10-1-2011

An Experimental Nexos Laboratory Using Virtual Xinu

Paul Ruth
Renaissance Computing Institute

Dennis Brylow
Marquette University, dennis.brylow@marquette.edu

Accepted version. Published as part of the proceedings *of the conference, FIE 2011: 41st Annual Conference on Frontiers in Education* (October 2011): S2E-1 - S2E-6. DOI.

An Experimental Nexos Laboratory Using Virtual Xinu

Paul Ruth
RENCI/UNC-Chapel Hill
Chapel Hill, NC
Dennis Brylow
Mathematics, Statistics and Computer Science
Marquette University
Milwaukee, WI

Abstract - The Nexos Project is a joint effort between Marquette University, the University of Buffalo, and the University of Mississippi to build curriculum materials and a supporting experimental laboratory for hands-on projects in computer systems courses. The approach focuses on inexpensive, flexible, commodity embedded hardware, freely available development and debugging tools, and a fresh implementation of a classic operating system, Embedded Xinu, that is ideal for student exploration.

This paper describes an extension to the Nexos laboratory that includes a new target platform composed of Qemu virtual machines. Virtual Xinu addresses two challenges that limit the effectiveness of Nexos. First, potential faculty adopters have clearly indicated that even with the current minimal monetary cost of installation, the hardware modifications, and time investment remain troublesome factors that scare off interested educators. Second, overcoming the inherent complications that arise due to the shared subnet that result in students' projects interfering with each other in ways that are difficult to recreate, debug, and understand.

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

Specifically, this paper discusses porting the Xinu operating systems to Qemu virtual hardware, developing the virtual networking platform, and results showing success using Virtual Xinu in the classroom during one semester of Operating Systems at the University of Mississippi.

Index Terms – Computer Science Education, Courseware, Computer Systems, Virtual Machines.

Introduction

The study of computer systems continues to be one of the core elements of mainstream Computer Science curricula. The Joint IEEE/ACM Task Force on the Model Curriculum for Computing suggests that Architecture, Organization, and Operating Systems compose 19% of the core hours of an undergraduate computer science program [1]. Most computer science programs require a series of systems courses that explore both the fundamental building blocks that underlie software development and the powerful abstractions that have allowed Computer Scientists and Software Engineers to build the sophisticated software that permeates the modern world.

Prior work has confirmed that core systems courses that include an experimental environment allow students to learn by doing, reinforce concepts covered in lectures, and encourage a concrete understanding of the details within the larger concepts [2]. However, constructing pedagogically appropriate infrastructure for hands-on experimentation at the lowest levels of systems software remains difficult and expensive, often requiring faculty and facility resources disproportionate to other core courses.

In this work we build upon Project Nexos [3], an established curriculum development effort that aims to add flexible, inexpensive embedded system laboratory experiences to a wide spectrum of core undergraduate courses. Nexos-based laboratories are in use or under construction at half a dozen colleges and universities, and are based upon freely-available open source tools combined with inexpensive consumer appliances like the Linksys WRT54GL wireless router.

This paper presents Virtual Xinu, a significant new platform for use in the Nexos curriculum effort. The novel contributions of this work include:

- Extensions to the Embedded Xinu educational operating system that allow the same core kernel to run on either physical

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

hardware or Qemu [4] virtual machine with a single compile-time option,
- Modifications to existing Project Nexos curriculum modules that allow users to use either physical or virtual machines, whichever best suit the activity at hand, and
- New curriculum modules specific to the virtual platform that enable students to work with new and interesting system components.

Our goal from the outset was not to replace the real embedded hardware that is the hallmark of a Nexos-based lab, but rather to supplement and extend it with the power of virtualization. There are multiple advantages to a combined laboratory with both physical and virtual targets, including improved scalability, lower construction costs, stricter network isolation, and expanded debugging capabilities.

The following sections of this paper discuss related work, present an overview of Virtual Xinu, and evaluate Virtual Xinu in the context of an upper-division course on operating systems.

