = j;
                                check = true;
                        }
                }
                if (check == false) {
                        t++;
                        continue;
                }
                rt[shortest]--;
                minm = rt[shortest];
                if (minm == 0)
                        minm = INT_MAX;
                if (rt[shortest] == 0) {
                        complete++;
                        check = false;
                        finish_time = t + 1;
                        wt[shortest] = finish_time -proc[shortest].bt -
proc[shortest].art;
                        if (wt[shortest] < 0)
                                wt[shortest] = 0;
                }
                t++;
        }
}
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 << "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 << " " << proc[i].pid << "\t\t"
                        << proc[i].bt << "\t\t " << wt[i]
                        << "\t\t " << tat[i] << endl;
        }
        cout << "\nAverage waiting time = "<< (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "<< (float)total_tat / (float)n;
}
int main()
{
        Process proc[] = { { 1, 6, 1 }, { 2, 8, 1 },{ 3, 7, 2 }, { 4, 3, 3 } };
        int n = sizeof(proc) / sizeof(proc[0]);
        findavgTime(proc, n);
        cout<<endl;
        return 0;
}
```

OUTPUT:



```
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ g++ srtf.cpp
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ ./a.out
Processes  Burst time  Waiting time  Turn around time
 1              6               3              9
 2              8               16             24
 3              7               8              15
 4              3               0              3

