New Directions in Operating Systems Courses Using Hardware Simulators

Renzo Davoli
Dept. of Computer Science
University of Bologna Italy
renzo@cs.unibo.it

Michael Goldweber
Dept. of Math and Computer Science
Xavier University
mikeyg@cs.xu.edu

Abstract

Operating Systems evolve quite rapidly thus teaching syllabi, pedagogic methods and especially software tools to support the Operating Systems course must keep pace with this evolution. Starting from an enumeration of what topics need to be covered in a modern Operating Systems course, this paper presents a survey of hardware simulators already available for supporting this course. The survey includes an analysis of each tool's strengths and weaknesses and appropriate course utilization strategies. Furthermore, we present a set of research directions for future hardware simulators focusing on recognized needs not currently filled by any of the available tools. Finally, we present plans to investigate how a new simulator might meet these needs.

TEACHING OPERATING SYSTEMS TODAY (AND TOMORROW)

The role of the Operating Systems (OS) course remains a vital one. While most students will not become OS developers, there is no doubt that a good knowledge of OS's is needed for any professional role within the Information Technology and Computer Science communities. Furthermore, many important topics in the discipline have come from, and continue to be generated from OS research and development/evolution; topics which are often only covered in the OS course. Unfortunately, these same fruitful research and development/evolution efforts in OS's have broadened the corpus of knowledge needed to be covered in the OS course.

To illustrate this, the authors, who combined have been involved in teaching OS courses for over 20 years, have enumerated a topics list for an OS course (or courses). The result of this exercise is presented in Figure 1 without any guarantee of completeness. Furthermore, we are very conscious that the topics will be covered with varying levels of depth.

Interestingly, a number of items in this enumeration are not discussed at all or are mentioned in passing or short notes in the most popular texts used to support the teaching of the OS course(s).[27, 25, 26, 21, 24]. We attribute this, in addition to the frequent edition update of these same texts, to the rapid and fertile development in the field.

HARDWARE SIMULATORS TO THE RESCUE

Areas of Application

The question that arises is what methodologies can be employed in OS courses to achieve success? Regardless of the ever-changing (or growing) list of topics, laboratory exercises must allow for student exploration and engagement of the theoretical concepts conveyed in the lectures. The answer that we propose is that simulation technology must be widely applied.

Simulation technology, or more specifically, hardware simulators would be useful in the following areas:

- Taxonomy of OS's: Clearly students need to interact with several different OS's to test their features, look-n-feel, and possibly programming interfaces. A heterogeneous lab with several OS's is inherently difficult (and expensive) to maintain and limits the resource sharing among students. (The hardware OS mapping is in some sense static.) Moreover some OS's, designed for single-user use (e.g. MS-Windows 95, 98, me) are not safe in a lab environment due to their lack of appropriate security facilities. Hardware emulation can be used to create complete virtual machines where students can experiment with various OS's within the framework of a safe and stable host environment.
- OS administration: The problem can be stated as follows: how can a student conduct experiments related to OS administration without being the actual administrator of a lab computer? Hardware emulation provides an entire machine in the student's user space, allowing the student

- Introduction to OS
  - Use of operating systems - taxonomy of OS
  - Structure of a modern OS
  - Special purpose OS's (e.g. embedded, portable phones, palmtop)
  - Principles of OS design
  - Multitasking/Multiuser OS principles of administration
  - Networked OS/Distributed Systems design and administration
  - Standardization processes for OS; open and closed standards
- OS Programming
  - The system call layer
  - Interfacing libraries
- Kernel Development
  - Theory of concurrency and concurrent programming
  - Theory of resource management
  - Kernel structure; including both macro and micro kernels
  - CPU scheduling
  - Kernel support for multithreading
  - Device drivers
  - Memory management/Virtual Memory
  - Caching systems
  - File systems
  - Loadable kernel modules
  - Dynamic link support and reentrant code
  - Virtual Machine support
  - Protection and security
  - Kernel support for networking
  - Kernel support for multiprocessor systems
  - Kernel support for graphical user interfaces
  - Kernel management of soft (and hard) real time requirements
- System Administration
  - Responsibility of administrators: Laws and Rules
  - Administrative tools and procedures
  - Scripting
  - Administration in the large, clustered and distributed system administration
  - Administration of security
- Environments and Tools
  - Languages, interpreters and compilers
  - Textual and graphical shells.
  - Editors, special purpose environments and other user development tools

Figure 1: A syllabus for a course in OS

to conduct administrator experiments on the virtual machine without any security danger.