Prior and Related Work

Our work is descended from Purdue University's Xinu Laboratory [5], first developed in the 1980's as a vehicle for systems education and research. Subsequent work in the 2000's produced Embedded Xinu [3], a modern port of the original Xinu design with a focus on open source tools and resource-constrained, embedded RISC platforms. Curriculum design and evaluation efforts have shown the utility of this teaching platform in undergraduate courses including computer organization [6], operating systems [7], embedded systems [3], networking [8], and compiler construction [9].

Harvey Mudd's TinkerNet Project [10] is another curriculum effort descended from the Xinu design, emphasizing experimental networking without the special-purpose hardware required by the original Xinu plan. However, the presumption of a functional networking stack on the system makes TinkerNet unsuitable for low-level operating system and hardware system work. TinkerNet's reliance on desktop PCs as hardware targets contributes significantly to the cost and space investments required.

Many alternative educational operating systems have been proposed, including those based on simulation (Nachos [11]), bare

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

hardware, and embedded systems (BabyOS[12]), but none of these systems have withstood the test of time to the extent that the Xinu kernel has.

Special sessions and papers have recently explored the growing role of virtualization in systems courses, including Xen Worlds [13], V-NetLab [14] and the SOFTICE [15] projects. These efforts concentrate on production operating system environments like Linux, and consequently report very limited scaling even on powerful hardware. In contrast, Virtual Xinu offers much greater scaling, and lower student learning curves.

Background

The simple but powerful RISC-based Embedded Xinu kernel provides an understandable platform for teaching low-level systems courses. Nexos-based laboratories support Embedded Xinu on several embedded hardware platforms, but the most common is the Linksys WRT54GL wireless router. The WRT54GL employs a Broadcom BCM 47XX/53XX system-on-a-chip with a 200MHz, 32-bit embedded MIPS architecture with 16 MB of RAM. Further, it has four 100 Mbit/s Ethernet interfaces and a WAN interface. Most importantly, the WRT54GL includes two UART serial ports, which are shipped unused. The use of a RISC architecture and a limited number of peripheral devices makes this real embedded device simple enough for advanced undergraduates.

Figure 1 shows a typical Nexos laboratory composed of a central Xinu server that manages a pool of dedicated backend targets, such as the WRT54GL. Each router is modified to include serial interfaces that the server uses to control the backend and provide access to the backend's console. The backends' consoles are connected through a serial annex to the server allowing simultaneous access to the serial interfaces to all backends. In addition, the server and backends share a private Ethernet. The private Ethernet serves dual purposes as both an isolated playground for networking projects and as the medium over which the backend can network boot the students' Xinu kernels. The server is connected to a production network through which students access the Xinu server. The production network is usually the general-purpose network connecting the machines in an existing computer laboratory and must be separated from the shared

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

playground to prevent misbehaving projects from negatively affecting the rest of the domain.

Most of the complexity of the laboratory is hidden from the students. They are given a package of source code that includes an appropriate makefile for invoking the cross-compiler that produces the MIPS kernel image. After building a kernel image, students access the backends through well defined software on the server that allows them to temporarily "checkout" a backend and transfer a kernel to the assigned backend, boot the kernel, and pipe console I/O to a terminal. The automated laboratory interface allows the students to focus on the concepts rather than struggling with the steps needed to run their code.

Project Nexos is more than an embedded systems laboratory. The integrated Nexos curriculum is a fundamental part of the project. The curriculum employs hands-on projects revolving around the Embedded Xinu Operating System. The Embedded Xinu kernel is designed to be small, elegant, and understandable by undergraduates that are new to kernel development. The simplicity and elegant design of the kernel allows for tractable hand-on projects that reinforce sophisticated concepts learned in the classroom. Instructors can use predefined projects from the Nexos curriculum or can easily remove or modify key functionality of the operating system. The partially implemented kernel can be given to the students who are asked to re-implement core functionality or modify kernel-level algorithms and data structures.

Virtual Xinu Laboratory

The goal of the virtual Xinu laboratory is to increase the range of curriculum that is possible without sacrificing any of the realistic hands-on feel of the traditional Embedded Xinu Laboratory. The contributions of Virtual Xinu are:

- Enabling the creation of an entire Embedded Xinu laboratory that exists only in software,
- Adding functionality and curriculum that take advantage of the flexibility of virtualization, and
- Simplify the adoption of the Nexos platform.

