Name-Aditya Jain

Reg No.-17BCE7066

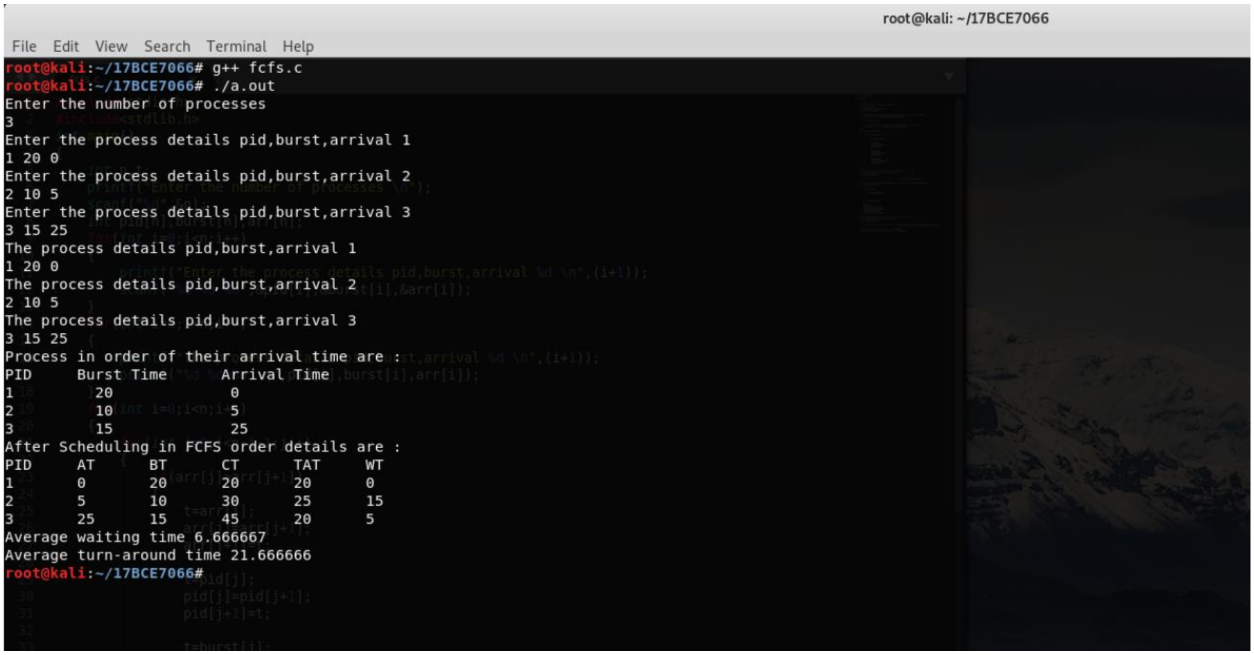
Process Scheduling

First Come First Serve (FCFS)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int n,t;
	printf("Enter the number of processes \n");
	scanf("%d",&n);
	int pid[n],burst[n],arr[n];
	for(int i=0;i<n;i++)
	{
		printf("Enter the process details pid,burst,arrival %d \n",(i+1));
		scanf("%d %d %d",&pid[i],&burst[i],&arr[i]);
	}
	for(int i=0;i<n;i++)
	{
		printf("The process details pid,burst,arrival %d \n",(i+1));
		printf("%d %d %d \n",pid[i],burst[i],arr[i]);
	}
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n-i-1;j++)
		{
			if(arr[j]>arr[j+1])
			{
				t=arr[j];
				arr[j]=arr[j+1];
				arr[j+1]=t;

				t=pid[j];
				pid[j]=pid[j+1];
				pid[j+1]=t;

				t=burst[j];
```

```
                burst[j]=burst[j+1];
                burst[j+1]=t;
            }
        }
    }
    printf("Process in order of their arrival time are : \n");
    printf("PID\tBurst Time\tArrival Time \n");
    for(int i=0;i<n;i++)
    {
        printf("%d  \t  %d        \t %d  \n",pid[i],burst[i],arr[i]);
    }
    int ct[n]={0};int c=0;int tat[n]={0};int wt[n]={0};float avgwt=0,avgtat=0;
    for(int i=0;i<n;i++)
    {
        if(c<arr[i])
        {
            c=arr[i]+burst[i];
        }
        else
        {
            c=c+burst[i];
        }
        ct[i]=c;
        tat[i]=ct[i]-arr[i];
        wt[i]=tat[i]-burst[i];
        avgwt=avgwt+wt[i];
        avgtat=avgtat+tat[i];
    }
    printf("After Scheduling in FCFS order details are :\n");
    printf("PID\tAT\tBT\tCT\tTAT\tWT \n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t \n",pid[i],arr[i],burst[i],ct[i],tat[i],wt[i]);
    }
    printf("Average waiting time %f \n",avgwt/n);
    printf("Average turn-around time %f \n",avgtat/n);
}
```

Output:



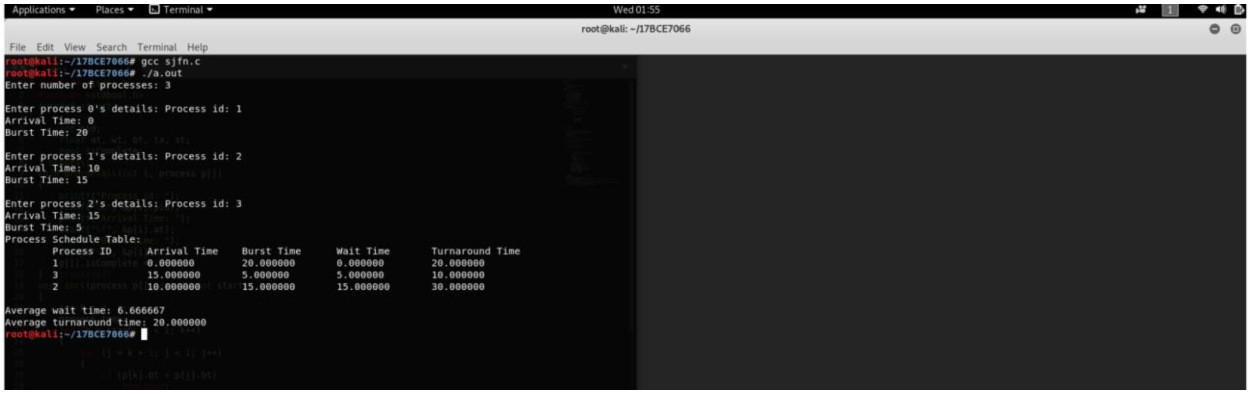
```
root@kali:~/17BCE7066# g++ fcfs.c
root@kali:~/17BCE7066# ./a.out
Enter the number of processes
3
Enter the process details pid,burst,arrival 1
1 20 0
Enter the process details pid,burst,arrival 2
2 10 5
Enter the process details pid,burst,arrival 3
3 15 25
The process details pid,burst,arrival 1
1 20 0
The process details pid,burst,arrival 2
2 10 5
The process details pid,burst,arrival 3
3 15 25
Process in order of their arrival time are :
PID     Burst Time      Arrival Time
1         20              0
2         10              5
3         15              25
After Scheduling in FCFS order details are :
PID     AT      BT      CT      TAT     WT
1       0       20      20      20      0
2       5       10      30      25      15
3       25      15      45      20      5
Average waiting time 6.666667
Average turn-around time 21.666666
root@kali:~/17BCE7066#
```

Shortest Job First (SJF)

```
#include <stdio.h>
#include <stdbool.h>
typedef struct
{
    int pid;
    float at, wt, bt, ta, st;
    bool isComplete;
} process;
void procdetail(int i, process p[])
{
    printf("Process id: ");
    scanf("%d", &p[i].pid);
    printf("Arrival Time: ");
    scanf("%f", &p[i].at);
    printf("Burst Time: ");
    scanf("%f", &p[i].bt);
    p[i].isComplete = false;
}
```

