A SURVEY OF CONTEMPORARY INSTRUCTIONAL OPERATING SYSTEMS FOR USE IN UNDERGRADUATE COURSES

Charles L. Anderson and Minh Nguyen
Division of Computer Science
Western Oregon University
Monmouth, OR 97361
{andersc,mnguyen}@wou.edu

ABSTRACT

An instructional operating system is one intended for use in teaching courses in operating systems. Student programming assignments develop or enhance the instructional operating system. This paper presents an overview of the issues related to using operating systems, especially instructional ones, in coursework. We also present results of a survey we conducted to determine which instructional operating systems are in use at campuses in the United States.

1. INTRODUCTION

When teaching an undergraduate operating systems class, the instructor needs to decide what type of assignments to give students. Assignments can be purely theoretical or book-based: questions and answers about the topics and concepts. Alternatively, an instructor may choose to provide hands-on programming assignments.

There are a number of possibilities for programming assignments. One type of assignment involves concepts related to operating systems without actually involving programming an operating system. For example, students might be asked to implement a solution to the dining philosophers problem. Alternatively, students may be given assignments to modify or develop an operating system. But which operating system should the students work on?

In this era of easy access to commercial-grade, open-source software, there are a number of choices for real-world operating systems that students can work on. The most obvious examples include Linux, BSD variants, and Darwin. Although some instructors use these systems for student assignments, there is a very real concern that a production-quality operating system is far too large for students (especially those in an introductory class) to comprehend. The complexities of the real-world implementation can easily obscure the underlying concepts that the instructor wants to present.

The obvious solution is to create a smaller operating system specifically to serve as a platform for instruction, rather than as a fully-functional, end-user operating system. We call these *instructional operating systems*. There has been a rich history of such operating systems, and there have been a number of operating systems textbooks published based on specific instructional operating systems (e.g., [3,4,10]).

Having decided to use an instructional operating system, the instructor is left with the process of choosing one. However, trying to find the "right" operating system to use in our coursework proved to be a non-trivial task. We could find no evaluation of contemporary instructional operating systems in the literature, although the topic had been discussed in a panel at the thirtieth SIGCSE technical symposium [7].

This paper is the result of our investigation. The second section discusses attributes of instructional operating systems. In the third section, we present the process we followed to identify instructional operating systems. The fourth section contains our findings about the instructional operating systems.

2. INSTRUCTIONAL OPERATING SYSTEMS

Platforms

The hardware on which an operating system runs is known as a *target platform*. Possibly the most significant component of the target platform is the CPU. Obviously, the most common CPU that students have contact with is the Intel IA-32 (or x86) processor found in most PCs today. The Intel architecture is blessed with a very diverse set of both commercial and open-source tools for developing software.

However, due in part to its CISC history and its roots with a segmented memory model, the IA-32 is rather complex. Students can be confused or bogged down by the complexities of the instruction set. Therefore, some instructional operating systems have chosen to target other architectures: The MIPS family of processors is a common choice. However, authors of the instructional operating system OS/161 [8] expressed some regrets about choosing the MIPS architecture due to a dearth of high quality, publicly available tools.

Regardless of the CPU instruction set, the target platform need not be actual hardware. With the advent of fast, modern processors, complete instruction-level simulation is well within the capability of most desktop systems. A major benefit that this provides is the opportunity to shield the students and the instructional operating system from unnecessary hardware complexity. Some complexity and realism can be a good thing; too much realistic complexity becomes an obstacle to learning.

Nachos [5] and OS/161 both run on simulated MIPS processors. The RISC architecture is comparatively simple to emulate. The simulators run under the *host* operating system on the host architecture–e.g., a desktop system running Windows on Intel. The simulators are written in high level languages (e.g., C/C++), which allows the simulator to run on almost any host platform.

Nachos and OS/161 take significantly different approaches in the relationship between the instructional OS and the simulator. In Nachos, the simulator and the OS are part of the same program under the host operating system. This means that the operating system is ultimately machine code for the host architecture, and only the programs running under Nachos need to be compiled into the target architecture's machine code (MIPS). In other words, assuming that the host platform (e.g., Windows or Linux on Intel) is different from the target platform (MIPS), a cross-compiler is only needed for the programs that run under the instructional OS. In fact, an instructor could provide those executable programs to the students to save them from having to install the cross-compiler (e.g., on their computers at home).

