CS 3243 – Operating Systems Phase 1 Report

Robbie Diaz, Kevin De Brito, David Butcher



Table of Contents

Introduction 3

System Architecture and Design 3

Driver 3

Loader 3

Long Term Scheduler 3

Short Term Scheduler 3

CPU 3

Disk 3

RAM 4

PCB 4

Process Queue 4

Logger 4

Compilation 4

Execution 4

Group Organization and Project Management 4

Work Division 4

Data Collection and Analysis 5

Results 5

Expected results 5

Unexpected results 5

Figure 1 6

Figure 2 6

Figure 3 7

Figure 4 8

# 

# Introduction

Our project is to design and implement an OS simulator. This simulator will run on a virtual CPU using a custom instruction set provided by the professor. The OS itself is composed of 9 classes and one utility class. These classes are: the CPU, Disk, Driver, Loader, LongTermScheduler, PCB , ProcessQueue, Ram, and ShortTermScheduler. The one utility class is the Logger.

# System Architecture and Design

## Driver

The driver is the main entry point to our simulator. In this class, we initialize all the other main classes, and begin creating processes. The newly created processes are then given to the various schedulers and eventually to the CPU itself. During the different execution phases of each process, statistics are gathered about the process and are recorded for later use.

## Loader

As the first main component called from the Driver class, the Loader is responsible for loading each process from a text file, creating a PCB for that process, filling in the PCB with the process’s information, and handing the completed PCB to the new queue.

## Long Term Scheduler

The Long Term Scheduler (LTS) is called when there are processes that need to be moved from the new queue to the ready queue. When the LTS is first called, it loads as many processes as the ram can hold from the disk. When it runs out of memory (after 15 processes), it stops attempting to load new processes. It is only when the ready queue is empty that the LTS is called again to load more processes into ram.

## Short Term Scheduler

The Short Term Scheduler (STS) is called when a CPU needs a new process to run. At the moment, the STS only moves the head of the ready queue to the tail of the running queue. This is by design, because that is what the project specifications have asked us to do. In a more advanced OS, the STS would have logic to assign processes based on priority and other factors.

## CPU

As the core of the OS, the CPU is responsible for executing all of the instructions for a given process. The CPU has the logic to decode instructions, modify internal registers, and manipulate the contents of Ram. A simplistic version of the CPU’s main run loop is as follows: load an instruction, break the instruction into its core components, execute the proper instruction based on the components, and repeat.

## Disk

The Disk class acts as the hard drive for the OS. When the simulator is first started, the Loader reads instructions and data into the Disk class’s internal array.

## RAM

The RAM class provides fast access read/write memory for the CPU. The Long Term Scheduler is responsible for loading the contents of the Disk into the RAM during setup.

## PCB

The main data structure in the OS. This class contains all the information about a process: its id, instruction location on disk and ram, the number of cycles it has run, etc. Without this class, there would be no way of representing a process to the OS.

## Process Queue

This is a wrapper class for an ArrayList of PCBs. There’s nothing more to it.

## Logger

This utility class is used as a convenience class to make logging simpler. It allows the OS to output its log messages to either a log file or to the console.

For a diagram of the project, please see Figure 4 at the bottom of this document.

# Compilation

There are two ways to compile the project. If you have Eclipse installed, then you can open the attached source code as a Java project. The project will have everything set up, except for passing in a data file, which isn’t hard. In the run configurations for the project, you have to specify an argument that points to the correct data file. Then hit run, and watch it go.

If you don’t have Eclipse installed, or just prefer to run it from a command line, then compile the classes with javac like you would any other java class. The main class is the edu.spsu.cs3243.Driver class. This class requires that a file path be given to it as an argument. This file path is the path to the data file that contains the processes we will be loading.

# Execution

To see the results of the OS simulator, you must simply run the simulator once. Depending on how the Logger class is configured, the output will either be put on the console, or in a file called “output.log”. This file is a simple text file, so Notepad or TextEdit will open it without problems.

# Group Organization and Project Management

## Work Division

We divided the work up as follows: Robbie worked on the Short Term Scheduler and CPU, Kevin worked on the Long Term Scheduler, RAM and Disk, David worked on the PCB and Loader. The Driver was a collaborative effort. Due to the different programming styles we had some trouble piecing everything together, but we did manage to get them working in tandem.

# Data Collection and Analysis

To collect the data from the simulator, we added some logging that printed out certain counters at specific times, and dumps the process’s information at the end of the simulation. We used these pieces of data to generate the graphs below. There is even more detailed information in the output.log file, which is obtained by running the simulation. At the bottom of the log file, there is a dump of all the processes in the terminated queue. All of the PCB’s information is in that print out. There are also a few lines that show the total time taken to run the simulation. The reason those lines are not in this document is because the lines are too long, and would word wrap, causing major readability issues and taking up a lot of space.

# Results

## Expected results

We expected that the processes would run and would complete successfully according the stated expectations in the project description. We also expected that the processes wouldn’t get stuck in any infinite loops during executing, which none did. Another expected result was that the process wait time would be horrible. This is solely because we are executing the processes in their entirety, instead of time slicing or some other technique. In the 2nd phase of the project, this result will disappear.

## Unexpected results

One unexpected result was that the RAM would fill up completely after loading the 15th process. The problem was fixed by clearing the contents of RAM after the 15th process finished executing. This could have been avoided if the RAM had more space, but the project specification was extremely precise on the amount of RAM we could have.

Figure 1

Figure 2

Figure 3

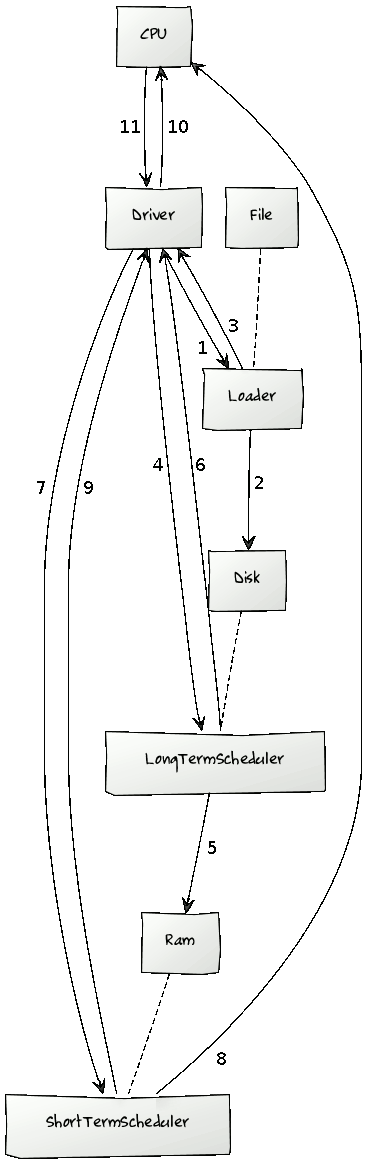


Figure 4