- Kernel development: This is the classical usage of hardware simulation in the teaching of OS courses. Clearly it is not effective to give real bare hardware to students for the testing of their student written/modified OS kernels. In addition to the difficulty of loading a new kernel on real hardware, there are a significant number of architectural details, which while pedagogically unimportant require a significant time investment. Finally, it may prove overly expensive to provide each student/team with an identical dedicated hardware platform.
- OS programming: While it is possible for the students to test system calls and libraries with standard programming tools [19, 20], experimentation is limited to the OS installed on the host machine. As with the first point above, hardware emulation provides the opportunity for students to the compare the programming interfaces of different OS's.

Issues to Consider

The choice of the right simulation tool or the design of a new tool for teaching a specific topic is very difficult as several factors and trade-offs must be taken into account.

- Architecture age: The simulated environment must be up to date. The simulation of obsolete architectures lead to a set of implementation issues weakly related to current realities. Outdated architectures focus on issues that may no longer be important (e.g. the scarcity of RAM) and don't contain support for new problems that demand attention (e.g. mobility). While Tannenbaum's argument on how ontogeny recapitulates phylogeny[26] is valuable, we believe that student interest and motivation is maximized when their experimental testbed closely mirrors current technology.
- Simplicity vs. Realism: There is a trade-off between simplicity and real world consistency. A simulator for teaching, in order to be useful, must reflect the salient behavior of a real system. Furthermore, it must avoid pedagogically unimportant technical details and tricks, which while possibly crucial to optimized hardware performance, only serve to cloud (or misdirect) student understanding of the fundamental issues.

  Determining the break-even point for this trade-off is further complicated by the observation that it changes depending on the topic under consideration. A simulator with a simplified model for I/O devices can suffice for experimental kernel development while a simulator to test a real world OS as a guest OS must simulate the I/O devices in all their myriad details.

  Sometimes it would be useful to have a different simulation model for the varying degrees of student sophistication/maturity or depth one can explore a given topic. As

an example, at the very early stages in testing experimental student written OS kernels the memory model should be that of a direct mapping. (i.e. Virtual memory turned "off" so as to avoid students getting lost in various layers of complexities during their initial kernel debugging experiences.) Later on a second memory model utilizing a simplified MMU may be employed. This unrealistic and poorly performing MMU would not support a TLB, but instead access memory resident page/segment tables for each address translation. Finally, after the simplified virtual memory model is mastered, one may elect a TLB-based memory model to explore the TLB management algorithms of modern OS's.

This is just one example. This technique can be applied to many other components of the simulated system. Optimally, a simulator should support different modules for each hardware component.

The model of processor used in educational hardware simulators in theory can be of any kind. Interestingly, the majority of the tools used to test student written OS kernels (or to learn CPU architecture) represent a RISC structure. MIPS, seemingly, is the preferred architecture. We believe this is due to the pedagogically important (clear) distinction between the hardware and the software a RISC architecture provides. Furthermore, the well structured instruction set of RISC architectures makes it easier to code in assembly when needed.

- HCI: The importance of the student simulator interaction is often overlooked. At a minimum students must have a set of tools for debugging the code they write/modify. Graphical interfaces seemingly aid students in the control of the simulated environment[1]. Nevertheless, tradeoffs must be considered between ease of use and the need for quick prototyping for skilled users.

  The importance for meaningful student interaction with their just created OS cannot be understated. Furthermore, their results or interactions must appear realistic. Being able to interact with their just created OS through a minimal shell inside a terminal window or by a telnet connection is a fantastic payback for their efforts. It is the artisan intellectual reward. Obviously the more complete and usable the interface the more satisfactory it is, taking into account that the workload must be compatible with the capabilities of the student and the duration of the course(s).
- Simulator proliferation vs. one size fits all: Even if in some cases requirements are too different to use a single simulator for different tests, it is the case to use at most two of them per course. Each simulator has its own interface, its well-known set of problems. The time spent to learn the structure of another simulator is partly subtracted to the learning time.
- Speed: Simulated machines are clearly much slower than real hardware. A worthy balance must be maintained between cost of emulation, user interface management (e.g. refresh time, richness of control information) and the overhead costs of simulation. Optimally, the sustainable speed of simulation should change depending on what is currently being tested and why.

