**Batch:C1 Roll No.:16010122221**

**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:

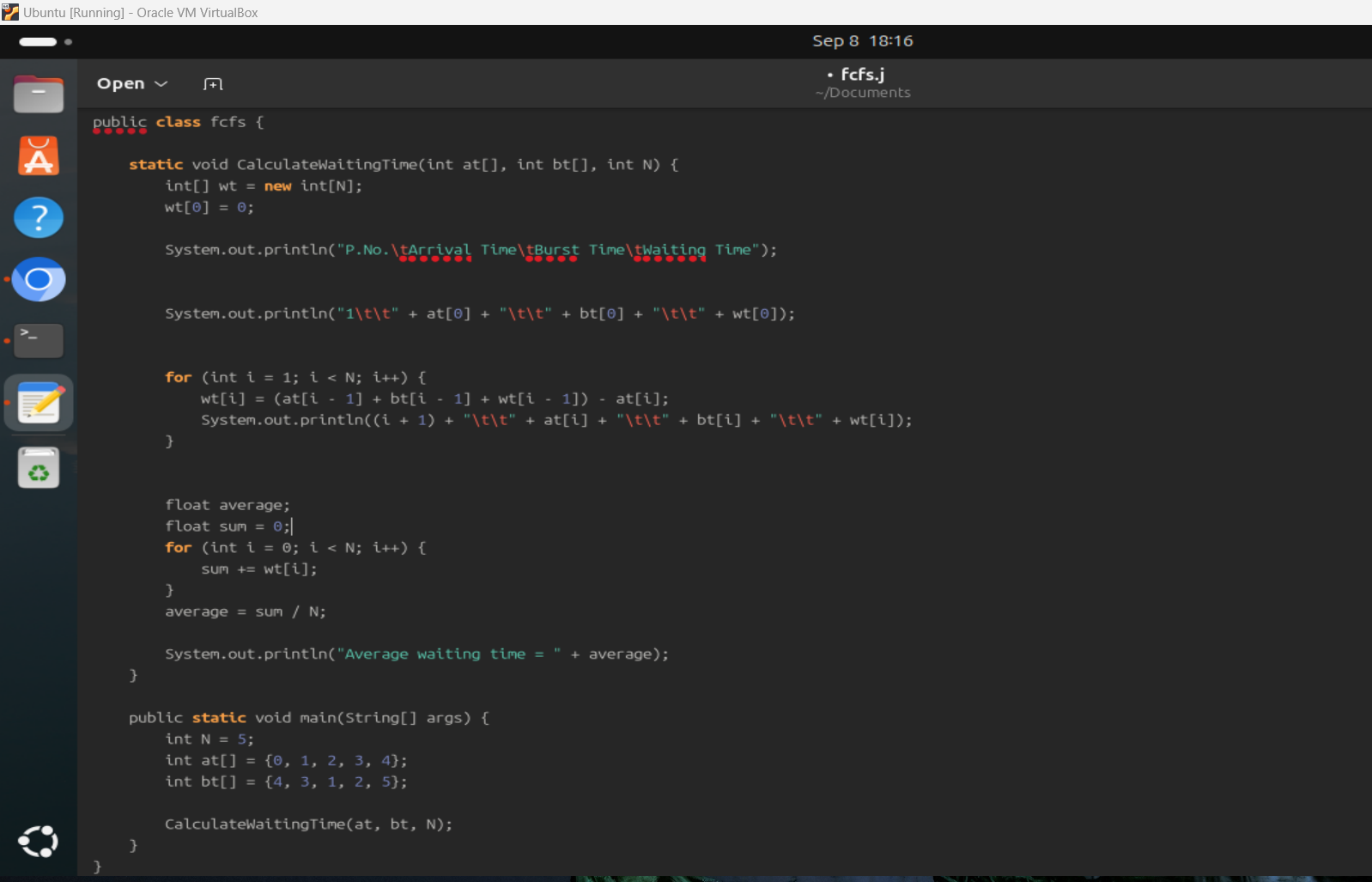
1. The current process issues an I/O request or system request, moving it from running to waiting.
2. The current process terminates.
3. A timer interrupt indicates the process has run for its allotted time, moving it from running to ready.
4. An I/O operation completes, moving the process from waiting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