```
void sort(process p[], int i, int start)
{
    int k = 0, j;
    process temp;
    for (k = start; k < i; k++)
    {
        for (j = k + 1; j < i; j++)
        {
            if (p[k].bt < p[j].bt)
                continue;
            else
            {
                temp = p[k];
                p[k] = p[j];
                p[j] = temp;
            }
        }
    }
}
void main()
{
    int n, i, k = 0, j = 0;
    float avgwt = 0.0, avgta = 0.0, tst = 0.0;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    process p[n];
    for (i = 0; i < n; i++)
    {
        printf("\nEnter process %d's details: ", i);
        procdetail(i, p);
    }
    for (i = 0; i < n; i++)
    {
        if (p[i].isComplete == true)
            continue;
        else
        {
            k = i;
            while (p[i].at <= tst && i < n)
                i++;
            sort(p, i, k);
            i = k;
            if (p[i].at <= tst)
                p[i].st = tst;
            else
                p[i].st = p[i].at;
            p[i].st = tst;
            p[i].isComplete = true;
            tst += p[i].bt;
            p[i].wt = p[i].st - p[i].at;
            p[i].ta = p[i].bt + p[i].wt;
            avgwt += p[i].wt;
            avgta += p[i].ta;
        }
    }
    avgwt /= n;
    avgta /= n;
    printf("Process Schedule Table: \n");
    printf("\tProcess ID\tArrival Time\tBurst Time\tWait Time\tTurnaround Time\n");
    for (i = 0; i < n; i++)
        printf("\t%d\t\t%f\t%f\t%f\t%f\n", p[i].pid, p[i].at, p[i].bt, p[i].wt, p[i].ta);
    printf("\nAverage wait time: %f", avgwt);
    printf("\nAverage turnaround time: %f\n", avgta);
}
```

Output:



```
root@kali:~/17BCE7066# gcc sjfn.c
root@kali:~/17BCE7066# ./a.out
Enter number of processes: 3

Enter process 0's details: Process id: 1
Arrival Time: 0
Burst Time: 20

Enter process 1's details: Process id: 2
Arrival Time: 10
Burst Time: 15

Enter process 2's details: Process id: 3
Arrival Time: 15
Burst Time: 5
Process Schedule Table:
        Process ID      Arrival Time    Burst Time      Wait Time       Turnaround Time
        1               0.000000        20.000000       0.000000        20.000000
        3               15.000000       5.000000        5.000000        10.000000
        2               10.000000       15.000000       15.000000       30.000000

Average wait time: 6.666667
Average turnaround time: 20.000000
root@kali:~/17BCE7066# 
```

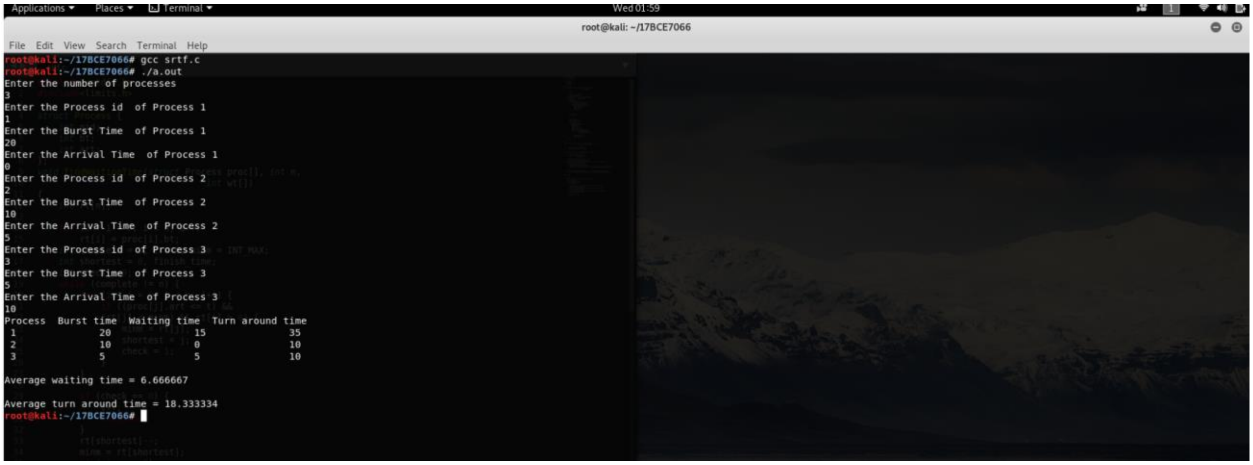
Shortest Remaining Time First (SRTF)

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

struct Process {
            int pid;
            int bt;
            int art;
};
void findWaitingTime(struct 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;
            int check = 0;
            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 = 1;
                     }
                }
                if (check == 0)
                {
                     t++;
                     continue;
                }
                rt[shortest]--;
                minm = rt[shortest];
                if (minm == 0)
                     minm = INT_MAX;
                if (rt[shortest] == 0) {
                     complete++;
                     check = 0;
                     finish_time = t + 1;
                     wt[shortest] = finish_time -
```

```
                                              proc[shortest].bt -
                                              proc[shortest].art;

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

void findavgTime(struct Process proc[], int n)
{
               int wt[n], tat[n], total_wt = 0,
                                        total_tat = 0;

               findWaitingTime(proc, n, wt);
               findTurnAroundTime(proc, n, 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\t\t%d\t\t%d\t\t%d\n",proc[i].pid,proc[i].bt,wt[i],tat[i]);

               }
  printf("\nAverage waiting time = %f \n",(float)total_wt/(float)n);
  printf("\nAverage turn around time = %f \n",(float)total_tat/(float)n);
}
int main()
{
               int n;
               printf("Enter the number of processes \n");
               scanf("%d",&n);
               struct Process proc[n];
               for(int i=0;i<n;i++)
               {
                    printf("Enter the Process id  of Process %d \n",(i+1));
```

```
                        scanf("%d",&proc[i].pid);
                        printf("Enter the Burst Time  of Process %d \n",(i+1));
                        scanf("%d",&proc[i].bt);
                        printf("Enter the Arrival Time  of Process %d \n",(i+1));
                        scanf("%d",&proc[i].art);
                }
                findavgTime(proc, n);
                return 0;
}
```

