# Project Report

# on

# SCHEDULING ALGORITHMS

**In the partial fulfillment of the term work for**

**Course: OSL (Sem-IV)**

**in**

Second Year Computer Engineering


**SUBMITTED BY:**

1. **Varun Singh (64)**
2. **Rohan Talele (67)**

**Under the Guidance of**

Prof. Kirti Motwani

**DEPARTMENT OF COMPUTER ENGINEERING**

**XAVIER INSTITUTE OF ENGINEERING,**

**MAHIM CAUSEWAY, MAHIM,**

**MUMBAI- 400 016**

**2018 - 2019**

# *CERTIFICATE*

*This is to certify that the project entitled "SCHEDULING ALGORITHMS" has been carried out as a part of term work by the team under my guidance in partial fulfillment for the course "OSL" in Second Year Computer Engineering (Sem-IV) of Mumbai University, Maharashtra during the academic year 2018-2019.*

**Team Members:**

1. **Varun Singh (64)**
2. **Rohan Talele (67)**

*Date: 23-10-2018*

*Place: Mumbai*

_____

*Subject In-charge*

(Prof. Kirti Motwani)

# Mini-Project on SCHEDULING ALGORITHMS

**Title:** SCHEDULING ALGORITHMS

**Objective:** The project entitled "CPU SCHEDULING", is basically a program which simulates the following scheduling algorithms:

1. FCFS (First Come First Served)
2. SJF (Shortest Job First)
3. SJF(Preemptive)
4. Priority Scheduling
5. Round-Robin

**Introduction:** The main purpose of this application is to compare and select the best algorithm for a particular set of input pertaining to the processes and in conclusion display the best algorithm. CPU SCHEDULING is a key concept in computer multitasking, multiprocessing operating system and real-time operating system designs. Scheduling refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs. CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. By switching the CPU among processes, the operating system can make the computer more productive. A multiprogramming operating system allows more than one processes to be loaded into the executable memory at a time and for the loaded processes to share the CPU using time-multiplexing.