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

The following sections present the mechanisms that enable Virtual Xinu including answering the primary challenges to porting the kernel to the Qemu platform, the techniques used to instantiate isolated virtual environments, and a description of the curriculum used in the classroom.

Fundamentally, Virtual Xinu differs from Embedded Xinu by using Qemu virtual machines instead of wireless routers. A virtual machine is a software emulation of computer hardware. An operating system can interact with a virtual machine as if it were real hardware. Figure 2 depicts the virtual laboratory. The virtual laboratory functions like the embedded laboratory but is consolidated into a single PC. Users interact with mutually isolated virtual environments of virtual backend targets that are created on-demand. Each virtual environment is owned by a single user and executes his or her software in an isolated sandbox. Typically each virtual environment is composed of a single virtual machine, however multiple virtual machines can be instantiated to support networking projects. Virtualization allows for backend targets to be a mix of architectures as depicted in the figure. However, to date, only the MIPS version of Xinu has been ported to the virtual laboratory.

Students use a modified makefile and compiler chain to build a Xinu kernel for the virtual target. Simple compile-time options have been added to the Nexos platform to instruct the compiler to build for the virtual target. However advanced students can implement kernels that identify the hardware at boot time and adapt the configuration. Students run their kernels by invoking the same well-defined software used to run on the embedded hardware. When a student wants to run their Xinu kernel, a virtual environment is instantiated on-demand and the console of the primary backend (often the only backend) is piped back to the user. From the point-of-view of the user, the basic functionality of the virtualized laboratory is accessed in nearly an identical way as the embedded laboratory.

Virtual Xinu provides additional functionality and curriculum that is not available in the current version of Embedded Xinu. The additional functionality includes:

Debugging. A standard debugger (e.g. gdb) can be attached to the executing Xinu kernel. For the first time students can step through executing Xinu code which not only aids in completing the

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

assignments, but also allows for experimentation with the internal data structures and flow control of fully functional kernels.

Isolated network. Students working on projects involving the networking stack can instantiate isolated virtual environments. Virtual machines within an environment can send arbitrary network traffic. The mutually isolated virtual environments eliminate the cross talk allowed in the current shared 'warzone' preventing the actions of one student from affecting other students.

Scale. Using virtualization, there is no hard limit on the number of virtual machines that can be instantiated. A modern PC can support hundreds simultaneously running student projects. Performance degrades as more virtual machines are instantiated. However learning is not affected by the performance of individual machines.

Porting Xinu to Qemu

The greatest amount of time and effort required to develop Virtual Xinu was in porting the Xinu Operating System to the Qemu virtual platform. Existing implementations of Xinu run on various wireless routers using MIPS CPUs, most notably the Linksys WRT54GL. Qemu can emulate a variety are architectures including MIPS. Qemu was chosen for its no-cost open source software, its prolific distribution (it is included with most Linux distributions) and the existing MIPS codebase for the wireless routers.

Admittedly, Qemu provides far less performance than the more popular paravirtualized and hardware supported virtualization mechanisms. However, Qemu's realistic hardware emulation and its ability to virtualize MIPS hardware on a standard x86 PC makes it ideal as a platform for experimenting and learning through the Nexos curriculum.

Although choosing a new machine that with the same CPU architecture reduces the complexity of porting the Xinu to a virtual platform only the CPU is the same. The motherboard and nearly all of the peripherals are different. In addition, the virtual hardware can emulate many more types of devices that are not available on the static embedded hardware. For example, the virtual machines can be instantiated with a PCI bus, IDE drives, sound cards, USB controllers, a mouse, a keyboard, and other devices. The current version of Virtual Xinu includes all of the modifications required to complete the existing

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

curriculum including the serial ports and network interface as well as adding the required support for the Programmable Interrupt Controller (PIC). Further, Virtual Xinu supports an IDE drive, which for the first gives Nexos support for a standard block device.