Output:



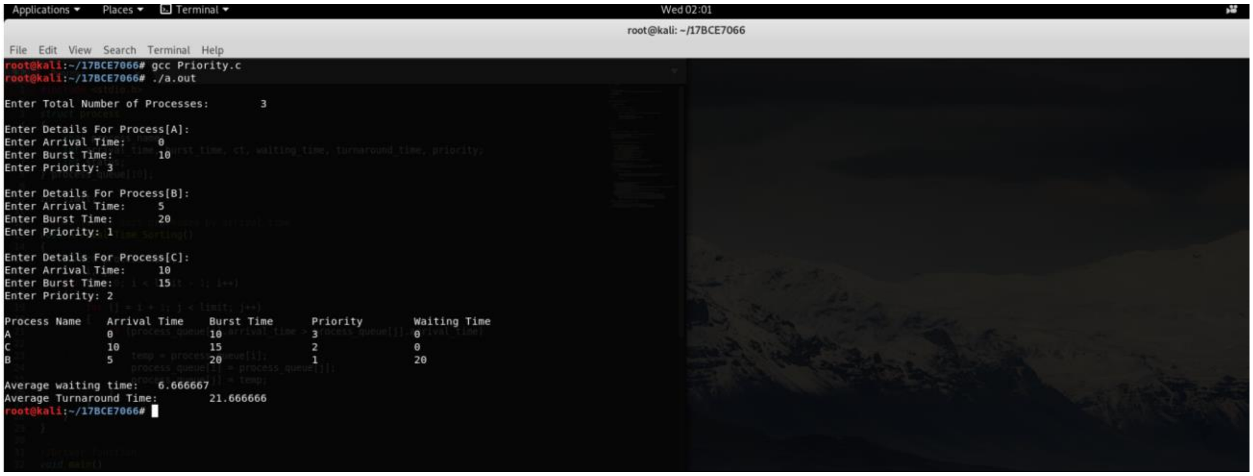
```
root@kali:~/17BCE7066# gcc srtf.c
root@kali:~/17BCE7066# ./a.out
Enter the number of processes
3
Enter the Process id  of Process 1
1
Enter the Burst Time  of Process 1
20
Enter the Arrival Time  of Process 1
0
Enter the Process id  of Process 2
2
Enter the Burst Time  of Process 2
10
Enter the Arrival Time  of Process 2
5
Enter the Process id  of Process 3
3
Enter the Burst Time  of Process 3
5
Enter the Arrival Time  of Process 3
10
Process  Burst time  Waiting time  Turn around time
 1              20              15              35
 2              10              0               10
 3              5               5               10

Average waiting time = 6.666667

Average turn around time = 18.333334
root@kali:~/17BCE7066# 
```

