UNOS Operating System Simulator

Zhong Tang

UNITEC Institute of Technology
Auckland, New Zealand

ztang@unitec.ac.nz

Proceedings of the 15th Annual NACCQ, Hamilton New Zealand July, 2002 www.naccq.ac.nz

ABSTRACT

The course 'Operating System Internals' (OSI) usually focuses on teaching the concepts and issues surrounding the design of an OS in general. Students are advised to learn by example and through experiments, which are fundamental for the real understanding of operating system. Although controlled experiments with a commercial OS is one of basic approaches in organizing the practical work of students, unfortunately, the sheer speed and complexity of modern computer systems make them really hard to understand by the average students. So providing students with a high-level OS simulator that can isolate the actually environment is an alternative way that will allow students to observe the work of various OS components and to explore various algorithms.

This paper describes our own developed UNOS Operating System Simulator, which is implemented as a standalone Java application simulating a basic multi-tasking non-pre-emptive operating system and requests least programming knowledge.

Key words: Courseware, Java, Operating system, Simulation, UML

1. INTRODUCTION

'Operating Systems Internals' is a level-6 course, which is offered on a semester basis in UNITEC Institute of Technology. The laboratory experiment is a main part of the course. Currently, the students experiment with the internals of the Windows NT operating system and its Win32 API. However, the course results and the previous students' feedback showed that students spent considerable time overcoming previous technical difficulties, and missed the more important conceptual knowledge. These are the problems and constraints the students are confronted with in their practical work:

- Performance: Spend considerable time on overcoming technical difficulties.
- Information: Concepts is hard to be abstracted from experiments.
- Economy: Students lose interest on the course.
- Control: Hard to manipulate the experiment to get desired results.
- Efficiency: Completion of an experiment is time consuming.
- Service: Lack of visual information of internal states and operations.

Therefore, there remains a need for the students to interact with an animated tool, i.e. an operating system simulator, in order to explore various algorithms that are used in operating systems, e.g. Job scheduling, CPU scheduling, Disk scheduling, Virtual memory management.

In this paper, the writer will make a short survey of existing 'teaching' OS, then describes an internally developed operating system animated tool: UNOS, which features a full-fledged graphic user interface running on Java2 runtime environment, complete source and documentation, followed by future extension to UNOS system in conclusion.

2. OVERVIEW OF EXISTING 'TEACHING' OPERATING SYSTEMS

Using an existing commercial OS for students practical work of the 'OSI' course has several limitations: 1) lack of a copy of the OS source code; 2) requiring more technical skills; 3) need of more programming knowledge. To tackle the issue of the complexity of commercial OS, a number of 'teaching' operating system have been created. These 'teaching' OS can be classified in two types: downsize operating system and simulated operating system. An example of the former is Minix, and of the latter is Nachos.

Minix is a free Unix clone of small size, which is designed as a teaching system by Andrew S. Tanenbaum. To work with Minix, users have to overcome the steep learning curve in installing and getting familiar with the system. Moreover, it does not provide a visual interface tool to manipulate the internals of the OS.

Nachos (Silbershatz and Galvin 1999) is an instructional software run on Unix. It allows users to study and modify a real operating system. It is a complex piece of software and it is difficult for beginning users to gain an overall understanding of the various system components and how they fit together.

The only difference between these 'teaching' operating systems and real operating systems is that the 'teaching' operating systems run as a single Unix process, where as real operating systems run on bare machines. Both Minix and Nachos simulates the general low-level facilities of typical machine, include interrupts, virtual memory and interrupt-driven I/O. Although it is possible to read and understand source code by using the code browsing tools (e.g. the emacs 'tags' facility), its intended audiences should have knowledge of common structures such as stacks, queues and linked list, and be familiar with current hardware architectures and knowledge of C, C++ and UNIX.

The requirement of the OSI course at UNITEC is to let the students to concentrate on the comparison between main ideas, rather than on the details of implementation. Using those 'teaching' operating systems mentioned above in the OSI course still does not address the issue of complexity. However, the writer's positive experience with the formers inspired the design and implementation of UNOS system as a high-level animated simulator.

