DYNAMIX: an Operating System Teaching Assistant

Edward Grabczewski and Xiaohui Liu
Department of Computer Science
Birkbeck College
University of London
Malet Street
London WC 1E 7HX

Tel. 071 631 6468
Fax: 071 636 4971
E-mail: hui@uk.ac.bbk.dcs

Our experience shows that students have difficulty understanding how operating systems work. A theoretical study of the main functions of an operating system does not give students a feel for how these functional components interact to provide a service to the user. Some way is needed to help teach the theoretical material more effectively. We have chosen a simple graphical technique to develop DYNAMIX, a tool that displays the internals of a popular operating system and demonstrates the interaction among its functional components. In this paper we describe the development of DYNAMIX and show how it can be used in the classroom.

Topics: Educational Tools, Learning and Cognition, Classroom Experiences
Format: Workshop
Computer: IBM PC XT/AT Compatible

E Grabczewski currently works for ORACLE Corporation (UK). Please direct any correspondence regarding this paper to X Liu.

1. Introduction

In text books, operating systems concepts are typically analysed under headings such as processes, memory management, file systems and input/output. Our experience in teaching operating systems over the past few years shows that most students can manage these isolated concepts well, however they have difficulty in synthesizing them back into a coherent picture of how an operating system works. This is partly due to the lack of adequate coverage of the interactions among various parts of an operating system in the literature. It is also partly due to the lack of a practical tool to model the interaction among different software components and allow lecturers to present the difficult concept in a simple way.

We have been looking at ways to address the problem. One approach has been to develop a software tool which helps visualise how the internal components of a real operating system cooperate with each other to execute a user command. As a result, we developed DYNAMIX, a modified version of the MINIX operating system.

MINIX (mini-UNIX) [10, 11] is based on UNIX Version 7. It was developed by Andrew Tanenbaum to give students an opportunity to study a real operating system program instead of just reading about them in books. MINIX has the virtue of being relatively small, simple, modular and freely available. It runs on a standard IBM PC XT/AT compatible.

DYNAMIX (dynamic-MINIX) is an extension of the MINIX kernel. It allows the user to see the internal workings of MINIX as it executes commands. DYNAMIX displays a dynamic "window" into the MINIX operating system, allowing the user to literally see the main components and how they interact.

Normally MINIX takes a tiny fraction of a second to execute a command but for classroom teaching this needs to be slowed down considerably, allowing the lecturer time to adequately explain what is happening. To this end a number of useful control functions are provided in DYNAMIX. For example, the lecturer can adjust the speed of groups of software components to get the best effect in the window. The entire demonstration can also be paused during lengthy explanations.

DYNAMIX has been used to assist in the teaching of operating systems concepts at Birkbeck College and initial findings are encouraging. In this paper we will describe the development of DYNAMIX and discuss how it can be used for classroom teaching. In section 2 the background and motivation for this work is given. This is followed by a description of the development of DYNAMIX in section 3. In section 4, we describe our experience in using DYNAMIX for classroom teaching and feedback from our students. In section 5, further improvements to DYNAMIX are suggested. Finally, the concluding section summarises our work.

2. Background

Those who teach operating systems concepts are rather spoilt for choice. Excellent textbooks have been written by: Bach [1], Comer [2], Deitel [3], Fortier [4], Leffer[6], Lister [7], Silberschatz [8], Switzer[9], and Tanenbaum [10, 12]. Each of these texts analyses to some extent the design and implementation of operating systems. Some texts contain sample program listings of operating systems for students to study. Most of these programs run on IBM PC XT/AT compatibles, making them easily available to students both at college and home.

There appear to be two principal methods to teach operating system concepts at academic institutions. One is the "theoretical" approach where the concepts are taught using textbooks and course notes. This is typical of short courses where no further time is available for a more in-depth study.

