Chapter 2. Background

Over the last few decades, several operating system courseware systems have been made. We classified these courseware into two categories: *kernel-level courseware* and *user-level courseware*. In this section, we explain kernel-level courseware and user-level courseware, and illustrate related researches, and then we briefly explain several educational operating systems which are used at the campus. Finally we briefly describe how the two categories in networking concept are applied.

2.1. Kernel-level Courseware

One main approach to providing hand-on experiences in operating system courses is implementing educational operating systems in kernel-level, we called it as kernel-level courseware. Kernel-level courseware are designed to run directly on a bare machine, such as Minix [28], Xinu [10], etc. Kernel-level educational operating systems require running in supervisor mode, so students can have experiences in real-world operating systems design and implementation.

Kernel-level educational operating systems contain essential operating system components since these are run on top of bare machines. Kernel-level courseware also provides several kernel-level programming projects related to complex concepts in textbook. Kernel-level projects can provide hands-on experience which is design and modifying a real operating system, so students can understand the complex concepts more clearly. In additional to understanding the complex concepts clearly, when student learn how to use the kernel-level courseware, they also have to study some issues which do not get a mention in textbook, such as bootstrapping, handling interrupts, device

driver. In other words, kernel-level educational operating systems are the same as commercial operating systems. Kernel-level courseware is a small real-world operating system which provides hands-on experience for learning operating system course.

Because Kernel-level educational operating system is a real operating system, it needs to run on a bare-machine. Every student must have a bare-machine to run the courseware. The laboratory would require providing every student with their own machine on which to run the courseware, which is impractical in most university setting and does not scale to large class sizes.

Furthermore, kernel-level courseware is dependence on hardware, whole courseware contain a large percentage of code which related to hardware. Students need to understand these hardware knowledge if they want to realize the source codes of the courseware. Moreover, kernel-level educational operating system is not easy to operate, it typically provides students with a manual book as a start, another text book indeed. Students need to read the manual book to realize how to operate the courseware. With another text book introduced, the learning curve is undoubtedly steepen.

Additionally, kernel-level educational operating system contains a great many operating systems related details, such as boot strapping and device driver, which are not mentioned in operating system textbook, so the courseware have a large of source codes. A large of source codes may make student confused because they need much extra time to find the concept which they want to understand. For undergraduate students, a huge and complex educational operating system may make it easy to lose sight of key ideas in a forest of details.

Furthermore, kernel-level courseware is difficult to debug. Students need learn how to debug a kernel-level program because they are unable to use their practiced debugger tools, such as *gdb*, to debug their kernel-level projects.

2.2. User-level Courseware

Another approach is designed educational operating system in user-level, we called as user-level courseware. User-level courseware does not run directly on a bare machine, it designed to run in unprivileged mode, as the same as general application program. User-level educational operating system provides a simpler courseware installation than kernel-level courseware. Students could modify and rebuild the courseware as the same as run an application program, so students can learn how to operate the courseware quickly. Furthermore, user-level courseware does not involve all operating system components completely, it just keep several essential components which be mentioned in textbook. User-level courseware divides the system into several modules, each of which performs an operating system concept, such as CPU scheduling, memory management. While students study a concept in textbook, they can read and hands-on the corresponding module directly without understanding other concepts. User-level courseware can be differentiated between with simulator and without simulator.

User-level courseware with simulator provide a platform which simulates hardware, the course ware can run directly on the simulator. Using hardware simulator can isolate user-level courseware from low-level machine-dependent concerns, such as device interrupt and driver. The simulator usually uses a set of simulation parameters to decide the simulation environment, such as the cpu time quantum and share of resource events. In addition, the simulator of some user-level courseware may calculate utilization of cpu time, I/O activity, and other resources, so it may accumulate statistics that can be used to estimate the performance of the system modified by the students.

User-level courseware without simulator is more briefly than with simulator, it is no dependence on any hardware platform or simulator. It does not build a “real”

educational operating system, For instance, Awk-Linux[30], use program instrumentation tool to simulate several hardware functionalities. Without understanding how to build a operating system, students can directly raise hands to do the projects which related to essential concept in textbook.

