First Come First Served (FCFS) Scheduling Algorithm

Student Name	ID
Ahmed Essam Taj	*2036930*
Mohammad Hamdan Alsefri	*2037161*
Fahad Hamad AlSifri	*1743998*

Course: CPIT 260

Section: A1

Instructor: Dr. Iftikhar Ahmad

Project name: First Come First Served (FCFS) Scheduling Algorithm

Date: 5/10/2022

Table of Contents

FCFS Algorithm Project

We are going to make a program that implements the FCFS algorithm. The program will start with an interface that ask the user to enter a number of process and then choose to wither you want to apply FCFS with ignoring the arrival time or not. Lastly the program will show the Gantt chart and the average waiting and turnaround time.

Checklist

- ✓ Title of the project
- ✓ Name of students
- ✓ Introduction
- ✓ Description of the project
- ✓ Java code
- ✓ Output screenshots

Introduction:

CPU Scheduling is an important concept in operating systems. A CPU must be scheduled in order to achieve CPU Utilization, and fairness. There are multiple techniques used to schedule a CPU such as First Come First Served (FCFS), Shortest Job First (SJF), and priority scheduling etc. In this report our mainly topic is FCFS.

The first-come, first-served (FCFS) algorithm is a type of scheduling algorithm that can be used by operating systems and networks to efficiently perform various tasks and processes. It is commonly referred to as a first-in, first-out (FIFO), or first-come, first-served (FCFS) algorithm.

FCFS algorithm is simple to implement, as it can handle various tasks and requests regardless of their complexity. It can also mimic the customer service experience of a grocery store by determining which patrons will be first served.

Unlike other scheduling algorithms, the FCFS algorithm does not require the use of AI or human intervention to perform its tasks. Also doesn't waste time prioritizing the requests and tasks due to their complexity. Additionally, the party responsible for the scheduling is the CPU itself instead of software or an alternate, more complex job scheduling algorithm.

FCFS algorithm does not prioritize the requests and tasks based on how long it will take to complete them, it does not perform as well when it comes to complex systems that need to handle a wide variety of requests at the same time. Use of the FCFS algorithm risks the possibility that a series of simple requests will get stuck in a central processing unit's queue for an unreasonably long wait time behind a single complex task just because the complex task arrived first.

The FCFS scheduling algorithm is nonprimitive

- Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.
- The FCFS algorithm is thus particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.
- It would be disastrous to allow one process to keep the CPU for an extended period.

Description:

How a FCFS Scheduling Algorithm Works:

- By far the simplest CPU-Scheduling algorithm.
- Process that requests the CPU first is allocated the CPU first.
- The implementation of the FCFS policy is easily managed with a FIFO queue.
- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue.

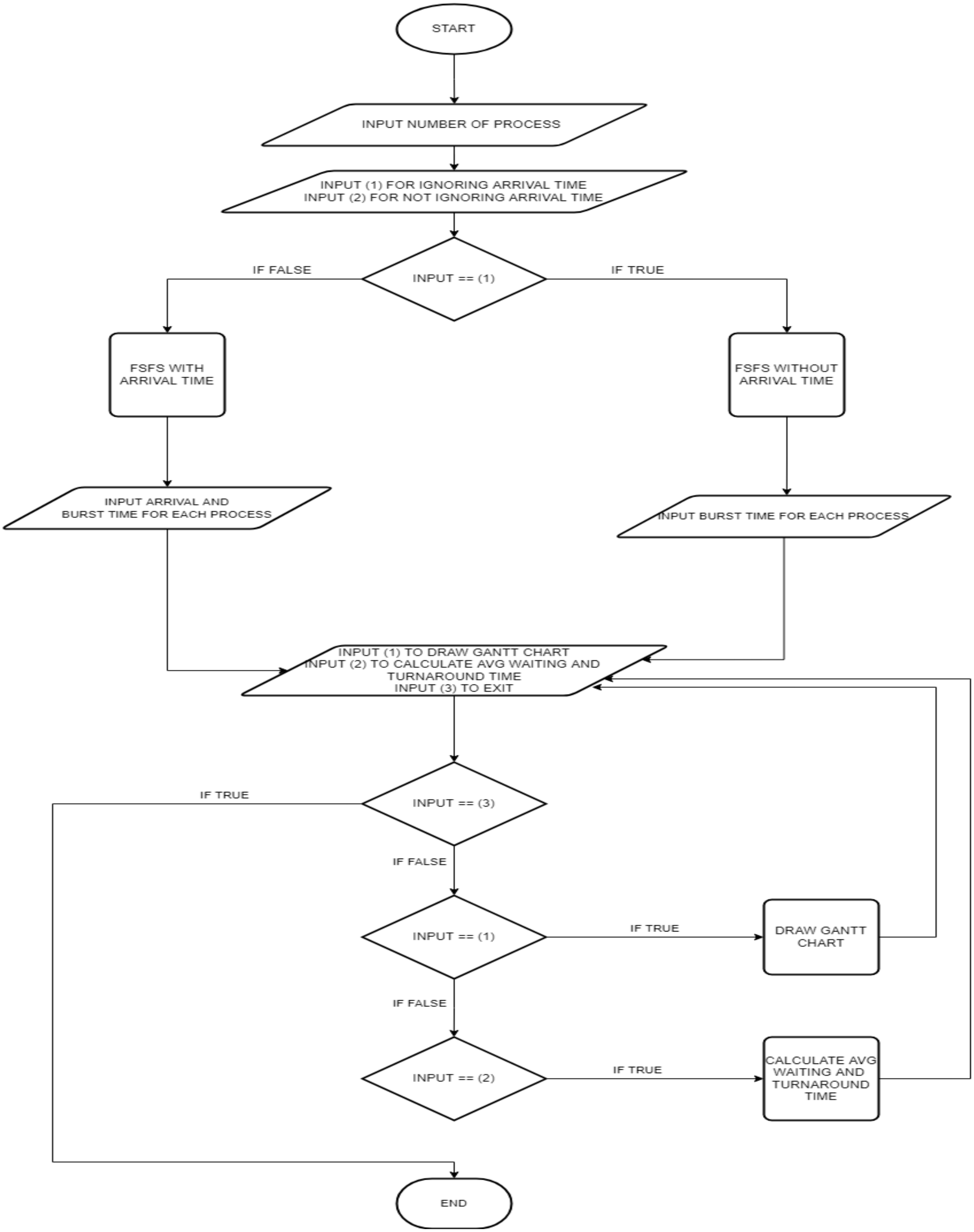
Here is an FCFS scheduling algorithm example. To begin, suppose there are three requests to process in the CPU's queue: P1, P2, and P3. Assume P1 is a complex process that requires approximately 25 seconds, P2 a much simpler request that requires only 10 seconds of processing, and P3 a moderately simple request that requires 15 seconds.