  The test of a real OS must have a simulator as performance tuned as possible. For the testing of an educational OS the primary need is that of debugging tools, which inevitably slows down the simulation. Furthermore one should be able to alter the overhead cost during run time. Increasing the richness of the information displayed by the simulator only when the execution gets close to an interesting point, say a suspected code error, is a tremendous advantage.
- Security: Simulators are executed within a host machine environment. Often hardware simulators run as plain user processes. Sometimes, though, for performance reasons or due to a need to access specific I/O entities (e.g. emulated networks) a hardware simulator needs direct access to kernel modules or to run with special permissions. System administrators must be very careful in configuring this kind of simulator in their institution's lab environments.
- Source code availability: For the exercises to be meaningful, students must be able to understand all the details of the hardware-kernel interface. Thus the availability of the source code of all the tools they are using is crucial, both for all the libraries, tools, drivers they use to write their code and for the simulator itself. As a side effect the students learn how to be part of the real software development process: students have been known to find bugs and write fixes for the simulators/tools.
- Documentation: Last, but certainly not least important is the documentation for the simulator. It is not our intention to forward an argument for quality documentation - it should be self evident. Nevertheless we wish to observe that there is a wide gap in expertise between the developers of OS hardware simulators and their potential adopters - many of whom have no disciplinary expertise in OS's. In order to insure a simulator's widest possible adoption, the documentation should be carefully written with the widest possible audience in mind. Too often simulators are created at research institutions where the threshold for documentation quality is informed by their own prodigious expertise. Lost is the observation that the vast majority of potential adopters are not at research institutions themselves and lack disciplinary expertise as well.

  Associated with this is the need for suggested assignments for which the simulator might be used. The arguments presented above for quality documentation extend here as well. The greatest simulator created, even if it is

accompanied by superior documentation will go unused unless the potential adopter can be shown exactly how the tool is to be used in support of the OS course.

SURVEY OF SIMULATORS

Designed for teaching

A number of simulators exist to support the OS course. Unfortunately, none of them support the full range of features outlined above. It has been observed by the authors that the most widely used simulators, OSP and then later Nachos, are the ones with the best user and instructor documentation. Evidently technical and pedagogical merit are less important to adopters than documentation quality and the concomitant shorter learning curve such superior documentation implies.

In addition to the issue of documentation quality, adopters seemingly migrate towards those systems with well defined student assignments. Simple hardware simulators which support multiple kernel designs (e.g. CHIP, MPS) are less popular than the OS/hardware simulator packages (e.g. Nachos, OSP) where students replace a module in an already existent OS with code of their own.

- Chip [2] Designed as courseware at Cornell University it simulates the architecture of a DEC PDP11 – IBM S/370 hybrid system. Supporting terminals, printers, disks, drums, and an interval timer, CHIP exports a simplified MMU for testing virtual memory implementations. The interface is character based (full screen, curses library) with an integrated debugger. A graphical Tcl/ Tk front-end has also been developed for CHIP.[1] Choosing simplicity over realism CHIP is very suitable for the study of kernel development. Student activities are mainly concerned with their writing of a complete, though shallow kernel. CHIP's architecture is now quite outdated. It was initially developed to run under VMS. It has been ported to Solaris and Linux.
- Nachos [8, 7] Created at U.C. Berkeley, Nachos simulates a MIPS machine as a set of C++ classes. Unfortunately, this implies that all requests to/for hardware functionality are managed as standard method calls. This means that students cannot get exposed to and therefore cannot deeply experiment with/ understand real kernel/hardware interactions. Furthermore, Nachos is both an OS kernel and a hardware simulator linked together into a single executable. Nachos supports OS programming in addition to kernel development. Kernel development is limited to students replacing various Nachos kernel modules with ones of their own construction In 1997 at Stanford University the kernel part of Nachos was ported to SimOS (see below).
- Spim [16] Designed at the University of Wisconsin, Spim is a pure MIPS CPU simulator for teaching assembly language programming in an Architecture or Computer Organization course. Since Spim does not implement any I/O devices or support a compiler (it interprets assembly source on the fly) it is not suitable to support any of the activities outlined above for an OS course.
- MPS [18] A computer simulator, like CHIP, developed at the University of Bologna, by one of the authors. MPS emulates a MIPS processor along with a a set of terminals, printers, disks, and tape devices. Currently up to four instances for each kind of I/O peripheral can be configured. MPS runs ELF executables and has its own integrated debugger. As in CHIP, the level of feedback provided during execution can be changed during runtime to increase/decrease performance. Choosing realism over simplicity, MPS is well suited for OS research. Nevertheless, its ease of use suggests a usefulness in the study of kernel development. Unfortunately, the attention to detail and real-world consistency necessary for OS research, especially those related to the MMU and TLB management make MPS too complicated for undergraduate OS courses. Future releases of MPS promise the implementation of various network interfaces and simplier MMU implementations.
- Sys161 [12] A relatively new simulator, developed at Harvard, it was also designed specifically to support the OS course. Sys161 simulates a MIPS processor together with several I/O devices, disks, a TLB-based MMU and a network interface. It runs ELF executables so it supports gcc cross-compiling and gdb. It has no graphical interface and no integrated debugger. Sys161 is in many respects similar to MPS, though it lacks MPS's sophisticated interface and debugging facilities.
- Netwire/edu [5, 6] Netwire is a network emulator also developed at the University of Bologna. It emulates complex computer networks of arbitrary topology. The network can be designed either with a graphical interface or by a description language similar to tcl. The communicating end nodes in this systems can be standard processes (through a specific Netwire library), real linux boxes using linux tuntap devices, or MPS nodes. (This latter interface has not yet been released.)
- MIPS/IDT R3052E Emulator [11, 28] This is another MIPS 3000 based emulator, developed at the ETZH in Zurich. Written completely in Java it is both highly portable and very slow performance-wise. Used to support the Topsy educational OS, it supports a console-like interaction within its emulation window and permits the use of external debuggers. Since Topsy is a fully functioning OS, this system is primarily used for concurrent thread programming. Hence its usefulness for supporting the OS course is more in line with systems like BACI[3, 4], then the type of emulators being proposed in

