**Batch: D - 1 Roll No.: 16010122096**

**Experiment No. 04**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of Basic Process management algorithms – Non Pre-emptive ( FCFS , SJF, priority) |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To implement basic Non –Pre-emptive Process management algorithms ( FCFS , SJF , Priority)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 2.** To understand the concept of process, thread and resource management.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Most systems handle numerous processes with short CPU bursts interspersed with I/O requests and a few processes with long CPU bursts. To ensure good time-sharing performance, a running process may be preempted to allow another to run. The ready list, or run queue, maintains all processes ready to run and not blocked by I/O or other system requests. Entries in this list point to the process control block, which stores all process information and state.

When an I/O request completes, the process moves from the waiting state to the ready state and is placed on the run queue. The process scheduler, a key component of the operating system, decides whether the current process should continue running or if another should take over. This decision is triggered by four events:

1. The current process issues an I/O request or system request, moving it from running to waiting.
2. The current process terminates.
3. A timer interrupt indicates the process has run for its allotted time, moving it from running to ready.
4. An I/O operation completes, moving the process from waiting to ready, potentially preempting the current process.

The scheduling algorithm, or policy, determines the sequence and duration of process execution, a complex task given the limited information about ready processes.

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

**Description of the application to be implemented**:

**First-Come, First-Served Scheduling:**

# The FCFS scheduling algorithm processes tasks in the order they arrive. Each process is added to a queue and executed sequentially, with no preemption. The simplicity of FCFS makes it easy to implement, but it can lead to the "convoy effect," where short processes are delayed by longer ones, resulting in potentially inefficient CPU utilization and increased average waiting times.

# Shortest job first :

The SJF scheduling algorithm selects the process with the shortest burst time for execution next, minimizing the average waiting time. This non-preemptive strategy is optimal when all processes are available simultaneously, as it ensures that the CPU spends less time waiting. However, its effectiveness can be compromised if process durations are not known in advance, and it can suffer from the "starvation" problem for longer processes.

**Implementation details:**

**FCFS:**

#include <bits/stdc++.h>

using namespace std;

void fcfs(vector<pair<int, int>> &processes, vector<int> &arrival)

{

   int completion = 0;

   int n = processes.size();

   vector<int> turnAround(n), waitingTime(n);

   cout << "Process Completion time:" << endl;

   for (int i = 0; i < n; ++i)

   {

      if (completion < arrival[i])

      {

         completion = arrival[i];

      }

      completion += processes[i].second;

      cout << "P" << processes[i].first << ": " << completion << endl;

      turnAround[i] = completion - arrival[i];

      waitingTime[i] = turnAround[i] - processes[i].second;

   }

   cout << endl

        << "Turn Around Time: " << endl;

   for (int i = 0; i < n; ++i)

   {

      cout << "P" << processes[i].first << ": " << turnAround[i] << endl;

   }

   cout << endl

        << "Waiting Time: " << endl;

   for (int i = 0; i < n; ++i)

   {

      cout << "P" << processes[i].first << ": " << waitingTime[i] << endl;

   }

   cout << endl;

   cout << "Average turn around time = " << (float)accumulate(turnAround.begin(), turnAround.end(), 0) / n << " ms" << endl;

   cout << "Average waiting time = " << (float)accumulate(waitingTime.begin(), waitingTime.end(), 0) / n << " ms" << endl;

}

void input()

{

   cout << "First Come First Serve (FCFS) Scheduling Algorithm" << endl;

   cout << "Enter the number of processes: ";

   int n;

   cin >> n;

   vector<pair<int, int>> processes(n); // Process ID and Burst

   cout << "Enter process ID and Burst time for each process: " << endl;

   for (int i = 0; i < n; i++)

   {

      cin >> processes[i].first >> processes[i].second;

