**Project – 1**

**Phase 1: Deadline 10/16/25**

**Scheduling Algorithms**

**Phase 1: First Two Algorithms**

**Objective:** Implement and test **two scheduling algorithms** FCFS and SJF.

**Tasks for Phase 1:**

1. Choose **two algorithms**:

* First-Come, First-Serve (FCFS)
* Shortest Job First (SJF)

1. Implement the chosen algorithms using the **provided starter files**.
2. Read the **input file (schedule.txt)** to get the list of tasks.
3. Generate **output files** showing:

* Process execution order
* Waiting times, turnaround times, and response times

1. Take **screenshots** of the program running with the output.
2. Submit **Phase 1 deliverables** by the Phase 1 deadline.

**Phase 1 Deliverables:**

* Source code for the two algorithms(.java files).
* Report that includes Output files for the two algorithms, full screenshots of output.

**Description:**

This project involves implementing several different process scheduling algorithms. The scheduler will be assigned a predefined set of tasks and will schedule the tasks based on the selected scheduling algorithm. Each task is assigned a priority and CPU burst. The following scheduling algorithms will be implemented:

* First-come, first-served (FCFS), which schedules tasks in the order in which they request the CPU.
* Shortest-job-first (SJF), which schedules tasks in order of the length of the tasks’ next CPU burst.

Priorities range from 1 to 10, where a higher numeric value indicates a higher relative priority. For round-robin scheduling, the length of a time quantum is 10 milliseconds.

1. **Implementation**

The implementation of this project may be completed in either C or Java, and program files supporting both of these languages are provided in the source code download for the text. These supporting files read in the schedule of tasks, insert the tasks into a list, and invoke the scheduler.

The schedule of tasks has the form [task name] [priority] [CPU burst], with the following example format:

A number with black text

AI-generated content may be incorrect.

Thus, task T1 has priority 4 and a CPU burst of 20 milliseconds, and so forth. It is assumed that all tasks arrive at the same time, so your scheduler algorithms do not have to support higher-priority processes preempting processes with lower priorities. In addition, tasks do not have to be placed into a queue or list in any particular order.

There are a few different strategies for organizing the list of tasks, as first presented in Section 5.1.2(refer textbook). One approach is to place all tasks in a single unordered list, where the strategy for task selection depends on the scheduling algorithm. For example, SJF scheduling would search the list to find the task with the shortest next CPU burst. Alternatively, a list could be ordered according to scheduling criteria (that is, by priority). One other strategy involves having a separate queue for each unique priority, as shown in Figure 5.7. These approaches are briefly discussed in Section 5.3.6. It is also worth highlighting that we are using the terms list and queue somewhat interchangeably. However, a queue has very specific FIFO functionality, whereas a list does not have such strict insertion and deletion requirements. You are likely to find the functionality of a general list to be more suitable when completing this project.

1. **C Implementation Details**

The file driver.c reads in the schedule of tasks, inserts each task into a linked list, and invokes the process scheduler by calling the schedule() function. The schedule() function executes each task according to the specified scheduling algorithm. Tasks selected for execution on the CPU are determined by the pickNextTask() function and are executed by invoking the run() function defined in the CPU.c file. AMakefile is used to determine the specific scheduling algorithm that will be invoked by driver. For example, to build the FCFS scheduler, we would enter



and would execute the scheduler (using the schedule of tasks schedule.txt) as follows:



Refer to the README file in the source code download for further details. Before proceeding, be sure to familiarize yourself with the source code provided as well as the Makefile.

1. **Java Implementation Details**

The file Driver.java reads in the schedule of tasks, inserts each task into a Java ArrayList, and invokes the process scheduler by calling the schedule() method. The following interface identifies a generic scheduling algorithm, which the five different scheduling algorithms will implement:

A white background with black text

AI-generated content may be incorrect.

The schedule() method obtains the next task to be run on the CPU by invoking the pickNextTask() method and then executes this Task by calling the static run() method in the CPU.java class.

The program is run as follows:



Refer to the README file in the source code download for further details. Before proceeding, be sure to familiarize yourself with all Java source files provided in the source code download.