**Proposed System:**

## System Requirements Specification

**HARDWARE:**

PROCESSOR : ANY INTEL PROCESSOR ABOVE 4$^{TH}$ GENERATION

RAM : 512MB DD RAM

HARD DISK : 25 GB

**SOFTWARE:**

FORNT END :JAVA, IDE NETBEANS 8.2

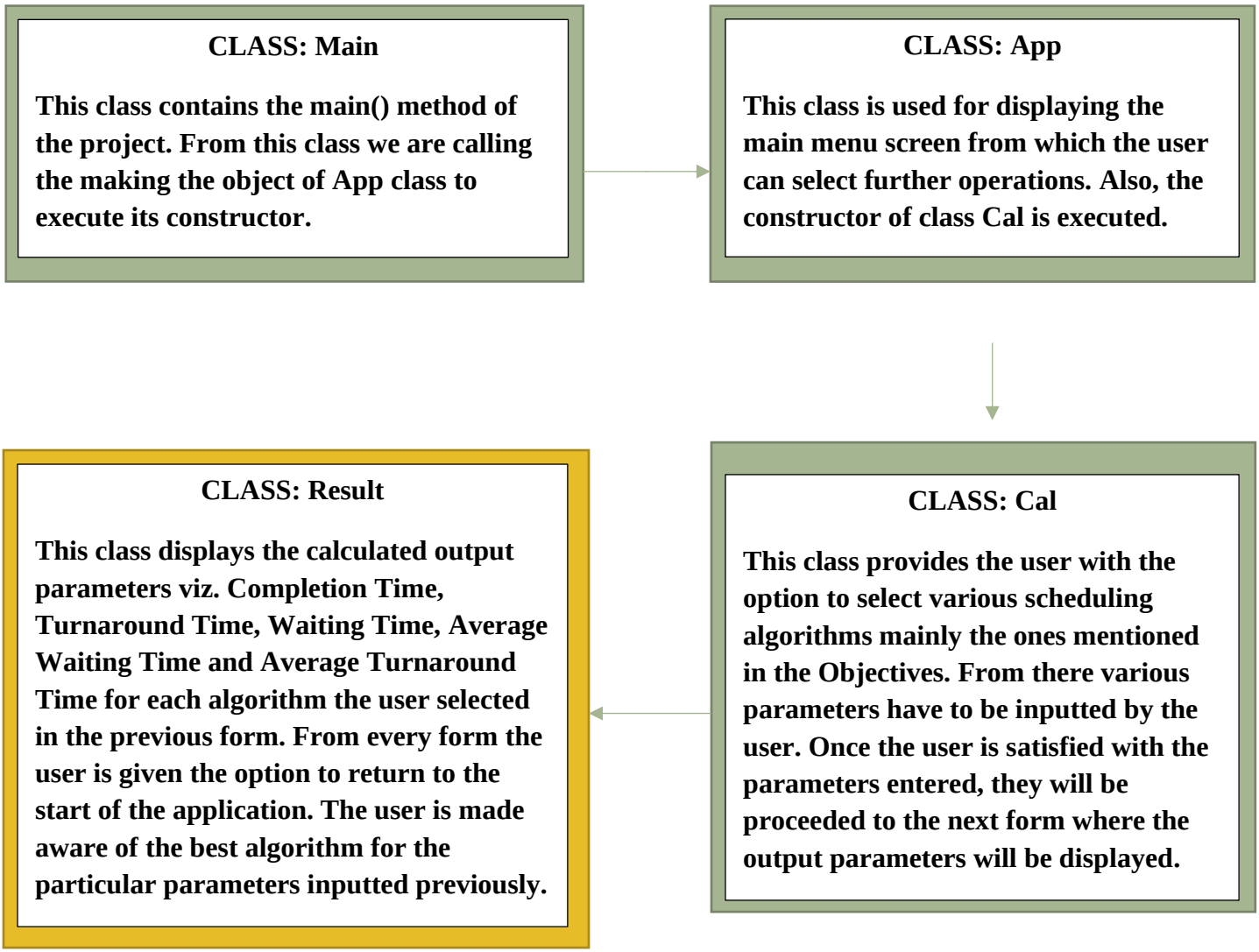
OS : WINDOWS 10

**Block diagram:** LIST OF CLASSES AND THEIR VARIOUS METHODS

**PACKAGE: 1. com.algo**

**CLASSES:**

1. **Main**
2. **App**
3. **Cal**
4. **Result**



**Implementation:**

The various list of methods and their implementation:

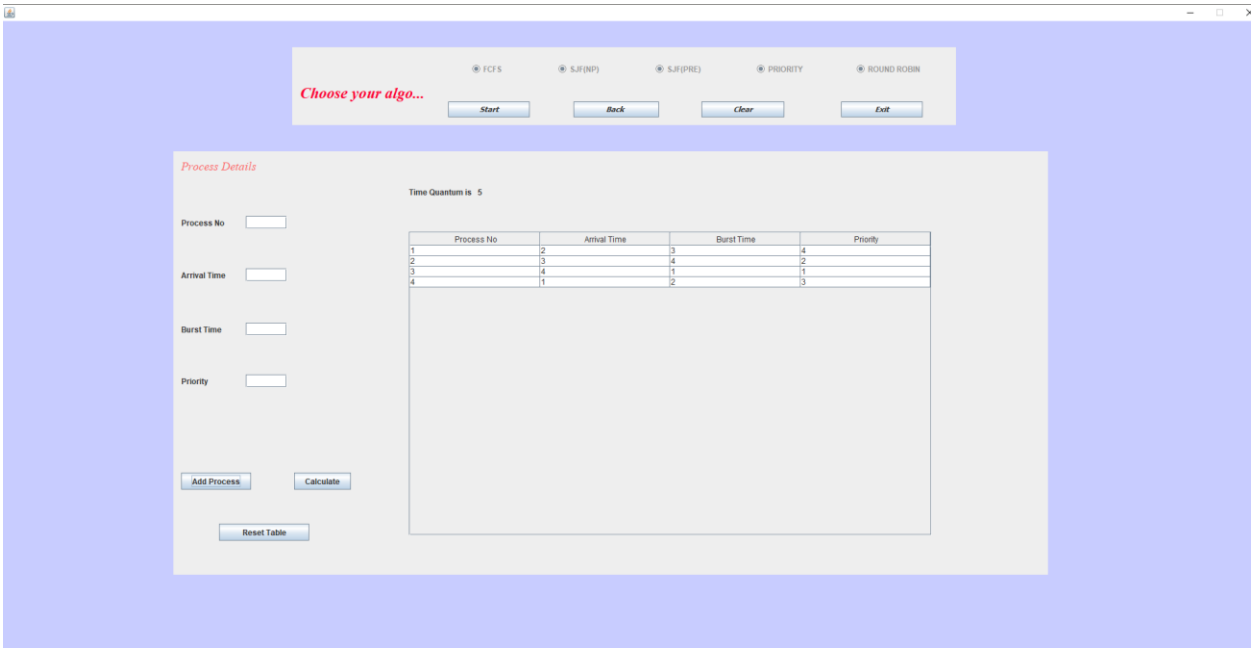
1) **public static void main (String[] args)** – It's the main method used to create the object of class App to execute its constructor.

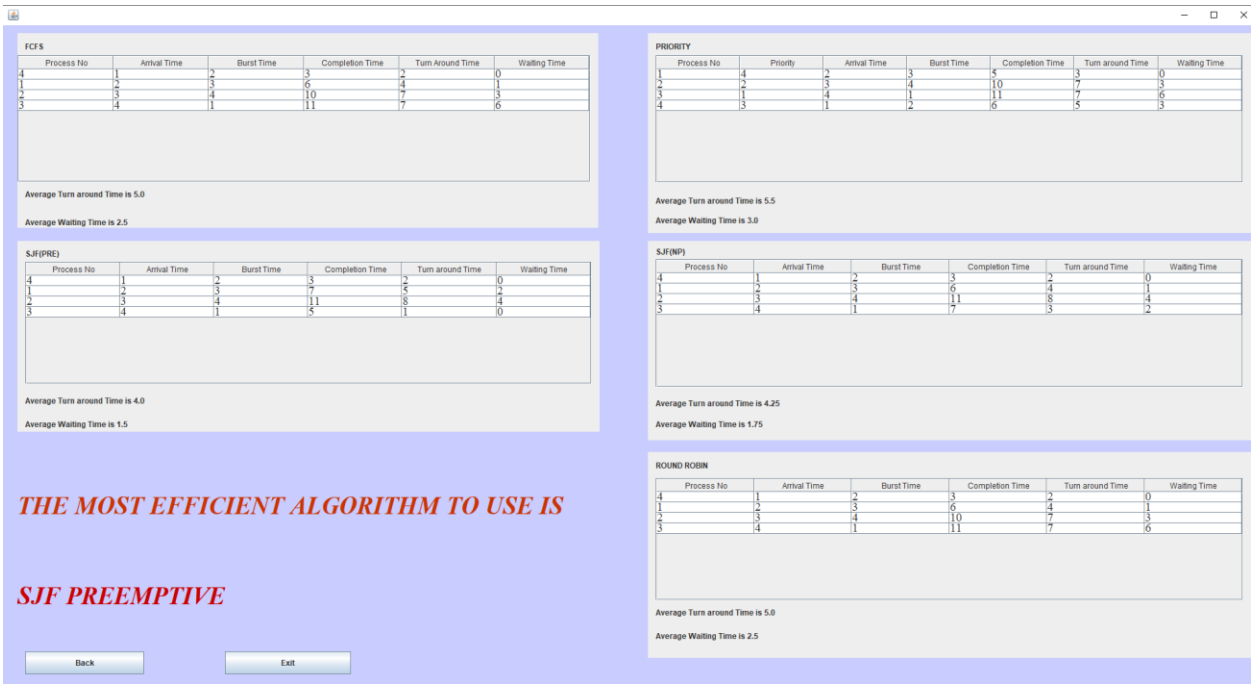
2) **public App() –** This method is used to display the Main Menu Screen and from here the user can choose whether they want proceed further or close the application. To proceed further an object of class Cal is created and its constructor is executed.

