Operating System Simulator Project

1. Introduction

This project is designed to be completed within approximately 10 weeks. You will learn much from this project if the proper time and thought is invested. Some of the learning objectives and goals are the following:

(a) to learn and implement some of the basic concepts of event-driven simulation;
(b) to learn fundamental issues of resource allocation and management in multiprogramming operating systems;
(c) to learn and implement the mechanism of context switching and interrupt handling;
(d) to learn and implement the basic flow of control within an operating system;
(e) to learn and implement different resource management algorithms;
(f) to learn and implement fundamental data structures of an operating system;
(g) to develop and practice good programming and debugging habits;
(h) to gain some experience implementing a complex program;
(i) to develop and sharpen your skills as a C programmer;

The value of this project as a learning experience is directly proportional to the time you give it. It will be very time consuming, there is no mistake about that. However, this project can give you an edge when you enter the work force after graduation - it should be documented on your resume'. I'll be glad to give you some pointers if you wish.

2. Project Overview.

The program you will write (hereafter called the "simulator") is designed to simulate the action of both hardware and software components of a simple time-sharing computing system. Your simulator will be organized so that C functions will model the behavior of hardware components as well as components of the operating system itself. In particular, simulator components will be written to model CPU and MEMORY hardware, interrupt handlers, the CPU scheduler, and process management functions of the system. Other components of the simulator will be provided to you at the beginning of the project.

One of the input files (CONFIG.DAT) to the simulator contains data describing the exact configuration of the system being simulated; for example, the number of interactive terminals, the number and characteristics of peripheral devices, the speed of the CPU, and the size of MEMORY, to name a few. The part of the simulator designed to read this file and initialize the simulation environment will be given to you at the outset.

Several other input files are needed to run the simulator. The LOGON.DAT file contains a description of user logon events by specifying the time they will occur and the ids of users logging on. To model the activity or behavior of each interactive user, a file called SCRIPT.DAT is input to the simulator. This file contains a "process script" identifying the sequence of programs run by each user during an interactive session. To model the behavior of each program designated in the process script, a "program script" must be input. There are five types of programs your simulator will model: EDITOR, PRINT Spooler, COMPILER, LINKER, and arbitrary USER programs. For each of these program types a file will be input containing a script for each program

"instance" of that type; eg, the EDITOR may be executed a total of five times by all users, therefore five EDITOR scripts will be input via EDITOR.DAT. Details on these files will be described later.

Program scripts will be "executed" by your CPU module to generate, in time sequence, the SVC calls made by that program. These calls become events processed by the interrupt hardware placing the operating system in execution under control of the Interrupt Handler(another function you will write). The Interrupt Handler calls the appropriate service routine, which may call the Scheduler, etc. Finally, when interrupt service is complete, control returns to the interrupted program and "execution" of that program continues until it terminates via an END SVC.

Other types of events can occur during simulation: EIO events generated by some device, and TIMER events generated by the system interval timer. These events also trigger the interrupt servicing mechanism described above.

Simulation terminates when all interactive users have "logged off" and all I/O events have been serviced. A user "logs off" when the corresponding process reaches the end of its process script. Thus, when all process scripts have been exhausted, and all I/O activity ceases, the simulator can terminate.

Output from the simulator will be to a single file called "name.OUT", where "name" is your last name (truncated to 8 characters). One of the CONFIG parameters is the name of your output file. In addition to a summary of all CONFIG.DAT parameters read as inputs, your output file will contain a log of all events processed during simulation. Furthermore, images of loaded process and program scripts will be echoed to this file. Finally, debug output and simulation statistics will be written to this file.

As an aid during the development of your simulator, you will have access to the "instructor's version" called OSSIM. OSSIM will be available on the LAN. Running or using OSSIM can be beneficial in two ways:

(1) it provides a guide and a benchmark for your output;
(2) it provides a working example to enhance your understanding of the dynamics of the simulation.

You should learn to use OSSIM as soon as possible.

3. The System Model.

The system being simulated is a single processor, time-sharing system with the following hardware and software components.

Hardware: N interactive terminals. One central processor.
Processor memory. An arbitrary number of peripheral devices.

Software: Interrupt handlers for LOGON, I/O completion, I/O-wait SVCs, I/O-Request SVCs, Program-end SVCs, Program-Abends; Memory manager; Loader; Scheduler; Editor; Compiler; Linker; and arbitrary user programs.

All input parameters describing the system being simulated are entered through the file CONFIG.DAT.

4. Simulator Specification

The simulation begins by processing events. The first event that occurs is an interrupt from a user terminal signaling a request to logon. The interrupt hardware changes the CPU and MEMORY states and gives control to the Interrupt Handler(IH). The IH examines the source and cause of the interrupt and calls the Logon Service routine.

The Logon Service routine creates a process control block (PCB) for the new terminal user and reads a process script from SCRIPT.DAT. It then allocates and loads the first program in the user's process script. Next, the service routine places the PCB in the CPU ready queue and signals the scheduler. When the very first logon event is serviced, the CPU will be idle. Consequently, the scheduler will assign the newly created PCB to the CPU. Then, the Dispatcher is called to give control to the new process. The Dispatcher prepares the program for execution and calls the CPU. The CPU interprets instructions contained in the program script until it encounters the next SVC call. It then creates an event corresponding to the SVC request and adds it to the event list. The CPU then terminates releasing control to the interrupt hardware to service the next hardware event.

The servicing of other events (interrupts) is similar. For example, when an SVC to start an I/O operation is serviced, an I/O request block (IORB) is allocated and queued for the requested device. An attempt is then made to start a new operation on the requested device. If the device is not busy, the next waiting IORB is de-queued and its request is initiated. The device operation is simulated by simply computing how long the I/O will take (based on byte count and device speed). The computed duration of the I/O transfer is then used to create an EIO event for the device which is added to the event list.

As you can see, the servicing of one event creates new future events. Eventually, as programs reach their end and I/O devices complete their requests, the event list will empty and the simulator will terminate.

The remainder of this section serves not only to amplify on the details of the simulation highlighted above, but to serve as a specification for the program you are to implement. Section 4.1 introduces the notion of Process Script, the model of how interactive users behave. Section 4.2 follows with the definition and discussion of Program Script, our model of how a typical program behaves from the Operating System's point of view. Section 4.3 describes the input files to the simulator, while Section 4.4 describes the output file.

4.1 The Process Model

The behavior of every terminal user must adhere to the process model,although the number and sequence of programs executed by each user will vary. A "process script" is any sequence of programs defined by the regular expression given in (1) that conforms to the transitions of the process model. It defines the system and user programs an interactive user will run during an interactive session. Each program specified in the process script must be allocated memory, loaded and run. The process script, in effect, defines the work load for the operating system determined by a given user.

(1)logon {editor, user, linker, compiler, printer}* logoff

Examples of valid process scripts: