****

**LAB EXERCISE 4**

**Implementation of CPU Scheduling Policies**

Name: Jayannthan P T

Dept: CSE ‘A’

Roll No.: 205001049

Develop a menu driven C program to implement the CPU Scheduling Algorithms

1. Priority (Non-Preemptive and Preemptive)
2. Round Robin

**Code:**

*/\*Develop a menu driven C program to implement the CPU Scheduling Algorithms - Priority*

*(Non-Preemptive and Preemptive) and Round Robin*

*Algorithm: 1. Read the following a. Number of p b. Process IDs c. Arrival time for each process d. Burst Time for each process 2. Design a menu with FCFSand SJFoptions 3. Upon selection of menu option apply the corresponding algorithm. 4. Compute the Turnaround Time, Average waiting Time for each of the algorithm. 5. Tabularize the results. 6. Display the Gantt Chart.\*/*

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

typedef **struct** process

{

**char** pid[3];

**int** arrival, burst,teempburst, turnaround, waiting, completion, priority;

} process;

**void** print\_gantt\_chart(process p**[]**, **int** n)

{

    printf("\n\nGantt-Chart\n");

**int** i, j;

    printf(" ");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < p[i].burst; j++)

            printf("--");

        printf(" ");

    }

    printf("\n|");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < p[i].burst - 1; j++)

            printf(" ");

        printf("P%s", p[i].pid);

        for (j = 0; j < p[i].burst - 1; j++)

            printf(" ");

        printf("|");

    }

    printf("\n ");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < p[i].burst; j++)

            printf("--");

        printf(" ");

    }

    printf("\n");

    printf("0");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < p[i].burst; j++)

            printf("  ");

        if (p[i].turnaround > 9)

            printf("\b");

        printf("%d", p[i].turnaround);

    }

    printf("\n");

}

**int** main()

{

**int** no\_of\_process;

**int** totalwaitingtime = 0, totalturnaround = 0;

**int** pos;

**char** ch = 'y';

    process p[100];

    while (ch == 'y' || ch == 'Y')

    {

**int** choice;

        printf("\nMenu\n\t1.Priority-Non Preemptive\n\t2.Priority-Preemptive\n\t3.Round Robin\n\t4.Exit\n\t\nEnter Choice:");

        scanf(" %d", &choice);

        switch (choice)

        {

        case 1:

        {

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

**int** no\_of\_process;

            printf("\nNumber of p :");

            scanf(" %d", &no\_of\_process);

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                printf("\n\nProcess %d\n", i + 1);

                printf("Process ID: ");

                scanf(" %s", p[i].pid);

                printf("Burst Time :");

                scanf(" %d", &p[i].burst);

                printf("Priority :");

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

            }

            process temppro;

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                pos = i;

                for (**int** j = i + 1; j < no\_of\_process; j++)

                {

                    if (p[j].priority < p[pos].priority)

                        pos = j;

                }

                temppro = p[i];

                p[i] = p[pos];

                p[pos] = temppro;

            }

            totalwaitingtime = 0, totalturnaround = 0;

            p[0].waiting = 0;

            p[0].turnaround = p[0].burst;

            totalturnaround += p[0].turnaround;

            for (**int** i = 1; i < no\_of\_process; i++)

            {

                if (p[i - 1].waiting + p[i - 1].burst - p[i].arrival > 0)

                {

                    p[i].waiting = p[i - 1].waiting + p[i - 1].burst - p[i].arrival;

                }

                else

                {

                    p[i].waiting = 0;

                }

                totalwaitingtime += p[i].waiting;

                p[i].turnaround = p[i].burst + p[i].waiting;

                totalturnaround += p[i].turnaround;

            }

            printf("\nP\_ID\tBurst Time\tPriority\tWaiting Time\t\tTurnaround Time\n");

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                printf("%s\t\t%d\t\t%d\t\t%d\t\t\t%d\n", p[i].pid, p[i].burst, p[i].priority, p[i].waiting, p[i].turnaround);

            }

**float** avgwaiting = (**float**)(totalwaitingtime / no\_of\_process);

**float** avgturnaround = (**float**)(totalturnaround / no\_of\_process);

            printf("\n\t\tAVERAGE \tWaitingTime =%.2f\t TurnaroundTime =%.2f\n", totalwaitingtime / no\_of\_process, totalturnaround / no\_of\_process);

            print\_gantt\_chart(p, no\_of\_process);

            break;

        }

        case 2:

        {

            printf("\nPriority-Preemptive\n");

**int** no\_of\_process;

            printf("\nNumber of process :");

            scanf(" %d", &no\_of\_process);

**int** tempburst[100];

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                printf("\n\nProcess %d\n", i + 1);

                printf("Process ID: ");

                scanf(" %s", p[i].pid);

                printf("Arrival Time :");

                scanf(" %d", &p[i].arrival);

                printf("Burst Time :");

                scanf(" %d", &p[i].burst);

                printf("Priority :");

