Teaching Operating Systems Using Android ABSTRACT

Article · May 2012

DOI: 10.1145/2157136.2157312

2 authors, including:

Jason Nieh
Columbia University
137 PUBLICATIONS **3,077** CITATIONS

Teaching Operating Systems Using Android

Jeremy Andrus
Dept of Computer Science
Columbia University
New York, NY
jeremya@cs.columbia.edu

Jason Nieh
Dept of Computer Science
Columbia University
New York, NY
nieh@cs.columbia.edu

ABSTRACT

The computing landscape is shifting towards mobile devices. To learn about operating systems, it is increasingly important for students to gain hands-on kernel programming experience in these environments, which are quite different from traditional desktops and servers. We present our work at Columbia University to teach operating systems using Android, an open, commercially supported software platform increasingly used on mobile and embedded devices. We introduce a series of five Android kernel programming projects suitable for a one semester introductory operating systems course. Each project teaches a core operating system concept infused with Android or mobile device specific context, such as Android specific process relationships, use of sensors, and design considerations for resource constrained mobile devices. We also introduce an Android virtual laboratory based on virtual appliances, distributed version control, and live demonstrations which gives students hands-on Android experience, with minimal computing infrastructure. We have used these Android kernel programming projects and the Android virtual lab to teach an introductory operating systems course. Although this was our first time teaching the course using Android, over 80% of students surveyed enjoyed using Android and the majority of students preferred Android to traditional desktop development.

Categories and Subject Descriptors: D.4.0 [Operating Systems]: General; K.3.1 [Computers and Education]: Computer Uses in Education–distance learning; K.3.2 [Computers and Education]: Computer and Information Science Education–computer science education

General Terms: Design, Experimentation, Human Factors

Keywords: Operating Systems, Android, Mobile Devices

1. INTRODUCTION

Hands-on learning through programming projects plays a key role in computer science education, and hands-on kernel programming projects are especially important in the area of operating systems (OS). Many approaches to designing these kernel programming projects have been proposed and implemented. Several pedagogical OSes exist [4,13,14] where students fill in or build missing subsystems. In recent years, many institutions have also begun using Linux to teach OS [1, 6, 11]. All these solutions focus primarily on desktop or server environments, whether physical or virtual.

The computing landscape, however, is shifting. The dominant computing platform is becoming the mobile device [5, 15]. The real-world constraints and operating environment of mobile devices are quite different from traditional desktop or server computers. It is important for students to learn in this new environment, and its prevalence and popularity can be used to create engaging programming projects.

We present our work using Android to teach OS through hands-on kernel programming projects. We chose Android for several reasons. First, as a production system it enables students to learn about real-world OS issues which are hard to glean from simplified pedagogical projects. Second, since Android is based on the open-source Linux kernel, students can leverage a wealth of Linux tools and documentation. Third, Android's use of the Linux kernel provides a familiar transition path from courses already using Linux to teach OS. Fourth, Android is the fastest growing mobile platform to date, and its popularity makes it of tremendous interest to students. Fifth, Android is open-source which allows exploration of a complete production system including the OS kernel, user space libraries, and a graphical user environment written in Java. Sixth, as a commercial platform, Android continues to be developed and improved which naturally evolves the platform as a pedagogical tool, enabling students to learn in a modern context. Finally, as a commercial platform, there is no need for us to maintain or update Android or any of its development tools. This allows us to focus limited resources on teaching rather than time consuming in-house OS development.

To facilitate our use of Android to teach OS, we created an Android virtual lab where students learn about operating systems using both emulated and physical mobile devices. We manage the complexity of device cross-compilation and production kernel development tools by providing a virtual appliance pre-configured with all the software tools necessary to develop an Android Linux kernel. A virtual appliance can be readily deployed, downloaded and used by students without the installation or configuration necessary to deploy Android development tools natively on their personal computers. Additionally, our Android virtual lab uses a distributed version control system and live demonstration infrastructure to develop, distribute, submit and grade homework projects. Our use of the Android emulator in con-

SIGCSE'12, February 29–March 3, 2012, Raleigh, North Carolina, USA.

junction with this infrastructure allows remote and distance learning students to take full advantage of our Android virtual lab, while also facilitating close collaboration with on-campus students using physical mobile devices.

Students work in groups to complete five Android Linux kernel programming projects. These projects require students to read and understand core Android Linux components, and then either modify or add components as required. The projects build in complexity, and cover various important OS topics: (1) system calls and process based on the unique process hierarchy of an Android device, (2) synchronization where a global resource such as the orientation sensor is shared amongst many processes, (3) scheduling by exploiting Android's single-application usage model to increase device responsiveness, (4) virtual memory and the exact size and nature of Android's inter-process shared memory, and (5) file systems with automatic geo-tagging.

We successfully used Android in a 100 student introductory OS course at Columbia University. Over 80% of students surveyed enjoyed applying OS concepts to the Android platform, and the majority of students preferred Android over traditional desktop development.