When P1 is first put in the queue, the wait time is zero and the CPU starts the processing immediately. P2, on the other hand, would have a wait time of 25 seconds. And P3 having arrived last, would have to wait 35 seconds. As a total of 25 seconds for P1 and 10 seconds for P2, an FCFS scheduling algorithm would need 50 seconds to complete all three requests and empty the queue, which would be the same as other sequential processing, mono- CPU systems.

Because FCFS does not evaluate requests before starting, it has fewer complete tasks per set period of time when compared to an intelligent scheduling algorithm. In this scenario, the FCFS scheduling algorithm would complete a single task in the first half of its run time of 25 seconds. Other algorithms that start from the simplest of requests, for example would have finished two requests.

In our project we will implement the First Come First Served scheduling technique by creating a JAVA program that will receive a certain number of processes provided from the user and then ask for the arrival and burst time for each process, the program will create a Gantt chart to show the order of processes and calculate the average Waiting Time and average Turn Around Time. We will also usejava swing and interfaces to create a graphical user interface (GUI).

Flowchart:


START
INPUT NUMBER OF PROCESS
INPUT (1) FOR IGNORING ARRIVAL TIME
INPUT (2) FOR NOT IGNORING ARRIVAL TIME
INPUT == (1)
IF FALSE
IF TRUE
FSFS WITH ARRIVAL TIME
FSFS WITHOUT ARRIVAL TIME
INPUT ARRIVAL AND BURST TIME FOR EACH PROCESS
INPUT BURST TIME FOR EACH PROCESS
INPUT (1) TO DRAW GANTT CHART
INPUT (2) TO CALCULATE AVG WAITING AND TURNAROUND TIME
INPUT (3) TO EXIT
INPUT == (3)
IF TRUE
IF FALSE
INPUT == (1)
IF TRUE
DRAW GANTT CHART
IF FALSE
INPUT == (2)
IF TRUE
CALCULATE AVG WAITING AND TURNAROUND TIME
END