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

                tempburst[i] = p[i].burst;

                p[i].teempburst = p[i].burst;

            }

**int** rem\_time[no\_of\_process];

            for (**int** i = 0; i < no\_of\_process; i++)

                rem\_time[i] = p[i].burst;

            process tempro[100];

**int** tempcount = 0;

**int** completed = 0;

**int** cur\_time = 0;

            while (completed < no\_of\_process)

            {

**int** idx = -1;

                for (**int** i = 0; i < no\_of\_process; i++)

                {

                    if (p[i].arrival <= cur\_time && rem\_time[i] > 0 && (idx == -1 || p[i].priority < p[idx].priority))

                        idx = i;

                }

                cur\_time++;

                if (tempcount != 0 && strcmp(tempro[tempcount-1].pid,p[idx].pid)) tempcount--;

                else

                {

                    tempro[tempcount] = p[idx];

                }

                strcpy(tempro[tempcount].pid,p[idx].pid);

                tempro[tempcount].burst++;

                tempro[tempcount].turnaround = cur\_time;

                tempcount++;

                rem\_time[idx]--;

                if(rem\_time[idx]==0)

                {

                    completed++;

                    p[idx].completion = cur\_time;

                    p[idx].waiting = p[idx].completion - p[idx].arrival - p[idx].burst;

                    p[idx].turnaround = p[idx].burst + p[idx].waiting;

                }

            }

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                totalwaitingtime+=p[i].waiting;

                totalturnaround+=p[i].turnaround;

            }

            printf("\nP\_ID\tArrival Time\tBurst Time\tPriority\tWaiting Time\t\tTurnaround Time\n");

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                printf("%s\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].priority, p[i].waiting, p[i].turnaround);

            }

**float** avgwaiting = (**float**)(totalwaitingtime / no\_of\_process);

**float** avgturnaround = (**float**)(totalturnaround / no\_of\_process);

            printf("\n\t\tAVERAGE \tWaitingTime =%.2f\t TurnaroundTime =%.2f\n", avgwaiting, avgturnaround);

            process temmptemppro[100];

**int** temptempcount=-1;

            for(**int** i=0;i<tempcount;i++)

            {

                if(strcmp(tempro[i+1].pid,tempro[i].pid)!=0)

                {

                    temmptemppro[++temptempcount]=tempro[i];

                }

            }

            print\_gantt\_chart(temmptemppro, temptempcount+1);

            print\_gantt\_chart(tempro, tempcount);

            break;

        }

        case 3:

        {

            printf("\nRound Robin\n");

**int** no\_of\_process;

**int** quantum;

            printf("\nNumber of p :");

            scanf(" %d", &no\_of\_process);

**int** temp\_nop = no\_of\_process;

            process temppro[100];

**int** tempburst[100];

**int** count = 0;

**int** tempcount = -1;

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                printf("\n\nProcess %d\n", i + 1);

                printf("Process ID: ");

                scanf(" %s", p[i].pid);

                printf("Arrival Time :");

                scanf(" %d", &p[i].arrival);

                printf("Burst Time :");

                scanf(" %d", &p[i].burst);

                tempburst[i] = p[i].burst;

            }

            process tempppro;

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                pos = i;

                for (**int** j = i + 1; j < no\_of\_process; j++)

                {

                    if (p[j].arrival < p[pos].arrival)

                        pos = j;

                }

                tempppro = p[i];

                p[i] = p[pos];

                p[pos] = tempppro;

            }

            totalwaitingtime = 0, totalturnaround = 0;

            p[0].waiting = 0;

            printf("\nTime Quantum :");

            scanf(" %d", &quantum);

            for (**int** sum = 0, i = 0; temp\_nop != 0;)

            {

                if (tempburst[i] <= quantum && tempburst[i] > 0)

                {

**int** temptempburst = tempburst[i];

                    sum = sum + tempburst[i];

                    tempburst[i] = 0;

                    count = 1;

                    tempcount++;

                    strcpy(temppro[tempcount].pid, p[i].pid);

                    temppro[tempcount].burst = temptempburst;

                    temppro[tempcount].arrival = p[i].arrival;

                    temppro[tempcount].turnaround = sum;

                }

                else if (tempburst[i] > 0)

                {

                    tempburst[i] = tempburst[i] - quantum;

                    sum = sum + quantum;

                    tempcount++;

                    strcpy(temppro[tempcount].pid, p[i].pid);

                    temppro[tempcount].burst = quantum;

                    temppro[tempcount].arrival = p[i].arrival;

                    temppro[tempcount].turnaround = sum;

                    temppro[tempcount].waiting = sum - p[i].arrival - quantum;

                }

                if (tempburst[i] == 0 && count == 1)