Serial Device Driver. The Nexos laboratory requires the use of a serial devices and many of the projects prescribed by the curriculum make use of a serial interface. The Qemu emulator provides a UART and, by default, uses it for console data. The WRT54GL uses a physical UART for console data and the first several projects prescribed by the curriculum are implemented using kernels that only use the console device. Further, one of the first projects is to implement a synchronous device driver for the UART. The Qemu virtualized UART requires very little modification of the existing projects and only requires a change to the location where the UART is mapped in memory space. Functionally, there are no other differences, however Qemu does not emulate concurrently executing devices. Although this is an observable difference between the platforms and does not properly represent real hardware, it does provide a teaching opportunity. A discussion of how this is incorporated into the curriculum is described in the section on curriculum.

Network Device Driver. Another benefit of Qemu is that it emulates an NE2000 NIC. The NE2000 was among the most popular of the early Ethernet NICs. This popularity has led to drivers in nearly all major operating systems as well as it being the choice of many virtual platforms. A completely new NE2000 network driver was implemented for the virtual platform. The new driver replaces the Broadcom driver used on the WRT54GL and works with the existing Xinu network stack. The driver sits below Xinu's Ethernet (layer-2) software. All networking curriculum for layer-2 or higher that was created for the Embedded Laboratory will work unchanged with Virtual Xinu.

Programmable Interrupt Controller. The existing embedded platforms are designed to be small static devices. There are usually a small number of integrated devices and no need to expand the system by adding devices after production. The small static number of devices simplifies the WRT54GL and other wireless routers and allows them use only the eight interrupt lines connected directly to the cpu. The Qemu virtual platform has a large array of possible devices and new devices can be implemented in the future. Qemu handles this additional complexity by providing a virtual Programmable Interrupt

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

Controller (PIC) that provides a large number of interrupts that are organized hierarchically.

The PIC requires significant modification to Xinu's interrupt handler. These modifications are complete and asynchronous device handling is possible. The NE2000, advanced serial devices, and the IDE device each make use of the modified interrupt handler.

IDE Block Device. One of the benefits of virtualization is the increased amount of available devices. Virtual Xinu adds an IDE block storage device to the curriculum. The virtual IDE device is connected using a virtual PCI bus. Virutal Xinu provides an implementation of the IDE driver, a block-level interface for reading and writing, as well as a simple file system that uses the block-level interface.

One of the advantages of the virtual block device is the ability to analyze and edit the block after the virtual machine shuts down (or crashes). Qemu emulates a block device by attaching to a file on the host machine. A user creates an empty file (probably using a utility like 'dd'). When the virtual machine is instantiated it attaches to the empty file. When a block is read or written by the virtual machine the emulator reads or writes to the file. A student who is implementing a file system debug their work by using a hex editor to view the blocks that was written by the kernel or manually write blocks for the virtual machine to read.

Isolated Virtual Environments

Students in a networking course are often implementing portions of the TCP/IP network stack or even designing and implementing their own network protocols. The Embedded Xinu Laboratory uses a single shared physical network over which students can send their experimental and developmental network traffic. This shared subnet results in students' projects interfering with each other in ways that are difficult to recreate, debug, and understand.

Virtual Xinu Environments have mutually isolated virtual networks. Multiple users can instantiate individual Virtual Xinu Environments on a shared server and will not interfere with each other's development and debugging tasks. Further, more complicated network topologies can be created on-demand. For example, a single user can create a Virtual Xinu Environment composed of multiple isolated virtual networks and connect them with a Virtual Xinu kernel

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

that can route layer-3 traffic between the virtual subnets. This can be extended to include on-demand deployment of a large number of subnets in an arbitrary topology.

Central to the effectiveness of Virtual Xinu Environments is the on-demand creation of isolated virtual networks. Nexos leverages existing techniques from the cloud computing community to create tunneled virtual networks [16, 17].

Figure 3 shows two mutually isolated Virtual Xinu Environments. In the figure, the Xinu Server is hosting four virtual machines (two blue virtual machines and two orange virtual machines). Each virtual machine is a Qemu process executing on the Xinu Server. From a Xinu kernel's point-of-view, the virtual machine presents an emulated NE2000 NIC referred to as ETH0. The physical host machine (Xinu Server) sees the virtual machine as a process. The virtual machine's externally facing emulated hardware must be accessible by the host, which is responsible for handling each device in an appropriate manner.

