Abdullah Ehsan,

Leeford Nduru, and Alan Castle

simulOS

CS 3502

with Dr. Patrick Bobbie

Table of Contents

[Abstract & Intro 2](#_Toc35467795)

[Design Approach 2](#_Toc35467796)

[Implementation 2](#_Toc35467797)

[Simulation 3](#_Toc35467798)

[Data 6](#_Toc35467799)

[Summary 7](#_Toc35467800)

# Abstract and Introduction

This report covers the semester course project of the Operating System simulator completed as per the given specification. This report covers the major as well as the auxiliary modules of the virtual machine in detail under the following major headings: the design, implementation, simulation, data and results.

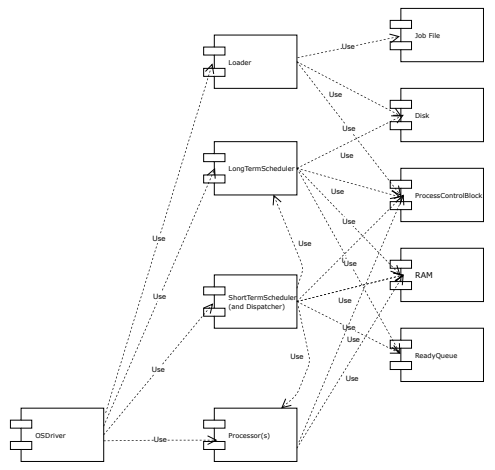
# Design Approach

The group decided to implement the operating system simulation in Java programming language. Initially C++ was considered more suitable choice for being a low-level language allowing pointers and efficient memory management but the ease, cross platform portability offered by Java and level of experience of the team favored Java.

The design architecture is shown in UML diagram in Figure 1. Each of the module in the diagram represents a hardware or a logical component of the virtual Operating System. These modules are organized in the three layers such that each layer generally uses its lower layer, with some exceptions. In the first layer there exist an OS Driver. OS Driver calls the Loader, Long Term Scheduler, Short Term Scheduler, and the Processor(s) from the underlying second layer. In the second layer there is some inter level use: Short Term Scheduler uses the Long Term Scheduler when it runs under 5 job and the Processor to dispatch the jobs. At the third layer, there is Job File, Disk, RAM, Process Control Block and Ready Queue which are called by the second layer modules.

# Implementation

The Driver is the component from where simulation of the virtual machine kicks off. It is implemented in the class OSDriver.java. It calls the loader, the long-term and short-term schedulers and the processors as discussed in detail below.

The Disk component is implemented in the Disk.java class as a virtual hard disk. It is an array of array of type int of fixed size 2048 words, with each word of 32 bits, accommodating 8 hex characters converted and stored as 0s and 1s at any one Figure 1: Design Approach - Architecture Diagram

time. There are two methods to writes data to given address, converting hex string to int array of bits and reads data from given address, returns int array of bits. Disk usage is tracked using Boolean variable and there is a method that gets chunk of free space in disk.

The Loader component is implemented in Loader.java that is called in OSDriver.java file. The Loader constructor requires Job File to read jobs from process list. The load() method reads lines from job file, strips control characters and stores the job on the Disk and adds the job in Process Control Block component with job’s priority, id, length, its address in file, input and output buffer address.

The Long Term Scheduler module is implemented in class LongTermScheduler.java. This class is instantiated in the OSDriver.java class by passing its constructor parameters of Disk, RAM, Process list of Process Control Block, and Ready Queue ArrayList of processes. Then its method schedule() is called that in turn calls the specific scheduling algorithm methods like fcfsScheduler(). The scheduling algorithm, fcfsScheduler(), adds each job from the disk into the ram and the ReadyQueue in the order of arrival; storing the job’s file address, input and output buffer address and its entry time and removes this job from Process Control Block.

The Short Term Scheduler module is implemented by a class called ShortTermScheduler.java. This is instantiated in the OSDriver.java, passing its constructor parameters of RAM, Processor, ReadyQueue ArrayList and LongTermScheduler. Short Term Scheduler also implements the Dispatcher module that dispatches the jobs from the Ready Queue to Processor according the specific scheduling algorithm. It also does context switching by using data from Processor and updates the Process Control Block by calling its method schedule(). Short Term Scheduler implements method dispatch(ProcessControlBlock pcb) that calls Long Term Scheduler also if the Ready queue falls under 5 jobs. This way the dispatcher always has a job to dispatch to Processor and keep Processor utilization high.