2. ANDROID VIRTUAL LAB

Central to any OS development lab is universal access to proper development tools for all students. Our Android virtual lab provides each student with a pre-configured VMware virtual appliance containing all the Android and Linux development tools necessary to complete each programming project. The set of tools includes all tools necessary to boot and test a real device as well as the Android SDK comprising the Android emulator, a tool to create virtual devices, and a device debug GUI tool. We also include a cross-compilation toolchain, Android's Bionic C library, and several shell scripts to mitigate the complexity of embedded development.

Although Android development tools are available for a wide variety of platforms, we provide a pre-configured virtual appliance for four important reasons. First, we avoid mistakes or incompatibilities in development tool installation. For example, when the course was offered, the Android development tools did not support Mac OS 10.6, Mac OS versions prior to 10.4, or Linux distributions other than Ubuntu. Students may also use non-standard PC configurations which would put unnecessary management burden on the instructional staff. Second, the virtual appliance can be used as a safety net for students who corrupt their development tool installation or experience complete system failure. The VMware Workstation snapshot feature can be used to make incremental backups of student work and any changes they make to the development environment. A snapshot is also an easy way for students to begin a homework assignment from a known good configuration. Third, by pre-configuring all of the development tools, we avoid complicated cross-compilation setup by providing simple, standard Makefiles and shell scripts for both kernel and user-level development. The Android SDK is designed primarily for GUI application development not kernel development, so some configuration of the compiler and Android runtime libraries is necessary. A standardized environment allows us to provide simple Makefile examples for user-level test programs and simple Linux kernel cross-compilation instructions. Finally, the virtual appliance gives us freedom to customize both the tools and the Android user space. Customizing the Android user space allows us to create more engaging projects as well as to overcome deficiencies in development tools. We use a customized version of the Android emulator which enables students to use the *Sensor Simulator* program from OpenIntents [12] to inject orientation and acceleration data into the emulator by interacting with a 3D model of a phone instead of manually entering numbers into a shell prompt. We also use a custom device drawing library to support the kernel scheduling project described in Section 3.3.

To manage homework project preparation, distribution, submission and grading, we use the Git distributed version control system, already used by Android and Linux for source code version control. Each homework assignment consists of a single Git repository that typically contains a complete Linux kernel tree, template user-space projects as necessary, any additional tools needed to complete the assignment, and a Makefile used to prepare the student's emulator or device for project development. Instructional staff prepare each project repository and push it to a central Git server. The repository is then replicated such that all student groups are given access to their own private repository on the Git server. The central Git server also facilitates distributed group collaboration even when one or more students are remote or distance learning students.

To maximize grading efficiency, we extend the previously developed concept of live demonstrations [7]. Three or four student groups are assigned an hour long demo time slot. During the first 20-30 minutes, all student groups perform a complete clone of their homework submission Git repository, cross-compile the Linux kernel for their mobile device, and install and boot the new kernel. In the last 30-40 minutes, the staff meets individually with each group. During this time, groups demonstrate functionality required by the homework, further explain their solution methodology, and participate in a basic code review. This time helps instructional staff to better understand the group's submission, correct common mistakes, and see how group members contributed to an assignment. Live demos also provide opportunities for instructional staff to explain solutions which can facilitate a more complete understanding of difficult and challenging assignments.

Our Android virtual lab is designed to support remote or distance learning students, though such students may only have access to the emulator and not a real Android device. These students can still participate in demonstrations using freely available screen-sharing applications such as *Skype*, *join.me* or *VNC*.

3. KERNEL PROJECTS

Using our Android virtual lab, students work in groups to complete five kernel programming projects. These projects require students to read, understand, and modify core Linux components. While some projects require writing a simple user space test program, students are never required to compile the entire Android code base or write any GUI applications. The five projects focus on five important OS concepts, and infuse Android or mobile device specific investigation into the assignment. The five areas covered by the assignments are: system calls and processes, synchronization, scheduling, virtual memory, and file systems. Corresponding Android-related topics incorporated in these areas are: the *zygote* process and Java worker threads, device sensors,

display-prioritized scheduling, multi-process working set via copy-on-write shared memory, and location aware file systems.

The assignments progress in complexity, building not only on OS principles learned in earlier assignments, but also on Android specific knowledge gained. For example, the first assignment requires students to investigate the Android process tree and note how all GUI programs are children of a process named zygote. In a subsequent homework, they investigate the cross-process memory sharing method used by the zygote to save system RAM. We formalize each project's Android and mobile device investigation by asking students to answer a small number of questions designed to make them reflect on how the particular OS concept was applied in the context of an Android or mobile device.

All assignments are intended to keep students focused on the OS principles being taught, and designed such that a group of two or three students can complete them with no prior kernel or Android experience. We provide detailed step-by-step instructions on cross-compiling and device or emulator use. Android specific aspects of the assignments are presented as practical application of the core principle, and most of the Linux kernel modifications necessary to complete the assignments are contained in architecture independent code. Thus, the solution complexity remains manageable and homework setup and prerequisite topic knowledge is kept to a minimum, yet students engage with a complex, real-world system.