Qemu network interfaces can be handled through the tun/tap device abstraction. A tun/tap device is a kernel abstraction that behaves as a NIC but is used to pass layer-2 (Ethernet) frames to a process instead of to an external physical network. When the virtual machine is instantiated, it can be configured to attach its emulated NIC to an existing tun/tap device. Once this connection is established, the Xinu virtual machines can route network traffic to its virtual ETH0 and the traffic will be passed to the host via the tun/tap device. The host can also route its traffic through the tun/tap device and that traffic will be sent to the virtual machine which will handle it as if is was arriving from a physical network. Finally, Nexos can connect multiple virtual machines by attaching their tun/tap devices using a standard Linux bridge. The result is an isolated layer-2 network connecting multiple virtual Xinu kernels.

It should be noted that it is possible to bridge a virtual network to an external physical network giving a virtual machine access to the local domain or even the global Internet. However, it is not recommended to allow students to generate network traffic that is released to a production network.

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

Curriculum Modifications and Experience

The similarities between real hardware and Qemu virtualization not only simplifies transitioning between Embedded and Virtual Xinu but provides a realistic platform on which students can learn systems concepts. As virtual machines become prolific in mainstream computing they blur the line between physical and virtual hardware. Experience using production quality virtual hardware is as valuable as experience using physical hardware. By changing a few compilation options and selecting a different set of device drivers, students are able to run the same experimental operating system kernel source code on both real router hardware and on virtual hardware. This bifurcated support for both real and virtual hardware keeps the curriculum grounded in genuine embedded systems, while leveraging the many advantages offered by virtualization.

The Virtual Xinu laboratory and its modified curriculum have been used over one semester of upper-level Operating Systems at the University of Mississippi (CSCI 423). The class contained 25 junior and senior level students that we placed in pairs and asked to complete five projects over the course of the semester. The projects were similar to ones used over the in previous semesters. For each project the students were given a partially implemented Xinu kernel and were asked to implement missing functionality or to modify an existing kernel algorithm and/or data structure.

The curriculum was modified to incorporate features enabled by the Virtual Xinu Laboratory. Specifically, for each project the students were asked to develop a kernel that would correctly execute on both the WRT54GL hardware and the Qemu virtual hardware. Most students were able to successfully develop a kernel that correctly executed on both platforms and were able to use the hands-on experience to reinforce operating systems concepts from lecture.

The experience using the Virtual Xinu in the classroom led to a few observations on the effect it has on the learning outcomes of students. Although each of these observations applies to many or all of the projects, they were made apparent during the first project, which is to write synchronous device driver for the UART. The driver is necessary so that the students can read/write console data for the remainder of the projects. It is essentially the Xinu "Hello, World" program. The solution involves repeatedly checking the UART to see if

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

data (a single character) is ready to be read then reading the character after it is ready and a similar process for writing data to the UART.

Using two different platforms helps the students better understand issues involved with writing software that interacts directly with hardware. Specifically, in order to write code that runs on both the virtual and physical hardware, the student had to understand that devices use memory mapped I/O and different hardware platforms can map I/O devices to different locations in memory. This not only reinforces I/O concepts learned in class but also forces students to truly understand low-level programming concepts such as pointers. For the synchronous serial driver the students must understand that the two platforms have mapped the UART to different location in the memory address space and to find the correct location the kernel much check the cpu's ID to determine the platform that they are using.

Qemu teaches a subtle (and unexpected) lesson due to its slightly unrealistic emulation of devices. Specifically, Qemu does not concurrently execute independent devices. The effect is that when a student uses the virtual platform he or she can write data to the UART and the data is completely processed by the UART before control is passed back to the kernel. For this reason student can incorrectly implement their device driver by not waiting for the device to be ready to receive data (if all data is processed immediately then the UART will always report being ready to accept data and does not need to be checked). If a student does this, he or she will get correct behavior on the virtual platform but incorrect behavior on the WRT54GL. When students make this mistake and ask questions, the answer reinforces the concept of concurrency in a way that is not possible using only lecture.

