CMSC 312 Project Description

Operating system simulator

Bartosz Krawczyk, Ph.D.
Department of Computer Science
School 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.

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 256 kB 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. Commands the user can issue are PROC, MEM, LOAD, EXE, RESET, and EXIT.

- PROC - shows all unfinished processes in the system and their information. The process information should include: current process state, amount of CPU time needed to complete, amount of CPU time already used, priority (if relevant), number of I/O requests performed.
- MEM - shows the current usage of memory space.
- LOAD - loads a program or job file into the simulator, and will also include the allocation of the program’s PCB and memory space.
- EXE - lets the simulation run on its own. The user can also specify the number of cycles to run before pausing. If there are no processes in the ready queue that are waiting to be scheduled, EXE will return to the command interface.

- RESET – allows the user to manually reset the simulator. All unfinished processes are terminated and the simulator clock returns to zero.
- EXIT – allows the user to end and exit the simulator.

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. They will accomplish this by a load and an exe command. The load command will load a program or job file into the simulator and will also include 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 or 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.

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 also will be most accepted. For advanced projects one may implement several solutions and compare their

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 256 kB. This value which is to be compared against the memory requirements of newly arrived processes. If total memory minus used memory (e.g. 256 kB – 180 kB = 76kB) 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.

2.10 Event Processing

To simulate the features of the Operating System simulator an event driven paradigm will be employed. Any event waiting for service will be represented by an ECB (i.e. Event Control Block) and will wait for service on the EventQueue. Content of the ECB will differ according to the event type. EventQueue will be a Priority Queue implemented as a Binary Heap data structure.

Each ECB will be created by the appropriate handler such as the I/O Scheduler, CPU (executing interruptible instructions), Process Scheduling, and so on. In the case of I/O, the Interrupt Processor will monitor the EventQueue for events matching the current clock time and service those events accordingly. Long and Short Term Schedulers (e.g. Dispatcher) will monitor the EventQueue for events matching the current clock time and service those events accordingly.

3. Operating System Classes and Methods

Please find below minimum required functions for specific OS components.

Command Interface: Executes commands to the Operating System and the simulators.

proc()
mem()
load()
exe()
reset()
promptUser()

Scheduler: Schedules processes according to round robin algorithm with a time quantum of 10 cycles.

insertPCB()
removePCB()
getState()
setState()
getWait()
setWait()
getArrival()
setArrival()
getCPUTime()
setCPUTime()

ExecutionQueue: Encapsulates the queue structure used by Scheduler.

enQueue()
deQueue()

CPU: Aggregates and supervises Clock, CacheMemory, InterruptProcessor, and eventQueue.

advanceClock()
detectInterrupt()
detectPreemption()

Clock: Simulates a CPU clock.

execute()
getClock()

InterruptProcessor: Simulates an interrupt processor.

signalInterrupt()

addEvent()
getEvent()

EventQueue: Encapsulates the event queue used by interrupt system.

enQueue()
deQueue()

IOScheduler: Simulates I/O devices.

scheduleIO()
startIO()

IOBurst: Randomly determines IO burst times.

generateIOBurst ()