3.1 System Calls and Processes

The first project lays the groundwork for future projects as students investigate the primary application abstraction, the process, and its primary interface into the kernel, the system call. Our focus in this assignment is process creation, termination, properties, and relationships in the context of a mobile device. In completing this assignment, students also gain an understanding of kernel data structures, such as linked lists, and their APIs. All subsequent assignments require students to be intimately familiar with these concepts.

Students write a new system call which returns the system process tree in DFS order. This involves modifying architecture specific system call entry points, manipulating and traversing the kernel data structures representing processes and threads, and managing data transfer to and from the kernel. To test the system call, they write a simple user space application to invoke this new system call and print out the process tree similar to the UNIX *ps* utility.

The system call allows students to examine Android's process tree and application startup method, which provide insight into a key system design that drives the entire Java-based user environment. Java applications are interpreted in the *Dalvik* virtual machine, and are represented in the OS as processes which are children of a special process called the zygote. The Dalvik virtual machine starts several worker threads for each application to handle things like input events and garbage collection. Thus, the process tree of Android devices shows not only the relationship between the *init* process and its children, but also the relationship that all Java applications have to the zygote and the symmetry of their component threads.

Students investigate the zygote process using their test program, and are asked to reason why an embedded or mobile system might use such a process. This reflection is designed to connect the pedagogical concept of process creation using copy-on-write memory to a real-world mobile device with memory and disk constraints. Benefits of the zygote include faster application startup time, and cross-process memory sharing of core library code and static data.

The Android emulator enables remote or distance learning students taking the class to complete this assignment without a physical device. The Android emulator provides a complete machine emulation in which a standard version of the Android runtime is installed and run. Thus, remote or distance learning students can investigate the Android process tree using the emulator in the same way on-campus students use real devices.

3.2 Synchronization

The second project focuses on synchronization, a critical aspect of a modern multi-tasking OS. The wealth of sensors available on modern mobile devices provides an excellent pedagogical vehicle to demonstrate real-world applications of synchronizing concurrent or interleaved access to a single resource. In completing this assignment, students also gain an appreciation for manipulating and interacting with embedded system sensors.

Students implement a novel synchronization primitive, the *orientation event*, which allows multiple processes to block until the mobile device has been put into a particular orientation. For example, a process can block until the phone is placed face down on a table. To accomplish this, they first write a user space daemon which reads device orientation through a standard Android hardware abstraction library, and then passes the data into the kernel through a new system call. The orientation event interface is implemented as a set of three new system calls: `orientevt_open`, `orientevt_close`, and `orientevt_wait`. The daemon process passing device orientation into the kernel functions as the signal which wakes up any blocked process. Students test this new interface by writing several small test programs. Each test program forks multiple children, and each child process blocks on an orientation event opened by its parent. When the device is moved within the range of the desired orientation, all child processes should be unblocked.

Orientation and acceleration sensors are an integral part of the mobile device experience, and incorporating them into a synchronization project gives students an experience that desktop or server machines cannot provide. The ability for multiple processes to wait for a device to enter a particular orientation or acceleration profile has many possibilities in real-world user applications and system services such as interactive game controls and pre-fall system shutdown.

As students investigate sensor-based synchronization on Android, they are also exposed to real device interaction using a hardware abstraction layer. This interaction necessarily includes basic understanding and manipulation of sensor data. We provide several helper functions to keep the students focused on the primary topic of synchronization, however the exposure to real-world device data is a valuable experience which can be directly applied in the workforce.

The Android emulator provides the ability for remote or distance learning students to complete this assignment without a physical device. We provide a modified version of the Android emulator, a daemon process to run on the emulator, and a Java-based host application which simulates [12] a mobile device using a 3D wire-frame model. As students

manipulate the model in the Java application, orientation information is sent to the daemon process which updates emulator state. The Android hardware abstraction layer reads this emulator state. In this way, remote students can have a similar experience to on-campus students

3.3 Scheduling

The third project focuses on scheduling. Mobile devices generally operate as a single user environment, and thus have significantly different scheduling requirements from desktop or server machines. For example, Android users typically view a single application at a time, not multiple applications in multiple windows.

In what is one of the most challenging projects, students write a new scheduling policy for the Linux kernel. This is the first assignment which requires students to manipulate a substantial portion of core Linux kernel code. To mitigate the daunting task of implementing a new scheduler, we leverage the modular scheduling framework in Linux which provides several examples of self-contained schedulers. We encourage students to use these existing schedulers as templates. The new scheduling policy trades fairness and throughput for responsiveness, a key metric in mobile devices. We call our new scheduler, "Display Boosted Multilevel Container" (*DBMC*) scheduling, and the primary objective of this scheduler is to "boost" the priority of foreground applications. Since mobile devices typically display a single application at at time, boosting the priority of this single, foreground, application decreases its execution time and increases the apparent device responsiveness.

To support DBMC scheduling, we made a small (15 line) change to the Android user space environment that leverages the Android application usage and security model. In contrast with desktop Linux systems, