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

Operating System Lab

CEN-493

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

Program - 4

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;

};

struct myCompBT

{

    bool operator()(Process &p1, Process const &p2)

    {

        return p1.BT > p2.BT;

    }

};

bool mycomp(Process P1, Process P2)

{

    if (P1.AT != P2.AT)

    {

        return P1.AT < P2.AT;

    }

    else if (P1.BT != P2.BT)

    {

        return P1.BT < P2.BT;

    }

    else

    {

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

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

        return num1 < num2;

    }

}

bool mycompInterval(pair<string, pair<int, int>> p1, pair<string, pair<int, int>> p2)

{

    return p1.second.first < p2.second.first;

}

vector<pair<string, pair<int, int>>> Merge\_Interval\_Helper(vector<pair<int, int>> Interval, string P\_Name)

{

    stack<pair<int, int>> helper;

    int Interval\_Length = Interval.size();

    helper.push(Interval[0]);

    for (int i = 1; i < Interval\_Length; i++)

    {

        if (Interval[i].first <= helper.top().second)

        {

            helper.top().second = Interval[i].second;

        }

        else

        {

            helper.push(Interval[i]);

        }

    }

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

    while (!helper.empty())

    {

        result.push\_back({P\_Name, {helper.top().first, helper.top().second}});

        helper.pop();

    }

    return result;

}

vector<pair<string, pair<int, int>>> Merge\_Interval(unordered\_map<string, vector<pair<int, int>>> &executionTime)

{

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

    for (auto &x : executionTime)

    {

        vector<pair<string, pair<int, int>>> intvl = Merge\_Interval\_Helper(x.second, x.first);

        for (auto &y : intvl)

        {

            Intervals.push\_back(y);

        }

    }

    sort(Intervals.begin(), Intervals.end(), mycompInterval);

    return Intervals;

}

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();

}

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

{

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

    priority\_queue<Process, vector<Process>, myCompBT> pque;

    unordered\_map<string, vector<pair<int, int>>> executionTime;

    int processItertor = 0;

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

    pque.push(P\_Array[processItertor]);

    if (timer != 0)

    {

        executionTime[P\_Array[processItertor].P\_Name].push\_back({0, timer});

    }

    processItertor++;

    while (!pque.empty() || processItertor < T\_Process)

    {

        timer++;

        if (!pque.empty())

        {

            Process process = pque.top();

            pque.pop();

            process.BT--;

            executionTime[process.P\_Name].push\_back({timer - 1, timer});

            if (process.BT != 0)

                pque.push(process);

        }

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

        {

            pque.push(P\_Array[processItertor++]);

        }

    }

    vector<pair<string, pair<int, int>>> All\_Interval = Merge\_Interval(executionTime);

    Timing(All\_Interval, P\_Array, T\_Process);

}

int main()

{

    system("cls");

    Print\_Bars();

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

    cout << "Shortest Job First Preemptive Process Scheduling Alogorithm\n";

    Print\_Bars();

    int T\_Process;

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

    cin >> T\_Process;

    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;

    }

    SJF\_Preemptive(P\_Array, T\_Process);

    Print\_Bars();

    cout << "Exited..\n";

    Print\_Bars();

    return 0;

}

Output :-

Graphical user interface

Description automatically generated