An alternative is to supplement the theoretical material with a practical element, such as studying the programs for a given operating system. This normally requires the student to have some pre-requisite knowledge of:

- Computer hardware (eg. 8086 micro-processor, memory)
- Low-level programming language (eg. 8086 assembler)
- High-level programming language (eg. C)
- Use of data-structures in programs (eg. linked-lists)
- Software tools to build or debug the programs (eg. cc, make, sdb)

Typically this "integrated" approach can only be considered in long-term courses where the student can devote much of their time to such a study; for example, a three year bachelor degree in computing science. Obviously, from an engineering point of view a practical approach would give the student a wealth of experience. This could result in a better understanding of the theory.

One issue which prevents the use of an *integrated* approach at Birkbeck is that the MSc conversion course in Computing Science accepts students from a wide variety of backgrounds. Not all of them meet the above requirements. In a year of full-time (or two years part-time) education they must develop an appreciation of all these pre-requisites to understand the process of software development. The full-time students sit the course on operating systems in parallel with other subjects, whereas the part-time students start operating systems in the second year.

Another issue is the number of lecturing hours for the course. Students have 20 hours of lectures - 15 lectures at 80 minutes each. Typically, a course which combines the theoretical and practical approaches to studying operating systems programs would need considerably more time than is available for our students; probably between 40 to 60 hours of lecture time and a number of practical sessions.

As a result, the *theoretical* approach was taken for operating systems teaching at Birkbeck, with course materials heavily based upon [2,6,7,9]. In this way students can refer to these texts for an alternative explanation of any particular area should they become stuck.

Operating systems comprise a number of sub-systems, most of which are complex. One of the problems we find when teaching operating systems to students is the problem of synthesizing the concepts about individual sub-systems back into a coherent picture of how an operating system works, e.g. how various components of an operating system interact to serve a user's request.

Unfortunately, this study of operating systems behaviour, crucial to the students' understanding of whole subject, is not really one of the issues that most text books treat in any depth. They tend to offer a static analysis of the system, breaking it down into smaller and smaller components but then forgetting to put them back together again to show the student how they interact, e.g, when executing a user command.

This deficiency has led us to develop a graphical tool to supplement the theoretical approach. We believe that one of the advantages in using a graphical tool is the efficiency with which the main concepts are transmitted. The display provides an animated picture of the major components of the operating system. With only a little theory and computing experience the student can appreciate how the basic components co-operate to perform the overall system operations. We call this tool DYNAMIX and introduce it next.

3. The Development of DYNAMIX

DYNAMIX started life as an MSc dissertation by Grabczewski, supervised by Liu [4]. It was developed largely for the purpose of teaching. We felt it would prove useful to students of operating systems and was unlike any other tool available at the time.

The approach we took was to base DYNAMIX on the MINIX operating system. By modifying and extending the kernel programs, written in C, we found it possible to inform the user about the internal state and behaviour of the operating system as it runs.

The MINIX implementation is influenced by more recent thinking about operating systems design. This design is based upon the old concept of a layered software architecture for operating systems and the newer concept of using client-server processes within the kernel itself. This implies a set of kernel processes which communicate with one another by passing messages.

DYNAMIX works by intercepting any messages and interrupts. It displays these in a graphical way on a character-based terminal.

The graphical display for DYNAMIX shows four layers. The top three are MINIX processes and the bottom layer is the kernel. Messages are passed among the top three layers and the mechanism for passing messages is implemented in the bottom kernel layer [Figure 1].

