\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Operating System Lab

CEN-493

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program - 5

Code :-

#include <iostream>

#include <algorithm>

#include <vector>

#include <queue>

#include <unordered\_map>

#include <stack>

using namespace std;

struct Process

{

    string P\_Name;

    int AT;

    int BT;

    int WT;

    int CT;

    int RT;

    int TAT;

};

bool mycomp(Process P1, Process P2)

{

    if (P1.AT != P2.AT)

    {

        return P1.AT < P2.AT;

    }

    else

    {

        int num1 = stoi(P1.P\_Name.substr(1));

        int num2 = stoi(P2.P\_Name.substr(1));

        return num1 < num2;

    }

}

void Print\_Bars()

{

    for (int i = 0; i < 120; i++)

        cout << "\_";

    cout << "\n";

}

void Average\_Time(Process P\_Array[], int T\_Process)

{

    double Av\_CT = 0, Av\_RT = 0, Av\_WT = 0, Av\_TAT = 0;

    for (int i = 0; i < T\_Process; i++)

    {

        Av\_CT += P\_Array[i].CT;

        Av\_RT += P\_Array[i].RT;

        Av\_TAT += P\_Array[i].TAT;

        Av\_WT += P\_Array[i].WT;

    }

    Av\_WT /= T\_Process;

    Av\_TAT /= T\_Process;

    Av\_RT /= T\_Process;

    Av\_CT /= T\_Process;

    cout << "Average Time For The Different Time In Process Scheduling\n\n";

    cout << "Average Completion Time -> " << Av\_CT << "\n";

    cout << "Average Waiting Time -> " << Av\_WT << "\n";

    cout << "Average Turn Around Time -> " << Av\_TAT << "\n";

    cout << "Average Respond Time -> " << Av\_RT << "\n";

}

void GanttChart(vector<pair<string, pair<int, int>>> &All\_Interval)

{

    int size = All\_Interval.size();

    cout << "Gantt Chart For Process Scheduling\n";

    cout << "\n";

    if (All\_Interval[0].second.first != 0)

    {

        cout << "|\t\t|  ";

    }

    else

    {

        cout << "|\t";

    }

    for (int i = 0; i < size; i++)

    {

        if (i != 0 && All\_Interval[i - 1].second.second < All\_Interval[i].second.first)

        {

            cout << "\t|\t";

        }

        cout << All\_Interval[i].first << "\t|\t";

    }

    cout << "\n";

    if (All\_Interval[0].second.first != 0)

    {

        cout << " 0\t";

        cout << All\_Interval[0].second.first << "\t";

    }

    else

    {

        cout << All\_Interval[0].second.first << "\t\t";

    }

    for (int i = 0; i < size; i++)

    {

        if (i != 0 && All\_Interval[i - 1].second.second < All\_Interval[i].second.first)

        {

            cout << All\_Interval[i].second.first << "\t\t";

        }

        cout << All\_Interval[i].second.second << "\t\t";

    }

    cout << "\n";

}

void Chart(Process P\_Array[], int T\_Process)

{

    cout << "Various Time's Related To Process Scheduling\n\n";

    cout << "+---------------------------------------------------------------------------------------------------------------+\n";

    cout << "|\tProcess\t|\tAT\t|\tBT\t|\tCT\t|\tWT\t|\tTAT\t|\tRT      |\n";

    cout << "+---------------------------------------------------------------------------------------------------------------+\n";

    for (int i = 0; i < T\_Process; i++)

    {

        cout << "|\t" << P\_Array[i].P\_Name << "\t|\t" << P\_Array[i].AT

             << "\t|\t" << P\_Array[i].BT << "\t|\t" << P\_Array[i].CT

             << "\t|\t" << P\_Array[i].WT << "\t|\t" << P\_Array[i].TAT

             << "\t|\t" << P\_Array[i].RT << "\t|\n";

    }

    cout << "+---------------------------------------------------------------------------------------------------------------+\n";

}

void Timing(vector<pair<string, pair<int, int>>> &All\_Interval, Process P\_Array[], int T\_Process)

{

    int size = All\_Interval.size();

    for (int i = 0; i < T\_Process; i++)

    {

        for (int j = size - 1; j >= 0; j--)

        {

            if (P\_Array[i].P\_Name == All\_Interval[j].first)

            {

                P\_Array[i].CT = All\_Interval[j].second.second;

                break;

            }

        }

        P\_Array[i].TAT = P\_Array[i].CT - P\_Array[i].AT;

        P\_Array[i].WT = P\_Array[i].TAT - P\_Array[i].BT;

        for (int j = 0; j < size; j++)

        {

            if (P\_Array[i].P\_Name == All\_Interval[j].first)

            {

                P\_Array[i].RT = All\_Interval[j].second.first;

                break;

            }

        }

    }

    Print\_Bars();

    Chart(P\_Array, T\_Process);

    Print\_Bars();

    Average\_Time(P\_Array, T\_Process);

    Print\_Bars();

    GanttChart(All\_Interval);

    Print\_Bars();

}

vector<pair<string, pair<int, int>>> Time\_Intervals(vector<string> &timeArray)

{

    vector<pair<string, pair<int, int>>> processTimeInterval;

    for (int i = 0; i < timeArray.size(); i++)

    {

        int end = timeArray.size();

        for (int j = i + 1; j < timeArray.size(); j++)

        {

            if (timeArray[i] != timeArray[j])

            {

                end = j;

                break;

            }

        }

        processTimeInterval.push\_back({timeArray[i], {i, end}});

        i = end - 1;

    }

    return processTimeInterval;

}

void AddTimeToArray(Process process, vector<string> &timeArray, int timer, int TQ)

{

    for (int i = timer; i < timer + TQ; i++)

    {

        timeArray.push\_back(process.P\_Name);

    }

}

void RoundRobin\_Preemptive(Process P\_Array[], int T\_Process, int TQ)

{

    sort(P\_Array, P\_Array + T\_Process, mycomp);

    queue<Process> que;

    int processIterator = 0;

    vector<string> timeArray;

    que.push(P\_Array[0]);

    int timer = P\_Array[processIterator].AT;

    if (timer != 0)

    {

        Process pnull;

        pnull.P\_Name = "--";

        AddTimeToArray(pnull, timeArray, 0, timer);

    }

    processIterator++;

    while (!que.empty() || processIterator < T\_Process)

    {

        if (!que.empty())

        {

            Process processCpuAllocated = que.front();

            que.pop();

            while (processIterator < T\_Process && timer + min(TQ, processCpuAllocated.BT) >= P\_Array[processIterator].AT)

            {

                que.push(P\_Array[processIterator++]);

            }

            if (processCpuAllocated.BT > TQ)

            {

                processCpuAllocated.BT -= TQ;

                AddTimeToArray(processCpuAllocated, timeArray, timer, TQ);

                que.push(processCpuAllocated);

                timer += TQ;

            }

            else

            {

                int remTime = processCpuAllocated.BT;

                AddTimeToArray(processCpuAllocated, timeArray, timer, remTime);

                timer += remTime;

            }

        }

        else

        {

            timeArray.push\_back("--");

            timer++;

            while (processIterator < T\_Process && timer >= P\_Array[processIterator].AT)

            {

                que.push(P\_Array[processIterator++]);

            }

        }

    }

    vector<pair<string, pair<int, int>>> Intervals = Time\_Intervals(timeArray);

    Timing(Intervals, P\_Array, T\_Process);

}

int main()

{

    system("cls");

    Print\_Bars();

    cout << "20BCS070\_Vicky\_Gupta\n";

    cout << "Round Robin Process Scheduling Alogorithm\n";

    Print\_Bars();

    int T\_Process;

    cout << "Enter The No Of Processes : ";

    cin >> T\_Process;

    int TQ;

    cout << "Enter The Time Quantum : ";

    cin >> TQ;

    fflush(stdin);

    Process P\_Array[T\_Process];

    Print\_Bars();

    cout << "Enter The Process Details...\n";

    cout << "| Process Name | Arival Time | Burst Time | \n";

    for (int i = 0; i < T\_Process; i++)

    {

        cin >> P\_Array[i].P\_Name;

        cin >> P\_Array[i].AT;

        cin >> P\_Array[i].BT;

    }

    RoundRobin\_Preemptive(P\_Array, T\_Process, TQ);

    Print\_Bars();

    cout << "Exited..\n";

    Print\_Bars();

    return 0;

}

Output :-

Graphical user interface

Description automatically generated