3. THE OUTLINE OF UNOS SIMULATOR

The UNOS operating system simulator is initially intended for use primarily by the students in open laboratory work, but its latest version can also be used by the instructor in the classroom to illustrate concepts of OS. This animated tool has a user-friendly interface, simple to use, both in term of incorporating student's algorithm and performing the simulation. At the same time it allows students to tune the various parameters, so that they can observe the effect on the overall performance. The system has two regimes of simulation — 'Single Step' or 'Continue' running modes. Its 'Single Step' running mode enables students to observe OS internal states at any particular phases and to display the current statistic information of CPU scheduling and memory usage. For functional use, the UNOS simulator consists of following components:

1. INPUT JOBS AND CREATE PCBS
   - ♦ Accept jobs;
   - ♦ Manipulate job descriptors (JD) and process control blocks (PCB);

2. TUNING OS PARAMETERS
   - ♦ Set CPU speed and OS size;
   - ♦ Set main memory and virtual memory sizes;
   - ♦ Set disk storage size;

3. SELECTING ALGORITHMS
   - ♦ Set job scheduling algorithm: First Come First Served (FCFS) or Priority
   - ♦ Set CPU scheduling algorithm: First Come First Served (FCFS), Shortest Job First (SJF), Round-Robin (RR) or Priority.
   - ♦ Set memory management algorithm: First-In First-Out (FIFO), Second Chance Page Replacement (SCPR) or Least Recently Used (LRU)
   - ♦ Set disk scheduling algorithm: First Come First Served (FCFS), Shortest Seek First (SSF) or Elevator

4. SIMULATION
   - ♦ Simulate program behaviors;
   - ♦ Simulate process behaviors;

5. PERFORMANCE REPORT
   - ♦ System simulation history
   - ♦ Frame Information

4. THE UNOS ARCHITECTURE

The UNOS operating system animated simulator is designed using the Unified Modeling Language (Booch *et al.* 1998) and developed in Java using its concurrent programming facilities. The Use Case Model captures all functional and/or non-functional requirements in the system. The writer first identified all actors that interface with the system, then analyzed the main functions of the system and realized the functions with these use cases.

4.1 PRIMARY USE CASE VIEW

The primary use case view captures the relationships among the different components of the UNOS system (Figure-1).