                {

                    temp\_nop--;

                    p[i].turnaround = sum - p[i].arrival;

                    p[i].waiting = sum - p[i].arrival - p[i].burst;

                    totalwaitingtime = totalwaitingtime + sum - p[i].arrival - p[i].burst;

                    totalturnaround = totalturnaround + sum - p[i].arrival;

                    count = 0;

                }

                if (p[i + 1].arrival <= sum)

                {

                    i++;

                }

                else

                {

                    i = 0;

                }

            }

            printf("\nP\_ID\tArrival Time\tBurst Time\tWaiting Time\t\tTurnaround Time\n");

            for (**int** i = 0; i < no\_of\_process; i++)

            {

                printf("%s\t\t%d\t\t%d\t\t%d\t\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].waiting, p[i].turnaround);

            }

**float** avgwaiting = (**float**)(totalwaitingtime / no\_of\_process);

**float** avgturnaround = (**float**)(totalturnaround / no\_of\_process);

            printf("\n\t\tAVERAGE \tWaitingTime =%.2f\t TurnaroundTime =%.2f\n", avgwaiting, avgturnaround);

            print\_gantt\_chart(temppro, tempcount + 1);

            break;

        }

        case 4:

            printf("Exiting...");

            return 0;

        }

        printf("\nWant to Continue (Y/N):");

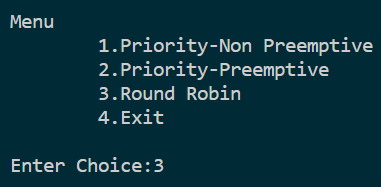
        scanf(" %c", &ch);

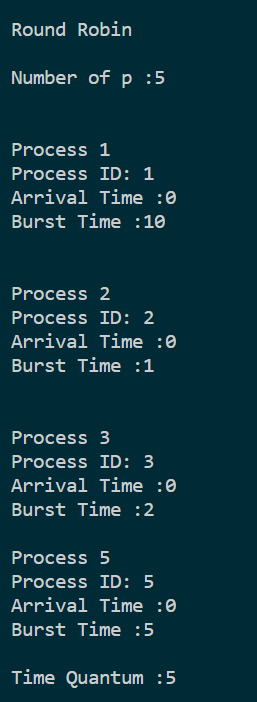
    }

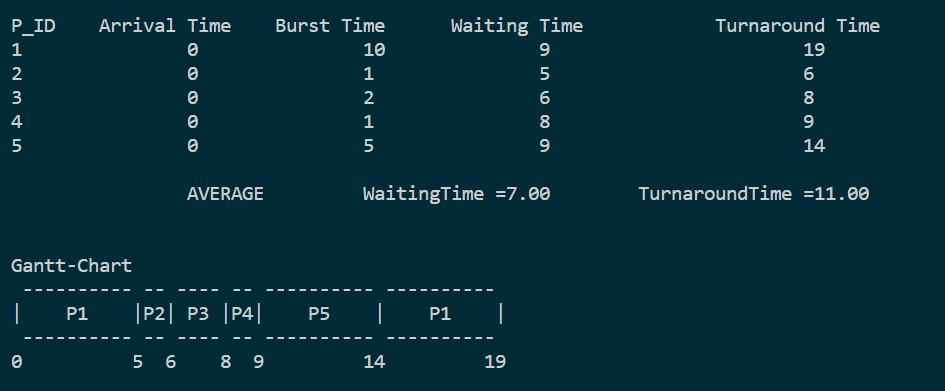
    return 0;

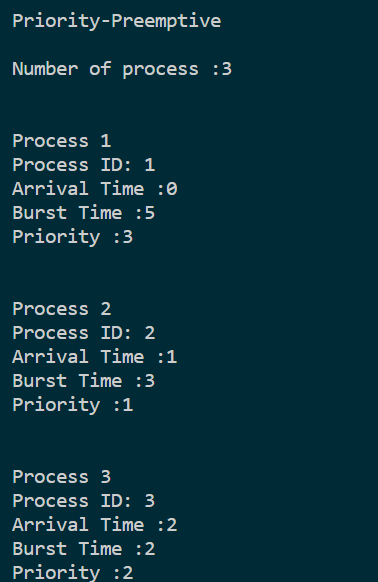
}

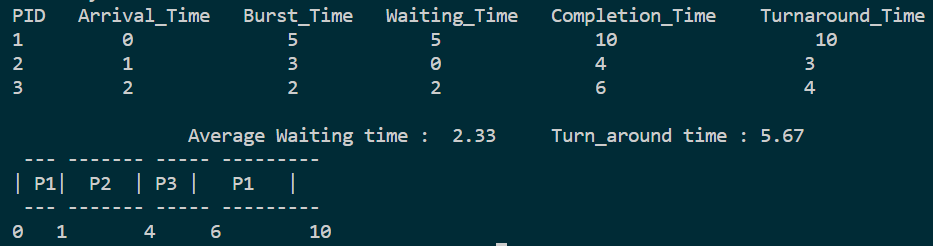
**Output:**

****

****

****

****

****