Average waiting time = 6.75
Average turn around time = 12.75
```

RESULT:

We have successfully found the waiting time and turn around time of n processes given their burst time using SRTF(Shortest Remaining Time First) Scheduling Algorithm

CONCLUSION:

In this program we implemented SRTF(Shortest Remaining Time First) Scheduling Algorithm and with the help of that successfully evaluated the waiting time and turn around time of certain processes.

PROGRAM NO. 4

AIM:

Write a program to find the waiting time and turn around time of n processes given their burst time using Round Robin Scheduling Algorithm

INTRODUCTION:

This algorithm is known as preemptive version of FCFS as discussed earlier, it executes the process on the basis of first come first serve, and the only difference here is it works on the principle of quantum time. Quantum time is defined by the amount of the time a CPU is assign to be executed is known as the quantum time independent of the actual burst time, a process will get scheduled in quantum parts values or we can say in quantum chunks.

ALGORITHM:

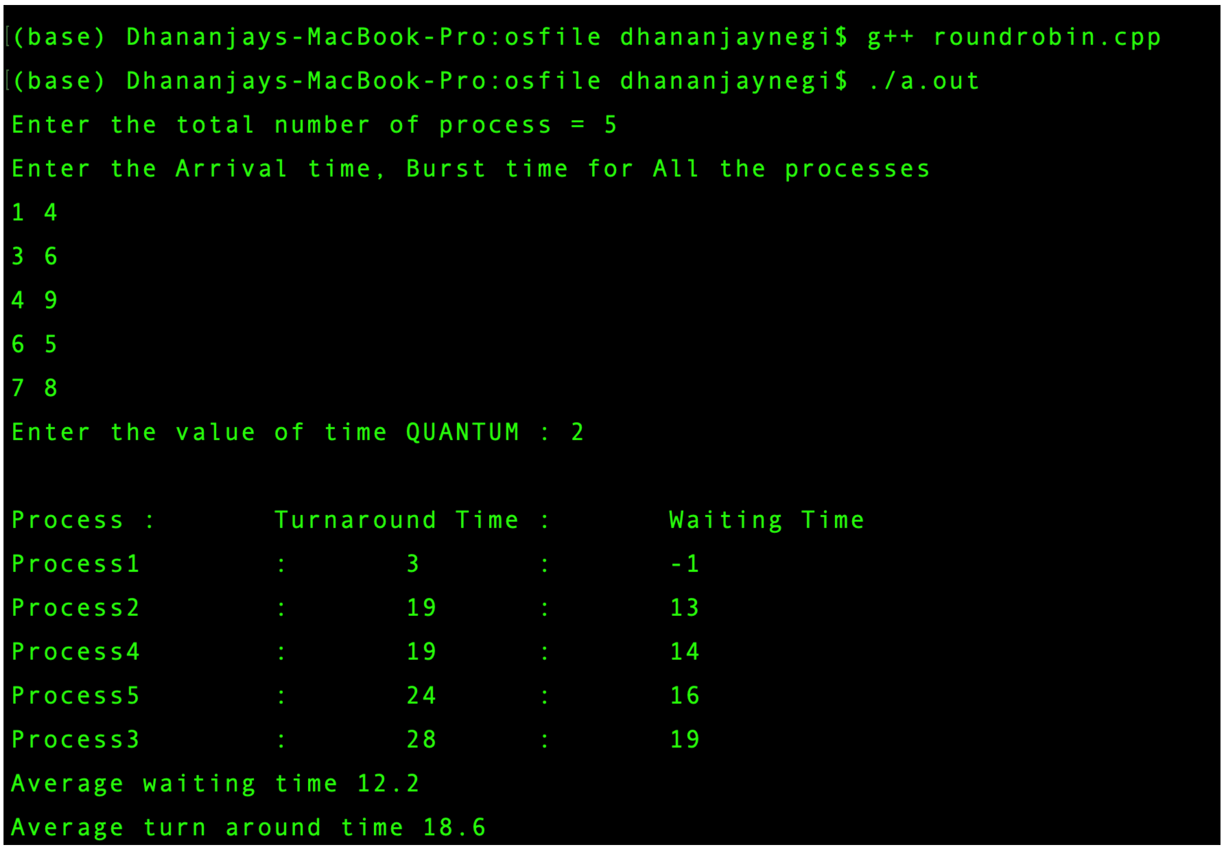
```
1. Start
2. Create a vector to keep track of remaining burst time of    processes.
3. Create another vector to store waiting times of processes and initialize
   this vector as 0.
4. Initialize time : t = 0
5. Keep traversing the all processes while all processes are not done.
6.  Do following for i'th process if it is not done yet.
   a- If remaining time for process i > quantum
          (i)  t = t + quantum
          (ii) reduce remaining time by quantum
   b- Else
          (i)  Increment t by amount of burst time remaining for process i
          (ii) Find the value of waiting time
          (ii) This process is over
7. Stop.
```

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
	int i,n,time,remain,temps=0,time_quantum;
	int wt=0,tat=0;
	cout<<"Enter the total number of process = ";
	cin>>n;
	remain=n;
	vector<int>at(n);
	vector<int>bt(n);
	vector<int>rt(n);
	cout<<"Enter the Arrival time, Burst time for All the processes"<<endl;

	for(i=0;i<n;i++)
	{
		cin>>at[i];
		cin>>bt[i];
		rt[i]=bt[i];
	}

	cout<<"Enter the value of time QUANTUM : ";
	cin>>time_quantum;

	cout<<"\nProcess\t:\tTurnaround Time\t:\tWaiting Time\n";
```

```
        for(time=0,i=0;remain!=0;)
        {
                if(rt[i]<=time_quantum && rt[i]>0)
                {
                        time += rt[i];
                        rt[i]=0;
                        temps=1;
                }
                else if(rt[i]>0)
                {
                        rt[i] -= time_quantum;
                        time += time_quantum;
                }

                if(rt[i]==0 && temps==1)
                {
                        remain--;
                        cout<<"Process"<< i+1 << "\t"<<":"<<"\t"<<time-at[i]<<"\t:
\t"<<time-at[i]-bt[i]<<endl;
                        wt += time-at[i]-bt[i];
                        tat += time-at[i];
                        temps=0;
                }
                if(i == n-1)
                        i=0;
                else if(at[i+1] <= time)
                        i++;
                else
                        i=0;
        }
        cout<<"Average waiting time "<<wt*1.0/n<<endl;
        cout<<"Average turn around time "<<tat*1.0/n;
        cout<<endl;

        return 0;
}
```

OUTPUT:



```
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ g++ roundrobin.cpp
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ ./a.out
 Enter the total number of process = 5
 Enter the Arrival time, Burst time for All the processes
 1 4
 3 6
 4 9
 6 5
 7 8
 Enter the value of time QUANTUM : 2

 Process :        Turnaround Time :       Waiting Time
 Process1         :       3       :       -1
 Process2         :       19      :       13
 Process4         :       19      :       14
 Process5         :       24      :       16
 Process3         :       28      :       19
 Average waiting time 12.2
 Average turn around time 18.6
```

RESULT:

We have successfully found the waiting time and turn around time of n processes given their burst time using Round Robin Scheduling Algorithm

CONCLUSION:

In this program we implemented Round Robin Scheduling Algorithm and with the help of that successfully evaluated the waiting time and turn around time of certain processes.

PROGRAM NO. 7

AIM:

Write a program to find the waiting time and turn around time of n processes given their burst time and priority using priority scheduling algorithm.