Finally, the ability to attach a debugger to a running Xinu kernel is very useful. Specifically, the debugger is useful for developing the synchronous driver because it allows a student to step through their code and examine variables and flow control before the kernel has the ability to print text to the console. However, debuggers are not only completing the projects but also for understanding existing kernel data structures and algorithms. Stopping the kernel on a strategic step and examining a queue of processes or a linked list of free memory blocks is often the best way to understand those data structures. Many

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

students reported that they do not know how they could have completed the projects without the debugger. However, others reported that using the debugger was too difficult.

Conclusion

The Nexos project continues to modernize computer systems education through the development of its integrated hands-on curriculum and laboratory. Virtual Xinu will become invaluable tool within the Nexos curriculum and will allow increased numbers of students and faculty to benefit from this proven hands-on learning environment.

The Virtual Xinu Laboratory described in this paper has several advantages over the existing Xinu Laboratories. Instructors who want to try Nexos before committing to it can use its simple, no-cost playground. Beginning student will be forced to understand differences between platform and subtle issues related to concurrency. More seasoned users will appreciate the debugger and the elimination of the network "warzone" that inhibited much of the advanced networking curriculum.

References

[1] Joint IEEE Computer Society / ACM Task Force on the Model Curricula for Computing. Approved final draft of the computer science volume, Dec 2001.

[2] S. Donovan and J. Bransford. How Students Learn: History, Mathematics, and Science in the Classroom. National Academies Press, January 2005.

[3] D. Brylow and B. Ramamurthy, "Nexos: A next generation embedded systems laboratory," *SIGBED Review*, 6(1), January 2009. ISSN 1551-3688.

[4] F. Bellard, "Qemu, a fast and portable dynamic translator," in *proceedings of the USENIX Annual Technical Conference*, 2005.

[5] D. E. Comer and T. V. Fossum, *Operating System Design: The XINU Approach*, PC ed. Prentice Hall, 1988.

[6] D. Brylow, "An experimental laboratory environment for teaching embedded hardware systems," in *WCAE 2007: Workshop on Computer Architecture Education*, ACM Press, June 2007, pp. 44-51, ISBN: 978-1-59593-797-1.

Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

[7] D. Brylow, "An experimental laboratory environment for teaching embedded operating systems," in *SIGCSE 2008: The 39th ACM Technical Symposium on Computer Science Education*, ACM Press, March 2008, pp. 192-196.

[8] D. Brylow and K. Thurow, "Hands-On Networking Labs With Embedded Routers," in *SIGCSE 2011: The 42nd ACM Technical Symposium on Computer Science Education*, ACM Press, March 2011, pp. 309-404.

[9] A. Mallen and D. Brylow, "Compiler Construction With A Dash of Concurrency and An Embedded Twist," in *SPLASH 2010: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems, Languages and Applications*, ACM Press, October 2010, pp. 161-168.

[10] T. Winters, R. Ausanka-Crues, M. Kegel, E. Shimshock, D. Turner, and M. Erlinger, "Tinkernet: A Low-cost and Ready-to-deploy Networking Laboratory Platform," in *Eighth Australasian Computing Education Conference (ACE2006)*, ser. Conferences in Research and Practice in Information Technology. Australian Computer Society, 2006.

[11] W. A. Christopher, S. J. Procter, and T. E. Anderson, "The Nachos Instructional Operating System," in *USENIX Winter*, 1993, pp. 481–488.

[12] H. Liu, X. Chen, and Y. Gong, "Babyos: a fresh start," in *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM Press, 2007, pp. 566–570.

[13] B.R. Anderson, A.K. Joines, and T. Daniels, "XenWorlds: Leveraging Virtualization in Distance Education," in *ITiCSE 2009: Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ACM Press, July 2009, pp. 293-297.

[14] W. Sun, V. Katta, K. Krishna, and R. Sekar, "V-NetLab: An Approach for Realizing Logically Isolated Networks for Security Experiments," in *Proceedings of CSET 2008: USENIX Conference on Cyber Security Experimentation and Test*, 2008, pp. 5:1-5:6.

[15] A. Gaspar, S. Langevin, W. Armitage, and M. Rideout, "Enabling New Pedagogies in Operating Systems and Networking Courses With State of the Art Open Source Kernel and Virtualization Technologies," *Journal of Computing Sciences in Colleges*, Vol 23, No 5, May 2008, pp. 189-198.

