**Operating Systems**

**Preemptive scheduling assignment**

**Name: Kartik Jolapara SAP ID: 60004200107**

**Div.: B1 Branch: Computer Engineering**

# Aim -

# CPU scheduling algorithms like Preemptive Priority, Round Robin etc.

# Problem Statement -

# Perform comparative assessment of various Scheduling Policies like

# Priority preemptive and Round Robin.

# Theory -

Scheduling algorithms are used when more than one process is executable and the OS has to decide which one to run first.

Terms used

1. Submit time: The process at which the process is given to CPU
2. Burst time: The amount of time each process takes for execution
3. Response time: The difference between the time when the process starts execution and the submit time.
4. Turnaround time: The difference between the time when the process completes execution and the submit time.

## Priority Scheduling

Each process is assigned a priority and executable process with highest priority

is allowed to run

## Code –

#include <bits/stdc++.h>

using namespace std;

#define ll long long int

void pro\_sort(vector<pair<pair<ll, ll>, pair<ll, ll>>> &*p*)

{

    for (int i = 0; i < *p*.size() - 1; i++)

    {

        for (int j = 0; j < *p*.size() - i - 1; j++)

        {

            if (*p*[j].second.first > *p*[j + 1].second.first)

            {

                swap(*p*[j], *p*[j + 1]);

            }

        }

    }

}

int main()

{

    ll n, calc = 0;

    cout << "Number of processes: ";

    cin >> n;

    cout << "Process"

         << " "

         << "Priority"

         << " "

         << "AT"

         << " "

         << "BT" << endl;

    vector<pair<pair<ll, ll>, pair<ll, ll>>> process, copy;

    for (int i = 0; i < n; i++)

    {

        int pro, pri, at, bt;

        cin >> pro >> pri >> at >> bt;

        process.push\_back({{pro, pri}, {at, bt}});

        calc += bt;

    }

    copy = process;

    pro\_sort(copy);

    pro\_sort(process);

    unordered\_map<ll, ll> ma;

    int flag = 1;

    int t = min(process[1].second.first, process[0].second.first + process[0].second.second);

    process[0].second.second -= (t - process[0].second.first);

    int start = process[0].second.first;

    calc += start;

    vector<pair<ll, ll>> gc;

    gc.push\_back({process[0].first.first, t});

    while (t < calc)

    {

        int maxp = INT\_MIN, idx;

        for (int i = 0; i < n; i++)

        {

            int pro = process[i].first.first;

            int pri = process[i].first.second;

            int at = process[i].second.first;

            if (ma[pro] != 1 && at <= t)

            {

                if (pri > maxp)

                {

                    maxp = pri;

                    idx = i;

                }

            }

        }

        int pro = process[idx].first.first;

        int pri = process[idx].first.second;

        int at = process[idx].second.first;

        int prev = gc[gc.size() - 1].second;

        gc.push\_back({process[idx].first.first, prev + 1});

        process[idx].second.second -= 1;

        t += 1;

        if (process[idx].second.second == 0)

        {

            ma[process[idx].first.first] = 1;

        }

    }

    unordered\_map<ll, ll> m;

    vector<ll> ct(n + 1), tat(n + 1), wt(n + 1);

    for (int i = gc.size() - 1; i >= 0; i--)

    {

        if (m[gc[i].first] == 0)

        {

            ct[gc[i].first] = gc[i].second;

            m[gc[i].first] = 1;

        }

    }

    for (int i = 0; i < n; i++)

    {

        int pro = process[i].first.first;

        tat[pro] = ct[pro] - process[i].second.first;

        wt[pro] = tat[pro] - copy[i].second.second;

    }

    float avgt = 0, avgw = 0;

    cout << "Process\t\t"

         << "Priority\t"

         << "AT\t"

         << "BT\t"

         << "CT\t"

         << "TAT\t"

         << "WT\t" << endl;

    for (int i = 0; i < n; i++)

    {

        int pro = copy[i].first.first;

        cout << copy[i].first.first << "\t\t" << copy[i].first.second << "\t\t" << copy[i].second.first << "\t" << copy[i].second.second << "\t" << ct[pro] << "\t" << tat[pro] << "\t" << wt[pro] << endl;

        avgt += tat[pro];

        avgw += wt[pro];

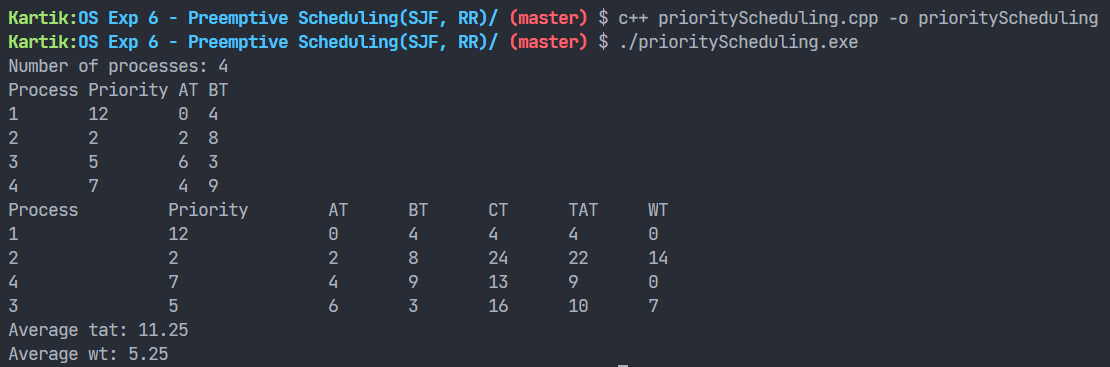
    }

    cout << "Average tat: " << avgt \* 1.0 / n << endl;

    cout << "Average wt: " << avgw \* 1.0 / n << endl;

}

## Output -



## Round Robin (RR)

* Each process is assigned a time interval called its quantum (time slice)
* If the process is still running at the end of the quantum the CPU is preempted and given to another process, and this continues in circular fashion, till all the processes are completely executed

## Code –

#include <iostream>

#include <cstdlib>

#include <queue>

#include <cstdio>

using namespace std;

*/\* C++ Program to Round Robin\*/*

typedef struct process

{

    int id, at, bt, st, ft, pr;

    float wt, tat;

} process;

process p[10], p1[10], temp;

queue<int> q1;

int accept(int *ch*);

void turnwait(int *n*);

void display(int *n*);

void ganttrr(int *n*);

int main()

{

    int i, n, ts, ch, j, x;

    p[0].tat = 0;

    p[0].wt = 0;

    n = accept(ch);

    ganttrr(n);

    turnwait(n);

    display(n);

    return 0;

}

int accept(int *ch*)

{

    int i, n;

    printf("Enter the Total Number of Process: ");

    scanf("%d", &n);

    if (n == 0)

    {

        printf("Invalid");

        exit(1);

    }

    cout << endl;

    for (i = 1; i <= n; i++)

    {

        printf("Enter an Arrival Time of the Process P%d: ", i);

        scanf("%d", &p[i].at);

        p[i].id = i;

    }

    cout << endl;

    for (i = 1; i <= n; i++)

    {

        printf("Enter a Burst Time of the Process P%d: ", i);

        scanf("%d", &p[i].bt);

    }

    for (i = 1; i <= n; i++)

    {

        p1[i] = p[i];

    }

    return n;

}

void ganttrr(int *n*)

{

    int i, ts, m, nextval, nextarr;

    nextval = p1[1].at;

    i = 1;

    cout << "\nEnter the Time Slice or Quantum: ";

    cin >> ts;

    for (i = 1; i <= *n* && p1[i].at <= nextval; i++)

    {

        q1.push(p1[i].id);

    }

    while (!q1.empty())

    {

        m = q1.front();

        q1.pop();

        if (p1[m].bt >= ts)

        {

            nextval = nextval + ts;

        }

        else

        {

            nextval = nextval + p1[m].bt;

        }

        if (p1[m].bt >= ts)

        {

            p1[m].bt = p1[m].bt - ts;

        }

        else

        {

            p1[m].bt = 0;

        }

        while (i <= *n* && p1[i].at <= nextval)

        {

            q1.push(p1[i].id);

            i++;

        }

        if (p1[m].bt > 0)

        {

            q1.push(m);

        }

        if (p1[m].bt <= 0)

        {

            p[m].ft = nextval;

        }

    }

}

void turnwait(int *n*)

{

    int i;

    for (i = 1; i <= *n*; i++)

    {

        p[i].tat = p[i].ft - p[i].at;

        p[i].wt = p[i].tat - p[i].bt;

        p[0].tat = p[0].tat + p[i].tat;

        p[0].wt = p[0].wt + p[i].wt;

    }

    p[0].tat = p[0].tat / *n*;

    p[0].wt = p[0].wt / *n*;

}

void display(int *n*)

{

    int i;

*/\**

*Here*

*at = Arrival time,*

*bt = Burst time,*

*time\_quantum= Quantum time*

*tat = Turn around time,*

*wt = Waiting time*

*\*/*

    cout << "\n=====================================================\n";

    cout << "\n\nHere AT = Arrival Time\nBT = Burst Time\nTAT = Turn Around Time\nWT = Waiting Time\n ";

    cout << "\n===================TABLE==============================\n";

    printf("\nProcess\tAT\tBT\tFT\tTAT\t\tWT");

    for (i = 1; i <= *n*; i++)

    {

        printf("\nP%d\t%d\t%d\t%d\t%f\t%f", p[i].id, p[i].at, p[i].bt, p[i].ft, p[i].tat, p[i].wt);

    }

    cout << "\n=====================================================\

n";

    printf("\nAverage Turn Around Time: %f", p[0].tat);

    printf("\nAverage Waiting Time: %f\n", p[0].wt);

}

## Output –

