Experiment Number: 9

Problem Statement: **Write a program to compute the finish time, turnaround time and waiting time for the following algorithms:**

**a) First come First serve**

**b) Shortest Job First (Preemptive and Non-Preemptive)**

**c) Priority (Preemptive and Non-Preemptive)**

**d) Round robin**

NAME: Aadesh Chawla ROLLNO: 12

CLASS: TY-IT-A BATCH: B1

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

1. **First come First serve**

**Code:**

#include <iostream>

#include <vector>

using namespace std;

void customSchedulingAlgorithm(vector<string> jobs, vector<int> executionTime)

{

    int numJobs = jobs.size();

    vector<int> completionTime(numJobs, 0);

    vector<int> turnaroundTime(numJobs, 0);

    vector<int> waitingTime(numJobs, 0);

    completionTime[0] = executionTime[0];

    turnaroundTime[0] = completionTime[0];

    waitingTime[0] = 0;

    for (int i = 1; i < numJobs; i++)

    {

        completionTime[i] = completionTime[i - 1] + executionTime[i];

        turnaroundTime[i] = completionTime[i];

        waitingTime[i] = turnaroundTime[i] - executionTime[i];

    }

    cout << "Job\tCompletion Time\tTurnaround Time\tWaiting Time\n";

    for (int i = 0; i < numJobs; i++)

    {

        cout << jobs[i] << "\t" << completionTime[i] << "\t\t" << turnaroundTime[i] << "\t\t" << waitingTime[i] << "\n";

    }

}

int main()

{

    vector<string> jobs = {"Job1", "Job2", "Job3", "Job4"};

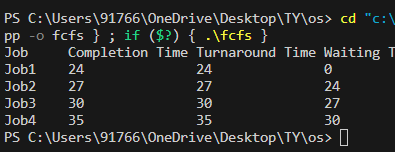
    vector<int> executionTime = {24, 3, 3, 5};

    customSchedulingAlgorithm(jobs, executionTime);

    return 0;

}

**Output:**

****

**b) Shortest Job First (Preemptive and Non-Preemptive)**

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void sjfNonPreemptive(vector<string> jobs, vector<int> executionTime) {

    int n = jobs.size();

    vector<int> finishTime(n, 0);

    vector<int> turnaroundTime(n, 0);

    vector<int> waitingTime(n, 0);

    vector<int> executionTimeSorted(n);

    for (int i = 0; i < n; i++) {

        executionTimeSorted[i] = executionTime[i];

    }

    sort(executionTimeSorted.begin(), executionTimeSorted.end());

    finishTime[0] = executionTime[0];

    turnaroundTime[0] = finishTime[0];

    waitingTime[0] = 0;

    for (int i = 1; i < n; i++) {

        int index = find(executionTime.begin(), executionTime.end(), executionTimeSorted[i]) - executionTime.begin();

        finishTime[i] = finishTime[i-1] + executionTime[index];

        turnaroundTime[i] = finishTime[i];

        waitingTime[i] = turnaroundTime[i] - executionTime[index];

    }

    cout << "Non-Preemptive SJF Scheduling:" << endl;

    cout << "Process\tFinish Time\tTurnaround Time\tWaiting Time\n";

    for (int i = 0; i < n; i++) {

        cout << jobs[i] << "\t" << finishTime[i] << "\t\t" << turnaroundTime[i] << "\t\t" << waitingTime[i] << "\n";

    }

}

void sjfPreemptive(vector<string> jobs, vector<int> executionTime) {

    int n = jobs.size();

    vector<int> finishTime(n, 0);

    vector<int> turnaroundTime(n, 0);

    vector<int> waitingTime(n, 0);

    vector<int> remainingExecutionTime(n);

    for (int i = 0; i < n; i++) {

        remainingExecutionTime[i] = executionTime[i];

    }

    int time = 0;

    while (true) {

        bool done = true;

        for (int i = 0; i < n; i++) {

            if (remainingExecutionTime[i] > 0) {

                done = false;

                if (remainingExecutionTime[i] > 1) {

                    time += 1;

                    remainingExecutionTime[i] -= 1;

                } else {

                    time += remainingExecutionTime[i];

                    waitingTime[i] = time - executionTime[i];

                    remainingExecutionTime[i] = 0;

                    finishTime[i] = time;

                }

            }

        }

        if (done) {

            break;

        }

    }

    for (int i = 0; i < n; i++) {

        turnaroundTime[i] = finishTime[i];

    }

    cout << "\nPreemptive SJF Scheduling:" << endl;

    cout << "Process\tFinish Time\tTurnaround Time\tWaiting Time\n";

    for (int i = 0; i < n; i++) {

        cout << jobs[i] << "\t" << finishTime[i] << "\t\t" << turnaroundTime[i] << "\t\t" << waitingTime[i] << "\n";

    }

}

int main() {

    vector<string> jobs = {"Job1", "Job2", "Job3", "Job4"};

    vector<int> executionTime = {6, 8, 7, 3};

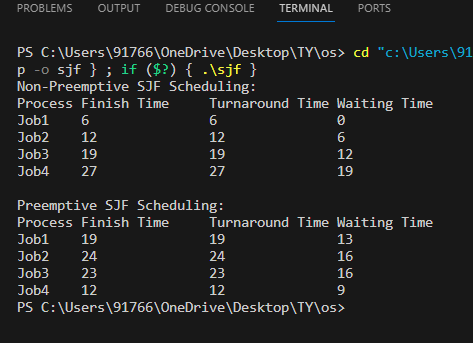
    sjfNonPreemptive(jobs, executionTime);

    sjfPreemptive(jobs, executionTime);

    return 0;

}

**Output:**

****

**c) Priority (Preemptive and Non-Preemptive)**

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void priorityNonPreemptive(vector<string> jobs, vector<int> burstTime, vector<int> jobPriority) {

    int n = jobs.size();

    vector<int> finishTime(n, 0);

    vector<int> turnaroundTime(n, 0);

    vector<int> waitingTime(n, 0);

    vector<int> prioritySorted(n);

    for (int i = 0; i < n; i++) {

        prioritySorted[i] = jobPriority[i];

    }

    sort(prioritySorted.begin(), prioritySorted.end(), greater<int>());

    finishTime[0] = burstTime[0];

    turnaroundTime[0] = finishTime[0];

    waitingTime[0] = 0;

    for (int i = 1; i < n; i++) {

        int index = find(jobPriority.begin(), jobPriority.end(), prioritySorted[i]) - jobPriority.begin();

        finishTime[i] = finishTime[i-1] + burstTime[index];

        turnaroundTime[i] = finishTime[i];

        waitingTime[i] = turnaroundTime[i] - burstTime[index];

    }

    cout << "Non-Preemptive Priority Scheduling:" << endl;

    cout << "Process\tFinish Time\tTurnaround Time\tWaiting Time\n";

    for (int i = 0; i < n; i++) {

        cout << jobs[i] << "\t" << finishTime[i] << "\t\t" << turnaroundTime[i] << "\t\t" << waitingTime[i] << "\n";

    }

}

void priorityPreemptive(vector<string> jobs, vector<int> burstTime, vector<int> jobPriority) {

    int n = jobs.size();

    vector<int> finishTime(n, 0);

    vector<int> turnaroundTime(n, 0);

    vector<int> waitingTime(n, 0);

    vector<int> remainingBurstTime(n);

    for (int i = 0; i < n; i++) {

        remainingBurstTime[i] = burstTime[i];

    }

    int time = 0;

    while (true) {

        int maxPriority = \*max\_element(jobPriority.begin(), jobPriority.end());

        if (maxPriority == INT\_MIN) {

            break;

        }

        int maxPriorityIndex = find(jobPriority.begin(), jobPriority.end(), maxPriority) - jobPriority.begin();

        if (remainingBurstTime[maxPriorityIndex] > 0) {

            finishTime[maxPriorityIndex] = time + 1;

            remainingBurstTime[maxPriorityIndex] -= 1;

        }

        if (remainingBurstTime[maxPriorityIndex] == 0) {

            jobPriority[maxPriorityIndex] = INT\_MIN;

            turnaroundTime[maxPriorityIndex] = finishTime[maxPriorityIndex];

            waitingTime[maxPriorityIndex] = turnaroundTime[maxPriorityIndex] - burstTime[maxPriorityIndex];

        }

        time += 1;

    }

    for (int i = 0; i < n; i++) {

        turnaroundTime[i] = finishTime[i];

    }

    cout << "\nPreemptive Priority Scheduling:" << endl;

    cout << "Process\tFinish Time\tTurnaround Time\tWaiting Time\n";

    for (int i = 0; i < n; i++) {

        cout << jobs[i] << "\t" << finishTime[i] << "\t\t" << turnaroundTime[i] << "\t\t" << waitingTime[i] << "\n";

    }

}

int main() {

    vector<string> jobs = {"Job1", "Job2", "Job3", "Job4"};

    vector<int> burstTime = {8, 4, 9, 5};

    vector<int> jobPriority = {2, 3, 1, 4};

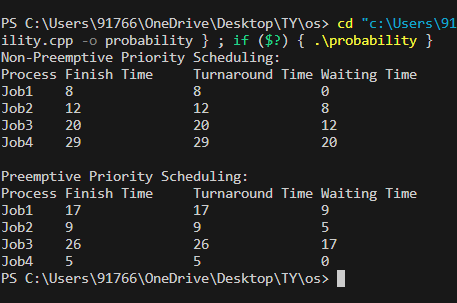
    priorityNonPreemptive(jobs, burstTime, jobPriority);

    priorityPreemptive(jobs, burstTime, jobPriority);

    return 0;

}

**Output:**

****

**d) Round robin**

**Code:**

#include <iostream>

#include <vector>

using namespace std;

void roundRobinScheduling(vector<string> jobs, vector<int> burstTime, int timeQuantum) {

    int n = jobs.size();

    vector<int> finishTime(n, 0);

    vector<int> turnaroundTime(n, 0);

    vector<int> waitingTime(n, 0);

    vector<int> remainingBurstTime(n);

    for (int i = 0; i < n; i++) {

        remainingBurstTime[i] = burstTime[i];

    }

    int time = 0;

    while (true) {

        bool done = true;

        for (int i = 0; i < n; i++) {

            if (remainingBurstTime[i] > 0) {

                done = false;

                if (remainingBurstTime[i] > timeQuantum) {

                    time += timeQuantum;

                    remainingBurstTime[i] -= timeQuantum;

                } else {

                    time += remainingBurstTime[i];

                    waitingTime[i] = time - burstTime[i];

                    remainingBurstTime[i] = 0;

                    finishTime[i] = time;

                }

            }

        }

        if (done) {

            break;

        }

    }

    for (int i = 0; i < n; i++) {

        turnaroundTime[i] = finishTime[i];

    }

    cout << "Round Robin Scheduling:" << endl;

    cout << "Process\tFinish Time\tTurnaround Time\tWaiting Time\n";

    for (int i = 0; i < n; i++) {

        cout << jobs[i] << "\t" << finishTime[i] << "\t\t" << turnaroundTime[i] << "\t\t" << waitingTime[i] << "\n";

    }

}

int main() {

    vector<string> jobs = {"Job1", "Job2", "Job3"};

    vector<int> burstTime = {10, 5, 8};

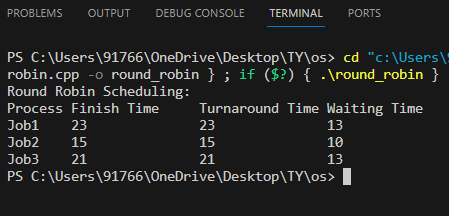
    int timeQuantum = 2;

    roundRobinScheduling(jobs, burstTime, timeQuantum);

    return 0;

}

**Output:**

****