INTRODUCTION:

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 the highest priority is to be executed first and so on.
Processes with the same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

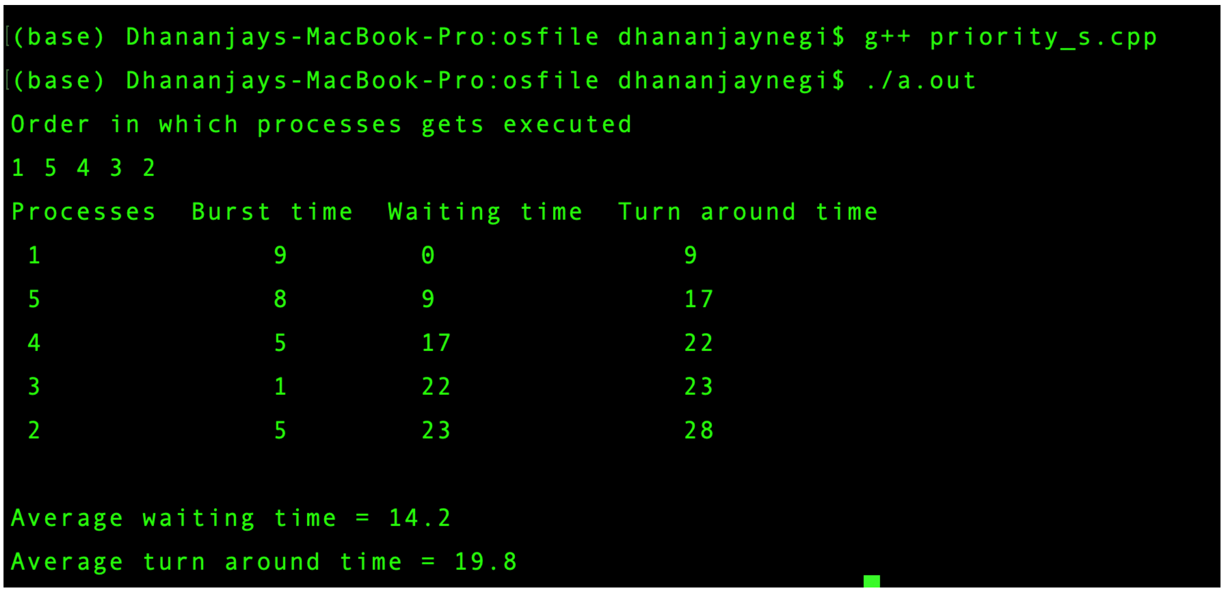
ALGORITHM:

1. Start
2. First Input the processes with their burst time and priority
3. Sort the processes, burst time and priority according to priority
4. Now apply the First Come First Serve Algorithm to find the order of execution
5. Find out the average waiting time and turn around time of given processes.
6. Stop.

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()
{
        Process proc[] = {{1, 9, 5}, {2, 5, 0}, {3, 1, 1}, {4, 5, 2}, {5, 8, 5}};
        int n = sizeof proc / sizeof proc[0];
        priorityScheduling(proc, n);
        cout<<endl;
        return 0;
}
```

OUTPUT:



```
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ g++ priority_s.cpp
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ ./a.out
Order in which processes gets executed
1 5 4 3 2
Processes  Burst time  Waiting time  Turn around time
 1              9        0                9
 5              8        9                17
 4              5        17               22
 3              1        22               23
 2              5        23               28

