**Name:**Vishal Shashikant Salvi.
**UID:** 2019230069
**Batch:** C
**Class:** SE Comps

## Experiment No. 2

**Aim:** Process scheduling Policy.

## Theory:

### Definition

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

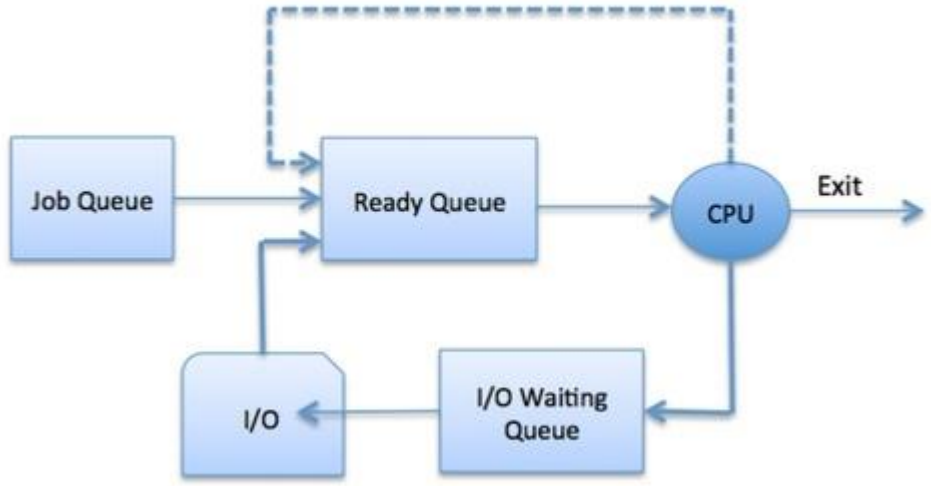
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

### Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.




The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

**Schedulers**

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

**Long Term Scheduler**

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

**Short Term Scheduler**

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of

the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

### Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

### Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

Program to implement First Come First Serve Scheduling Algorithm. Calculate average waiting time, average

turnaround time. (Given List of processes, their CPU burst times and Arrival times).

| Process | Burst Time | Arrival |
|---|---|---|
| 1 | 12 | 0 |
| 2 | 6 | 1 |
| 3 | 9 | 4 |

**Theory**:

## First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-pre-emptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

- First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.
- A perfect real life example of FCFS scheduling is **buying tickets at ticket counter**.

By far the simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the

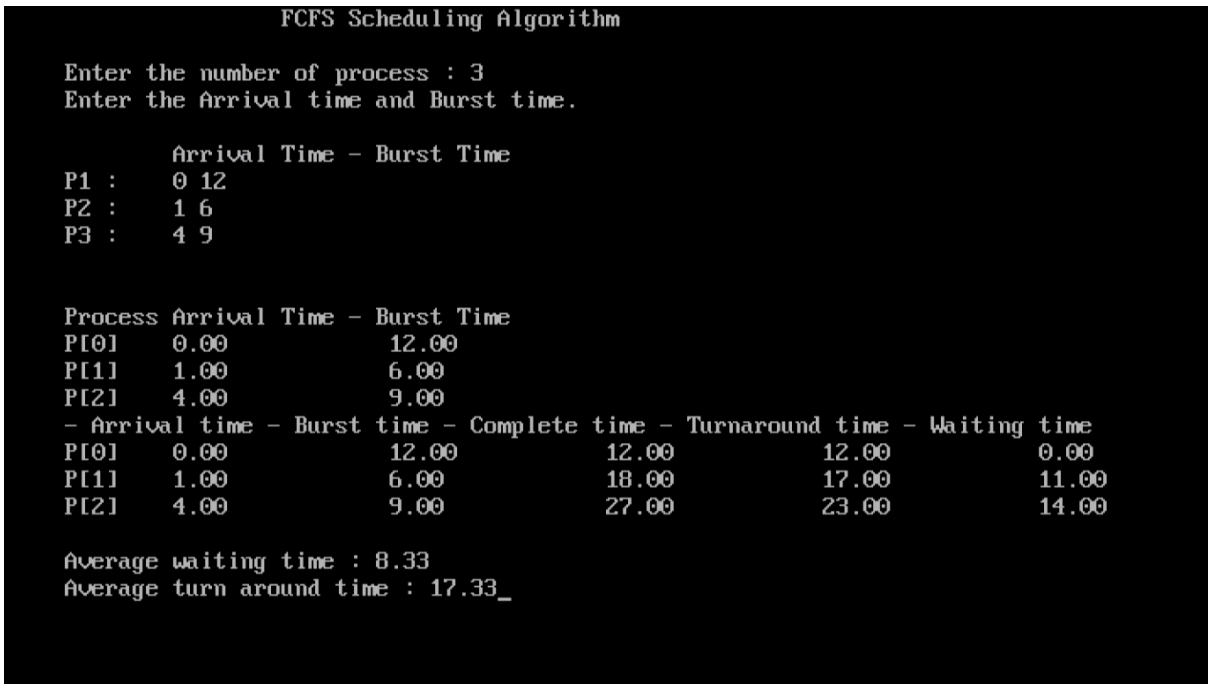
tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

