OS LAB 2

AIM: Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.
1. Priority (pre-emptive & Non-pre-emptive)
2. Round Robin (Experiment with different quantum sizes for RR algorithm)

SOURCE CODE

```c
#include<stdio.h>
void findWaitingTime(int processes[], int n, int burstTime[], int priority[], int waitingTime[])
{
    int remainingTime[n];
    for (int i = 0; i < n; i++)
        remainingTime[i] = burstTime[i];


    int completed = 0;
    int currentTime = 0;


    while (completed != n)
    {
        int highestPriorityIndex = -1;
        int highestPriority = -1;


        for (int i = 0; i < n; i++)
        {
            if (remainingTime[i] > 0 && priority[i] > highestPriority)
            {
                highestPriority = priority[i];
                highestPriorityIndex = i;
            }
        }
```

```
    if (highestPriorityIndex == -1)

    {

        currentTime++;

        continue;

    }


    remainingTime[highestPriorityIndex]--;


    if (remainingTime[highestPriorityIndex] == 0)

    {

        completed++;


        waitingTime[highestPriorityIndex] = currentTime + 1 - burstTime[highestPriorityIndex];


        if (waitingTime[highestPriorityIndex] < 0)

            waitingTime[highestPriorityIndex] = 0;

    }


    currentTime++;

  }

}


void findTurnaroundTime(int processes[], int n, int burstTime[], int waitingTime[], int
turnaroundTime[])

{

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

    turnaroundTime[i] = burstTime[i] + waitingTime[i];

}


void findAverageTime(int processes[], int n, int burstTime[], int priority[])

{
```

```c
    int waitingTime[n], turnaroundTime[n], totalWaitingTime = 0, totalTurnaroundTime = 0;


    findWaitingTime(processes, n, burstTime, priority, waitingTime);


    findTurnaroundTime(processes, n, burstTime, waitingTime, turnaroundTime);


    printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");


    for (int i = 0; i < n; i++)
    {
        totalWaitingTime += waitingTime[i];

        totalTurnaroundTime += turnaroundTime[i];

        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], burstTime[i], priority[i], waitingTime[i],
turnaroundTime[i]);

    }


    printf("\nAverage Waiting Time: %.2f", (float)totalWaitingTime / n);

    printf("\nAverage Turnaround Time: %.2f", (float)totalTurnaroundTime / n);
}

void priorityNonPreemptiveScheduling(int processes[], int n, int burstTime[], int priority[])
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (priority[j] > priority[j + 1])
            {
                int temp = priority[j];

                priority[j] = priority[j + 1];

                priority[j + 1] = temp;
```

```c
            temp = burstTime[j];

            burstTime[j] = burstTime[j + 1];

            burstTime[j + 1] = temp;


            temp = processes[j];

            processes[j] = processes[j + 1];

            processes[j + 1] = temp;

        }

     }

   }


   printf("\nPriority Non-Preemptive Scheduling:\n");

   findAverageTime(processes, n, burstTime, priority);

}


void priorityPreemptiveScheduling(int processes[], int n, int burstTime[], int priority[])

{

   int remainingTime[n];

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

      remainingTime[i] = burstTime[i];


   int completed = 0;

   int currentTime = 0;


   while (completed != n)

   {

      int highestPriorityIndex = -1;

      int highestPriority = -1;


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

```c
    {
        if (remainingTime[i] > 0 && priority[i] > highestPriority)

        {
            highestPriority = priority[i];

            highestPriorityIndex = i;

        }
    }


    if (highestPriorityIndex == -1)

    {
        currentTime++;

        continue;

    }


    remainingTime[highestPriorityIndex]--;


    if (remainingTime[highestPriorityIndex] == 0)

    {
        completed++;


        int finishTime = currentTime + 1;


        int waitingTime = finishTime - burstTime[highestPriorityIndex];


        if (waitingTime < 0)

            waitingTime = 0;


        printf("Process %d:\n", processes[highestPriorityIndex]);

        printf("Waiting Time: %d\n", waitingTime);

        printf("Turnaround Time: %d\n", finishTime);