Average waiting time = 14.2
Average turn around time = 19.8
```

RESULT:

We have successfully found the waiting time and turn around time of n processes given their burst time and priority using Priority based Scheduling Algorithm

CONCLUSION:

In this program we implemented Priority Based Scheduling Algorithm and with the help of that successfully evaluated the waiting time and turn around time of certain processes.

PROGRAM NO. 6

AIM:

Write a program to implement Banker’s Algorithm.

INTRODUCTION:

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

ALGORITHM:

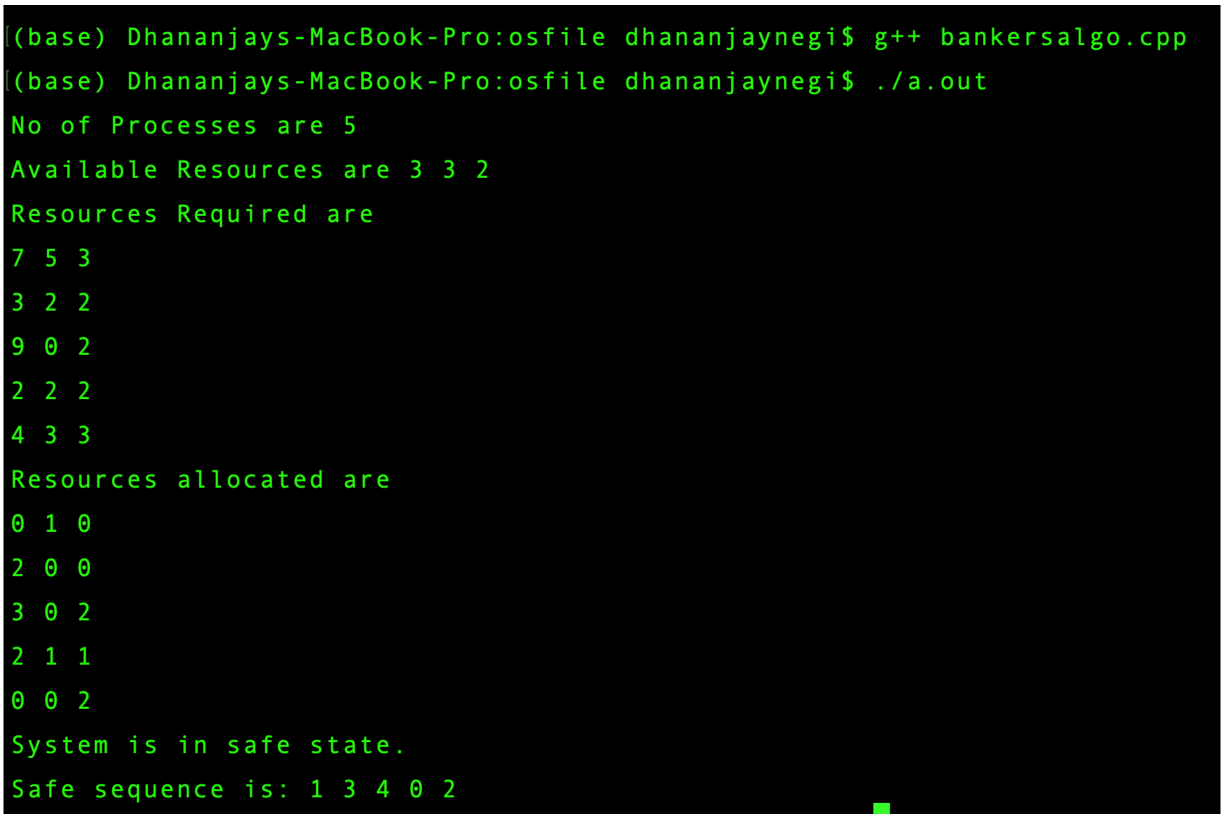
```
1. Start
2. Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
   Initialize: Work= Available
   Finish [i]=false; for i=1,2,……,n
3. Find an i such that both
   a) Finish [i]=false
   b) Need_i<=work
   if no such i exists goto step (4)
4. Work=Work + Allocation_i Finish[i]= truegoto step(2)
5. If Finish[i]=true for all i, then the system is in safe state.
6. Stop.
```

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};
        cout<<"No of Processes are "<<5<<endl;
        int avail[] = {3, 3, 2};
        cout<<"Available Resources are "<<avail[0]<<" "<<avail[1]<<"
"<<avail[2]<<endl;
        int maxm[][R] = {{7, 5, 3},
                                        {3, 2, 2},
                                        {9, 0, 2},
                                        {2, 2, 2},
                                        {4, 3, 3}};
   cout<<"Resources Required are"<<endl;
   for(int i=0;i<5;i++)
   {
      cout<<maxm[i][0]<<" "<<maxm[i][1]<<" "<<maxm[i][2]<<endl;
   }
        int allot[][R] = {{0, 1, 0},
                                         {2, 0, 0},
                                         {3, 0, 2},
                                         {2, 1, 1},
                                         {0, 0, 2}};
        cout<<"Resources allocated are"<<endl;
   for(int i=0;i<5;i++)
   {
      cout<<allot[i][0]<<" "<<allot[i][1]<<" "<<allot[i][2]<<endl;
   }
        isSafe(processes, avail, maxm, allot);
        cout<<endl;
        return 0;
}
```

OUTPUT:



```
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ g++ bankersalgo.cpp
[(base) Dhananjays-MacBook-Pro:osfile dhananjaynegi$ ./a.out
No of Processes are 5
Available Resources are 3 3 2
Resources Required are
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Resources allocated are
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
System is in safe state.
Safe sequence is: 1 3 4 0 2
```

RESULT:

We have successfully found the waiting time and turn around time of n processes given their burst time and priority using Priority based Scheduling Algorithm.

CONCLUSION:

In this program we implemented Priority Based Scheduling Algorithm and with the help of that successfully evaluated the waiting time and turn around time of certain processes.