this paper.

- OSP[14, 15, 13] A system similar in philosophy to Nachos it was developed at SUNY Stonybrook. OSP predates Nachos, though a java version, OSP2 has recently been released. OSP only supports kernel development since the compiled kernel/emulator, which contains student written modules, always runs a simulated set of users tasks. There is no user interaction during execution, nor an integrated debugger. All use feedback is in the form of copious runtime statistics generated by OSP during the simulated execution of the predefined user tasks.

Designed for common use but useful for education

There are also a number of simulators, that while not designed specifically for the OS course, may nevertheless prove useful.

We present here some examples of useful tools.

- VMWare, Bochs [17] and Plex86 [22]: These three simulators realistically emulate a complete Intel-based PC. VMware is a commercial product while Bochs and Plex86 are freeware. VMware and Plex86 simulate the I/O structure while they utilize the host processor to run the code for increased performances. Hence they need access to kernel modules and will therefore only run on real Intel (or ISA compatible) machines. Bochs, on the other hand, is a complete simulation of both processor and I/O devices. While it runs completely in user space, it is rather slow.

  These tools are useful when exploring the taxonomy of OS's and OS administration issues, especially when exploring OS's that have no multiuser or serious security support. Furthermore, these tools can also be used to run historically important PC-based educational OS's. (i.e. XINU or Minix)
- SimOS.[23] Not designed for educational purposes, SimOS's main goal is to simulate a MIPS R4000 or R10000 in enough detail to realistically run commercial operating systems. As such, SimOS supports multiple processors, busses, disks and ethernet interfaces. Nevertheless, the Nachos kernel was ported to run under it for the support of the OS course. Unfortunately, SimOS exposes students to too much realism and hardware idiosyncratic behavior to be pedagogically valuable.
- UML [9, 10] User mode linux (UML) can also be viewed as a simulator of sorts. It allows a kernel to be run as a GNU-linux process in the user space emulating the hardware I/O structure. The simulation takes place in the UML kernel, a linux kernel patched to work at the user mode layer. This solution, while obviously tailored to linux, could in theory be ported to other open source kernels. The primary goal of this tool is to allow several virtual machines to run on a single box all behaving in a highly usable manner. UML, therefore, is a good platform to teach system administration. Students can configure and run their own virtual machine in their user space without affecting the overall security of the real system.

A SIMULATION STRATEGY

Hopefully one can see that due to the diverse and complex nature of the topics in an OS course, the use of simulators is crucial for the delivery of a successful OS course(s). Furthermore, as the topics list demonstrates, there are too many topics to be adequately covered in one course. In many institutions there are several OS courses:

- Operating Systems: Where the theoretical foundations and the basic concepts are introduced and explored.
- Laboratory for Operating Systems (sometimes named Practicum in OS): Where students can test their knowledge on real OS's and write or modify the kernel, device drivers, tools, etc. This course can be taken concurrently with Operating Systems, have Operating Systems as its prerequisite.
- Advanced Operating Systems: Where new developments in the OS field are explored.

It is clear that both the practicum and advanced courses must use hardware simulators. It would be advantageous to coordinate tool choice between these tool courses. Moreover the same tools (maybe with a different set of modules) could be used in other courses in the CS curriculum such as Hardware Architectures, Computer Organization, and Computer Networks. These complicated tools often have steep learning curves, the reduction of which leaves more time for pedagogically valuable exploration and engagement.