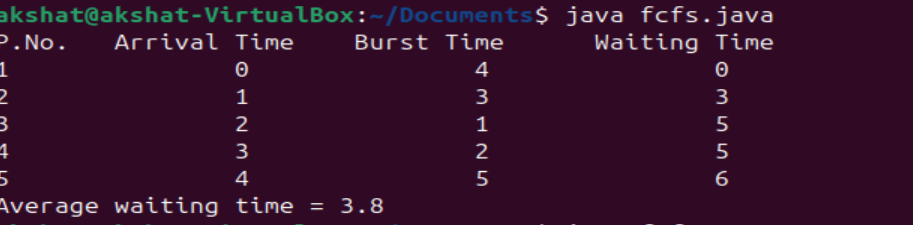
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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



**Output:**

****

**Shortest job First:**

import java.util.\*;

public class sjf {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of processes:");

int n = sc.nextInt();

int[] pid = new int[n];

int[] at = new int[n];

int[] bt = new int[n];

int[] ct = new int[n];

int[] ta = new int[n];

int[] wt = new int[n];

int[] f = new int[n];

int st = 0;

int tot = 0;

float avgwt = 0, avgta = 0;

for (int i = 0; i < n; i++) {

System.out.println("Enter process " + (i + 1) + " arrival time:");

at[i] = sc.nextInt();

System.out.println("Enter process " + (i + 1) + " burst time:");

bt[i] = sc.nextInt();

pid[i] = i + 1;

f[i] = 0;

}

while (true) {

int c = n;

int min = Integer.MAX\_VALUE;

if (tot == n)

break;

for (int i = 0; i < n; i++) {

if ((at[i] <= st) && (f[i] == 0) && (bt[i] < min)) {

min = bt[i];

c = i;

}

}

if (c == n) {

st++;

} else {

ct[c] = st + bt[c];

st += bt[c];

ta[c] = ct[c] - at[c];

wt[c] = ta[c] - bt[c];

f[c] = 1;

tot++;

}

}

System.out.println("\nPID\tArrival\tBurst\tComplete\tTurnaround\tWaiting");

for (int i = 0; i < n; i++) {

avgwt += wt[i];

avgta += ta[i];

System.out.println(pid[i] + "\t" + at[i] + "\t" + bt[i] + "\t" + ct[i] + "\t\t" + ta[i] + "\t\t" + wt[i]);

}

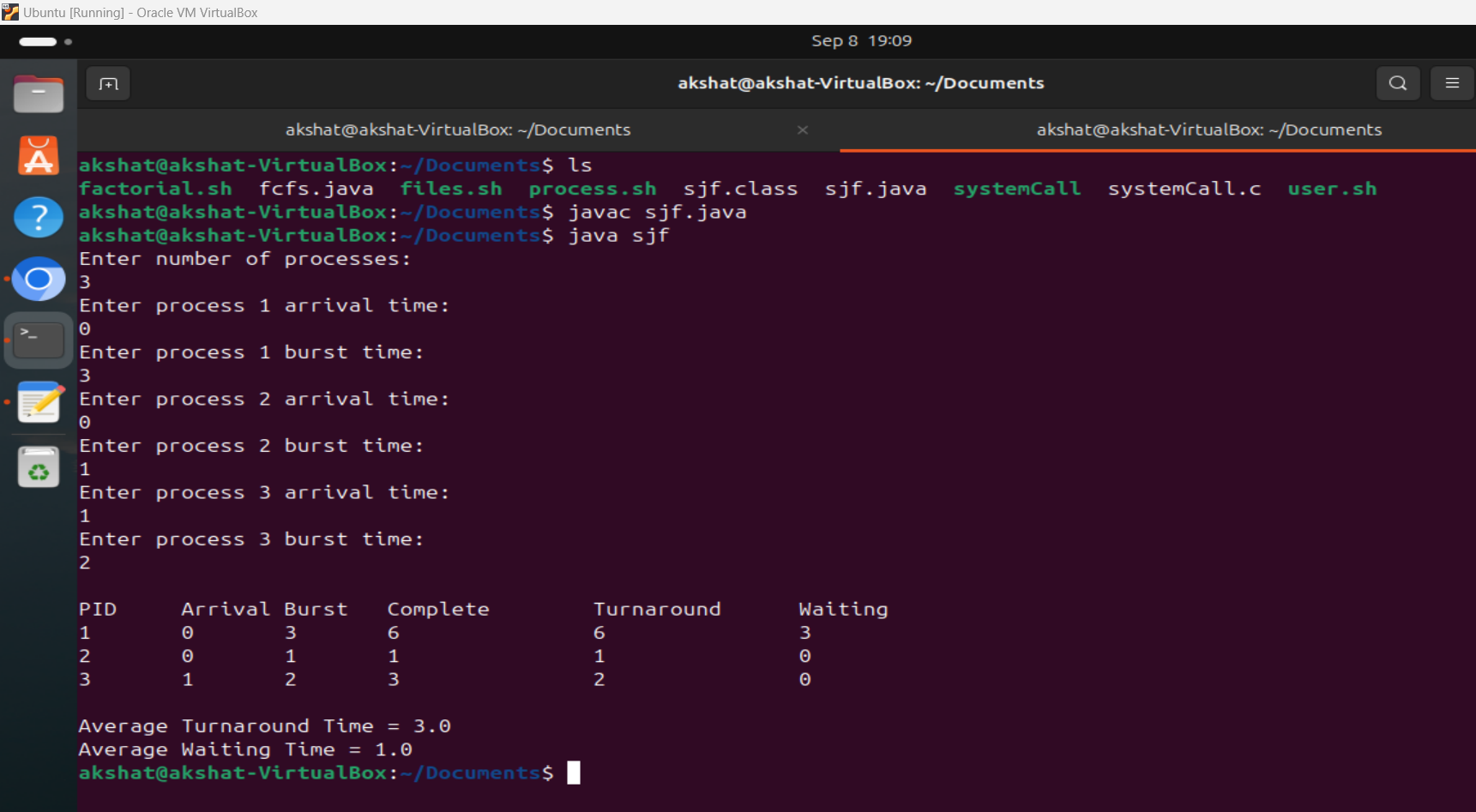
System.out.println("\nAverage Turnaround Time = " + (float) (avgta / n));

