Experiment No: 03

Experiment Title: Priority Scheduling.

Theory: Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority, Priority scheduling involves priority assignment to every process, and processes with higher priorities are carried out first, whereas tasks with equal priorities are carried out on a first-come-first-served (FCFS) basic.

Code:

```
// Priority In Operating System
#include<stdio.h>
int main ()
{
    int n, i, j, temp, temp2, temp3;
    int c [10], w [10], t [10], priority [10], process [10];
    printf("Enter the number of process: ");
    scanf("%d",&n);

    for (i =0; i<n; i++)
    {
        printf("\nEnter the cpu time of  P%d: ",i+1);
        scanf("%d", &c[i]);
        process[i] = i+1;
        printf("Enter the Priority Value of P%d:  ", i+1);
        scanf("%d", &priority[i]);
    }
    for (i=0; i<n-1; i++)
    {
```

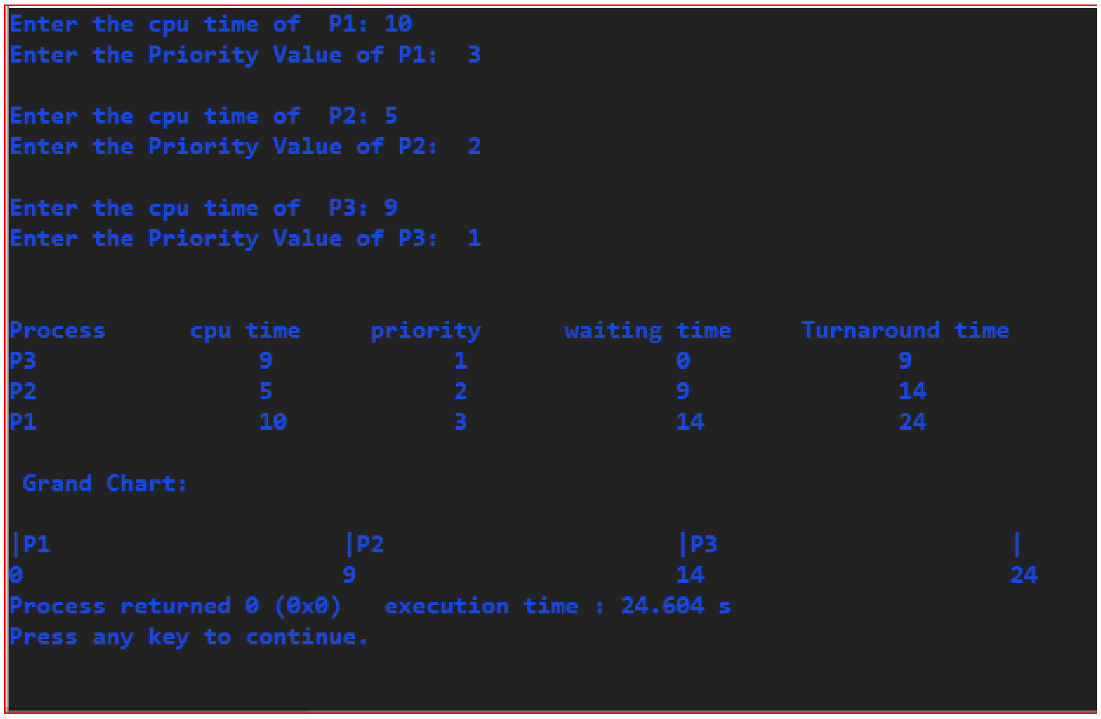
```
    for (j=i+1; j<n; j++)
    {
        if(priority[j]< priority[i])
        {
            temp = priority[j];
            priority[j] = priority[i];
            priority[i] = temp;

            temp2 = c[j];
            c[j] = c[i];
            c[i] = temp2;

            temp3 = process[j];
            process[j] = process[i];
            process[i] = temp3;
        }
    }
}
w [0] =0;
for (i = 1; i<n; i++)
{
    w[i] = w[i-1] +c[i-1];
}
for (i =0; i<n; i++)
{
    t[i] = w[i] + c[i];
}
printf("\n\nProcess\t     cpu time\t  priority\t waiting time\t Turnaround time\n");
```

```
for (i=0; i<n; i++)
   {
      printf("P%d\t         %d\t\t%d\t\t%d\t\t%d\n", process[i],  c[i],  priority[i], w[i],  t[i]);
   }
   printf("\n Grand Chart: \n\n|");
   for (i=0; i<n; i++)
   {
      printf("P%d\t\t\t|",  i+1);
   }
   printf("\n0");
   for (i =0; i<n; i++)
   {
      printf(" \t\t\t%d", t[i]);
   }
}
```

Input and Output:



```
Enter the cpu time of  P1: 10
Enter the Priority Value of P1:  3

Enter the cpu time of  P2: 5
Enter the Priority Value of P2:  2

Enter the cpu time of  P3: 9
Enter the Priority Value of P3:  1


Process      cpu time     priority      waiting time     Turnaround time
P3                9            1               0                9
P2                5            2               9                14
P1                10           3               14               24

 Grand Chart:

|P1                     |P2                     |P3                     |
0                       9                       14                       24
Process returned 0 (0x0)   execution time : 24.604 s
Press any key to continue.
```

Experiment Title: Round Robin Scheduling.

Theory: Round robin scheduling (RRS) is a job-scheduling algorithm that is considered to be very fair, as it uses time slices that are assigned to each process in the queue or line. Each process is then allowed to use the CPU for a given amount of time, and if it does not finish within the allotted time, it is preempted and then moved at the back of the line so that the next process in line is able to use the CPU for the same amount of time.

Code:

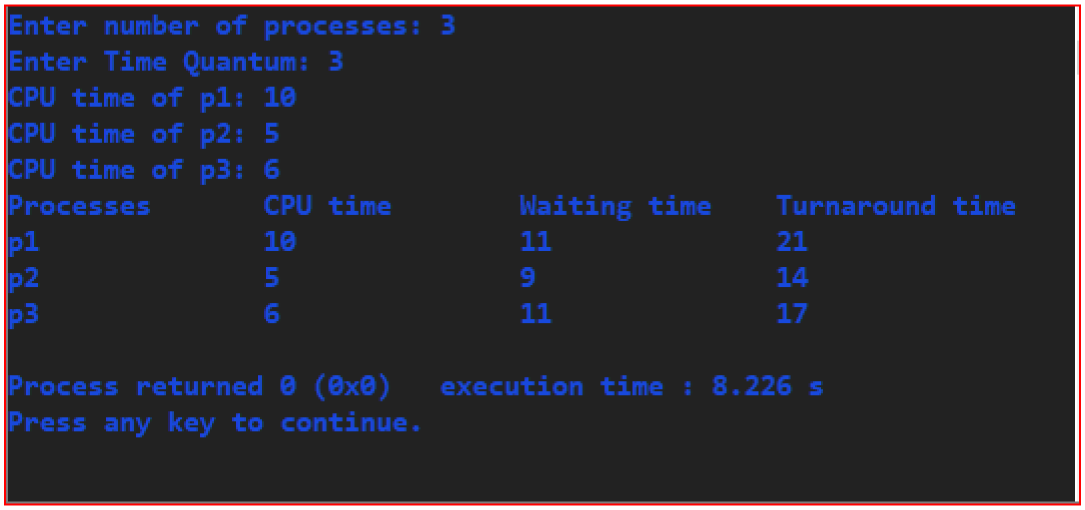
```
// Round Robin Experiment
#include<stdio.h>
int main () {
int cpu_time[10], rem_cpu_time[10], waiting_time[10], turnaround_time[10], time_quantum,
process[10], n, i,j,temp, flag, counter = 0, timer;


float avg_w = 0, avg_t=0;


   printf("Enter number of processes: ");
   scanf("%d", &n);
   printf("Enter Time Quantum: ");
   scanf("%d", &time_quantum);


   for(i=0;i<n;i++){
      printf("CPU time of p%d: ", i+1);
      scanf("%d", &cpu_time[i]);
      rem_cpu_time[i] = cpu_time[i];
   }
   for (timer=0, i=0; counter! =n;) {
           flag = 0;
           if(rem_cpu_time[i] <= time_quantum && rem_cpu_time[i] > 0) {
               timer += rem_cpu_time[i];
               rem_cpu_time[i] = 0;
               flag = 1;
           } else if(rem_cpu_time[i] > 0) {
                 timer += time_quantum;
                 rem_cpu_time[i] = rem_cpu_time[i] - time_quantum;
           }
           if(rem_cpu_time[i] == 0 && flag == 1) {
               counter++;
               turnaround_time[i] = timer;
           }
           i++;
           i = i%n;
   }
   for (i=0; i<n;i++){
      waiting_time[i] = turnaround_time[i] - cpu_time[i];
   }
   printf("Processes\tCPU time\tWaiting time\tTurnaround time\n");
   for (i=0; i<n; i++) {
      printf("p%d\t\t%d\t\t%d\t\t%d\n", i+1, cpu_time[i], waiting_time[i], turnaround_time[i]);
   }
}
```

Input and Output:



```
Enter number of processes: 3
Enter Time Quantum: 3
CPU time of p1: 10
CPU time of p2: 5
CPU time of p3: 6
Processes       CPU time        Waiting time    Turnaround time
p1              10              11              21
p2              5               9               14
p3              6               11              17

Process returned 0 (0x0)   execution time : 8.226 s
Press any key to continue.
```