    }
```

```c
        currentTime++;

    }

}


void roundRobinScheduling(int processes[], int n, int burstTime[], int quantum)

{

    int remainingTime[n];

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

        remainingTime[i] = burstTime[i];


    int completed = 0;

    int currentTime = 0;


    while (completed != n)

    {

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

        {

            if (remainingTime[i] > 0)

            {

                if (remainingTime[i] <= quantum)

                {

                    currentTime += remainingTime[i];

                    remainingTime[i] = 0;

                    completed++;


                    printf("Process %d:\n", processes[i]);

                    printf("Waiting Time: %d\n", currentTime - burstTime[i]);

                    printf("Turnaround Time: %d\n", currentTime);

                }

                else
```

```c
        {
            currentTime += quantum;

            remainingTime[i] -= quantum;

        }

      }

    }

  }

}


int main()

{

    int n, choice;

    printf("Enter the number of processes: ");

    scanf("%d", &n);


    int processes[n], burstTime[n], priority[n];


    printf("Choose the scheduling algorithm:\n");

    printf("1. Priority Non-Preemptive\n");

    printf("2. Priority Preemptive\n");

    printf("3. Round Robin\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);


    switch (choice)

    {

      case 1:

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

        {

          printf("Enter burst time for process %d: ", i + 1);

          scanf("%d", &burstTime[i]);
```

```c
        printf("Enter priority for process %d: ", i + 1);

        scanf("%d", &priority[i]);

        processes[i] = i + 1;

    }

    priorityNonPreemptiveScheduling(processes, n, burstTime, priority);

    break;


case 2:

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

    {

        printf("Enter burst time for process %d: ", i + 1);

        scanf("%d", &burstTime[i]);

        printf("Enter priority for process %d: ", i + 1);

        scanf("%d", &priority[i]);

        processes[i] = i + 1;

    }

    priorityPreemptiveScheduling(processes, n, burstTime, priority);

    break;


case 3:

    {

        int quantum;

        printf("Enter the time quantum for Round Robin: ");

        scanf("%d", &quantum);

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

        {

            printf("Enter burst time for process %d: ", i + 1);

            scanf("%d", &burstTime[i]);

            processes[i] = i + 1;

        }

        roundRobinScheduling(processes, n, burstTime, quantum);
```

```c
            }
            break;


        default:
            printf("Invalid choice. Exiting...\n");
            return 0;
    }


    return 0;
}
```

OUTPUT SCREENSHOTS


```
Enter the number of processes: 5
Choose the scheduling algorithm:
1. Priority Non-Preemptive
2. Priority Preemptive
3. Round Robin
Enter your choice: 1
Enter burst time for process 1: 1
Enter priority for process 1: 2
Enter burst time for process 2: 3
Enter priority for process 2: 1
Enter burst time for process 3: 2
Enter priority for process 3: 14
Enter burst time for process 4: 2
Enter priority for process 4: 4
Enter burst time for process 5: 1
Enter priority for process 5: 3

Priority Non-Preemptive Scheduling:

Process Burst Time      Priority       Waiting Time    Turnaround Time
2       3               1              6               9
1       1               2              5               6
5       1               3              4               5
4       2               4              2               4
3       2               14             0               2

Average Waiting Time: 3.40
Average Turnaround Time: 5.20
Process returned 0 (0x0)   execution time : 19.607 s
Press any key to continue.
```



```
Enter the number of processes: 3
Choose the scheduling algorithm:
1. Priority Non-Preemptive
2. Priority Preemptive
3. Round Robin
Enter your choice: 3
Enter the time quantum for Round Robin: 2
Enter burst time for process 1: 4
Enter burst time for process 2: 3
Enter burst time for process 3: 5
Process 1:
Waiting Time: 4
Turnaround Time: 8
Process 2:
Waiting Time: 6
Turnaround Time: 9
Process 3:
Waiting Time: 7
Turnaround Time: 12

Process returned 0 (0x0)   execution time : 23.237 s
Press any key to continue.
```