System.out.println("Average Waiting Time = " + (float) (avgwt / n));

sc.close();

}

}



**Conclusion :**

Through this experiment we understood the concept of non-pre-emptive scheduling algorithm and implemented First Come First Serve and Shortest Job First algorithm in C++ language.

**Post Lab Questions**

1. Criterion to Evaluate a Scheduling Algorithm

Scheduling algorithms can be evaluated based on several criteria, including:

- CPU Utilization: Measures how effectively the CPU is utilized. The goal is to maximize CPU usage.

- Throughput: Number of processes completed per unit time. Higher throughput is preferred.

- Turnaround Time: The total time taken from submission to completion of a process. Lower turnaround time is desirable.

- Waiting Time: The total time a process spends waiting in the ready queue. Lower waiting time improves system responsiveness.

- Response Time: Time from request submission to the first response. Important for interactive systems.

- Fairness: Ensures all processes get an equal opportunity to execute without significant delays.

2. Analysis of FCFS, SJF, and Priority Scheduling Algorithms

- First-Come, First-Served (FCFS):

- Efficiency: Simple to implement. Processes are executed in the order they arrive.

- Suitability: Suitable for batch systems but inefficient for time-sharing or real-time systems.

- Drawbacks: Causes the convoy effect, where long processes delay shorter ones, leading to high waiting times for subsequent processes.

- Shortest Job First (SJF):

- Efficiency: Minimizes average waiting time since shorter jobs are prioritized.

- Suitability: Best for systems with predictable job lengths, such as batch processing.

- Drawbacks: Cannot handle real-time or interactive processes well. Risk of starvation for longer processes.

- Priority Scheduling:

- Efficiency: Prioritizes processes based on importance (priority value).

- Suitability: Good for time-critical applications where certain processes must be executed first.

- Drawbacks: Can lead to starvation of lower-priority processes. Requires careful management of priority levels.

3. Starvation in SJF Scheduling and How to Avoid It

Starvation occurs in \*\*Shortest Job First (SJF) scheduling when longer processes continuously get postponed due to the arrival of shorter processes. Since SJF always favors the process with the shortest burst time, processes with longer execution times may never get executed if new short jobs keep arriving.

Ways to Avoid Starvation:

- Aging: Increase the priority of a process the longer it waits. This ensures that eventually, even long processes get executed.

- Hybrid Algorithms: Combine SJF with time-based priority adjustments or round-robin scheduling to ensure fairness.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**