AN OPERATING SYSTEM MODEL

Larry L. Wear, Ph.D.
Thomas P. Vayda, M.A.
Kirk MacKenzie
Ross Yakulis
Department of Computer Science
California State University, Chico
Chico, California 95929

Abstract

This paper describes an operating system (OS) simulator designed to be used as an instructional tool in the teaching of OS performance issues. Design criteria, assumptions used, development, and implementation of the model are discussed. The model provides a hands-on laboratory tool for experimentation with a layered, modular, multi-tasking, event driven OS. Incorporated in the model is the capability to experiment with a variety of job mixes. A flexible report generator presents the results of the simulation in a readable form. Empirical validation indicates that the model simulates typical OS behavior, and that it has potential for becoming an effective teaching tool.

INTRODUCTION

One of the main difficulties encountered both in the teaching and the study of Operating Systems (OS) is the lack of a laboratory environment for direct experimentation with a sophisticated, multi-tasking, event driven OS. One of the best ways to familiarize oneself with the properties and behavior of an OS is to vary the basic characteristics of the system, and the job mix. It is prohibitively expensive, difficult, and hazardous to allow students to perform such experiments directly on a functional system already allocated for other computing uses. These were the factors that motivated us to develop an OS model that would serve as a laboratory, allowing students to perform the types of experiments described above.

We decided that our model should be easily transportable and modifiable, thus an Apple III computer was selected, and UCSD Pascal was the programming language chosen. Since a team of ten programmers was to work on the project, it was essential to design the Operating System Model (OSM) as a collection of independent cooperating modules. This design choice should also facilitate future modifications to the OSM. All the programmers contributed during the design phase, and module assignments were made. After the individual modules were completed, a supervisor program, called the User Job Processor (UJP), was written; the separate modules were linked together and the OSM was completed.

It is the purpose of this paper to give an overview of the OSM, then to describe the three major sections of the OSM, the types of experiments that can be performed using the OSM, the degree of success achieved, and planned future enhancements to the current capabilities of the OSM.

OVERVIEW OF THE OPERATING SYSTEM MODEL

The purpose of the OSM is to simulate an OS with specified parameters and job stream. The OS simulated is a layered modular system such as described by Lister [1] or Calingaert [2]. The specification of the system parameters and job stream should be done interactively, thus allowing for and encouraging experimentation with the simulated OS.

The inputs to OSM will consist of several data files, previously generated by the user. These data files contain:

1. parameters specifying system characteristics such as memory size, memory placement algorithm to be used, the number and type of I/O devices attached to the system, etc.,
2. the stream of jobs to be processed by the OS. Each job will have randomly generated characteristics (within limits and distributions specified by the user) such as processor use vs. I/O ratio, I/O devices used, etc., and
3. specification of output reports to be generated by the OSM.

The output generated by the OSM will be reports consisting of statistics describing how the specified job stream was processed by the simulated OS. The statistics include such items as total CPU

Proceedings of the 1982
Winter Simulation Conference
Highland * Chao * Madrigal, Editors

82CH1844-0/82/0000-0323 $00.75 © 1982 IEEE

time used, number and size of I/O transfers handled by each device, percentage utilization of various system resources, and "snapshots" of the state of the system at specified times.

The OSM consists of three distinct logical modules as illustrated in Figure 1. They are the System Configuration/Job Stream generation module, the Simulator module, and the Report Generator module. The first of these modules interacts with the user to create initialization, system configuration, and job stream data files. The job stream file is created using random variables from user specified statistical distributions. The Simulator module processes the job stream data, by simulating the internal workings of an OS, and collects specified statistics. The Report Generator module produces readable reports that were specified by the user. These reports are relevant to OS performance and summarize statistics collected by the Simulator module.

The UCSD Pascal system was chosen not only because it supports modular design and coding techniques, but hopefully will allow the OSM to be easily transported to other sites and systems. The OSM currently runs on an Apple III with 160K main memory and a Profiler 5 megabyte hard disk.

The simulation is controlled by an imaginary timeline which records the sequence of events to be performed by the simulator module. The events on the timeline represent either future interrupts to be serviced by the OS, "snapshot" statistic reports to be output, or the end of the simulation. Many of the modules can alter the timeline by adding or deleting events from the timeline. The simulation proceeds by executing the next event indicated by the timeline and updating the simulation clock. When one event occurs, it may be necessary to add one or more future events to the timeline. This choice of simulation technique allows the simulation to proceed in an efficient manner, by eliminating unnecessary monitoring of the system behavior.

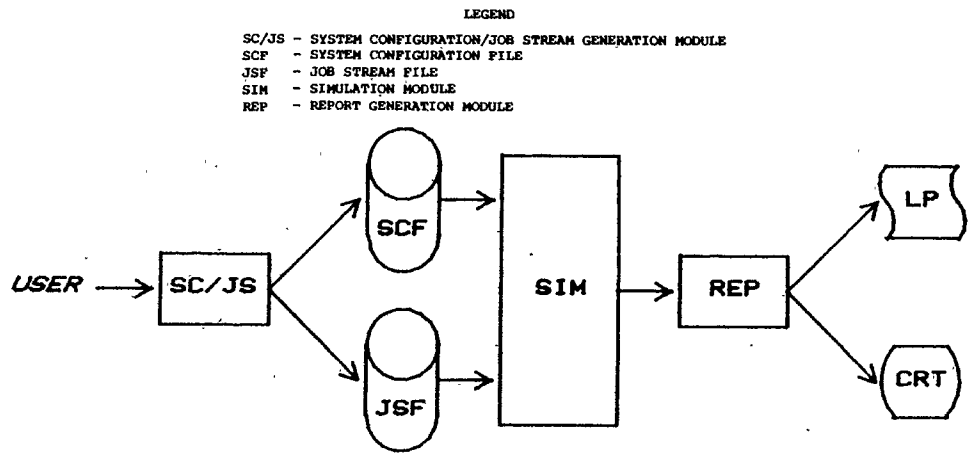

FIGURE 1 – MAIN MODULES OF THE OSM

To allow for future modifications and expansion of the OSM, it was decided that the Simulator module should consist of several independent, interacting submodules. This design methodology lent itself well to structured, modular methods of coding, and also allowed us to easily coordinate and specify tasks to be performed by the ten programmers involved in the project. The facilities provided by the UCSD Pascal system allowed the separately coded and debugged modules to be easily incorporated into a single set of cooperating programs.

As an example of the advantage of using independent modules, if a future user wanted to simulate a paged memory system instead of the currently used segmented memory, only the memory management submodule would have to be modified, leaving all other parts of the OSM unaltered. Similarly, different memory replacement policies could be easily implemented without affecting other modules of the system.

During the design phase the following assumptions were made about the OS to be simulated:

1. The simulation should be able to carry out a minimal amount of error checking, such as detecting and properly handling a request for a non-existant resource.

2. The OSM will model a layered modular OS of contemporary design.

3. The OSM will model a multi-tasking, event driven OS.

4. A vectored priority interrupt system will be supported.

5. Memory management will assume a variable length, segmented, non-paged system.

6. The I/O facilities will support direct memory access (DMA) and channels.

7. Static resource allocation will be used in the first version of the OSM.

The next three sections provide detailed descriptions of each of the main modules and submodules used in the OSM.

SYSTEM CONFIGURATION/JOB STREAM GENERATION MODULE

This interactive menu-driven module allows the user to create two data files needed as inputs to the OSM: the system configuration file, and the job stream file. The file names are specified by the user, and the files created are automatically saved on disk by the module.

The system configuration file specifies parameters for the computer system to be simulated. The user is prompted to specify the size of main memory, the memory placement policy to be used (currently first-fit, best-fit, or worst-fit), and I/O facilities to be provided. Up to 30 I/O devices of varying speeds can be specified. The devices are grouped into the three categories based on their transfer rate, and no more than ten devices from each category can be used by a job. This allows the simulation of slow, medium, or high speed I/O devices.

The job stream file contains the description of the set of jobs to be processed by the simulation module during one simulation run. To create the job stream file first the user is prompted to discribe a single job type by specifying job characteristics such as upper and lower bounds for average time between I/O requests, upper and lower bounds for the average size of an I/O request, what percent of I/O requests use slow, medium, or fast I/O devices, average code and data segment size, and average CPU time required. After a particular job type has been described, the user is asked how many jobs of this type are to be included in the job stream. After all the job types have been similarly described the module randomly chooses from the pool of job types specified, and creates the job stream file.

SIMULATION MODULE

The simulation module consists of several utility procedures and five main submodules, each of which models the following parts of an OS respectively: The Executive submodule (EXEC), Memory Manager (MEMMGR), User Job Processor (UJP), First Level Interrupt Handler (FLIH), I/O Processor (IOP), and the Scheduler. Figure 2 illustrates the logical flow of jobs through the simulation module.

Several utility procedures are shared by the submodules. They provide commonly used functions such as managing the many linked lists and other data structures used by the OSM, calculating overhead times needed for each OS function, controlling and formatting I/O for communicating with the

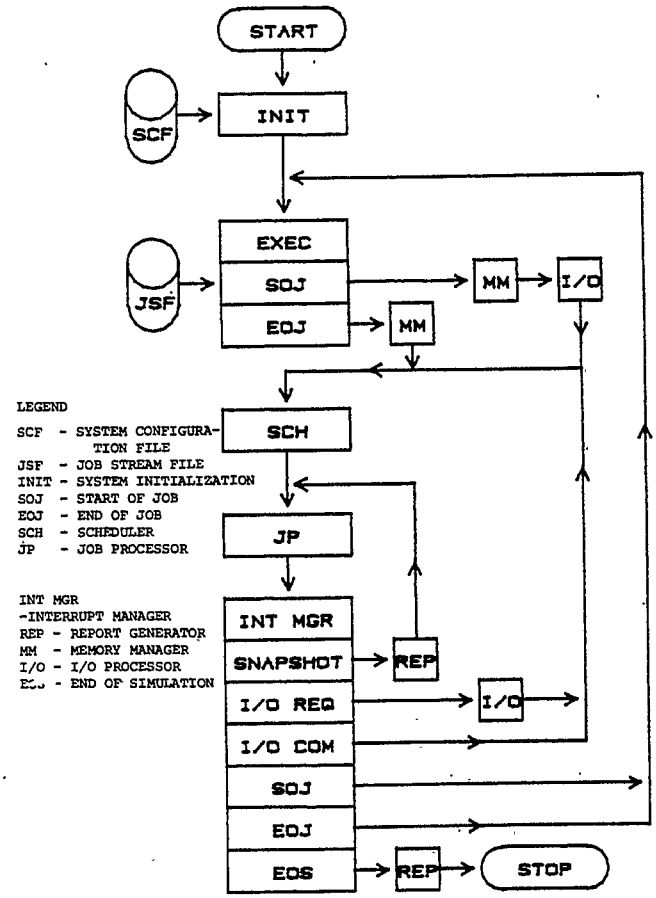

FIGURE 2 - THE SIMULATION MODULE

user, reading the data files, and generating random variables from specified distributions.

A simulation run begins by initializing global variables, then the EXEC module is called to start the first job in the job stream file. The EXEC submodule is responsible for reading and then initiating new jobs, as well as terminating completed or invalid jobs. The MEMMGR is called by EXEC to allocate memory for a new job entering the system. The MEMMGR can either allocate memory (using first, best, or worst fit) to the job, or suspend the job if sufficient memory is not available. When sufficient memory becomes available, the suspended job will be activated by the MEMMGR.

The UJP simulates the execution of the currently active task. It increments the system clock until the task either requests I/O, is interrupted, or completes execution. When one of these events occurs, control is passed to the FLIH. The FLIH determines the cause of the interrupt, decides which module is to process the interrupt, then updates some relevant statistics and passes control to the appropriate module.

The IOP is responsible for initiating and supervising I/O operations, and for activating the correct job when an I/O completion interrupt occurs. It also maintains a (possibly empty) queue of jobs waiting on each I/O device. Appropriate global statistics are updated by the IOP whenever an I/O transfer occurs.

The scheduler is called by several other modules, and is responsible for maintaining the ready to run queue. It is activated after every interrupt and reorders the ready to run queue as needed.

The simulation module continues to process jobs in the manner described above, all the jobs in the job stream file have been processed and the last job exits the system. At this time the report generator module is called to produce a summary of OS activities performed during the run, and then the simulation run is over.

REPORT GENERATOR MODULE

The module is invoked by the interrupt handler to print out statistics gathered by specified simulation submodules during a given run, or to print out specific summary statistics at the end of a run. Many of the statistics that we chose to gather were suggested as useful performance measures by Coffman and Denning [3] and Hansen [4].

There are two types of reports available: "snapshots" of the current state of various components of the OSM or cumulative statistics indicating OS performance and other relevant measurements during a given run. The "snapshot" reports allow the user to examine interrupts remaining to be processed, the state of the currently executing task, the current memory map, various device and suspend queues, the ready to run queue, and the state of all jobs currently in the system.

The cumulative statistics show the work performed by the various simulation submodules including the memory manager, exec, interrupt handler, I/O processor, and the scheduler. We can get measurements of overhead incurred in all the modules, the number of jobs that have entered or left the system, and the number and type of interrupts processed. The I/O module reports idle time, utilization time, number of jobs processed, number of jobs in the device queue, and the current job utilizing the device, for each device. The memory manager reports the number of jobs currently in memory, the number of jobs blocked for memory, amount of wasted memory, and the number of times compaction has been performed.

There are several other reports available that were not mentioned above, and more will be added as users start asking questions about other aspects of the OSM.

EXPERIENCES USING THE OSM

The OSM appears to be an effective OS laboratory. A user can easily test the effect of changing the job mix offered to a particular OS, or the effect of changing some OS characteristics while processing a particular job stream, or possibly to vary both of the above. Once a library of system configuration and job stream files has been created, it may not even be necessary to create new files for each run. The flexible report generation module allows the user to closely examine the performance of a particular subsystem (such as the memory manager or the scheduler) or to observe the overall workings of the simulated system. Thus using the OSM, the user is free to perform an unlimited number and type of experiments, and to study the impact of changing job or system parameters on any particular subsystem.

The OSM has been extensively tested by our group. We have found that the system works very satisfactorily within confines of the design assumptions listed in a previous section. Most of the reports produced are very readable and provide important statistics. Since the simulation time units do not correspond to real time units, it has been difficult to obtain quanitative comparisons with actual statistics from existing systems, but the qualitative results appear to be very realistic. For example, if the number of I/O devices is increased, the amount of time spent waiting in device queues is reduced. The fact that this and numerous other results agree with typical OS behavior as described by Habermann [5] and Shaw [6] lead one to have confidence in the correct operation of the model. How well the OSM helps students of operating systems to understand the tradeoffs inherent in OS design remains to be decided in the near future, when it is released for general use by an advanced OS class.

The OSM has certain limitations, some of which can be overcome by enhancements to the OSM (see next section) and some inherent limitations imposed by the choice of hardware. The current version of the OSM does not allow simulating a paged memory system, choosing various scheduling algorithms, dynamic resource allocation, dedicated or protected resources, and other similar desirable OS features.

Since all reports are presented in tabular form, some of them are time consuming to read and interpret. The simulation time units do not correspond to actual time units, thus it is difficult to interpret the numerical results.

The main limitation imposed by the hardware, is that given a large job stream, and many reports to be produced, a simulation run can take a considerable amount of time to complete. The next section proposes remedies for some of these limitations.

PLANNED FUTURE ENHANCEMENTS TO THE OSM

As mentioned above, the current version of the OSM has several limitations that can be overcome by enhancing certain features of the OSM. Some of the enhancements that we are currently working on are listed below.

1. Adding the capability for a paged, or a partial memory system.

2. Expanding the available choices for memory placement and replacement policies.

3. Allowing the selection of a variety of scheduling algorithms in addition to the currently used simple round-robin algorithm.

4. Allowing for dynamic resource allocation.

5. Incorporating an inter task communication facility similar to the I/O processor.

6. Adding an input/output job spooler to the OSM.

7. Replacing tabular reports by graphic output.

There are undoubtedly many other enhancements or improvements that could be added to the OSM, thus there is much work left to be done. We hope that some of the future users of the OSM will help to contribute to this effort.

SUMMARY

The OSM described in this paper presents one possible solution to the problem of providing hands-on experimentation opportunities for persons studying various aspects of OS evaluation, design, or implementation. While the current version of the OSM suffers from limitations described above, it can still provide many valuable insights into the numerous tradeoffs and design compromises that each OS designer must resolve. It also promises to be an effective tool to be used by instructors of the arcane art of OS design.

BIBLIOGRAPHY

1. Lister, A.M., Fundamentals of Operating Systems, Springer-Verlag Inc., New York

2. Calingaert, Operating System Elements - A User Perspective, Prentice-Hall, Inc., New Jersey

3. Coffman, E. G. and Denning, P. J., Operating Systems Theory, Prentice-Hall Inc., New Jersey

4. Hansen, P. B., Operating Systems Principles, Prentice-Hall, Inc., New Jersey

5. Habermann, A. N., Introduction to Operating System Design, Science Research Associates, Inc., Chicago

6. Shaw, A. C., The Logical Design of Operating Systems, Prentice-Hall, Inc., New Jersey