CMSC 312 Project Description

Operating system simulator

Bartosz Krawczyk, Ph.D.
Department of Computer Science
College of Engineering
Virginia Commonwealth University

1. Summary

This project requires to design and implement a simulation of an operating system. Students are expected to prepare a computer program that will serve as a simulator of loading and execution of programs on a theoretical operating system. These programs will originate from job files created by the programmers. Jobs or simulated programs may consist of a dummy software like word processor, web browser, virus scan software, media player and photo editing software. **These programs need not to be coded, only simulated**. These job files will have simulated instruction cycles and I/O that is proportional to the type and size of program that it is. **Simulator must be able to both take user-defined input files, as well as create a requested number of job files randomly using pre-defined generator.**

Classes are to be created in order to mimic the role of different hardware components and software embedded in the operating system. These classes will have methods in them to perform operating system duties and will rely or call on other classes or methods, thus creating a fully interconnected system.

2. Project Description

2.1 Requirements

The project requires that the hardware consist of one central processing unit and 2048 MB of memory. Furthermore, the simulator shall have I/O operations taking between 25 and 50 cycles. The simulator runs in steps defined as “looped” cycles.

2.2 Features

A command line embedded within GUI is required, which prompts the user for input. The user will initiate the start of a program with a command line in the simulator. User will specify the number of cycles the program can run for testing & observational purposes.

2.3 Basic Operations

Four basic operations are required in this project program. They are calculation (processing), I/O simulation, yielding (interrupts), and output. Job files will contain the random sequence of these operations or program instructions to simulate activity of a program. Each job or simulated program in the text file will be like a “script”; text lines read one after another by the simulator. These operations or instructions will be stored into an array of strings along with any parameters that follow them. The number of cycles represents a “run-time” or the length of all the instructions and math operations to be simulated for that particular program.

2.4 User Interface

This program/simulator will incorporate a user interface so that he/she can control the flow of the operating system and observe the "running" of it for testing purposes. It must be possible to automatically and manually load a program or job file into the simulator thus to conduct the allocation of the program's PCB and memory space. The exe command will let the simulator run on its own. The user can also specify the number of cycles to run before pausing.

2.5 Graphical User Interface

The GUI will display real-time statistics, visualizations and data on all currently running processes. The PCB information of the jobs that are in memory and in the run state will be shown in some sort of GUI data table. This GUI table is always "refreshing" or updating as the simulator runs on its own or steps through the code.

2.6 Job and Program Files

"Job files" are text files that open and load the mock programs found in "program files". The lines in a job file will be LOAD commands specifying the exact cycle time a process (in the program files) is to be loaded and put into the NEW queue or state. Also, the job file contains the string name of the process so it can be identified to the user. The EXE command is the last line in each job file script and will cause the simulator to load all other lines and information, storing it into the proper fields of the PCB.

Once loaded, the first line in a program file will input the process' memory requirement (possibly an INT value). The following lines will be the "script" of operations: CALCULATE, I/O, YIELD, and OUT:

- CALCULATE – When this is read in, the simulator will run the process in the run state for the number of cycles specified as a parameter
- I/O – The simulator reads in this command, but a random number generator creates the parameter value in the main code or simulator (likely an INT value). This will put the process in the blocked state.
- YIELD – This command yields from or pauses the running process, halts its accumulating cycle time and gives priority to another process (possibly through the use of round-robin scheduling). These can act as random interrupts in the system.
- OUT – This will print out a message to the screen, possibly to indicate which process is running and all of its PCB information. This is similar to the user entered PROC command, but is generated by the system.

Each program must contain at least one critical section and at least a single critical section resolving scheme must be implemented.

Additionally, for grade B and higher it is required for a at least single interprocess communication scheme must be implemented. Single level child-parent relationship must be used by processes.

Additionally for grade A both a message passing scheme must be implemented. Multi-level child-parent relationship must be used by processes.

2.7 Scheduling Algorithm

Project team must choose and implement a scheduling algorithm. Easiest solution would be to go with Round Robin approach due to the ease of implementation, however other solutions (with exception of a trivial First Come First Served) also will be accepted.

Additionally, for grade B and higher it is required to implement at least two different schedulers must and compare them with regard to their performance. Processes must have assigned priorities.

Additionally, for grade A multi-level queue scheduling and, process resources, and deadlock avoidance algorithms must be added.

2.8 Process States

The state implementation of our operating system simulator is as follows:

- NEW – The program or process is being created or loaded (but not yet in memory).
- READY – The program is loaded into memory and is waiting to run on the CPU.
- RUN – Instructions are being executed (or simulated).
- WAIT (BLOCKED) – The program is waiting for some event to occur (such as an I/O completion).
- EXIT – The program has finished execution on the CPU (all instructions and I/O complete) and leaves memory.

2.9 Memory Management

Memory Management will be implemented by keeping a running total of all processes in main memory by not letting the overall memory size exceed the set limit. This value which is to be compared against the memory requirements of newly arrived processes. If total memory minus used memory is more than the newly arrived job's memory requirement, it may enter the READY state (queue). Otherwise, the process would be entered into a waiting queue.

Additionally, for grade B and higher memory must be divided into main memory, storage, caches and registers. Paging is to be implemented.

Additionally for grade A virtual memory with paging is to be implemented.

2.10 I/O

Two types of I/O events must be **generated – caused by external events (e.g., hardware peripherals) that may happen at any moment and caused by processes being executed by CPU.** They must be handled accordingly.

2.11 Multi-threading and multi-CPU

System must allow for concurrent execution of at least 4 threads (simulating multi-threading).

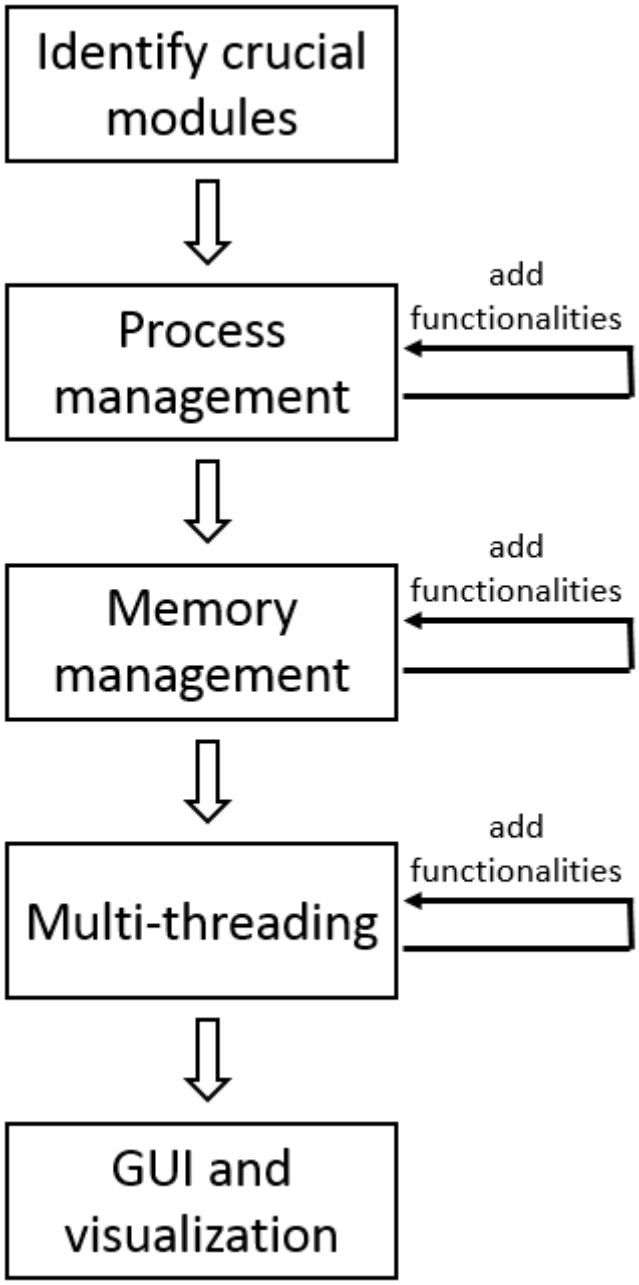
Additionally, for grade B and higher this must be implemented using real multi-threading, not single-thread based simulation.

Additionally, for grade A simulation of multi-core and multi-thread architecture with at least two cores and four threads per each must be implemented.

3. General rules for the project:

- This is an **open-ended project** and the design skills, capability of adapting to specifications where necessary, as well as imagination and creativity of solving the required tasks play a crucial role.
- Project must be fully functional and the code of project must compile and run.
- Project must be a compound system of modules working together, not a collection of non-integrated parts.
- Project must be done individually, not code sharing or copying / modifying is allowed.
- Students are allowed to discuss and consult theoretical solutions and approaches among themselves.
- Source code must be delivered for grading, as well as an executable version of the project.
- Student is obliged to deliver a .pdf documentation of the project, discussing the implemented solutions, software and hardware requirements for the project to run, as well as guidelines on how to run the project.

Task	D/C grade	B grade	A grade
Process implementation and PCB	✓	✓	✓
Critical section within each process	✓	✓	✓
Critical section resolving scheme	✓	✓	✓
Single interprocess communication method		✓	✓
Single level parent-child relationship		✓	✓
Two interprocess communication methods			✓
Multi-level parent-child relationship			✓
Scheduler	✓	✓	✓
Two schedulers and comparison		✓	✓
Process priorities		✓	✓
Multi-level queue scheduling			✓
Process resources			✓
Deadlock avoidance algorithm			✓
Basic memory and operations on it	✓	✓	✓
Memory divided into hierarchy		✓	✓
Paging		✓	✓
Virtual memory with paging			✓
I/O interrupts and handlers	✓	✓	✓
Multi-threading via simulation	✓	✓	✓
Multi-threading via hardware		✓	✓
Multi-threading via hardware and multi-core via simulation			✓
GUI	✓	✓	✓
GUI with real-time monitoring and visualizations of simulator performance / operations	✓	✓	✓
Loading external processes and generating new ones on user request	✓	✓	✓


Identify crucial modules
Process management
add functionalities
Memory management
add functionalities
Multi-threading
add functionalities
GUI and visualization