[16] Paul Ruth, Junghwan Rhee, Dongyan Xu, Rick Kennell, Sebastien *Goasguen "Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure", Proceedings of The 3rd IEEE International Conference on Autonomic Computing* (ICAC'06), Dublin,Ireland, June 2006.

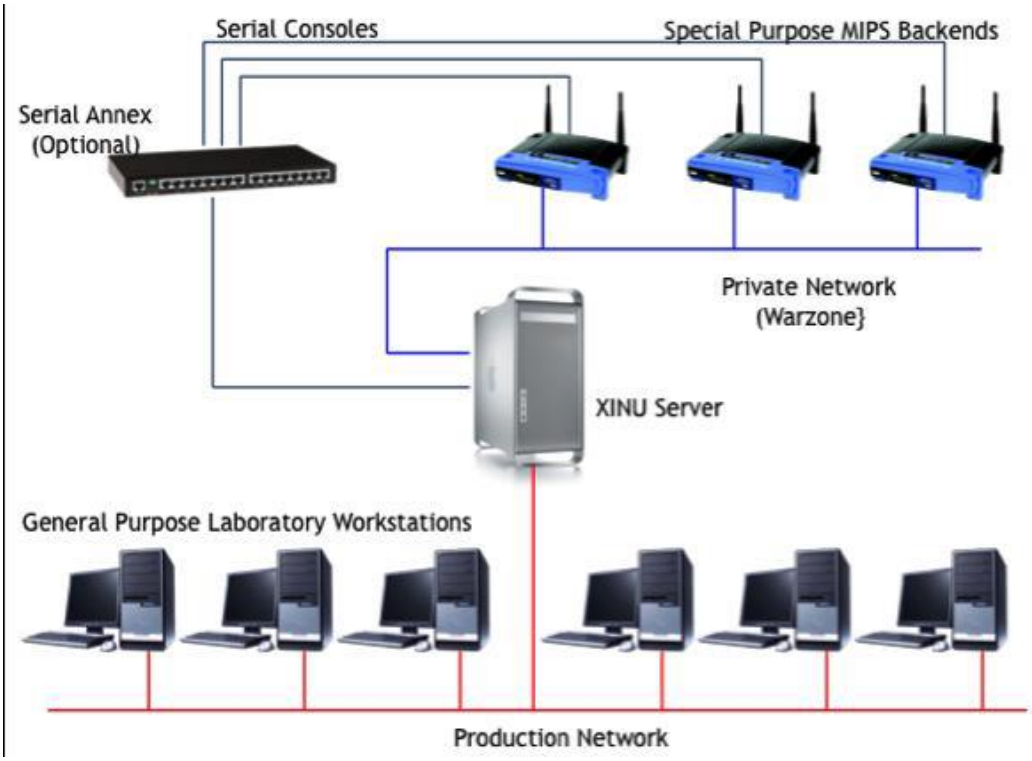
Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

[17] Paul Ruth, Xuxian Jiang, Dongyan Xu, Sebastien Goasguen, "Towards Virtual Distributed Environments in a Shared Infrastructure", *IEEE Computer, Special Issue on Virtualization Technologies*, May, 2005.

Author Information

Paul Ruth, Senior Distributed Systems Researcher, RENCI/UNC-Chapel Hill, pruth@renci.org. (Formerly Assistant Professor of Computer Science, University of Mississippi)
Dennis Brylow, Assistant Professor of Computer Science, Marquette University, brylow@mscs.mu.edu.

Figure 1. Embedded Xinu Laboratory.



Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.