**Code :**

```
#include<stdio.h>
#include<conio.h>
void main()
{
   float process[500],aTime[500],bTime[500],abTime[500],wTime[500],tat_time[500];
   int n = 0,i = 0 ;
   float aw_time = 0, atat_time = 0;
   clrscr();
   printf("\t\tFCFS Scheduling Algorithm\n");
   printf("\nEnter the number of process : ");
   scanf("%d",&n);
   printf("Enter the Arrival time and Burst time.\n\n");
   printf("\tArrival Time - Burst Time\n");
   for(i = 0 ; i < n ; i++){
         process[i]=i+1;
         printf("P%d :\t", i+1);
         scanf("%f\t%f",&aTime[i],&bTime[i]);
   }
   printf("\n\nProcess\tArrival Time - Burst Time\n");
   for(i = 0 ; i < n ; i++){
         printf("P[%d]\t%.2f\t\t%.2f\n",i,aTime[i],bTime[i]);
   }
   wTime[0] = 0;
   tat_time[0] = bTime[0];
   abTime[0] = bTime[0]+aTime[0];
   for( i = 1 ; i < n ; i++){
         abTime[i] = abTime[i-1] + bTime[i];
         tat_time[i] = abTime[i] - aTime[i];
         wTime[i] = tat_time[i] - bTime[i];
   }
   for(i = 0 ; i < n ; i++){
         aw_time = aw_time + wTime[i];
         atat_time = atat_time + tat_time[i];
```

```
    }
    printf("- Arrival time - Burst time - Complete time - Turnaround time - Waiting time\n");
    for(i = 0 ; i < n ; i++){
        printf("P[%d]\t%.2f\t\t%0.2f\t\t%0.2f\t\t%0.2f\t\t%0.2f\n",i,aTime[i],bTime[i],abTim
e[i],tat_time[i],wTime[i]);
    }
    printf("\nAverage waiting time : %0.2f",aw_time/n);
    printf("\nAverage turn around time : %0.2f",atat_time/n);
    getch();
}
```

**Output:**



Program to implement Shortest Job First Scheduling Algorithm. Calculate average waiting time, average

turnaround time. (Given List of processes, their CPU burst times and Arrival times).

| Processes | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

**Theory:**

## Shortest Job First (SJF)

Shortest job first is a scheduling algorithm in which the process with the smallest execution time is selected for execution next. Shortest job first can be either pre-emptive or non-preemptive. Owing to its simple nature, shortest job first is considered optimal. It also reduces the average waiting time for other processes awaiting execution.
Shortest job first is also known as shortest job next (SJN) and shortest process next (SPN).

A different approach to CPU scheduling is the shortest-job-first (SJF) scheduling algorithm. This algorithm associates with each process the length of the process's next CPU burst. When the CPU is available, it is assigned to the process 190 Chapter 5 that has the smallest next CPU burst. If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie. Note that a more appropriate term for this scheduling method would be the shortest-next-CPU-burst algorithm, because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will t

Shortest job first depends on the average running time of the processes. The accurate estimates of these measures help in the implementation of the shortest job first in an environment, which otherwise makes the same nearly impossible to implement. This is because often the execution burst of processes does not happen beforehand. It can be used in interactive environments where past patterns are available to determine the average time between the waiting time and the commands.

Although it is disadvantageous to use the shortest-job-first concept in short-term CPU scheduling, it is considered highly advantageous in long-term CPU scheduling. Moreover, the throughput is high in the case of shortest job first.

Shortest job first also has its share of disadvantages. For one, it can cause process starvation for longer jobs if there are a large number of shorter processes. Another is the need to know the execution time for each process beforehand. Often, this is almost impossible in many environments.