FUTURE DEVELOPMENT

It is clear that there is not yet a modular hardware simulation tool able to satisfy all the requirements expressed in this paper. Our survey of existing systems has shown that each tool has serious shortcomings and drawbacks.

The authors are planning to work on the MPS and Netwire tools, integrating them together and subdividing them into modules. Each feature or device (e.g. memory management) should be implemented by different interchangeable modules. This will allow the same device or feature to be used in different phases of the project with varying degrees of detail and complexity in a graceful manner. Finally, the authors intend to create high quality documentation for the new system and a set of suggested assignments.

REFERENCES

[1] L. Alvisi and F. Schneider. A graphical interface for CHIP. Technical report, Cornell University, 1996. Technical Report TR 96-1587.

[2] O. Babaoglu, M. Bussan, R. Drummond, and F. B. Schneider. Documentation for the chip computer system. Technical report, Department of Computer Science, Cornell University, 1988.

[3] B. Bynum and T. Camp. After you, alfonse: A mutual exclusion toolkit. In *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, 1996.

[4] T. Camp. An introduction to BACI. http://www.mines.edu/fs_home/tcamp/baci/index.html.

[5] E. Carniani and R. Davoli. The netwire emulator: A tool for teaching and understanding networks. In *ACM 6th Conference on Innovation and Technology in Computer Science Education, ITiCSE 2001*, pages 153–156, Canterbury, England, 2001.

[6] E. Carniani and R. Davoli. The netwire emulator: Teaching and understanding networks in both synthetic and real scenarios. In M. Roccetti, editor, *Simulation Series*, volume (34)1, pages 29–33. SCS, 2002. Presented at the International Conference on Simulation and Multimedia in Engineering Education, ICSEE 2002.

[7] W. A. Christopher, S. J. Procter, and T. E. Anderson. Nachos. http://www.cs.washington.edu/homes/tom/nachos/.

[8] W. A. Christopher, S. J. Procter, and T. E. Anderson. The nachos instructional operating system. In *USENIX Winter 1993 Conference Proceedings*, 1993. Best Paper Award.

[9] J. D. Dike. User-mode linux. In *Proc. of 2001 Ottawa Linux Symposium (OLS)*, Ottawa, 2001.

[10] J. D. Dike. Making linux safe for virtual machines. In *Proc. of 2002 Ottawa Linux Symposium (OLS)*, Ottawa, 2002.

[11] G. Fankhauser. A mips r3000 simulator. http://www.tik.ee.etzh.ch/ gfa/sim/simulator.html.

[12] D. A. Holland, A. T. Lim, and M. I. Seltzer. A new instructional operating system. In *ACM 33rd Technical Simposium on Computer Science Education SIGCSE 2002 Proceedings*, pages 111–115, 2002.

[13] M. Kifer and S. Smolka. OSP2 – an environment for operating systems projects.

[14] M. Kifer and S. Smolka. *OSP An Environment for Operating System Projects*. Addison–Wesley, 1991.

[15] M. Kifer and S. Smolka. *OSP An Environment for Operating System Projects: Instructor's Manual*. Addison–Wesley, 1991.

[16] J. Larus. Spim: a mips r2000/r3000 simulator. http://www.cs.wisc.edu/ larus/spim.html.

[17] K. Lawton. Bochs project home page. http://bochs.sourceforge.net.

[18] M. Morsiani and R. Davoli. Learning operating system structure and implementation through the MPS computer system simulator. In *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education*, pages 63–67, New Orleans, 1999.

[19] G. Nutt. *Operating System Projects Using Windows NT*. Addison Wesley, 1999.

[20] G. Nutt. *Kernel Projects for Linux*. Addison Wesley, 2001.

[21] G. Nutt. *Operating Systems: A Modern Perspective*. Addison Wesley, 2nd edition, 2002.

[22] The plex86 Team. Plex86 project home page. http://www.plex86.org.

[23] M. RosenBlum, S. A. Herrods, E. Witchel, and A. Gupta. Complete computer system simulation: The simos approach. In *IEEE Parallel and Distributed Technology: System and Applications*, pages 34–43, 1995.

[24] A. Silbershatz, P. Galvin, and G. Gagne. *Operating System Concepts*. Wiley, 6th edition, 2003.

[25] W. Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall, 4th edition, 2001.

[26] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2nd edition, 2001.

[27] A. S. Tanenbaum and A. S. Woodhull. *Operating Systems Design and Implementation*. Prentice Hall, 2nd edition, 1997.

[28] The Topsy Team. Topsy - a teachable operating system. http://www.tik.ee.etzh.ch/ topsy.