In contrast, OS/161 runs on a simulator called System/161, which is separate from the operating system. The operating system code is machine language for the target architecture (MIPS). This implies that a cross-compiler is needed both for the programs that run under the operating system, as well as the operating system itself; students cannot avoid having to use a cross-compiler.

GeekOS [9] takes a unique approach to the issue of simulated vs. real hardware by specifically supporting both. GeekOS can run on real hardware (Intel-based PCs) by booting from a floppy disk. This provides complete realism with all of the issues that come with it. However, GeekOS is also intended to run under the Bochs [2] simulator. Bochs provides a nearly complete emulation of the IA-32 instruction set, as well as a number of peripherals available on PCs. It runs on Intel and non-Intel platforms including Mac OS X and Sun Solaris. In the case where the host environment is an Intel processor, the host and target architectures are the same, so no cross-compilers are needed.

Language

The language that the instructional OS is written in can be a significant factor in choosing an OS. Most instructional operating systems, as well as commercial ones, are written in C/C++ (with some assembly language). However, some undergraduate computer science programs, such as ours, have moved away from C/C++ and provide most or all instruction in Java. Expecting such students to learn C while studying operating systems can be a considerable hurdle, especially in a ten week quarter.

Fortunately, there is an implementation of Nachos written in Java. The operating system including the code written by students and the MIPS simulator are both written in Java. Programs that run under Nachos are typically written in C and compiled into MIPS machine language. However, as noted above, an instructor could provide those executable programs to students, which would shield them from C, in addition to saving them from having to install and use the cross-compiler tools.

OS Functionality versus Student Projects

Within instructional operating systems, there are two basic philosophies with regard to how much functionality is given to the students and how much is developed by the students. In the first case, the OS as first presented to the students is fairly complete and fully functional. Examples of this include Minix [10] and Xinu [3]. Student work can be modifications to existing code such as modifying the scheduler or adding new instances of components already in the OS–e.g., adding a new type of file system.

Most recent instructional operating systems follow the second philosophy where students are provided with a very minimal skeleton of the OS, and they write all the major components such as process management, file systems, and virtual memory. Examples of this include Nachos and GeekOS. In some cases, very trivial implementations of these components might need to exist to support even the skeleton functionality (e.g., a read-only FAT-like file system to boot from), but the students' work would produce something much more functional.

Development Environment

As noted above, most instructional operating systems are written in C/C++ and some assembly language. These are typically built with the GNU tool chain, possibly using the NASM for Intel assembly language. Often the assumption is that the operating system is built under Unix or Linux. This can be a problem for students who lack experience with Unix-like operating systems or for students who want or need to work off campus, but do not have a machine on which they can run a Unix-like operating system.

A possible option for such students is to use the Cygwin [6] emulation package under Windows. This provides the GNU tool chain including compilers and the *make* utility. Well-structured "make files" can be adapted to work well under Cygwin. Some instructional operating systems provide such make files; in other cases, the original make files can be adapted.

A notable exception to Unix-centric development environments is PortOS [1] that targets Windows platforms. It is intended to be built under Microsoft Visual Studio, which can cause its own set of problems. For example, we had a different version of Visual Studio than the authors of PortOS used, and thus we had some problems compiling the source code.

3. SURVEY METHODOLOGY

We employed a multi-pronged search for instructional operating systems and institutions using them. We identified most of the operating systems in this paper through a traditional literature search. We then searched the Internet for institutions using the specific operating systems that we had identified.

However, we also wanted to find instructional operating systems that have not been described in the literature – i.e., some under-documented operating system with an underground or cult following. We used an online listing of the 100 top Computer Science universities [12] and visited each school's web site in an attempt to discern what, if any, instructional operating system they were using. We also added schools that we knew of that were not on the list to those that we searched. When we identified a new operating system that we had not previously heard of, we then searched for other institutions using it.

Between the two techniques (using the OS names to search for schools and looking up schools directly), we came up with a sample of 98 institutions.

4. SURVEY

In this section we present our findings about what type of hands-on assignments are used at the campuses we surveyed. We also provide some basic information about some of the more common instructional operating systems. Below is a table of our findings for the 98 samples. (The reported numbers do not sum to 100% because we omitted samples below 3%.) "Unix-oriented" assignments typically involve writing programs that use the Unix API. "Java-based without an OS" refers to programming concepts related to operating systems (e.g., concurrency) in Java without using an (instructional) operating system – just the Java language and API.

Programming	**Usage**	**Programming**	**Usage**
Unix-oriented	27%	OS/161	5%
Nachos	17%	Xinu	4%
Linux kernel	14%	GeekOS	3%
Java-based without an OS	8%	JOS	3%
Minix	7%	Yalnix	3%

As one can see, instructional operating systems are actually not all that popular, and no one instructional operating system is used by a majority of institutions. Nearly half of all institutions use no instructional operating system at all (the Unix-oriented, Linux kernel, and Java-based entries in the table).

Nachos

Nachos [5] is an instructional operating system developed at University of California, Berkeley. The operating system was originally developed in C++; however, a Java version is also available. Nachos runs on its own simulator, which simulates the MIPS R2000/3000 instruction set. As discussed earlier, the simulator is co-resident with the operating system, which means that only the programs running under Nachos need to be compiled for the MIPS instruction set. The operating system accesses the simulated hardware through various C++ (or Java) classes.

Nachos is provided as a skeleton. The assignments are intended for teams composed of two students working in a 15-week semester. Student assignments develop code for thread management, file systems, multiprogramming, virtual memory, and networking. Each successive phase requires working code from the previous phase.

The original version of Nachos was about 2,500 lines of C++ code, approximately half of which were comments or interface descriptions. The simulator took another 2,500 lines of code.

Minix

Minix [10] is a complete, fully functional operating system that even provides Posix (IEEE 1003.1) compatibility. It presents a Unix-like operating system to users and programmers, and it includes many Unix utilities (from the GNU project). The first versions ran on 8088-based PCs in 16-bit real mode. Newer versions run on any IA-32 processor in 32-bit protected mode. It also runs under Bochs [2].

With all of this functionality, Minix is comparatively large: the kernel is over 30,000 lines of code, including comments. This could be overwhelming to students [9]. However, this is mitigated somewhat by the fact that Minix is based on a microkernel architecture in which operating system services are provided by separate, cooperating processes. This can help students understand the system because it is, by definition, partitioned into a number of small, easier-to-understand components.

OS/161

OS/161 [8] is a newer instructional operating system. It was developed at Harvard University based on previous experience using Nachos. The system and student assignments are heavily influenced by Nachos, but OS/161 aims to correct certain drawbacks of Nachos while maintaining support, realism and simplicity of a modern OS. The system's organization and structure is intentionally similar to BSD Unix. The base, skeletal kernel is about 11,000 lines of C code.

OS/161 runs on a simulator called System/161, which emulates the MIPS R2000 processor. System/161 also emulates a number of system devices such as disks, serial ports, and timers. As noted above, OS/161 is architecturally very different from Nachos in that the operating system runs on the simulator rather than being co-resident with it. Thus, the operating system has to be cross-compiled to MIPS machine language.

The assignments are similar to those in Nachos. There are five assignments, plus a preliminary exercise (e.g., compiling the code) intended for one semester. For the last four assignments, students work in pairs. The assignments cover synchronization, processes and system calls, virtual memory, file system, and a student project.

GeekOS

GeekOS [9] is an instructional operating system developed at the University of Marryland, College Park. It is written in C and runs on Intel-based PCs, as well as the Bochs emulator. The primary development environment is Linux (on Intel), but it can also be built under Cygwin.

GeekOS is another skeleton operating system where the students develop the functionality of the operating system through individual assignments during a semester-long course. The first project is intended just to get students familiar with building and running GeekOS. Subsequent projects have the students implement processes based on protected-mode segmentation, a multi-level feedback scheduler, virtual memory, a file system, and interprocess communication (IPC).

Others

Xinu was developed in the mid-1980's at Purdue University. It was originally developed on the Digital Equipment LSI-11 microcomputer and later ported to run under MS-DOS and MacOS (before OS X). The operating system was described in two textbooks [3,4], the second being devoted to internetworking and TCP/IP. Although it is still in use at a few campuses and has been used by companies such as Zilog, it appears that Xinu is no longer being actively developed.

Topsy [13] is an instructional microkernel operating system developed at the Swiss Federal Institute of Technology in the Computer Engineering and Networks Laboratory (TIK). It does not appear to be used at any US universities, but is used at a few campuses in Europe. It runs on the MIPS R3000 processor, either a physical CPU or a number of simulators. One pleasant feature of Topsy is that it includes an extensive (70+ pages) manual written in English.

JOS provides "UNIX-like functions (e.g., fork, exec), but is implemented in an exokernel style (i.e., the UNIX functions are implemented mostly as user-level library instead of built-in to the kernel)" [14]. JOS stands for Josh's Operating System and should not be confused with Java OS, which is not an instructional OS.

Yalnix is developed from the ground-up (i.e., without a skeleton) by students. It "includes the main process management features of UNIX, plus a few miscellaneous kernel primitives to form a complete system…. [It is not necessarily] required that Yalnix support signals or threads" [11]. Yalnix runs on an emulator called UtePC, which does not support disks or demand paging.

5. CONCLUSION

Motivated by the desire to find an instructional operating system to use at our institution, we discovered that there is no single dominant instructional operating system. In fact, nearly half of the campuses surveyed do not use an instructional operating system at all. The concept of skeletal operating systems (e.g., Nachos and OS/161) seems to be gaining in popularity over previous complete instructional operating systems (e.g., Minix and Xinu). Newer instructional operating systems can be run on software simulators to shield students from some of the complexities and inconveniences of real hardware. C is still the most common implementation language for such operating systems, although Nachos is available in Java.

Due to space constraints here, we cannot present all of the information we collected. Therefore, we have built a Wiki site with more information about these operating systems and others. We welcome additional information and comments about instructional operating systems. The URL is: **`http://cs.wou.edu/EdOS`**.

6. REFERENCES

[1] Atkin, Benjamin and Sirer, Emin Gün, 2002. PortOS: an Educational Operating System for the Post-PC Environment. *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education,* pp.116-120.

[2] bochs: The Open Source IA-32 Emulation Project. http://bochs.sourceforge.net. January 2005.

[3] Comer, Douglas 1984. *Operating System Design, the XINU Approach,* Prentice Hall.

[4] Comer, Douglas 1987. *Operating System Design Volume 2: Internetworking with XINU,,* Prentice Hall.

[5] Christopher, W. A., Procter, S. J., and Anderson, T. E., 1993. The Nachos Instructional Operating System. *USENIX Winter (1993),* pp. 481-488.

[6] Cygwin Information and Installation. http://www.cygwin.com. March 2005.

[7] Goldweber, Michael, Barr, John, Camp, Tracy, Grahm, John, and Hartley, Stephen, 1999. A comparison of Operating Systems Courseware. *SIGCSE Bulletin Volume 31, Issue 1*, pp. 348-349.

[8] Holland, David, Lim, Ada, and Seltzer, Margo, February 2002. A New Instructional Operating System. *SIGCSE Bulletin, Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, Volume 34 Issue 1,* pp. 111-115.

[9] Hovenmeyer, David, Hollingsworth, Jeffery, and Bhattacharjee, Bobby, 2004. Running on the Bare Metal with GeekOS. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education,* pp. 315-319.

[10] Tanenbaum, Andrew and Woodhull, Albert 1997. *Operating Systems: Design and Implementation,* Prentice Hall.

[11] The Yalnix Kernel. http://www.cs.utah.edu/classes/cs5460/handouts/yalnix.pdf. October 2004.
[12] Top 100 Universities of Computer Science. http://www.vyomworld.com/gre/cslist.asp. 2005.
[13] Topsy – A Teachable Operating System. http://www.tik.ee.ethz.ch/~topsy. April 2002.
[14] 6.828: Operating System Engineering. http://pdos.lcs.mit.edu/6.828/2004/overview.html. October 2004.