Figure 2. Virtual Xinu Laboratory

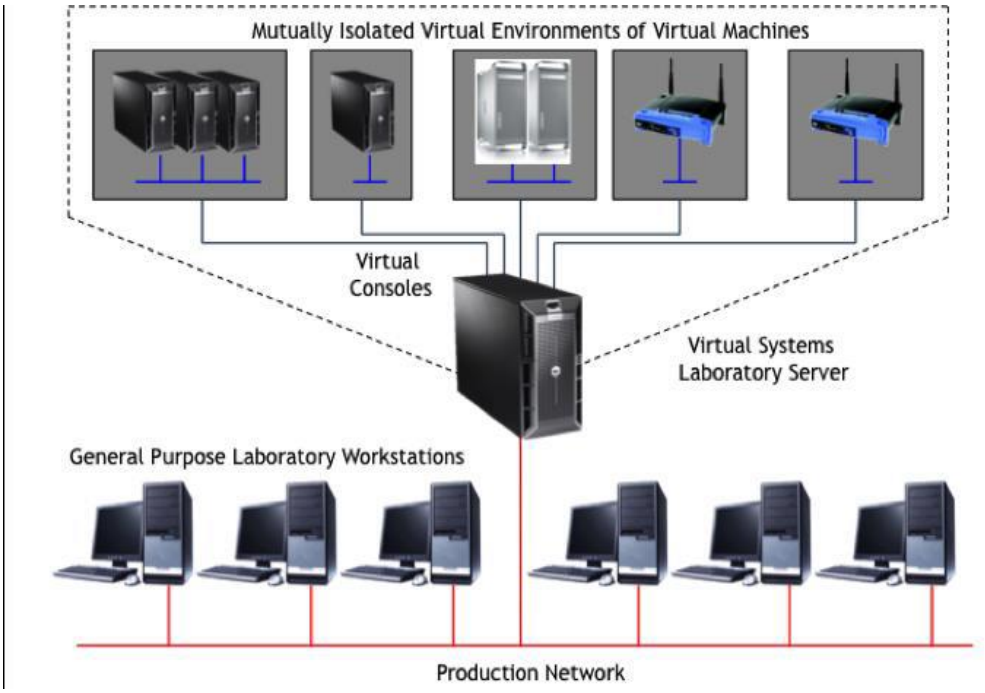
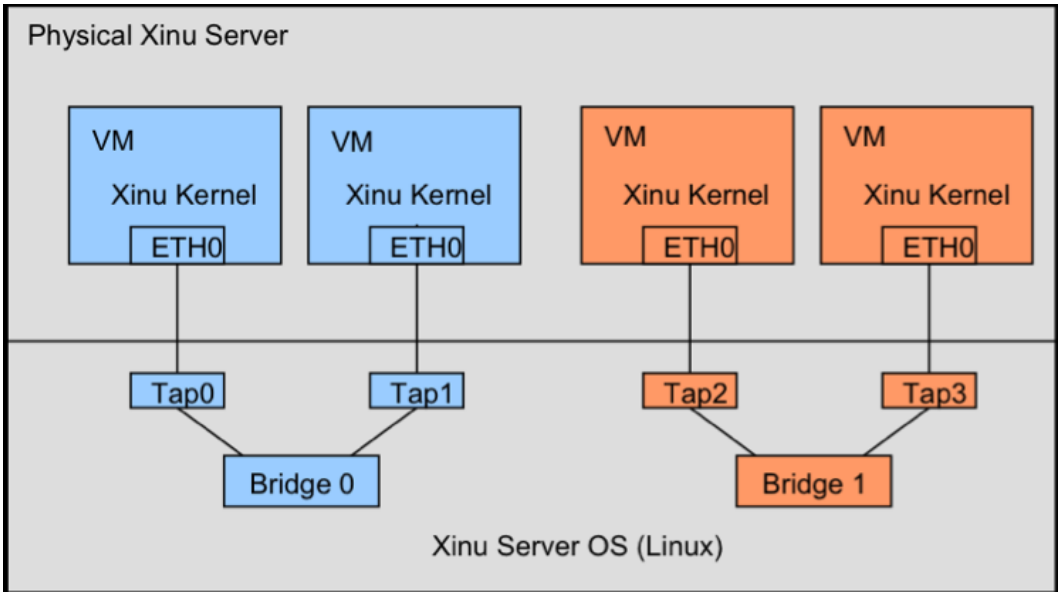


Figure 3. Mutually Isolated Virtual Xinu



Proceedings of FIE 2011: 41st Annual Conference on Frontiers in Education, Rapid City, SD, 12-15 Oct. 2011, (October 12-15, 2011): pg. S2E-1 - S2E-6. DOI.