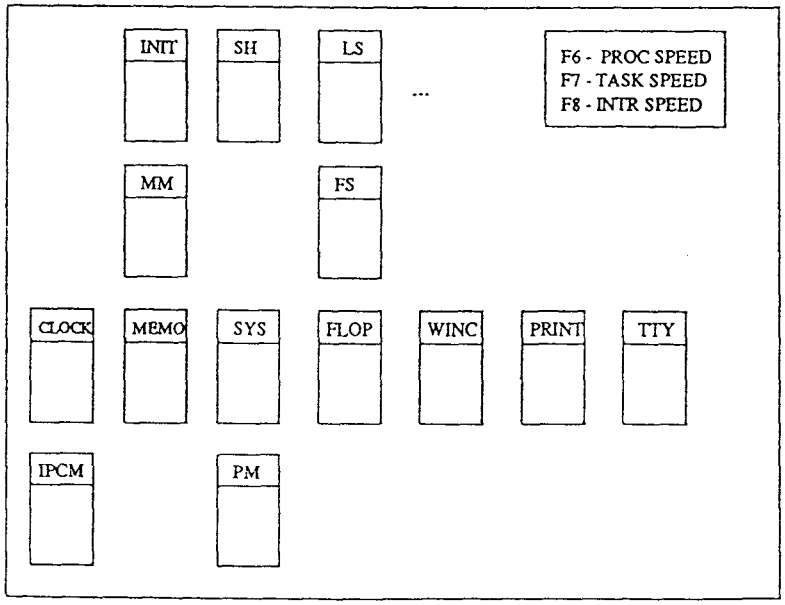

FIGURE 1: THE DYNAMIX WINDOW

3.1. MINIX

The architecture of MINIX is based upon a traditional onion skin model, building up layers of functionality. The bottom layer is the most fundamental and the higher layers give more and more value to the system by a hierarchy of virtual machines. Eventually

we reach the top layers, which are closest to what we think of when we use a computer. It comprises application programs, command interpreters and tools such as compilers.

The layered architecture is as follows:

l. User Processes
2. Server Processes
3. Input/Output (I/O) Processes
4. Kernel

Contained within each layer are the MINIX processes shown in Figure 1. The top layer shows those MINIX user processes, created by the user either directly or indirectly. The only exception to this is the "init" process, which is created by the MINIX operating system when it starts or "boots".

Server processes include the Memory Manager and File system. These processes handle all system calls made by user processes.

The I/O processes are responsible for driving various I/O devices such as disks and terminals. Tanenbaum also calls these processes "tasks".

The kernel is a single controlling process. It comprises the Process Manager (PM) and the Inter-Process Communication Manager (IPCM) functions.

User processes and server processes are preemptable by the MINIX scheduler whereas the I/O processes are not. This means that once an I/O process is allocated to the processor it uses it for as long as necessary without interruption from other processes, including other I/O processes.

Logically, the highest level of control in the operating system rests with the kernel. It is possible to think of it as sitting above all the MINIX processes at some meta-level, from which it takes charge of the underlying hardware and software. Unfortunately the layered architecture model for MINIX inverts this, implying that the processes in the top layers have ultimate control. Since this can confuse students we normally show them a diagram to help visualise the controlling forces within the operating system [Figure 2].

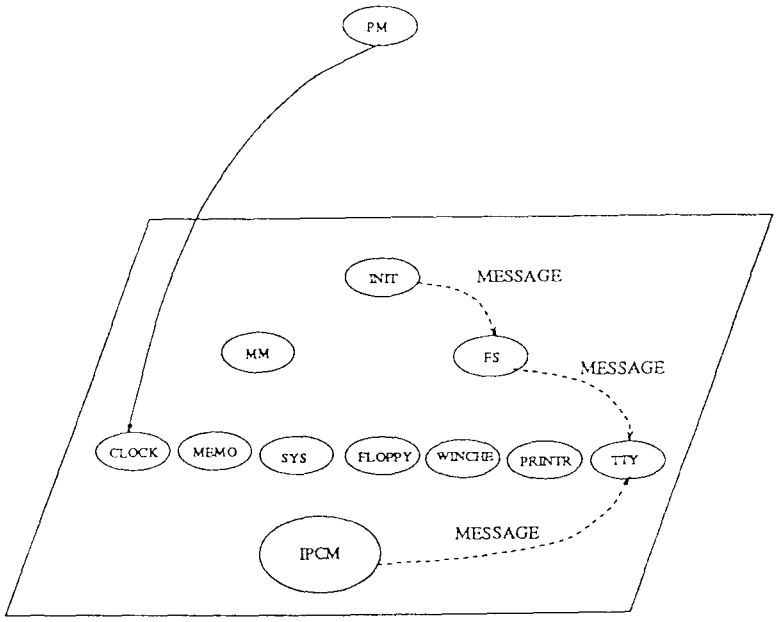

FIGURE 2: MINIX PROCESSES

Interrupts are displayed entering the system at the bottom left-hand corner of the screen. They are handled by the Inter-Process Communication Manager (IPCM) module and converted into messages for one of the MINIX processes.

The Process Manager (PM) is simply a name given to the scheduling functions which model the current state of all MINIX processes on the system.

MINIX processes interact in only one way, by passing messages to one another. A message usually contains control data and raw data in some specific format.

Any MINIX process can both send and receive messages using a special protocol incorporating the rendezvous mechanism. Hence, if a message is sent to a MINIX process that is already processing another one, the sender waits for the receiver to finish processing the previous message. Conversely if a receiver finds there is no message from a sender, it waits until one arrives. At some point in time the sender and the receiver meet to pass-on the message, hence the term "rendezvous".

An important concept to grasp is the way messages are transformed into interrupts and I/O data and vice versa, depending on whether a process is sending or receiving messages. This is done by I/O processes when they wish to communicate with devices in the external world. Thus one of the main functions of DYNAMIX is its ability to intercept messages and interrupts and to display them in a meaningful way on the screen.

3.2. Extending MINIX to give DYNAMIX

The standard MINIX operating system routines for message passing, terminal output and input need to be modified to create the DYNAMIX display. Moreover, a number of control functions need to be provided for the use of DYNAMIX in the classroom. Below we outline these developments and the details can be found in [4].

3.2.1. Displaying Messages and Interrupts

Let us suppose that the file system (FS) server process sends a message to the memory (MEMORY) I/O process. DYNAMIX displays this message in a concise notation on the screen, as follows:

Example	Field	COLOUR
FS	Sender Name	CYAN
1	Sender Process Number	GREEN
MEMORY	Destination Name	CYAN
DISK_WRITE	Function requested	CYAN
RAM_DEV	Function parameter	CYAN

Table 1: Sending Message

Table 1 shows that FS is the sender of the message and MEMORY is the receiver or destination of the message. Each process has a process number in MINIX, as well as (but distinct from) a process identifier. The FS process wishes to write to a disk in memory, which is shown by the requested function, DISK WRITE. An additional parameter is passed to the MEMORY process telling it which particular memory device is to be written to; in this case the RAM disk. The colour of the message on the screen, cyan, shows

the message is being sent, while green is used to indicate a process number.

To display this sent message we have embedded a DYNAMIX C function, dmp_smess(), inside the standard MINIX message sending function mini send(). Likewise, whenever a process wishes to receive a message it displays this in the following way:

Example	Field	COLOUR
CLOCK	receiver Name	RED
-3	Receiver Process Number	GREEN

Table 2: Receiving Message

Table 2 shows the CLOCK process is ready to accept messages from any other process on the system. The process number is shown again in green and the colour of the message on the screen is red to indicate that the process is ready to receive a message.

The received message is displayed by embedding another DYNAMIX function, dmp_rmess(), inside the MINIX message receiving function mini_rec().

Finally, any incoming interrupts or traps are displayed by embedding the DYNAMIX function dmp_interrupt() inside the MINIX C functions interrupt() and sys call(). The interrupts and traps are shown in the bottom left-hand corner of the display.

3.2.2. Displaying Graphics

Under normal circumstances, if a user process wishes to print some text to the terminal screen it will call a C runtime library function such as printf(). This would involve the user process passing a message to the file system (FS) server process, which in turn would pass a message to the terminal (TTY) I/O process requesting that some characters are output to the screen.

As we can see, a number of messages are passed among operating system processes. These would all be visible on the DYNAMIX screen, a fact which prevents us from using any runtime library routines to print characters when implementing DYNAMIX. Instead we use the MINIX printk() function, which outputs directly to the terminal video RAM, thus allowing output to be displayed on the screen without affecting the normal operation of MINIX.

The terminal in MINIX supports a subset of the ANSI X3.64 escape sequences which allows setting colour attributes and positioning text on the screen. Special characters can be embedded in strings to control the screen layout.

The following example defines a C pre-processor macro with arguments. This positions the screen cursor at a particular row and column:

```
#define POS(r,c)        " 33[%d;%dH",r,c

printk(POS(20,12));
```

This is the basic technique used to create the DYNAMIX screen. Other macros are defined where necessary. None of them are essential, they simply aid program readability.

The next example is taken directly from the DYNAMIX source code and shows the function responsible for displaying the list of blocked processes owned by the Process Manager:

```
#define LABEL(lab)          "6s",lab
#define VALUE(val)          "6s",val
#define CLEOLN              " 33[OK"
#define COMMAND_HOME        " 33[24;1H"

PRIVATE dmp_blkd_list()
{
   /* display blocked process list */
   printk(POS(22,21));
   printk(LABEL("BLKD_L:"));
   printk(CLEOLN);
   print_blocked_queue();
   printk(COMMAND HOME);
```

When dmp_blkd_list() is called it positions the screen cursor at row 22, column 21 and prints the string "BLKD L:". After clearing the rest of the current line it calls a function to display the blocked process numbers. Finally it positions the cursor at the bottom left hand corner of the screen and waits.

3.2.3. Controlling Graphics

Special function keys are defined in DYNAMIX to help control the display. The following functions have proved to be useful for classroom teaching:

- User process message-passing speeds [F6]/[Ctr1+F6]
- Server/task process message-passing speeds [F7]/[Ctr1+F7]
- Interrupt/trap speeds [F8]
- Disable/enable clock interrupts [F4]
- Stop/start the demonstration [F10]/[F9]
- Redraw DYNAMIX screen [Ctrl+F5]
- Enter/exit DYNAMIX screen [FS]

4. The Use of DYNAMIX in the Classroom

In this section, we discuss how DYNAMIX can be used to help lecturers explain the dynamics of operating systems behaviour. In particular, we address the questions of *when* it should be used, what can be demonstrated, and how the demonstration can be given. We shall also consider the feedback from students and discuss the pros and cons of DYNAMIX.

DYNAMIX can be used at various points in the course to supplement the teaching material, especially at the beginning and end of a course. At the start of the course DYNAMIX can be used to help students get a feel for the complexity of a real operating system and how its various components interact when serving a user's request. This initial impression of the dynamic system behaviour can then be used as a point of reference to clarify ideas and help unify concepts throughout the course.

To achieve this objective, there are a number of things we need to do during the pre-course demonstration:

- To introduce the basic components of an operating system
- To show how the components co-operate to do some useful work
- To demonstrate the concept of an interrupt
- To explain the concept of a user process and how they are scheduled

We also envisage that another demonstration of DYNAMIX at the end of the course could help students consolidate many of the topics, add a behavioural dimension to the theory and obtain a coherent picture of how various parts of operating systems work.

To achieve this in the post-course demonstration we need to:

- Understand the main functions performed by the components
- Explain how the components co-operate (by messages)
- Explain what the messages are and what they mean
- Explain the clock interrupt cycle and how time is kept
- Show how interrupts are converted into messages for a component
- Demonstrate preemptive scheduling of processes and quantum expiry

The post-course demonstration is fairly comprehensive, showing how a command is typed, read, forked, executed, scheduled and run. Much of the demonstration consists of an explanation of selected messages passed among components of the operating system. This gives the student some idea about how the system works at the interface level. A more detailed examination of the system would reveal how each message is processed by a component and would lead naturally to a study of the operating system programs themselves. At the time of writing we are about to give a post-course demonstration and should be able to report its impact on our students to the conference.