   }

   cout << endl;

   int choice;

   cout << "Does all the processes arrive at same time (enter 0) else (enter 1)" << endl;

   cin >> choice;

   cout << endl;

   vector<int> arrivalTimes(n);

   switch (choice)

   {

   case 0:

      cout << "Enter the arrival time processes: " << endl;

      int x;

      cin >> x;

      for (int i = 0; i < n; ++i)

         arrivalTimes[i] = x;

      fcfs(processes, arrivalTimes);

      break;

   case 1:

      cout << "Enter the arrival time for each process: " << endl;

      for (int i = 0; i < n; i++)

         cin >> arrivalTimes[i];

      cout << endl;

      fcfs(processes, arrivalTimes);

      break;

   default:

      cout << "Invalid choice. Please try again." << endl;

      return;

   }

}

int main()

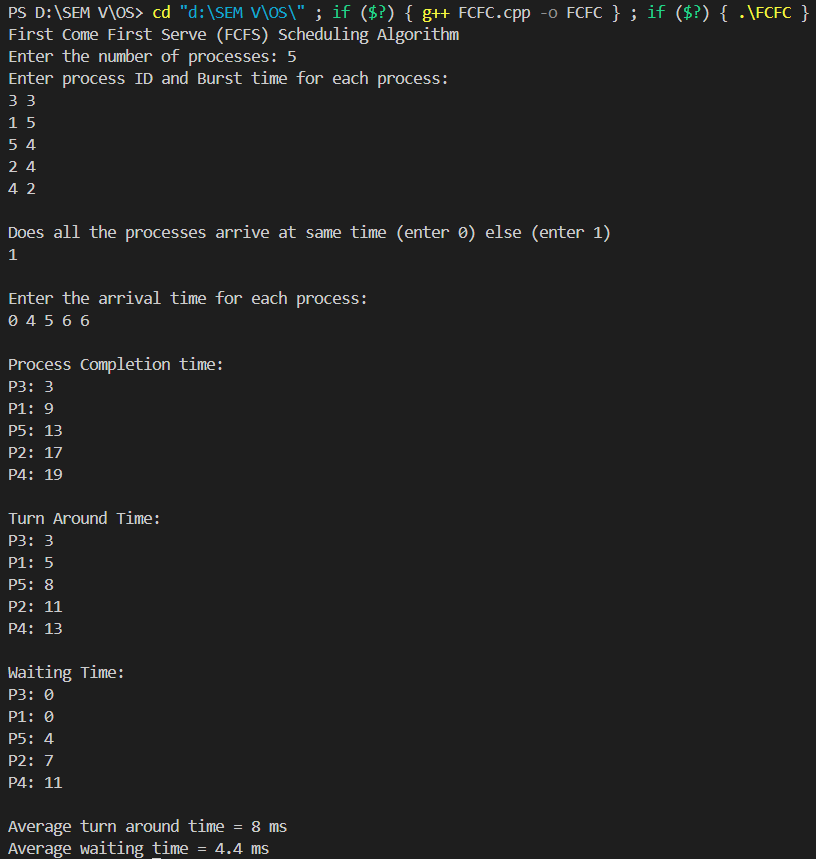
{

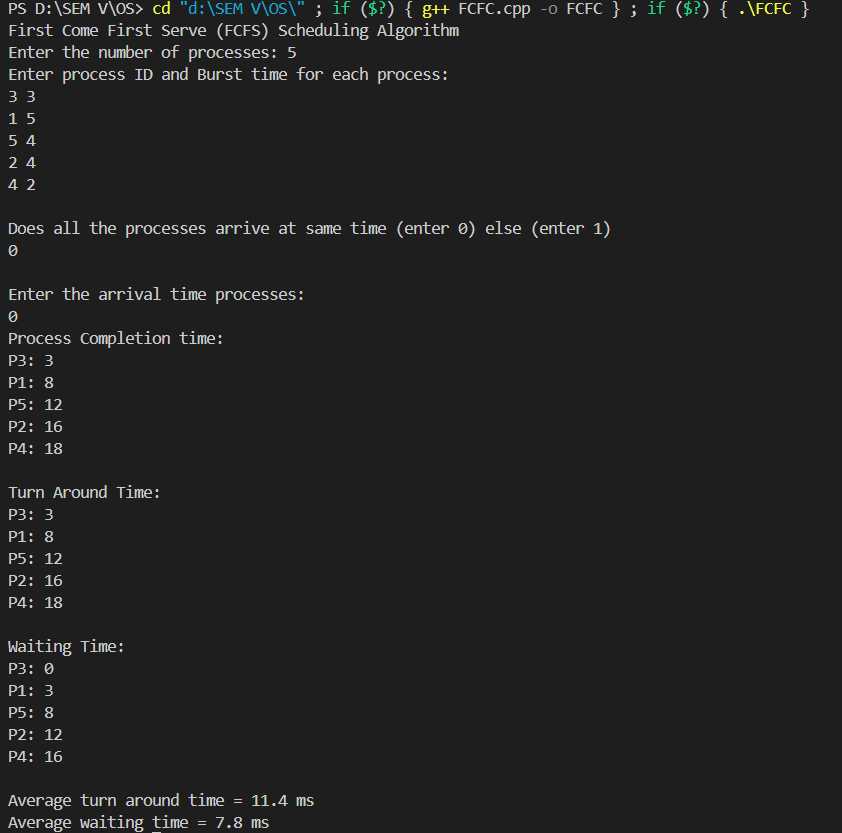
   input();

   return 0;

}

**Output:**





**SJF:**

#include <bits/stdc++.h>

using namespace std;

void sjf(vector<pair<int, int>> &processes, vector<int> &arrival)

{

   int completion = 0;

   int n = processes.size();

   vector<int> turnAround(n), waitingTime(n);

   vector<bool> isCompleted(n, false);

   cout << "Process Completion time:" << endl;

   for (int i = 0; i < n; ++i)

   {

      int idx = -1;

      int minBurst = INT\_MAX;

      // Find the process with the shortest burst time that has arrived

      for (int j = 0; j < n; ++j)

      {

         if (!isCompleted[j] && arrival[j] <= completion && processes[j].second < minBurst)

         {

            minBurst = processes[j].second;

            idx = j;

         }

      }

      // If no process has arrived yet, move completion to the arrival time of the next process

      if (idx == -1)

      {

         int earliestArrival = INT\_MAX;

         for (int j = 0; j < n; ++j)

         {

            if (!isCompleted[j] && arrival[j] < earliestArrival)

            {

               earliestArrival = arrival[j];

               idx = j;

            }

         }

         completion = arrival[idx]; // Update completion to the earliest arrival time

      }

      // Process the selected process

      completion += processes[idx].second;

      cout << "P" << processes[idx].first << ": " << completion << endl;

      turnAround[idx] = completion - arrival[idx];

      waitingTime[idx] = turnAround[idx] - processes[idx].second;

      isCompleted[idx] = true;

   }

   cout << endl

        << "Turn Around Time: " << endl;

   for (int i = 0; i < n; ++i)

   {

      cout << "P" << processes[i].first << ": " << turnAround[i] << endl;

   }

   cout << endl

        << "Waiting Time: " << endl;

   for (int i = 0; i < n; ++i)

   {

      cout << "P" << processes[i].first << ": " << waitingTime[i] << endl;

   }

   cout << endl;

   cout << "Average turn around time = " << (float)accumulate(turnAround.begin(), turnAround.end(), 0) / n << " ms" << endl;

   cout << "Average waiting time = " << (float)accumulate(waitingTime.begin(), waitingTime.end(), 0) / n << " ms" << endl;

}

void input()