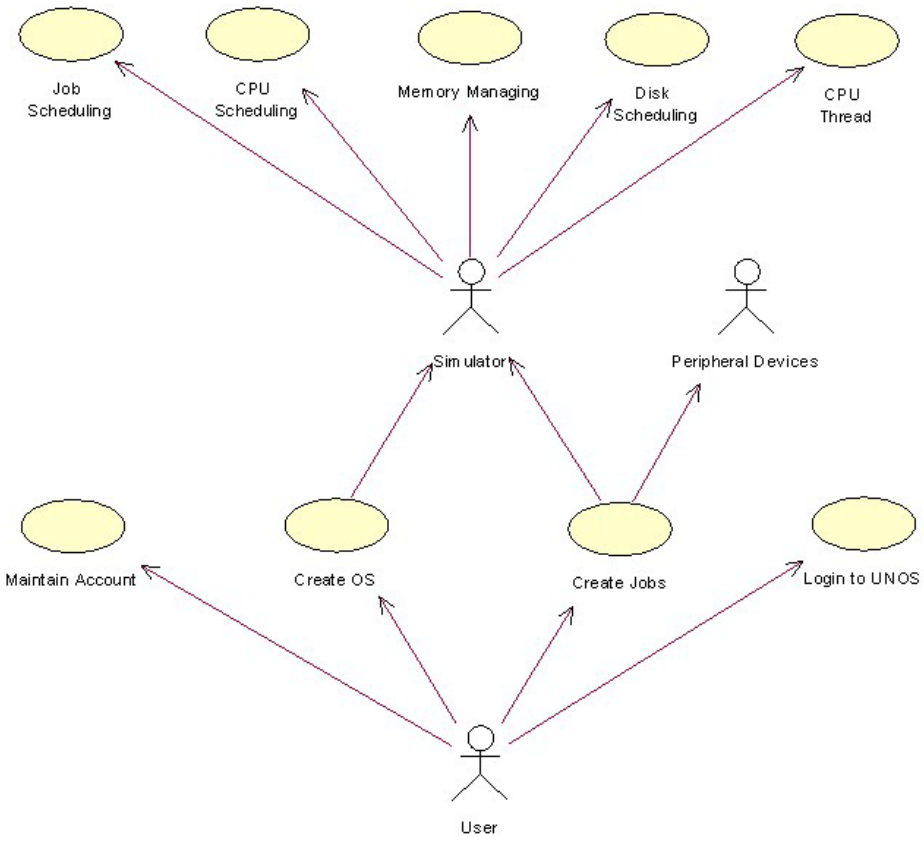

Figure-1 Primary Use Case View

The function of each use case in the primary use case view as described in the following sections.

4.1.1 CREATE JOBS

The system provides editing ability for users to create a job batch file, which is saved as a text file. After the simulator is configured and launched users can either load the job batch file into the job queue or create jobs at run time. The simulator manipulates job descriptors (JD) and process control blocks (PCB) representing the programs executing on a computer. JD contain such job characteristics as average CPU execution time, I/O burst times, priority, memory requirement and service request frequency (I/O rate %). PCBs extend JDs and are constructed when a job is loaded into the virtual memory.

4.1.2 CREATE OS

This function allows users to configure the OS simulator by setting OS parameters (time slice, page size, physical memory size and virtual memory size, etc.) and choosing different scheduling algorithms for jobs, processes, memory and disk operations.

4.1.3 JOB SCHEDULING

The job scheduler loads jobs from the job queue to the virtual memory, creates process control blocks (PCB) from the job descriptions (JD), and place them on the ready queue. The order in which jobs are selected from the job queue is determined by the job-scheduling algorithm supplied by the user.

4.1.4 CPU SCHEDULING

The CPU scheduler 'consumes' PCBs from the ready queue by placing them, one at a time, on the CPU, according to the CPU scheduling algorithm. The CPU scheduler also allocates a time slice to each process's execution.

4.1.5 MEMORY MANAGEMENT

When receiving a request from CPU thread, which asks for a new frame for the current running process, the memory management checks the memory free list to find an available frame. If there is no such available frame, it carries out the page replacement according to the given memory management algorithm.

4.1.6 DISK SCHEDULING

When receiving a request from the CPU thread, which asks for a disk writing or reading operation for the current running process, the disk scheduling scans the disk to allocate the tracks according to the given scheduling algorithm. The screen shot (Figure-4) illustrates the implementation.

4.1.7 CPU THREAD

The CPU thread is a major function of the UNOS system. It simulates the execution of the running process by monitoring the simulation time and reacting to messages that are generated randomly (such as new page, I/O request), time-dependent events (such as time slice expiration or end of task), and messages triggered when the user types a system command (such as suspend, resume or kill) in the system console. The message (I/O request, wait, sleep, or suspend) results in placing the running process' PCB on its corresponding queues, which represent 'monitors' (Hoare, 1974) and are implemented as Java synchronized objects. A polling watch (Lea 1977) is attached to each of these queues and monitors the absolute simulation time and compares it with the time at which the PCB should be removed from the corresponding queue. The possible states of a process are Ready, Running, Blocked, Waiting, Suspended, Sleeping, and Finished.

4.2 SECONDARY USE CASE VIEW

The secondary use case view captures the relationships between the UNOS system and its external environment (Figure-2).