2.3. Kernel-level Courseware Versus User-level Courseware

The two main approaches both have advantages. Kernel-level courseware provides hands-on kernel-level project experience with a real operating system, and user-level courseware provides a simpler and more concise system than kernel-level courseware, which does not need to spend time learning how to operate and it is not difficult to porting or maintain user-level courseware.

In the case of an academic term, using kernel-level courseware to teach the concepts in operating system textbook is unrealistic, because students have to spend much time to learn how to use the kernel-level courseware. In additional to learning how to operate is more difficult, porting and maintenance of kernel-level courseware are also significant problems. Laboratories may need porting the kernel-level courseware for fitting other hardware or for other reasons, because kernel-level courseware has much hardware dependence codes, revising the courseware needs spend lots of time. Although user-level courseware is not a real operating system, it contains concise essential concepts in operating system textbook and isolates the courseware from low-level machine-dependent concerns. User-level courseware expose students to essential concepts of operating system, while at the same time isolating them from the details which not mention in textbook. Because students can learn quickly how to use the courseware, they could concentrate on the essential concept of operating system in textbook. Furthermore, because user-level courseware runs as an user process, porting

and maintenance of the courseware is much more easy than kernel-level courseware, laboratories can easily modify the source code of the courseware to fit their demands. Table 2.1 is a list of some feature comparison of kernel-level courseware with user-level courseware.

Table 2.1 A comparison of kernel-level and user-level courseware

	Kernel-level courseware	User-level courseware
Demand to bare machine	Yes	No
Size of code	Big	Small
Porting cost	High	Low
Maintenance cost	High	Low
Debug tool	Kernel-level debugger	User-level-debugger
Difficulty	High	Low

2.4. Several Well-known Courseware

Over the past decades, there are many instructional operating systems have been used to assist in operating system course. In this section, we briefly explain several well-known kernel-level and user-level courseware which are used in operating system course at campus.

2.4.1. XINU

Xinu [10] was developed in the mid-1980's at Purdue University. It was originally

developed for teaching operating system course. Xinu [10] is also well-suited for teaching advanced programming concepts, such as IPC, Client-Sever Interaction, etc. The educational operating system was described in two textbooks, the second being devoted to internetworking and TCP/IP.

Xinu [10] “includes all the components that constitute an ordinary operating system: memory management, process management, process coordination and synchronization, inter process communication, real-time clock management, …, and a file system” [Comer].

The system supports device-independent abstract input/output operations, such as init, open, close, read, write, on a number of physical and logical devices. The courseware was designed initially for the Digital Equipment Corporations LSI-11 computer. Now there are many versions which can run on other hardware, such as SPARC, SUN 3/50, and x86 architecture. Major advantages of Xinu [10] are its simplicity, in comparison with commercial operating system, and that the source code can be modified as needed. The details of the system of Xinu [10] are illustrated in Chapter 3

2.4.2. MINIX

Minix [28] is an open source, Unix-like operating system. The system developer is Andrew S. Tanenbaum. Minix [28] was originally developed for compatibility with the IBM PC and IBM PC/AT microcomputers. Later version was also ported to the Motorola 68000, SPARC, and x86 architecture. The system was designed for teaching operating system course. The architecture of Minix [28] contains four layers: process management, task, servers, and user process. Figure 2.1 show four layers of the architecture of Minix [28]. Layer 1 and layer 2 are kernel-level layer, layer 3 contains

memory management and the file system, layer 4 contains a unix-like process hierarchy rooted at process 1. Major advantages are the same as Xinu [10], Minix [28] provide a simplicity system, in comparison with commercial O.S. and a hands-on system O.S for teaching operating system course.