{

   cout << "Shortest Job First (SJF) Scheduling Algorithm" << endl;

   cout << "Enter the number of processes: ";

   int n;

   cin >> n;

   vector<pair<int, int>> processes(n); // Process ID and Burst

   cout << "Enter process ID and Burst time for each process: " << endl;

   for (int i = 0; i < n; i++)

   {

      cin >> processes[i].first >> processes[i].second;

   }

   cout << endl;

   int choice;

   cout << "Does all the processes arrive at same time (enter 0) else (enter 1)" << endl;

   cin >> choice;

   cout << endl;

   vector<int> arrivalTimes(n);

   switch (choice)

   {

   case 0:

      cout << "Enter the arrival time processes: " << endl;

      int x;

      cin >> x;

      for (int i = 0; i < n; ++i)

         arrivalTimes[i] = x;

      sjf(processes, arrivalTimes);

      break;

   case 1:

      cout << "Enter the arrival time for each process: " << endl;

      for (int i = 0; i < n; i++)

         cin >> arrivalTimes[i];

      cout << endl;

      sjf(processes, arrivalTimes);

      break;

   default:

      cout << "Invalid choice. Please try again." << endl;

      return;

   }

}

int main()

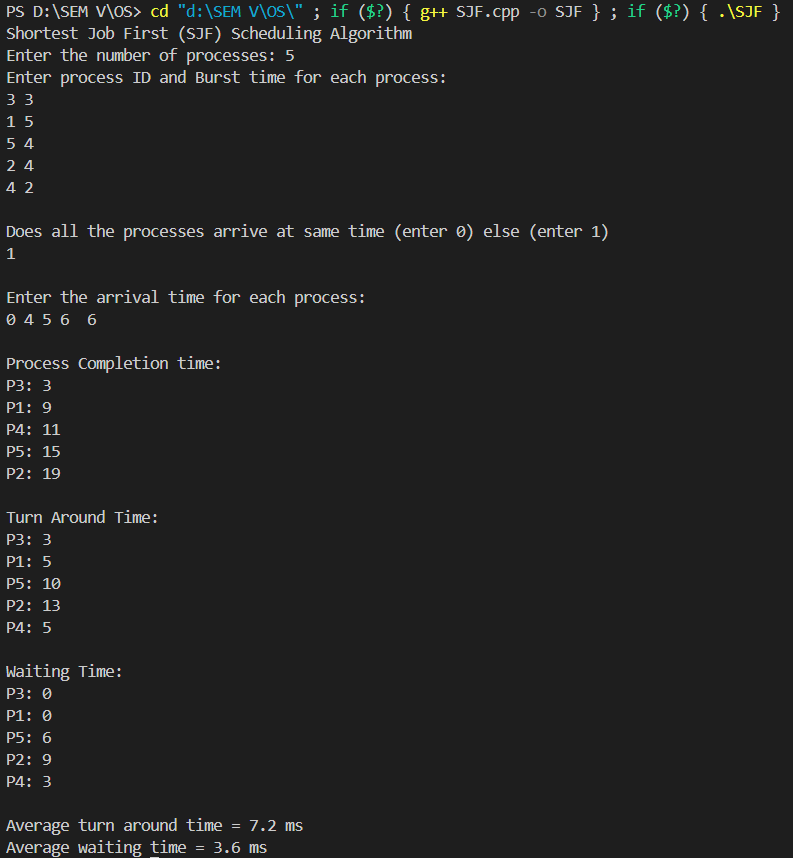
{

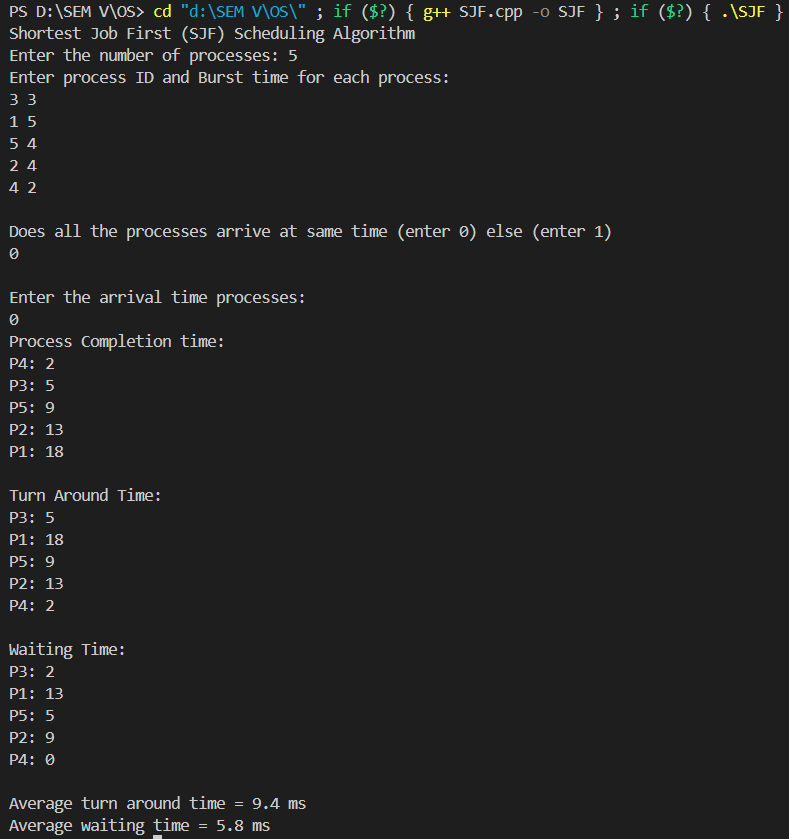
   input();

   return 0;

}

**Output:**





**Conclusion:**

The experiment successfully demonstrated the implementation of non-preemptive process management algorithms, FCFS and SJF, helping understand key concepts in process scheduling, resource management, and their impact on system performance.

**Post Lab Questions**

1. What is a criterion to evaluate a scheduling algorithm?

* **CPU Utilization:** Ensures the CPU is kept as busy as possible.
* **Throughput:** Measures the number of processes completed in a unit of time.
* **Turnaround Time:** The total time taken from submission to completion of a process.