Process Control Block has been implemented in ProcessControlBlock.java. It keeps track of the state of every process as they are being accessed by Loader, LongTermScheduler, ShortTermScheduler and Processor modules. These statistics include jobID, jobLength, jobPriority, jobAddress; inputBufferSize, outputBufferSize, tempBufferSize, totalBufferSize, bufferAddress; startTime, endWaitTime, jobCompletedTime on Processor.

The RAM module simulates the main memory implemented via the RAM.java class. It is an array of size 1024 of an array of size 32 bits word. The constructor RAM() initializes a new memory data[][] each time the simulator is run and each indexed position in used[RAMSIZE] is set to false. This module has following methods: void write(int address, String data) writes data to given address, converting hex string to int array of bits. Likewise, int[] read(int address) reads data from given address, returns int array of bits. Next, String readAsString(int address) reads data from given address, returns string of hex. And, boolean isFull() checks whether RAM is full. Also, int[] getFreeMemory() returns largest chunk of memory. It searches in memory and returns the largest chunk of free space found. Finally, freeSpace(int beginningAddress, int length) frees up space in RAM and updates 'used' array to mark that it is empty and available.

The processor is implemented in the class CPU.java. It is instantiated in the OSDrive.java class. The Processor has an array of array for 16 register , each 32 bit long and an Arraylist<int[]> for the Cache component. Its constructor grabs the instances of the RAM ram and continues fetching jobs into cache to decode and execute by DecodeAndExecute() method. The implementation uses the following methods: void assignPCB(ProcessControlBlock p) accepts pcb from dispatcher, void fetch() fetches next instruction of a certain job i.e. process, void decodeAndExecute() decodes and executes instruction, void decodeAndExecuteArithmetic(String opCode) decodes and executes arithmetic instructions, decodeAndExecuteConditionalBranchAndImmediate(String opCode) decodes and executes conditional branch and immediate instructions, decodeAndExecuteUnconditionalJump(String opCode) decodes and executes unconditional jump instructions, decodeAndExecuteIO(String opCode) decodes and executes I/O instructions, void getStats() gets the stats like job ID, IO Count, process wait time, process completion time.

Next, The Processor threading has implemented to run the four processors and and isAvailable flag is used to prevent the deadlock.

Also, some helper classes have been used for example Conversion from one type of data to other. The implementation runs through CLI.

# Simulation

The flow and behavior of the simulator has been discussed in the implementation as well, that lead to the successful completion of the simulation execution. The statistics have been collected and produced under data heading. The complete code and program modules implementation have been placed in the Dropbox.

A screenshot of a computer

Description automatically generated

Figure 2: Screenshot of one CPU simulation

A screenshot of a computer

Description automatically generated

Figure 3: Screenshot of Four CPU simulation

# Implementation Data and Results

Every time simulation run provides different outputs for the jobs. Mostly, output with same datafile is similar except completion time, wait time, average wait time, and average completion time. The results have been obtained using FCFS algorithm. The following data has been provided from running the simulation. It displays the average values of all the four metrics across all jobs: average wait time, average completion time, average IO request, average ram usage, average cache usage.

The following charts provide comparison between one and four processors from the simulation.

#### Graph1: Four Processor vs One processor - Wait Time

#### Graph 2 Four Processor vs One processor - Completion Time

#### Table 1: Four Processor vs One processor comparing data – completion time, waiting time

|  |  |  |
| --- | --- | --- |
|  | 1 CPU | 4 CPU |
| Average Completion Time |  |  |
| Average IO request |  |  |
| Average Wait Time |  |  |
| Average Ram Usage |  |  |
| Average Cache Usage |  |  |

Please see output.pdf attached for results from one processor and 4 processor results.

# 

# Demonstration

The video has been submitted.

# Summary

Implementing this virtual machine, has enhanced our understanding of different modules and components required in the design and implementation of the operating system. It also exposed the challenges and level of complexity in the design of a real operating system that makes applications run on actual machines. The multiple simulation runs show that the average completion time and average wait time vary for every run as it depends on number of factors including the machine you run on as it has multi-processor and is loaded differently at different times.