The function of each use case in the secondary use case view can be described as overpage:

4.2.1 INTERACTING SIMULATOR

The function to interact with UNOS system is carried out by the system console. The users can type a system command in the console to control the flow of system execution. The system commands includes: 1) Create a new job; 2) Load a job from job queue to ready queue; 3) Suspend a given process; 4) Resume a given process; 5) Wait a given process until another process invoke it; 6) Sleep a process for a given time; 7) Kill a given process; 8) Cls to clear the screen of the console; 9) Shutdown to exit the simulator; 10) Help to list all commands and their formats.

4.2.2 CONNECTING TO DATABASE

This application has a MS Access database to hold user accounts (i.e. user ID and password). Users need to connect to the database before running the simulator.

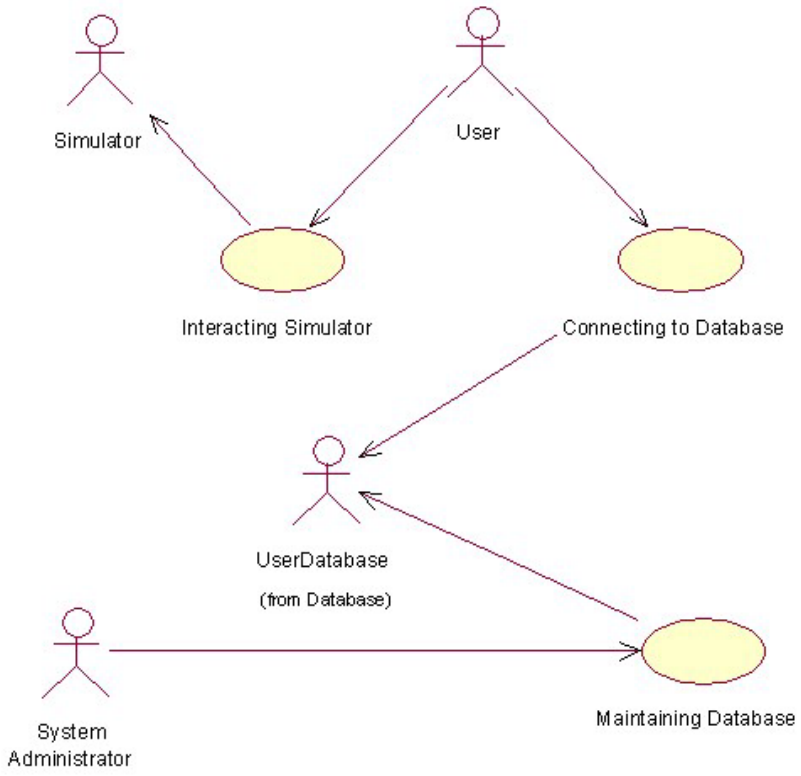

Figure-2 Secondary Use Case View.

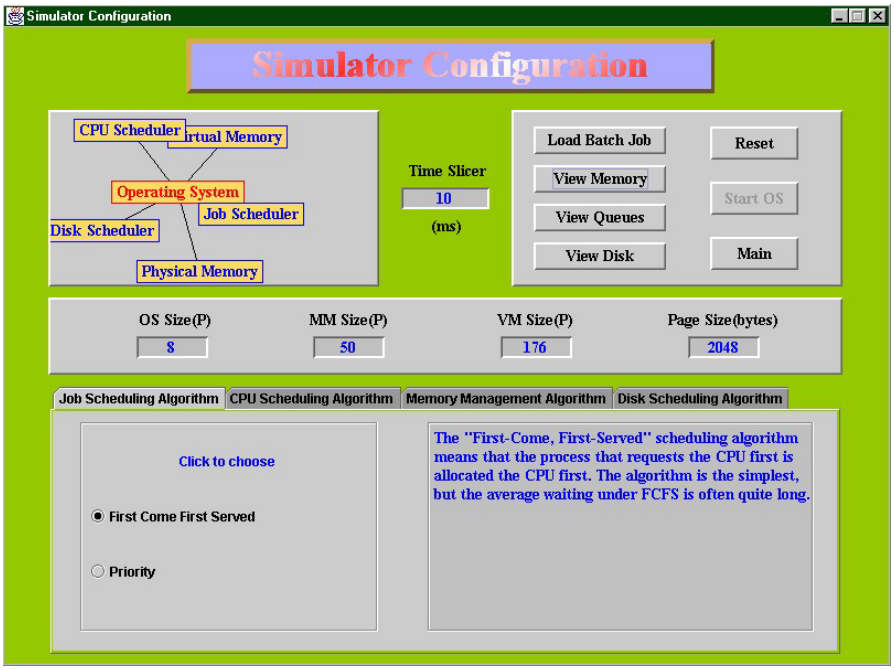

Figure-3 Simulator Configuration.