Priority Scheduling (Non-Premptive)

```
#include <stdio.h>
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;
         }
      }
   }
}
void main()
{
   int i, time = 0, burst_time = 0, largest;
   char c;
   float wait_time = 0, turnaround_time = 0, average_waiting_time;
   float 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);
}
```

Output:



```
root@kali:~/17BCE7066# gcc Priority.c
root@kali:~/17BCE7066# ./a.out

Enter Total Number of Processes:        3

Enter Details For Process[A]:
Enter Arrival Time:     0
Enter Burst Time:       10
Enter Priority: 3

Enter Details For Process[B]:
Enter Arrival Time:     5
Enter Burst Time:       20
Enter Priority: 1

Enter Details For Process[C]:
Enter Arrival Time:     10
Enter Burst Time:       15
Enter Priority: 2

Process Name    Arrival Time    Burst Time      Priority        Waiting Time
A               0               10              3               0
C               10              15              2               0
B               5               20              1               20

Average waiting time:   6.666667
Average Turnaround Time:        21.666666
root@kali:~/17BCE7066# 
```

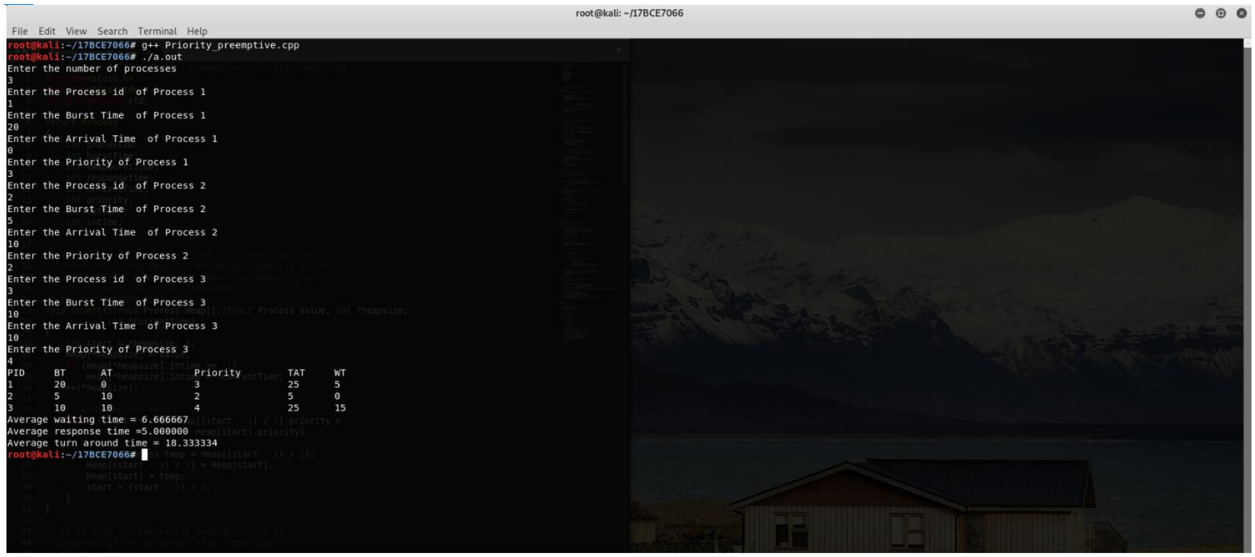
Priority Scheduling (Premptive)

```
#include<stdio.h>
#include<bits/stdc++.h>
using namespace std;
struct Process
{
   int processID;
   int burstTime;
   int tempburstTime;
   int responsetime;
   int arrivalTime;
   int priority;
   int outtime;
   int intime;
};
void insert(struct Process Heap[],struct Process value, int *heapsize,
         int *currentTime)
{
   int start = *heapsize, i;
   Heap[*heapsize] = value;
   if (Heap[*heapsize].intime == -1)
      Heap[*heapsize].intime = *currentTime;
   ++(*heapsize);
   while (start != 0 && Heap[(start - 1) / 2].priority >
                        Heap[start].priority)
   {
        struct Process temp = Heap[(start - 1) / 2];
        Heap[(start - 1) / 2] = Heap[start];
        Heap[start] = temp;
        start = (start - 1) / 2;
    }
}
void order(struct Process Heap[], int *heapsize, int start)
{
    int smallest = start;
    int left = 2 * start + 1;
    int right = 2 * start + 2;
    if (left < *heapsize && Heap[left].priority <
                               Heap[smallest].priority)
        smallest = left;
    if (right < *heapsize && Heap[right].priority <
                                Heap[smallest].priority)
        smallest = right;

    if (smallest != start)
    {
        struct Process temp = Heap[smallest];
        Heap[smallest] = Heap[start];
        Heap[start] = temp;
        order(Heap, heapsize, smallest);
    }
}

struct Process extractminimum(struct Process Heap[], int *heapsize,
                      int *currentTime)
{
    struct Process min = Heap[0];
    if (min.responsetime == -1)
        min.responsetime = *currentTime - min.arrivalTime;
    --(*heapsize);
    if (*heapsize >= 1)
    {
        Heap[0] = Heap[*heapsize];
        order(Heap, heapsize, 0);
    }
    return min;
}
```

```
bool compare(Process p1, Process p2)
{
   return (p1.arrivalTime < p2.arrivalTime);
}

void scheduling(struct Process Heap[],struct Process array[], int n,
            int *heapsize, int *currentTime)
{
   if (heapsize == 0)
      return;

   struct Process min = extractminimum(Heap, heapsize, currentTime);
   min.outtime = *currentTime + 1;
   --min.burstTime;
   if (min.burstTime > 0)
   {
      insert(Heap, min, heapsize, currentTime);
      return;
   }

   for (int i = 0; i < n; i++)
      if (array[i].processID == min.processID)
      {
         array[i] = min;
         break;
      }
}
void priority(struct Process array[], int n)
{
   sort(array,array+n,compare);

   int totalwaitingtime = 0, totalbursttime = 0,
      totalturnaroundtime = 0, i, insertedprocess = 0,
      heapsize = 0, currentTime = array[0].arrivalTime,
      totalresponsetime = 0;

   struct Process Heap[4 * n];
   for (int i = 0; i < n; i++)
   {
      totalbursttime += array[i].burstTime;
```

```
        array[i].tempburstTime = array[i].burstTime;
    }
    do
    {
        if (insertedprocess != n)
        {
            for (i = 0; i < n; i++)
            {
                if (array[i].arrivalTime == currentTime)
                {
                    ++insertedprocess;
                    array[i].intime = -1;
                    array[i].responsetime = -1;
                    insert(Heap, array[i], &heapsize, &currentTime);
                }
            }
        }
        scheduling(Heap, array, n, &heapsize, &currentTime);
        ++currentTime;
        if (heapsize == 0 && insertedprocess == n)
            break;
    } while (1);


    printf("PID\tBT\tAT\t\tPriority\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
    {
        totalresponsetime += array[i].responsetime;
        totalwaitingtime += (array[i].outtime - array[i].intime -
                             array[i].tempburstTime);
        totalbursttime += array[i].burstTime;

printf("%d\t%d\t%d\t\t%d\t\t%d\t%d\n",array[i].processID,array[i].tempburstTime,array[i].arrivalTime,array[i].priority,array[i].tempburstTime+(array[i].outtime - array[i].intime - array[i].tempburstTime),(array[i].outtime - array[i].intime -array[i].tempburstTime));
    }
    printf("Average waiting time = %f\n",
           ((float)totalwaitingtime / (float)n));
    printf("Average response time =%f\n",
           ((float)totalresponsetime / (float)n));
    printf("Average turn around time = %f\n",
        ((float)(totalwaitingtime + totalbursttime) / (float)n));
}
int main()
{
    int n, i;
    printf("Enter the number of processes \n");
    scanf("%d",&n);
    struct Process proc[n];
    for(int i=0;i<n;i++)
    {
        printf("Enter the Process id  of Process %d \n",(i+1));
        scanf("%d",&proc[i].processID);
        printf("Enter the Burst Time  of Process %d \n",(i+1));
        scanf("%d",&proc[i].burstTime);
        printf("Enter the Arrival Time  of Process %d \n",(i+1));
        scanf("%d",&proc[i].arrivalTime);
        printf("Enter the Priority of Process %d \n",(i+1));
        scanf("%d",&proc[i].priority);
    }
    priority(proc, n);
    return 0;
}
```

Output:

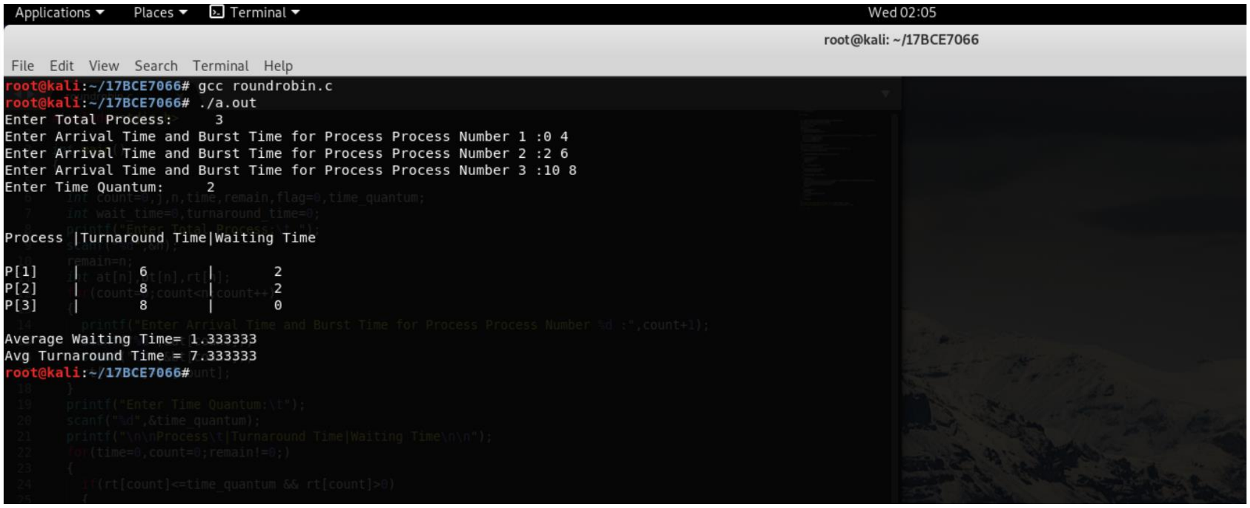


```
root@kali:~/17BCE7066# g++ Priority_preemptive.cpp
root@kali:~/17BCE7066# ./a.out
Enter the number of processes
3
Enter the Process id  of Process 1
1
Enter the Burst Time  of Process 1
20
Enter the Arrival Time  of Process 1
0
Enter the Priority of Process 1
3
Enter the Process id  of Process 2
2
Enter the Burst Time  of Process 2
5
Enter the Arrival Time  of Process 2
10
Enter the Priority of Process 2
2
Enter the Process id  of Process 3
3
Enter the Burst Time  of Process 3
10
Enter the Arrival Time  of Process 3
10
Enter the Priority of Process 3
4
PID     BT      AT              Priority        TAT     WT
1       20      0               3               25      5
2       5       10              2               5       0
3       10      10              4               25      15
Average waiting time = 6.666667
Average response time =5.000000
Average turn around time = 18.333334
root@kali:~/17BCE7066# 
```

Round Robin Scheduling (RR)

```
#include<stdio.h>
int main()
{

  int count=0,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0;
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  int at[n],bt[n],rt[n];
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcess\t|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("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f \n",turnaround_time*1.0/n);

  return 0;
}
```

Output:



```
root@kali:~/17BCE7066# gcc roundrobin.c
root@kali:~/17BCE7066# ./a.out
Enter Total Process:     3
Enter Arrival Time and Burst Time for Process Process Number 1 :0 4
Enter Arrival Time and Burst Time for Process Process Number 2 :2 6
Enter Arrival Time and Burst Time for Process Process Number 3 :10 8
Enter Time Quantum:     2


Process |Turnaround Time|Waiting Time

P[1]    |       6       |       2
P[2]    |       8       |       2
P[3]    |       8       |       0

Average Waiting Time= 1.333333
Avg Turnaround Time = 7.333333
root@kali:~/17BCE7066#
```