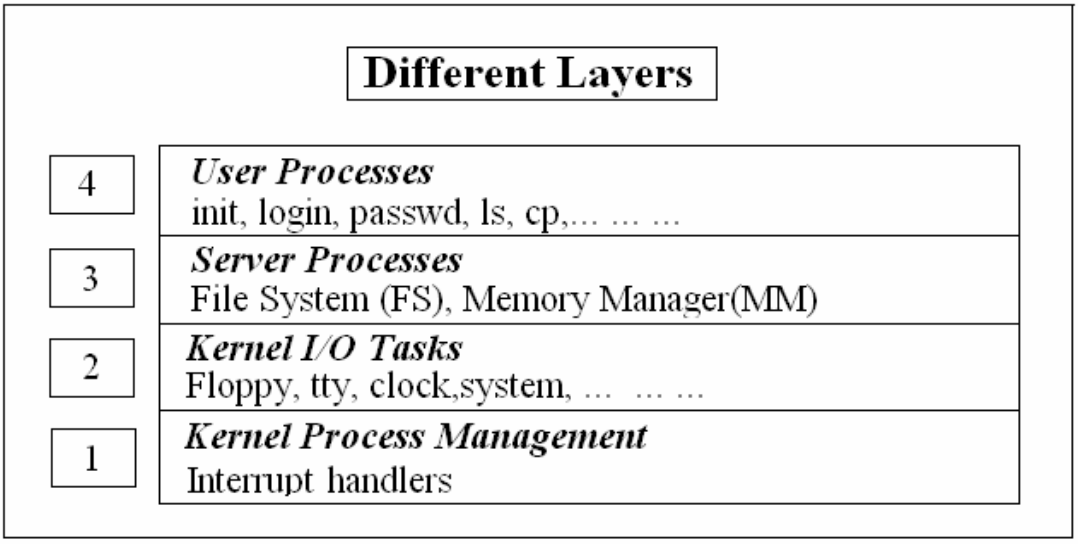

Figure 2.1 The architecture of MINIX

2.4.3. NACHOS

Nachos [26] is an user-level instructional operating system developed by Thomas Anderson at University of California, Berkeley. Nachos [26] is a simple baseline code courseware for a working operating system. The system was originally developed in C++ and runs on its own simulator which simulates MIPS instruction set. For this reason, only the user programs running under Nachos [26] need to be compiled for the MIPS instruction set. Figure 2.2 shows the major part of the architecture of Nachos [26]. The operating system accesses the simulated hardware through various C++ classes. The assignments are intended to illustrate and explore essential concepts of modern operating systems, such as threads and concurrency, multiprogramming, system calls, virtual memory. The Nachos [26] kernel and machine simulator run directly on the host

machine, students do not need to prepare a bare machine to install it. Major advantages of Nachos [26] are its much more simplicity than Kernel-level courseware, and students do not need an extra computer to use the system.