Code:

```
/*

PROJECT: First-Come First-Served (FCFS) Scheduling
COURSE: CPIT-260
SECTION: A1
INSTRUCTOR: Dr. Iftikhar Ahmad
STUDENT: Ahmed Essam Taj / Mohammad Hamdan Alsefri / Fahad Hamad AlSifri

 */
package fcfs;

import java.awt.Color;
import java.awt.Graphics;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class FCFS {

    static int minimumArrivalTime, sumCPUBurstTime;
    static int lengthOfEachBlock;
    static final int SCREEN_WIDTH = 700, SCREEN_HEIGHT = 280;
    static final int rectangleUpperPadding = 50, rectangleHeight = 100;
    static int numberOfProcesses;
    static int CPUBurstTime[], arrivalTime[];
    static BufferedReader br;
    static FCFS obj;
    static ImageIcon icon = new ImageIcon("icon.jpg");

    FCFS() {
        this.obj = this;
    }

    public static void main(String[] args) throws NumberFormatException, IOException {

        br = new BufferedReader(new InputStreamReader(System.in));

        Object pNum = JOptionPane.showInputDialog(null, "Enter Number of Process", 
"First-Come First-Served (FCFS) Scheduling", JOptionPane.CANCEL_OPTION, icon, 
null, "");
```

```
        numberOfProcesses = Integer.parseInt((String) pNum);
        CPUBurstTime = new int[numberOfProcesses];
        arrivalTime = new int[numberOfProcesses];
        int ct[] = new int[numberOfProcesses];     // completion times
        int ta[] = new int[numberOfProcesses];     // turn around times
        int wt[] = new int[numberOfProcesses];     // waiting times
        int pid[] = new int[numberOfProcesses];   // process ids
        float avgwt = 0, avgta = 0;
        int option;

        Object userChoice = JOptionPane.showInputDialog(null, "Enter (1) for FCFS with
Ignoring Arrival Time" + "\n" + "Enter (2) for FCFS without Ignoring Arrival Time",
                "First-Come First-Served (FCFS) Scheduling",
                JOptionPane.CANCEL_OPTION, icon, null, "");
        option = Integer.parseInt((String) userChoice);
        switch (option) {
            case 1:
                withoutArrivalTime(ct, ta, wt, pid, avgwt, avgta);
                break;
            case 2:
                withArrivalTime(ct, ta, wt, pid, avgwt, avgta);
                break;

            default:
                JOptionPane.showMessageDialog(null, "Wrong Choice, Try Again");
        }

    }

    public static void withArrivalTime(int ct[], int ta[], int wt[], int pid[], float avgwt, float
avgta) throws NumberFormatException, IOException {
        for (int i = 0; i < numberOfProcesses; i++) {

            JTextField text1 = new JTextField();
            JTextField text2 = new JTextField();
            Object[] fields = {
                "Arrival Time for Process " + (i + 1) + " : ", text1,
                "Burst Time for Process " + (i + 1) + " : ", text2
            };

            int input = JOptionPane.showConfirmDialog(null, fields, "First-Come First-Served
(FCFS) Scheduling", JOptionPane.OK_CANCEL_OPTION,
JOptionPane.QUESTION_MESSAGE, icon);

            arrivalTime[i] = Integer.parseInt(text1.getText());
```

```
            CPUBurstTime[i] = Integer.parseInt(text2.getText());
        }

        while (true) {
            int expression;
            Object choice = JOptionPane.showInputDialog(null, "Enter (1) to Draw the Gantt
Chart" + "\n" + "Enter (2) to calculate the AVG Waiting Time & TurnAround Time" +
"\n" + "Enter (3) to Exit",
                    "First-Come First-Served (FCFS) Scheduling",
JOptionPane.CANCEL_OPTION, icon, null, "");
            expression = Integer.parseInt((String) choice);
            switch (expression) {
                case 1:
                    drawGanttChart();
                    break;
                case 2:
                    calculateWaiting_turnAround_Time(ct, ta, wt, pid, CPUBurstTime,
arrivalTime, numberOfProcesses, avgwt, avgta);
                    break;
                case 3:
                    System.exit(0);
                default:
                    JOptionPane.showMessageDialog(null, "Wrong Choice, Try Again");
            }
        }
    }

    public static void withoutArrivalTime(int ct[], int ta[], int wt[], int pid[], float avgwt, float
avgta) throws NumberFormatException, IOException {

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

            JTextField text2 = new JTextField();
            Object[] fields = {
                "Burst Time for Process " + (i + 1) + " : ", text2
            };

            int input = JOptionPane.showConfirmDialog(null, fields, "First-Come First-Served
(FCFS) Scheduling", JOptionPane.OK_CANCEL_OPTION,
JOptionPane.QUESTION_MESSAGE, icon);

            arrivalTime[i] = 0;
            CPUBurstTime[i] = Integer.parseInt(text2.getText());
        }
```

```
        while (true) {
            int expression;
            Object choice = JOptionPane.showInputDialog(null, "Enter (1) to Draw the Gantt
Chart" + "\n" + "Enter (2) to calculate the AVG Waiting Time & TurnAround Time" +
"\n" + "Enter (3) to Exit",
                    "First-Come First-Served (FCFS) Scheduling",
JOptionPane.CANCEL_OPTION, icon, null, "");
            expression = Integer.parseInt((String) choice);
            switch (expression) {
                case 1:
                    drawGanttChart();
                    break;
                case 2:
                    calculateWaiting_turnAround_Time_without_arrival(ct, ta, wt, pid,
CPUBurstTime, arrivalTime, numberOfProcesses, avgwt, avgta);
                    break;
                case 3:
                    System.exit(0);
                default:
                    JOptionPane.showMessageDialog(null, "Wrong Choice, Try Again");
            }
        }
    }

    public static void drawGanttChart() throws NumberFormatException, IOException {
        //  int choice;
        sumCPUBurstTime = 0;


        /* calculating the sum of all cpu burst time */
        for (int i = 0; i < numberOfProcesses; i++) {
            sumCPUBurstTime += CPUBurstTime[i];
        }

        /* now the total width of the screen is to be divided into sumCPUBurstTime equal parts
*/
        lengthOfEachBlock = SCREEN_WIDTH / sumCPUBurstTime;

        /*
                 * claculating the minimum arrival time
         */
        minimumArrivalTime = Integer.MAX_VALUE;
        for (int i = 0; i < numberOfProcesses; i++) {
            if (minimumArrivalTime > arrivalTime[i]) {
                minimumArrivalTime = arrivalTime[i];
```

```
        }
      }

      drawGanttChartForFCFS();

   }

   public static void calculateWaiting_turnAround_Time_without_arrival(int ct[], int ta[],
int wt[], int pid[], int CPUBurstTime[],
         int arrivalTime[], int numberOfProcesses, float avgwt, float avgta) {

      int temp = 0;

      int total_tt = 0;
      for (int i = 0; i < numberOfProcesses; i++) {
         temp += CPUBurstTime[i];
         total_tt += temp;
      }

      int temp_wait = 0;
      int before = 0;
      int total_wt = 0;
      for (int i = 0; i < numberOfProcesses; i++) {

         if (i == 0) {
            before = CPUBurstTime[i];
            continue;
         }
         temp_wait += before;
         before = CPUBurstTime[i];
         total_wt += temp_wait;
      }
      avgta = total_tt;
      avgwt = total_wt;

      int input = JOptionPane.showConfirmDialog(null, "average waiting time: " + (avgwt /
numberOfProcesses) + " m/s " + "\naverage turnaround time: " + (avgta /
numberOfProcesses) + " m/s ", "Results",
            JOptionPane.CLOSED_OPTION, JOptionPane.QUESTION_MESSAGE, icon);
   }

   public static void calculateWaiting_turnAround_Time(int ct[], int ta[], int wt[], int pid[],
int CPUBurstTime[], int arrivalTime[], int numberOfProcesses, float avgwt, float avgta) {

      int temp;
```

```
        //sorting according to arrival times
        for (int i = 0; i < numberOfProcesses; i++) {
            for (int j = 0; j < numberOfProcesses - (i + 1); j++) {
                if (arrivalTime[j] > arrivalTime[j + 1]) {
                    temp = arrivalTime[j];
                    arrivalTime[j] = arrivalTime[j + 1];
                    arrivalTime[j + 1] = temp;
                    temp = CPUBurstTime[j];
                    CPUBurstTime[j] = CPUBurstTime[j + 1];
                    CPUBurstTime[j + 1] = temp;
                    temp = pid[j];
                    pid[j] = pid[j + 1];
                    pid[j + 1] = temp;
                }
            }
        }
        // finding completion times
        for (int i = 0; i < numberOfProcesses; i++) {
            if (i == 0) {
                ct[i] = arrivalTime[i] + CPUBurstTime[i];
            } else {
                if (arrivalTime[i] > ct[i - 1]) {
                    ct[i] = arrivalTime[i] + CPUBurstTime[i];
                } else {
                    ct[i] = ct[i - 1] + CPUBurstTime[i];
                }
            }
            ta[i] = ct[i] - arrivalTime[i];          // turnaround time= completion time- arrival time
            wt[i] = ta[i] - CPUBurstTime[i];          // waiting time= turnaround time- burst time
            avgwt += wt[i];              // total waiting time
            avgta += ta[i];              // total turnaround time
        }
        int input = JOptionPane.showConfirmDialog(null, "average waiting time: " + (avgwt /
numberOfProcesses) + " m/s " + "\naverage turnaround time: " + (avgta /
numberOfProcesses) + " m/s ", "Results",
                JOptionPane.CLOSED_OPTION, JOptionPane.QUESTION_MESSAGE, icon);

    }

    public static void drawGanttChartForFCFS() {
        FrameForFCFS frameForFCFS = new FrameForFCFS(obj);
    }

    static class FrameForFCFS extends JFrame {
```