5. SIMULATION MODELS AND PERFORMANCE REVIEW

UNOS simulator is an interactive one, with simple, intuitive user interface, to allow users to tune various parameters and select incorporated algorithms of four distinct components of the simulated OS, i.e. Job Scheduling, CPU Scheduling, Virtual Memory Management and Disk Scheduling (Figure- 3).

For each of these components, a number of algorithms are discussed in class, and calculation exercises are performed in tutorials. In open laboratory work, users can use the UNOS simulator to observe the effect of the algorithms that they supply for the UNOS animated models, and compare the results of the calculation exercises with the outputs produced by UNOS simulation models.

5.1 CPU SIMULATION MODEL

The CPU simulation model provides a graphical description of the process scheduling and job scheduling operation of an OS (Figure-4). It is also incorporated with a console-like interface for user to manipulate the process and interact with the operation of the simulator. During UNOS simulation period, users can press the button 'History' to suspend the CPU thread at any time. This will pop up the 'System Simulation History' window (Figure-5), which shows relevant history information of processes running up to this point. Such system history information include system time, process number, start time, finish time, load time and total wait time.

Meanwhile if necessary, users can click any cell on the job queue to show a particular job's information or click any cell on the other queue, e.g. Ready Queue, to show the process state (Figure-6). The 'Continue/ Single' button allows the user to set simulator-running mode. In 'Single' mode, the user clicks the 'CPU' button (a big square button labeled with current PCB number) each time to enable the system run a time slice.

When all jobs in the job queue are successful done, the simulator is in idle state. The user can enter the command 'Report' in the system console to display the statistic information of the overall performance of the simulated system (Figure-7).

5.2 MEMORY SIMULATION MODEL

The memory simulation model provides a graphical description of physical memory and virtual memory allocation and dynamically reflects the page replacement operations of an operating system (Figure-8). The message new page is forwarded to the Page Table object, which refreshes every CPU time to reflect the updated paging information, such as page frame number, logical page number, load time, recently used time, presence bit and valid-invalid bit. This simulation model also allows user to directly choose a logical page, by clicking a row on the current running PCB's page table, as the next page that the corresponding process is going to reference so as to highlight the procedure of page replacement. To track the usage of frames in physical memory, users can press the button 'Frame Info' to suspend the CPU thread at any time and to show the frame usage information (Figure-9).

5.3 DISK SIMULATION MODEL

The disk simulation model provides a graphical description of disk scheduling and track operations of an OS (Figure-10). The tracks are structured in a plane diagram to illustrate the procedure. The 'Auto/ Step' button allows the user to set the track-scanning mode. In 'Step' mode, the user clicking the button 'One Step' each time causes the scanner scan a track according to the selected scheduling algorithm, which highlights the concept of disk scheduling.

6. CONCLUSION

In summary, the UNOS operating system animated simulator provides a useful opportunity for supplementing the use of a real OS for teaching the OSI course. This paper describes the applicability and the main design and implementation of the UNOS interactive animated simulator. UNOS has been tested with almost course demos/examples in the textbook (Operating System Concepts, 5th, Silberschatz). The simulator was designed and developed using Object-Oriented methodology, which ensures the extensibility of the system. The future work on UNOS will include the functions, such as inter-process communication, file system, etc.