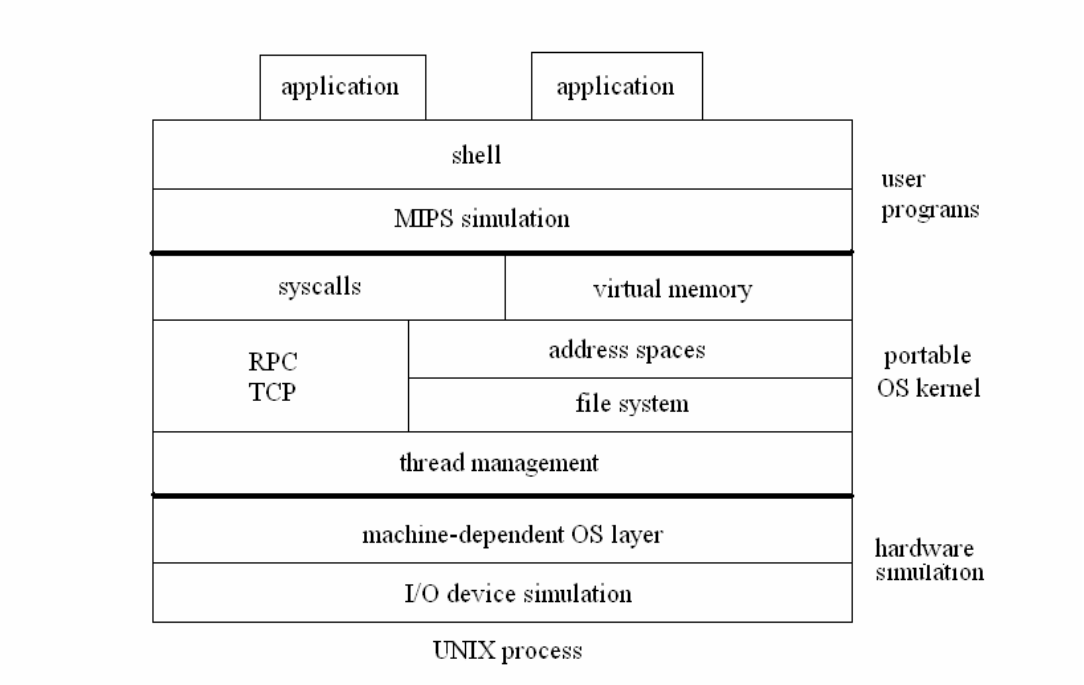

Figure 2.2 The architecture of Nachos

2.4.4. AWK-LINUX

Awk-Linux [30] is another user-level courseware, and was developed by Yung-Pin Cheng at National Taiwan Normal University. Awk-Linux [30] is different from above educational operating systems, it does not build a completely operating system. In above reason, the source code of Awk-Linux [30] is the smallest than other above courseware. Awk-Linux [30] use Awk, which is a UNIX string matching processor, as a program instrumentation tool to simulate several hardware functionalities, and according the instrumentation tool to design several essential concepts in operating system textbook, such as CPU scheduling, virtual memory, etc. Program instrumentation is a software engineering technique which inserts additional code to a

program. Awk-Linux [30] uses program instrumentation to simulate hardware functionalities, so it is no dependence on any hardware simulator or platform. Students execute the courseware in common with general user programs, so they do not spend time and effort to constructing the courseware as the same as above courseware. Because Awk-Linux [30] just implements operating system fundamental components on a needy basis, the courseware is the simplest than other courseware. The disadvantage is that student cannot get a glimpse of a full view of real operating system.

2.5. Networking course

Networking is an important concept in operating system. Because networking is a very large and complex concept, computer science department involve networking as a semester course in computer science curriculums. TCP/IP protocol is a popular communication protocol in Network, thus many networking courses choice TCP/IP protocol to introduce networking. As other operating system concept, students use hands to do some project results in better understanding. TCP/IP protocol consist of many protocols, such as *arp*, *ip*, etc, which run in the same place, but almost all network textbooks discuss each protocol independently, without considering how multiple protocols run together. Students could not understand clearly the TCP/IP technology from textbook because of "The TCP/IP technology comprises many protocols that all interact. To fully understand the details and implementation of a protocol, one must consider its interaction with other protocols in the suite." (comer, internetworking with TCP/IP)

Above educational operating systems are not appropriate for TCP/IP course, Xinu [10] and Minix [28] excepted, others do not mention to TCP/IP protocol completely.

Although Xinu [10] and Minix [28] provide complete TCP/IP protocol, however, the courseware are not suite in teaching Networking course because student need understand many extra knowledge about operating system. Furthermore, if we use kernel-level courseware in teaching networking course, then we need build a network topology environment to see how two hosts interact in network topology environment. It is so difficult to laboratories provide a real network topology environment because build a network topology needs many hosts.

An instructional networking system could help students to learn network concepts more clearly and particularly. For above reason, we design a courseware for teaching TCP/IP networking. We draw out the TCP/IP networking module from Xinu [10], and modify the module to become a user-level courseware. Students could use this courseware easily because they do not need other knowledge unrelated networking besides C programming knowledge. In addition, we provide a network topology simulator, which could simulate network topology environment. Students just need one Linux based computer to run the courseware. To put it briefly, we implement a user-level TCP/IP networking and network topology simulator for networking course.