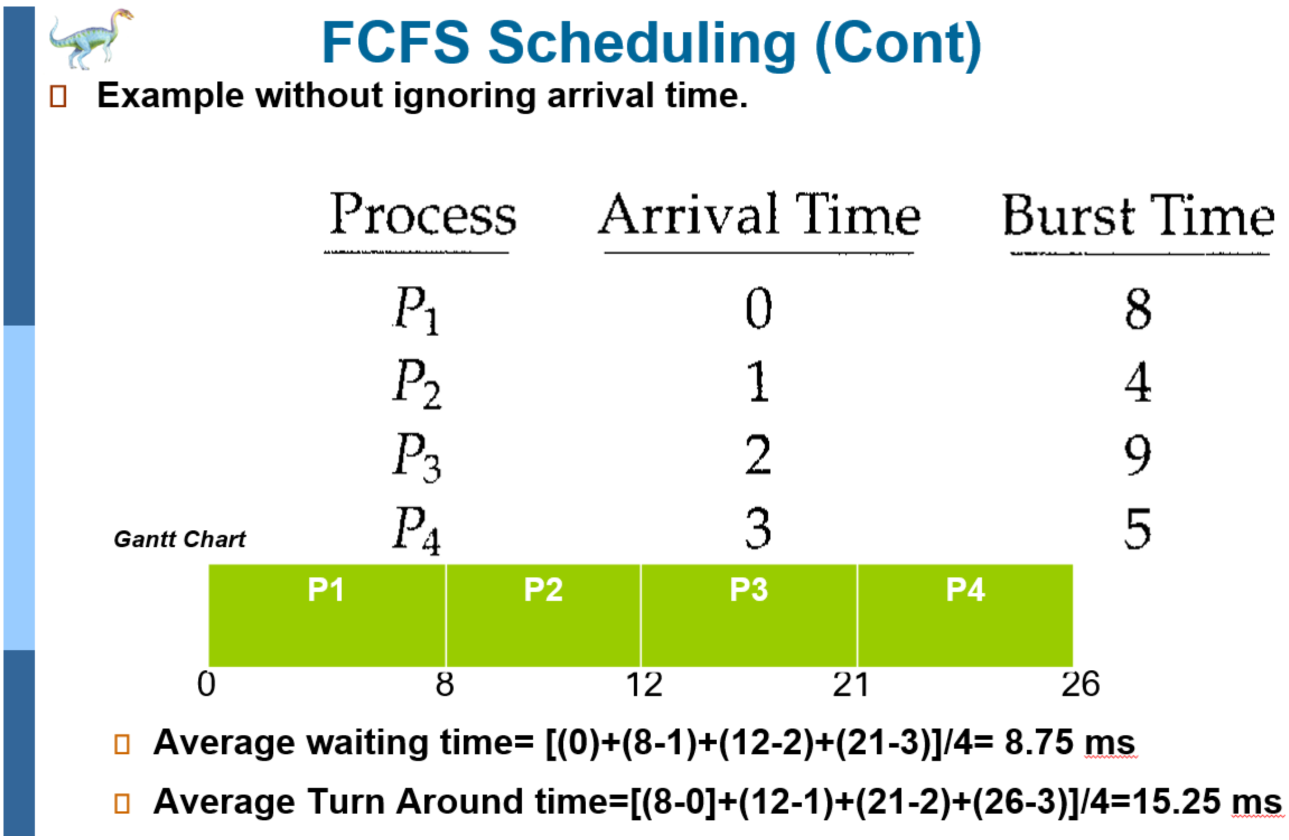
```
        int arrivalTime[];
        FCFS obj;

        FrameForFCFS(FCFS obj) {
            super("First-Come First-Served (FCFS) Scheduling");
            this.obj = obj;
            //this.setResizable(false);
            this.setVisible(true);
            this.setSize(obj.SCREEN_WIDTH + 100, obj.SCREEN_HEIGHT);
            arrivalTime = obj.arrivalTime.clone();
        }

        @Override
        public void paint(Graphics g) {
            super.paint(g);
            this.getContentPane().setBackground(Color.white);

            int currentTime = obj.minimumArrivalTime;
            arrivalTime = obj.arrivalTime.clone();

            int i, j, min, mini = 0;
            int leftStart = 50;
            g = this.getContentPane().getGraphics();
            g.drawString("" + obj.minimumArrivalTime, leftStart, obj.rectangleUpperPadding +
obj.rectangleHeight + 20);
            for (j = 0; j < obj.numberOfProcesses; j++) {
                min = Integer.MAX_VALUE;
                for (i = 0; i < obj.numberOfProcesses; i++) {
                    if (min > arrivalTime[i]) {
                        min = arrivalTime[i];
                        mini = i;
                    }
                }
                arrivalTime[mini] = Integer.MAX_VALUE;

                g = this.getContentPane().getGraphics();
                g.drawRect(leftStart, obj.rectangleUpperPadding, obj.lengthOfEachBlock *
obj.CPUBurstTime[mini], obj.rectangleHeight);
                g.drawString("P" + (mini + 1), leftStart + 5, obj.rectangleUpperPadding + 50);
                leftStart += obj.lengthOfEachBlock * obj.CPUBurstTime[mini];

                currentTime += obj.CPUBurstTime[mini];
                g.drawString("" + currentTime, leftStart, obj.rectangleUpperPadding + obj.rectangleHeight + 20);

            }
        }
    }
}
```

End of code

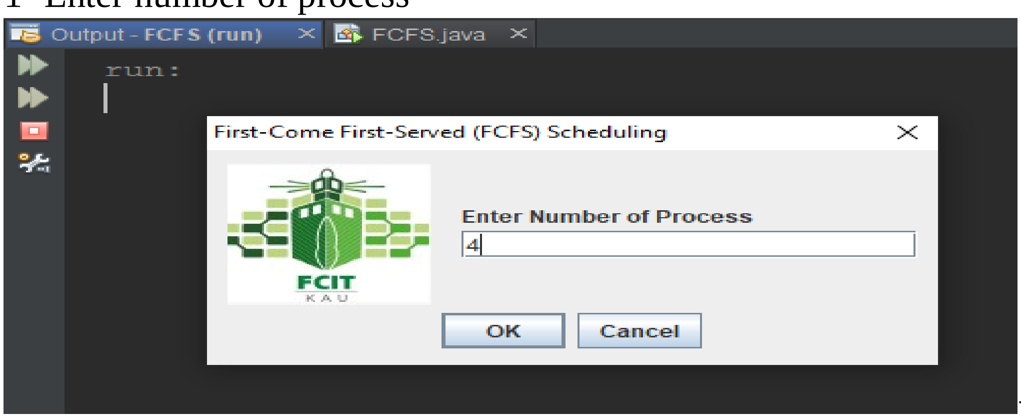
Note: Java project file is attached with this file