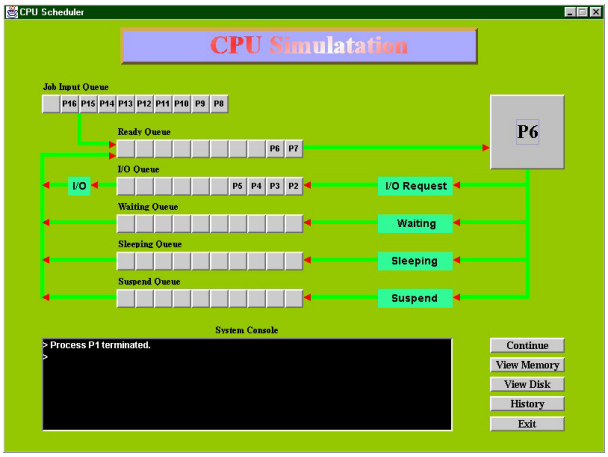

Figure-4 CPU Scheduling

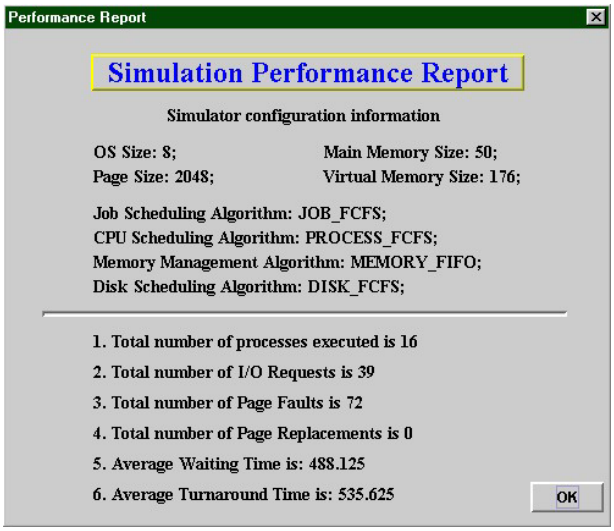

Figure-7 Statistics Calculated

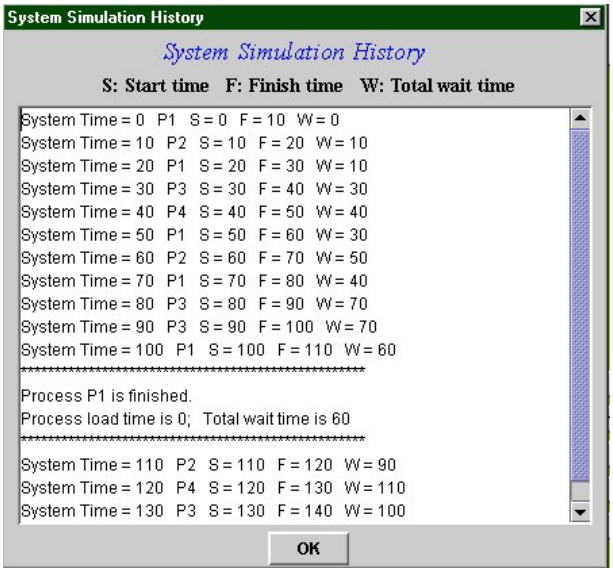

Figure-5 System Simulation History

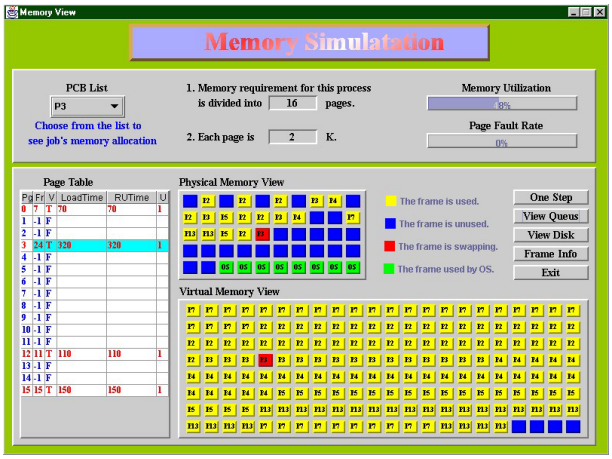

Figure-8 Memory Management

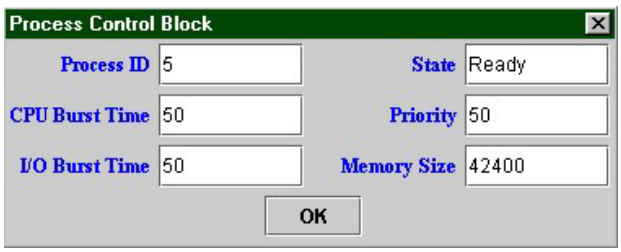

Figure-6 Process State Information

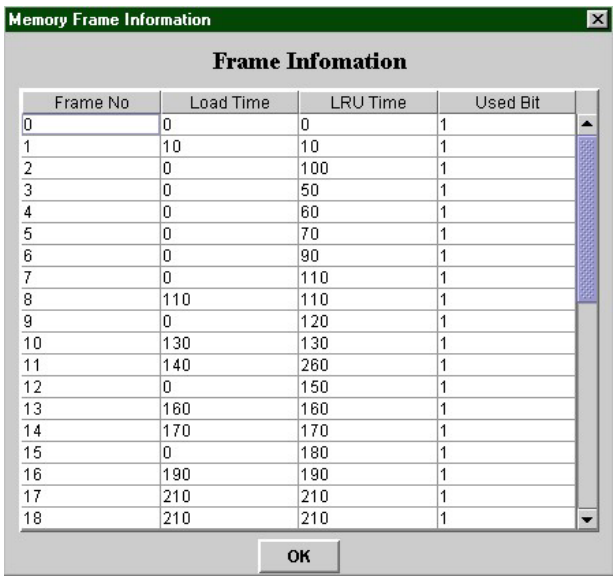

Figure-9 Frame Usage Information

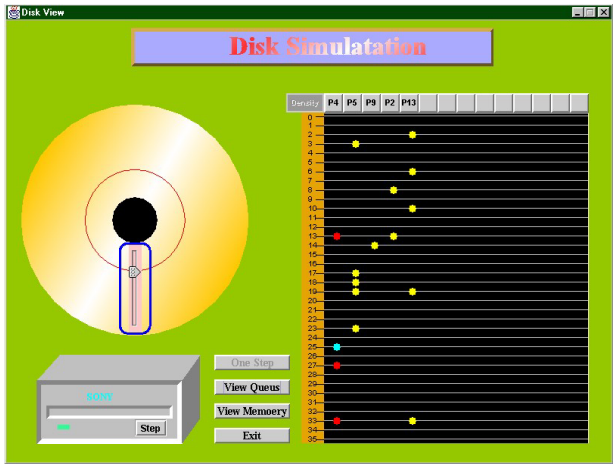

Figure-10 Disk Scheduling

REFERENCE

Booch, G., *et al.* (1998). Unifield Modeling Language, User Guide, Addison Wesley

Coad, P., Yourdon E, (1991). Object Oriented Analysis, Prentice Hall

Gancho Ganchev. "An Operating System Simulator for Teaching OS Internals", 12th Annual Conference of the NACCQ

G. Agrirre, M. Errecalde, R. Guerrero, C, Kavka. Texperiencing MINIX as a Didactical Aid for Operating System Courses, Operating System Review, 25:32-39, July 1991. Dunedin, New Zealand

Jacobson, I., Booch, G. & Rumbaugh, J. (1999). The Unified Software Development Process, Addison - Wesley

Jose M. Garrido, (2001). "Object-Oriented Discrete-Event Simulation with Java".

Silberschatz, A., and Galvin, P. (1998). Operating System Concepts, 5th edition

Stallings, W. (1998). Operating Systems; Internals and Design Principles, 3rd ed. Prentice Hall.

Tanenbaum A. (1987). Operating System Design and Implementation, Prentice Hall

poster papers