**Code :**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,btime=0;
int bt[10],temp,j,at[10],wt[10],tt[10],ta=0,sum=0;
float wavg=0,tavg=0,tsum=0,wsum=0;
clrscr();
printf(" -------Shortest Job First Scheduling -------\n");
printf("\nEnter the No. of processes :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
        printf("\tEnter the burst time of %d process :",i+1);
        scanf(" %d",&bt[i]);
        printf("\tEnter the arrival time of %d process :",i+1);
        scanf(" %d",&at[i]);
}
for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        {
                if(at[i]<at[j])
                {
                temp=p[j];
                p[j]=p[i];
                p[i]=temp;
                temp=at[j];
                at[j]=at[i];
```

```
                at[i]=temp;
                temp=bt[j];
                bt[j]=bt[i];
                bt[i]=temp;
                }
        }
}
for(j=0;j<n;j++)
        {
                btime=btime+bt[j];
                min=bt[k];
        for(i=k;i<n;i++)
        {
                if (btime>=at[i] && bt[i]<min)
                {
                        temp=p[k];
                        p[k]=p[i];
                        p[i]=temp;
                        temp=at[k];
                        at[k]=at[i];
                        at[i]=temp;
                        temp=bt[k];
                        bt[k]=bt[i];
                        bt[i]=temp;
                }
        }
        k++;
}
        wt[0]=0;
        for(i=1;i<n;i++)
        {
        sum=sum+bt[i-1];
        wt[i]=sum-at[i];
        wsum=wsum+wt[i];
        }
        wavg=(wsum/n);
        for(i=0;i<n;i++)
        {
        ta=ta+bt[i];
```

```
        tt[i]=ta-at[i];
        tsum=tsum+tt[i];
        }
tavg=(tsum/n);
printf("************************");
printf("\n RESULT:-");
printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around" );
for(i=0;i<n;i++)
        {
                printf("\n p%d\t %d\t %d\t\t %d\t\t\t%d",p[i],bt[i],at[i],wt[i],tt[i]);
        }
printf("\n\nAVERAGE WAITING TIME : %f",wavg);
printf("\nAVERAGE TURN AROUND TIME : %f",tavg);
getch();
}
```

**Output :**

```
    -------Shortest Job First Scheduling -------

   Enter the No. of processes :4
           Enter the burst time of 1 process :7
           Enter the arrival time of 1 process :0
           Enter the burst time of 2 process :4
           Enter the arrival time of 2 process :2
           Enter the burst time of 3 process :1
           Enter the arrival time of 3 process :4
           Enter the burst time of 4 process :4
           Enter the arrival time of 4 process :5
   ************************
    RESULT:-
   Process  Burst   Arrival         Waiting         Turn-around
    p1      7       0               0                       7
    p3      1       4               3                       4
    p2      4       2               6                       10
    p4      4       5               7                       11

   AVERAGE WAITING TIME : 4.000000
   AVERAGE TURN AROUND TIME : 8.000000
```

Program to implement Shortest Remaining Time First Scheduling Algorithm. Calculate average

waiting time, average turnaround time. (Given List of processes, their CPU burst times and Arrival times).

| Processes | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

**Theory:**

This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

### Shortest Remaining Time

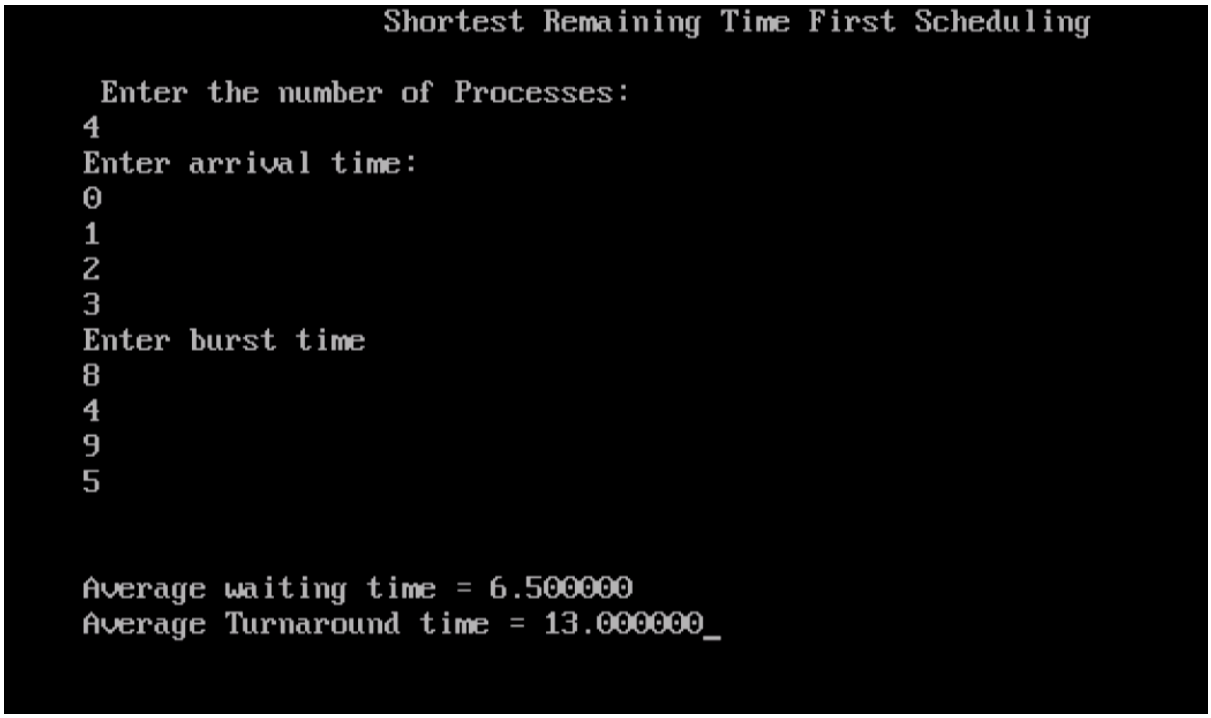
- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

**Code :**

```
#include <stdio.h>
#include<conio.h>
void main()
{
 int a[10],b[10],x[10],i,j,smallest,count=0,time,n;
 double avg=0,tt=0,end;
 clrscr();
  printf("\t\t Shortest Remaining Time First Scheduling\n");
  printf("\n Enter the number of Processes:\n");
```

```
 scanf("%d",&n);
printf("Enter arrival time:\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Enter burst time\n");
for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
x[i]=b[i];
 b[9]=9999;
for(time=0;count!=n;time++)
        {
                smallest=9;
        for(i=0;i<n;i++)
         {
                 if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
                smallest=i;
        }
 b[smallest]--;
 if(b[smallest]==0)
         {
                count++;
                 end=time+1;
                  avg=avg+end-a[smallest]-x[smallest];
                 tt= tt+end-a[smallest];
 }
}
printf("\n\nAverage waiting time = %lf\n",avg/n);
   printf("Average Turnaround time = %lf",tt/n);
  getch();
}
```

**Output :**


Shortest Remaining Time First Scheduling
Enter the number of Processes:
4
Enter arrival time:
0
1
2
3
Enter burst time
8
4
9
5
Average waiting time = 6.500000
Average Turnaround time = 13.000000

Program to implement Round Robin Scheduling Algorithm. Calculate average waiting time, average turnaround time. (Given List of processes, their CPU burst times and Arrival times).

| Processes | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 5 |
| P2 | 1 | 3 |
| P3 | 2 | 8 |
| P4 | 3 | 6 |

**Theory:**

The round-robin (RR) scheduling algorithm is designed especially for timesharing systems. It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes. A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally fronc 10 to 100 milliseconds in length. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. To implement RR scheduling, we keep the ready queue as a FIFO queue o£ processes.

New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process. One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail o£ the ready queue. The CPU scheduler will then select the next process in the ready queue.

## Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

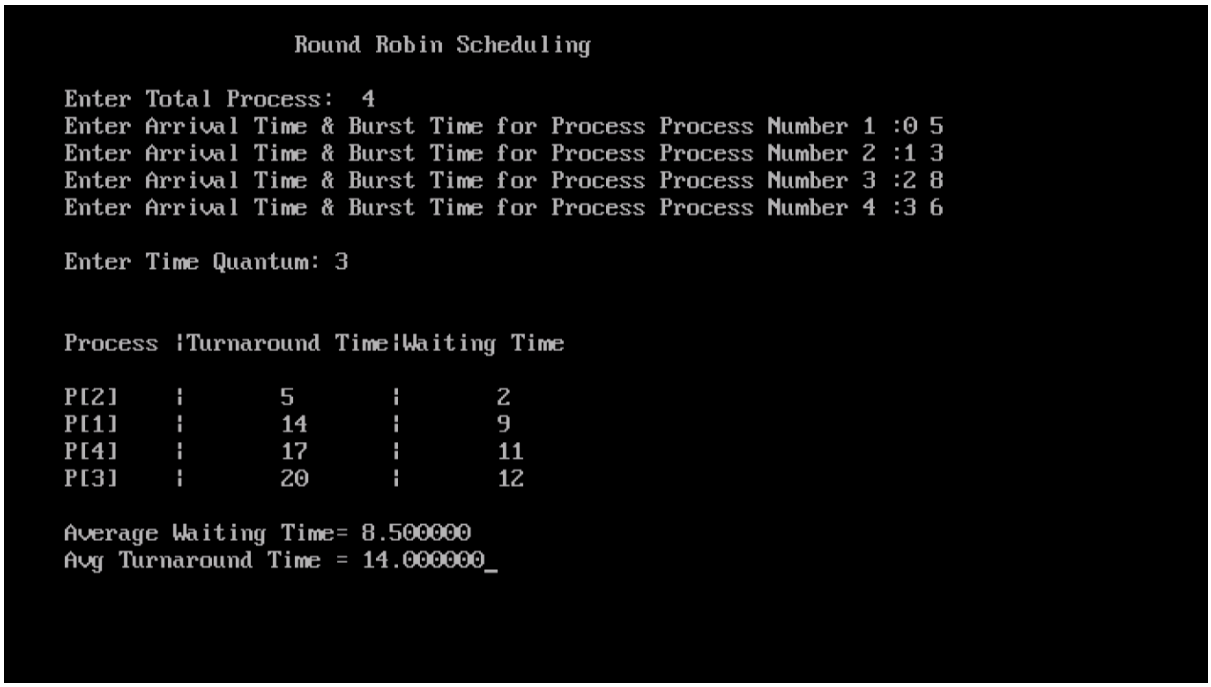
**Code :**

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
```

```
clrscr();
printf("\n\t\t Round Robin Scheduling");
printf("\n\nEnter Total Process:  ");
scanf("%d",&n);
remain=n;
for(count=0;count<n;count++)
{
  printf("Enter Arrival Time & Burst Time for Process Process Number %d :",count+1);
  scanf("%d",&at[count]);
  scanf("%d",&bt[count]);
  rt[count]=bt[count];
}
printf("\nEnter Time Quantum: ");
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",turnaround_time*1.0/n);
 getch();
}
```