Program and Output screenshots:

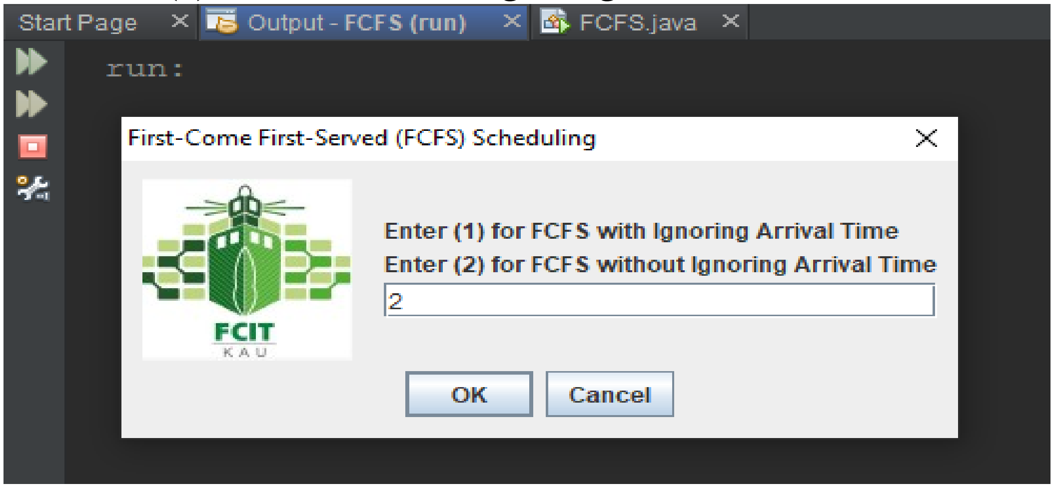
We will use an example from Chapter 5: CPU Scheduling slide 12



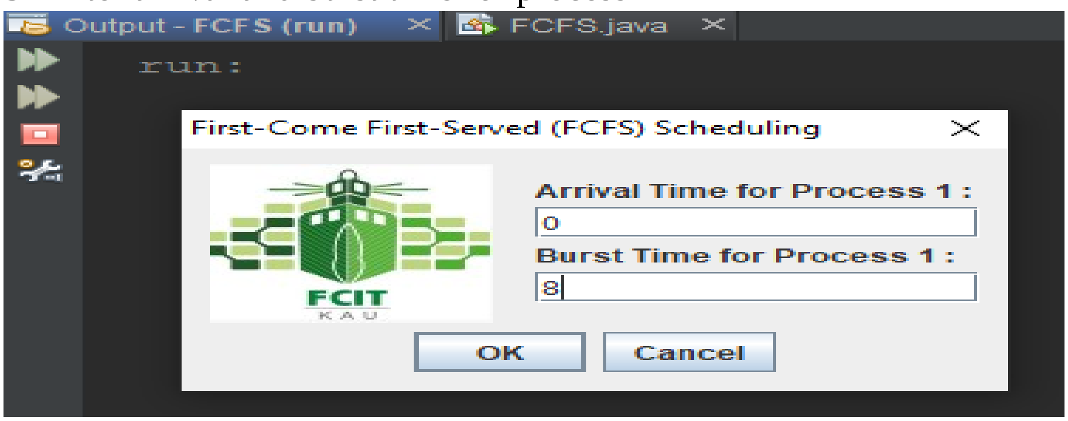
1- Enter number of process



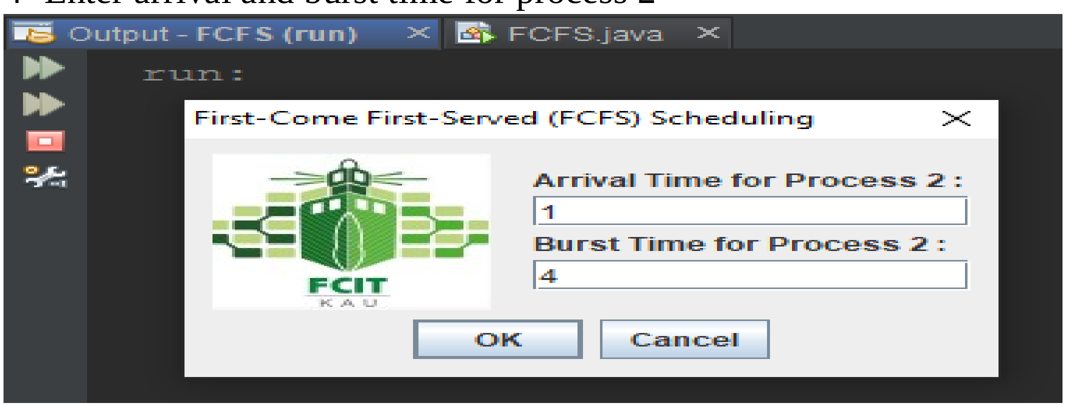
2- Choose (2) for FCFS without ignoring arrival time