3) **public Cal() –** This method is used by the user for choosing the desired scheduling algorithm and inputting the various parameters. These parameters are stored in a table for the user to perceive and they can at any instance change the parameters or completely discard the previously entered parameters and start afresh. After the user is satisfied with the parameters they entered, they can click on Calculate Button to instantiate an object of class Result and proceed to the next form where the results are displayed.

4) **public Result() –** This method is used by the application to display the Average Waiting Time and Average Turnaround Time for each of the algorithms the user previously selected. These parameters are displayed in a table, along with a message declaring which algorithm is the best to use. However due to combination of input parameters entered the Average Waiting Time and Average Turnaround Time for two or more algorithms can be same but only one algorithm will be displayed.

**Sample input/output:**



FCFS
SJF(NP)
SJF(PRE)
PRIORITY
ROUND ROBIN
Choose your algo...
Start
Back
Clear
Exit
Process Details
Time Quantum is 5
Process No
Arrival Time
Burst Time
Priority
Add Process
Calculate
Reset Table
Process No | Arrival Time | Burst Time | Priority
1 | 2 | 3 | 4
2 | 3 | 4 | 2
3 | 4 | 1 | 1
4 | 1 | 2 | 3


FCFS
Process No | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time
4 | 1 | 2 | 3 | 2 | 0
1 | 2 | 3 | 6 | 4 | 1
2 | 3 | 4 | 10 | 7 | 3
3 | 4 | 1 | 11 | 7 | 6
Average Turn around Time is 5.0
Average Waiting Time is 2.5
PRIORITY
Process No | Priority | Arrival Time | Burst Time | Completion Time | Turn around Time | Waiting Time
1 | 4 | 2 | 3 | 5 | 3 | 0
2 | 2 | 3 | 4 | 10 | 7 | 3
3 | 1 | 4 | 1 | 11 | 7 | 6
4 | 3 | 1 | 2 | 6 | 5 | 3
Average Turn around Time is 5.5
Average Waiting Time is 3.0
SJF(PRE)
Process No | Arrival Time | Burst Time | Completion Time | Turn around Time | Waiting Time
4 | 1 | 2 | 3 | 2 | 0
1 | 2 | 3 | 7 | 5 | 2
2 | 3 | 4 | 11 | 8 | 4
3 | 4 | 1 | 5 | 1 | 0
Average Turn around Time is 4.0
Average Waiting Time is 1.5
SJF(NP)
Process No | Arrival Time | Burst Time | Completion Time | Turn around Time | Waiting Time
4 | 1 | 2 | 3 | 2 | 0
1 | 2 | 3 | 6 | 4 | 1
2 | 3 | 4 | 11 | 8 | 4
3 | 4 | 1 | 7 | 3 | 2
Average Turn around Time is 4.25
Average Waiting Time is 1.75
THE MOST EFFICIENT ALGORITHM TO USE IS
SJF PREEMPTIVE
ROUND ROBIN
Process No | Arrival Time | Burst Time | Completion Time | Turn around Time | Waiting Time
4 | 1 | 2 | 3 | 2 | 0
1 | 2 | 3 | 6 | 4 | 1
2 | 3 | 4 | 10 | 7 | 3
3 | 4 | 1 | 11 | 7 | 6
Average Turn around Time is 5.0
Average Waiting Time is 2.5
Back
Exit

**Conclusion:** Since context-switching is a costly necessity, it is important to keep its frequency to as minimum as possible. Malicious code can induce excessive context-switching by generating successive processes with increasing priorities. Linux implements preemptive kernel. The most acclaimed advantage of the preemptive kernel is the protection it affords the OS. For instance, system calls are given high priority to enable them to execute ahead of other tasks. In this way they can reset failure points like race conditions and deadlocks. Systems with preemptive kernels have to contend with context-switching. The rationale behind this study is that since the rates of both preemption and context-switches can be monitored, then either conditions to allow context-switching or preemption can be tied to the levels of the monitored statistics.

**References:**


1) **http://www.javaengineeringprograms.com/round-robin-scheduling-algorithm-program-in-java/**

2) **https://www.geeksforgeeks.org/**

3) **s://www.javatpoint.com/os-round-robin-scheduling-algorithm**