**Output :**



Program to implement Priority Scheduling Algorithm. Calculate average waiting time, average

turnaround time. (Given List of processes, their CPU burst times and Arrival times=0 for all processes).

| Processes | Burst Time | Priority |
|---|---|---|
| P1 | 21 | 2 |
| P2 | 3 | 1 |
| P3 | 6 | 4 |
| P4 | 2 | 3 |

**Theory:**

The SJF algorithm is a special case of the general priority scheduling algorithm. A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order. An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa. Note that we discuss scheduling in terms of high priority and low priority. Priorities are generally indicated by some fixed range of numbers, such as 0 to 7 or 0 to 4,095. However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion. In this text, we assume that low numbers represent high priority.

## Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

**Code :**

```
#include<stdio.h>
#include<conio.h>
void main(){
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   clrscr();
   printf("\n\t\t Priority Scheduling Algorithm");
   printf("\nEnter Total Number of Process:");
   scanf("%d",&n);
   printf("\nEnter Burst Time and Priority:");
```

```
for(i=0;i<n;i++)
{
        printf("\nP[%d] ",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("P[%d] ",i+1);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;
}
for(i=0;i<n;i++)
{        pos=i;
        for(j=i+1;j<n;j++)
        {  if(pr[j]<pr[pos])
                pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
}    wt[0]=0;
for(i=1;i<n;i++)
{        wt[i]=0;
        for(j=0;j<i;j++)
           wt[i]+=bt[j];
        total+=wt[i];
}
avg_wt=total/n;
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
```

```
    }
    avg_tat=total/n;
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);
    getch();
}
```

**Output :**

```
                    Priority Scheduling Algorithm
    Enter Total Number of Process:4

    Enter Burst Time and Priority:
    P[1] Burst Time:21
    P[1] Priority:2

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

    P[3] Burst Time:6
    P[3] Priority:4

    P[4] Burst Time:2
    P[4] Priority:3

    Process     Burst Time          Waiting Time    Turnaround Time
    P[2]              3                 0                 3
    P[1]              21                3                 24
    P[4]              2                 24                26
    P[3]              6                 26                32

    Average Waiting Time=13
    Average Turnaround Time=21
```

**Conclusion:**

Thus, From this Experiment we learn more about FCFS, Shortest Path First, Shortest Remaining Time First, Round Robin, Priority Scheduling Algorithm and also implement it.