Ausawin Saehaan
V00797462

Repository: https://github.com/saehaana/cmsc312

commit id: 315188b1b772e4f1776bfe859e2d425890681182

Summary

This is a semester long project where students are to simulate an operating system by means of a program. Done in any language of choosing, the operating system simulator was worked on in three phases and consisted of four main components.

Phase 1 - Process Management: Handling of all resource allocation and scheduling; anything related to CPU and its processes.

Phase 2 - Memory Management: Handling of all memory allocation and deallocation associated with process operations.

Phase 3 - Bonus components and GUI: GUI will be implementation of a front end system using javaFX rather than console based print statements used in phase 1 and 2.

How to Run

The OS simulator was created using IntelliJ IDE and as of right now there is no executable. So the only way to run the program is to **save all files to a folder and execute Main.java from within the IDE**. One thing to note is the inclusion of the BufferedReader class to read text from specified templates/program files. **Your version of the OS simulator will not run correctly if you do not change the file paths of the templates to your download's directory**. If you do not change the file path directory, BufferedReader will not be able to read correctly and therefore processes and their associated operations cannot be properly set.

Once OS simulator has been launched, terminal UI will appear indicating to enter numbers and keywords corresponding to certain functions of the program.

**view readme within repo if additional instructions needed

Implemented Solutions

To simulate an operating system and its components, the idea of separate java files working together was an ideal approach. Code was encapsulated into separate classes and represented by names such as PCB, Processor, Scheduler, etc. Each class had their own setter and getter methods which was used to manipulate their class's object(s) and return certain I/O.

Ausawin Saehaan
V00797462

Components

Main: Contains a temporary UI as well as other class objects and lists used to navigate and create functionality of the program. The main portion of code in this class is contained within a while loop to simulate the OS being turned on and running and uses a switch statement with multiple cases for the user to navigate parts of the OS simulation.

Process: Process component and all the following components below will be manipulated by objects of their class type. As an example, object named process of Process type is created in Main.java and is used to call methods from Process.java. The methods called from Process.java will be used to manipulate the process object as a simulation of what would happen to a real process and processor in an actual operating system environment. When process is created from template, process object will give process operations random cycle lengths decided by method getRandomNumber().

PCB: Will retrieve and store all information pertaining to OS components using getter and setter methods; mainly consisting of information about each process e.g., process state, pid, CPU registers, etc

Scheduler: The first scheduling algorithm used is the Round Robin (RR) approach. This simulator's RR contains a time quantum of q (equal to ~75% greater than most CPU bursts). Once the first process enters the queue, the scheduler will execute the process for one time quantum. If there are other processes in the queue, then the current process being executed will be preempted and the next process in queue will execute for one time quantum. If no other processes exist but the first in queue, then more time quanta will be given for the process to complete its execution and terminate or return to ready state.

Critical Section: Will also include critical section resolving scheme. This component ensures that shared resources are to be accessed by one process at a time. If one process is executing, then others have to wait to execute in their own critical sections. Indicated by keywords “CRIT_START” and “CRIT_END” within all templates, crit_start will tell program process to acquire semaphore (imitation lock) and simulate manipulating a shared resource by incrementing Process.resource, with crit_end releasing the semaphore. Current implementation of critical section is incomplete as multithreading has not been introduced into the OS simulator until phase 2 begins.

Software Requirement(s): Latest version of IntelliJ IDE

Hardware Requirement(s): Working computer