Below, we outline the main issues involved in the pre-course demonstration given in the second lecture. In the first lecture, students were given some appreciation of what operating systems are and why we need them.

The lecture was given in the college lecture theatre where a colour video (Barco) system is used to project the images from the PC onto a big screen. A DYNAMIX demonstration can be run on an IBM PC XT/AT compatible computer with the following minimum configuration:

- Colour EGA/VGA monitor
- 640 KB of RAM
- Chip compatible hardware

Having explained the components of MINIX, the graphics of DYNAMIX, and other important concepts such as interrupts and processes, the pre-course demonstration was primarily concerned with studying how a user command is executed and of how various components of MINIX interact with each other during execution. This demonstration was brief and students were asked to take particular note of the operating system components, the scheduling queues and how interrupts bypass the normal processing mechanism.

Following our pre-course demonstration we questioned students about their initial impression of DYNAMIX. After explaining what we hoped to achieve by using it, we asked them if they felt it met the objectives.

We had 39 responses; 12 from the part-time and 27 from the full-time students and classified the responses into the following five categories.

GROUP	RESPONSE
1.	Entirely positive
2.	Positive but with remarks on demonstration speed
3.	Positive and with some helpful suggestions
4.	Neutral and with some helpful suggestions
5.	Entirely negative

Table 3. Group Classification

The results of the pre-course demonstration are summarised in Table 4.

GROUPS	PART-TIME	FULL-TIME	TOTAL	%
1	0	4	4	10
2	1	3	4	10
3	6	14	20	51
4	4	4	8	21
5	1	2	3	8
All	12	27	39	100

Table 4. Feedback from the Students

We consider the first three groups to be a positive response and so conclude that 71% of the students found our DYNAMIX demonstration useful at the start of the course. Of the remainder, 21% were neutral but made some suggestions to improve the demonstration and 8% reacted negatively.

On the whole, the results are encouraging. Apart from creating a lasting impression on students of how a real operating system works, DYNAMIX can also be used to support some form of interactive learning sessions. For example, at any stage the student is encouraged to experiment with and ask questions about DYNAMIX, to explain what has just happened and to predict what might happen next. For example, the student might be asked:

- What components will become active when we mount a floppy disk?
- Let us see what happens when I print a file
- Will the system hang if I try to kill the init process?

Our students were also particularly helpful with comments about the pre-course demonstration and we have noted them for future improvements. Several points have arisen from this demonstration.

Firstly, the pace of the demonstration should be sufficiently slow to allow students time to adjust to the DYNAMIX screen. More time is needed to familiarise students with the screen layout just prior to the demonstration. This could be incorporated into the preparatory talk just before the demonstration.

Secondly, the demonstration equipment needs to be chosen carefully to give a satisfactory performance. One of the major problems is the choice of IBM PC. There are many hardware configurations which will not work with MINIX. For example, MINIX 1.3 will not run on the latest IBM PS/2 hardware. Even on compatible hardware there can be problems with the demonstration speed.

Thirdly, the clock interrupt cycle needs to be carefully tuned to provide a fluid flow of messages in the DYNAMIX display. This is something of an art at the moment.

Finally, there is a need to filter and abstract some of the current output of DYNAMIX to suit the particular demonstration so that lecturers don't have to show irrelevant details. For example, some of the messages sent in MINIX do not need to be shown for many demonstrations.

These concerns have led to a number of possible improvements in DYNAMIX which are discussed below.

5. Potential Improvements

There are a number of ways DYNAMIX can be developed further.

Hardware independence: By implementing the DYNAMIX concept in MINIX we have constrained ourselves to a particular type of hardware. It would be preferable to implement DYNAMIX on a more hardware independent platform. One approach is to embed DYNAMIX into an operating system which itself runs on a virtual machine. Switzer [8] has used this approach to develop an operating system kernel that runs on a standard UNIX System V Release 3.2 operating system. This allows a level of portability not available on the current system.

Extra Functionality: The virtual machine upon which MINIX and DYNAMIX run could be extended to provide a memory paging mechanism. This would allow us to demonstrate the concept of virtual memory to students. Process swapping should also be incorporated into the operating system.

Extended graphics: By implementing DYNAMIX in an operating system that runs on a virtual machine, such as the UNIX operating system, we can use any graphics facilities available on the virtual machine system (eg. X) to create a clearer graphical model of the operating system internals.

Towards a Tutoring System: In its present form DYNAMIX is an assistant or tool to aid the teacher transmit concepts effectively. A long-term objective of this work would be to move DYNAMIX into a tutoring environment providing an interactive learning program for students without the aid of a real teacher. This tutoring system can then be made available for students to study at their own pace.

A considerable amount of work needs to be done to develop such a system. One of the pre-requisites of developing a tutoring system for operating systems is to develop a simulator that models operating system components at various levels of detail and abstraction. Our experience of using DYNAMIX in the classroom has given us a starting point from which to analyse the requirements for such a simulator and tutoring system. This analysis of requirements should always be the first step to developing a tutoring system.

6. Concluding Remarks

In this paper we have described the development of DYNAMIX, a tool to assist in the teaching of operating systems. DYNAMIX was developed to meet the real needs of students who were finding the subject complicated to learn in the short time available.

The graphical display of DYNAMIX comprises an animated screen showing all the relatively static processes of a real operating system and their dynamic interface interactions. We have some evidence to show that DYNAMIX helps students to comprehend the subject of operating systems, especially in obtaining a complete picture of how an operating system works. This is so because the key concepts are already abstracted and visualised for the student in the display itself. This allows the student to literally see the ideas they are expected to grasp. The display is a concise way of representing a lot of information all at once and the use of vision allows us to do this without overwhelming the student with details.

This method is contrasted with the pure textbook approach where the student is given a detailed analysis of operating systems theory and expected to synthesize the mental model for themselves. Unfortunately, there is no guarantee that all students will come up with the same synthesis. This is sometimes viewed as an advantage, especially in the arts; but in the precise world of technical subjects it can be an obstacle to learning.

References

1. Comer, D and Fossum, T V, *Operating System Design: The XINU Approach,* Prentice-Hall, PC Edition, 1988.
2. Deitel, H M, *Operating Systems,* Addison Wesley, 2nd Ed, 1990.
3. Fortier, P J, *Design of Distributed Operating Systems - Concepts and Technology,* McGraw-Hill, 1986.
4. Grabczewski, E, *DYNAMIX - A Window into the Internals of MINIX*, MSc Dissertation, Birkbeck College, London, 1991.
5. Leffer, S J and etc., *The Design and Implementation of the 4.3 BSD UNIX Operating System,* Addison Wesley, 1989.
6. Lister, A M and Eager, R D, *Fundamentals of Operating Systems,* Macmillan, 4th ed 1988.
7. Silberschatz, A, Peterson, J, and Galvin, P, *Operating Systems Concepts,* Addison Wesley, 3rd Ed 1991.
8. Switzer, R, *Operating Systems - A Practical Approach,* Prentice-Hall, 1993.
9. Tanenbaum, A S, *Operating Systems - Design and Implementation,* Prentice-Hall, 1987.
10. Tanenbaum, A S, *MINIX 1.3 Binaries and Sources for IBM PC's,* Prentice-Hall, 1988.
11. Tanenbaum, A S, *Modern Operating Systems,* Prentice-Hall, 1992.