* **Waiting Time:** The time a process spends waiting in the ready queue.
* **Response Time:** The time from submission to the first response of a process.
* **Fairness:** Ensures that each process gets a fair share of CPU time.

1. Analyse the efficiency and suitability of FCFS, SJF, and Priority scheduling algorithms.
   * **FCFS (First-Come, First-Served):**

* **Efficiency:** Simple and easy to implement, but can lead to long waiting times for short processes due to the "convoy effect."
* **Suitability:** Suitable for batch systems with similar job lengths, but not ideal for interactive or time-critical tasks.
  + **SJF (Shortest Job First):**
* **Efficiency:** Optimal in minimizing average waiting time, making it efficient for systems where process execution times are known in advance.
* **Suitability:** Best for environments where short jobs are common, but not ideal for systems with unpredictable or varying process times.
  + **Priority Scheduling:**
  + **Efficiency:** Efficient when prioritizing critical tasks, but can lead to longer wait times for lower-priority processes.
  + **Suitability:** Suitable for systems where certain tasks must be prioritized, such as real-time systems, but care must be taken to avoid starvation of lower-priority processes.

1. A brief explanation of the concept of "starvation" in SJF scheduling and how to avoid it.

* **Concept:** Starvation occurs in SJF when longer processes are continually postponed because shorter processes keep arriving, resulting in the longer processes never getting CPU time.
* **Avoidance:** One way to avoid starvation is by implementing **aging**, where the priority of a process increases the longer it waits, eventually ensuring that all processes are executed regardless of their initial burst time. This can help balance efficiency with fairness.

**Date: 31 / 08 / 2024 Signature of faculty in-charge**