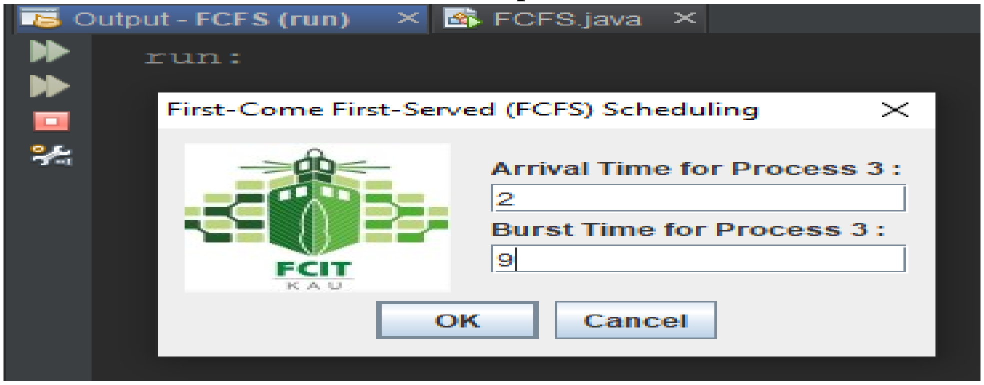
3- Enter arrival and burst time for process 1



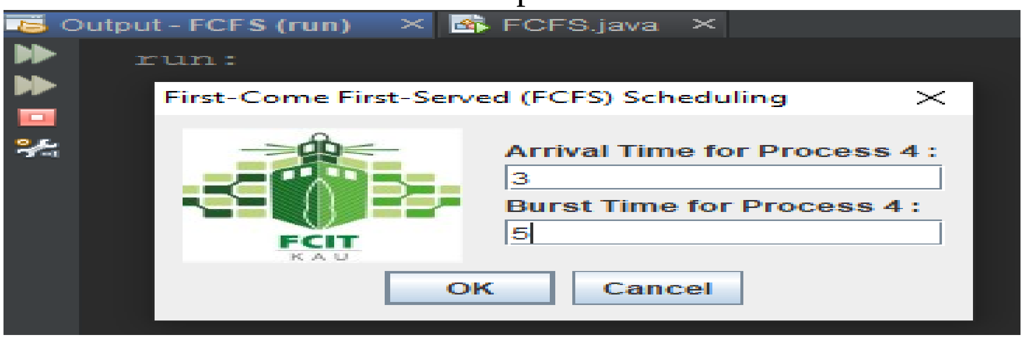
4- Enter arrival and burst time for process 2



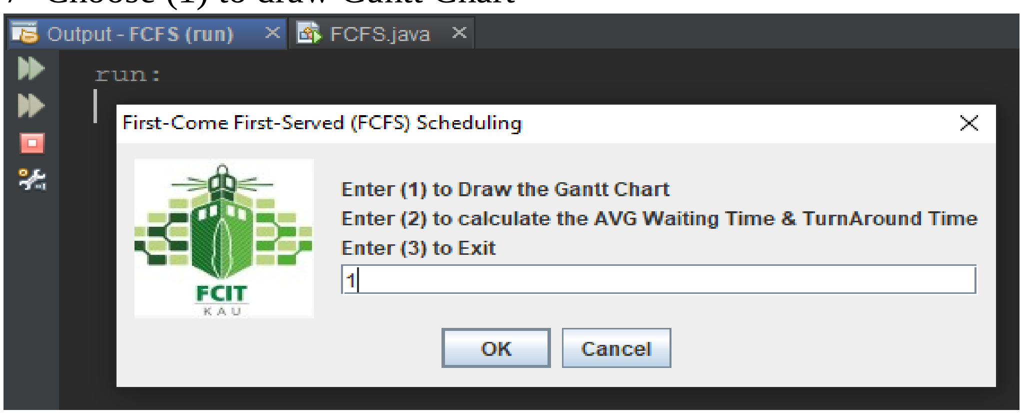
5- Enter arrival and burst time for process 3



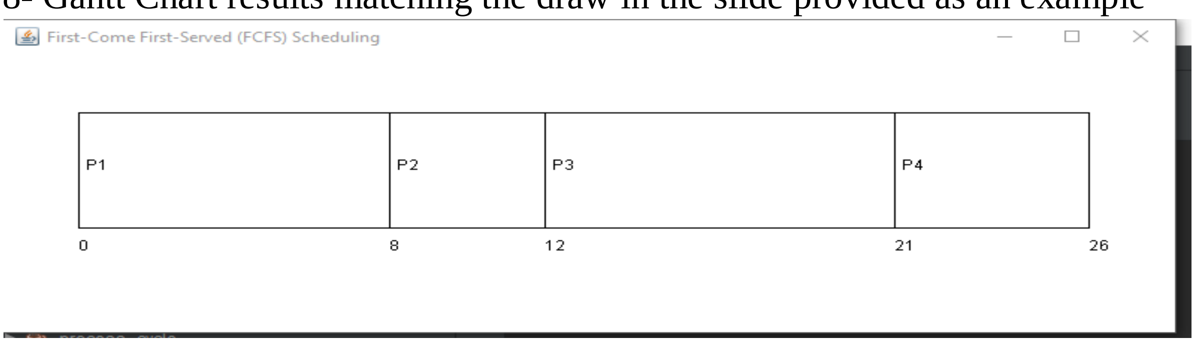
6- Enter arrival and burst time for process 4



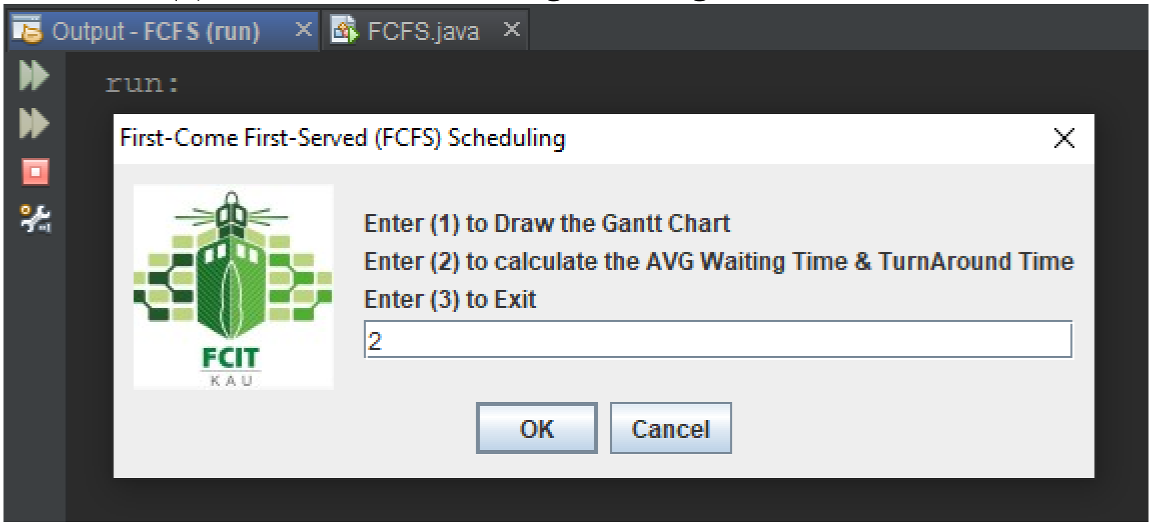
7- Choose (1) to draw Gantt Chart



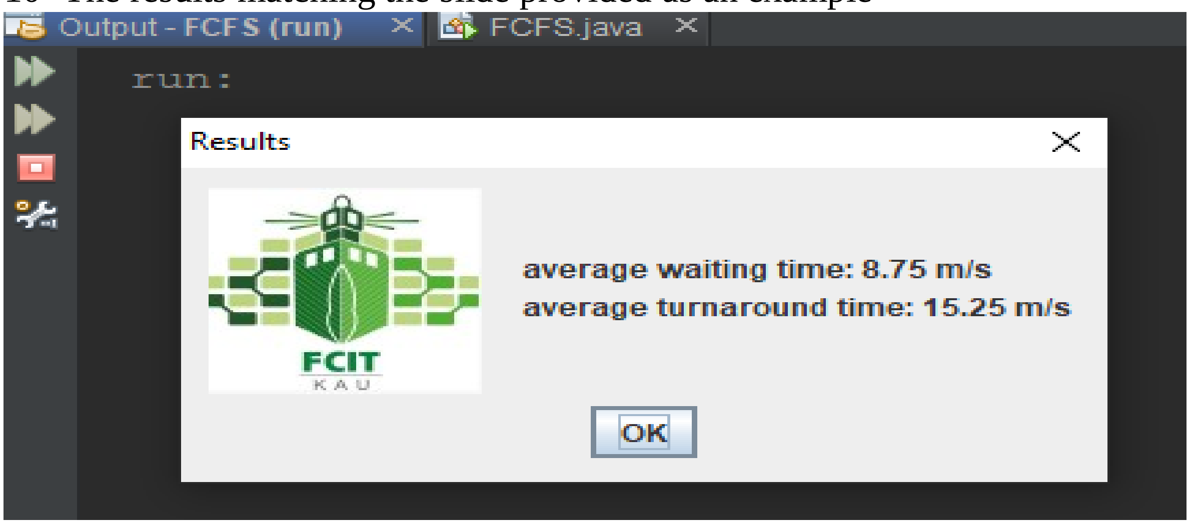
8- Gantt Chart results matching the draw in the slide provided as an example



9- Choose (2) to calculate the Average Waiting Time & Turnaround Time



10- The results matching the slide provided as an example



11- finally Enter (3) to exit the program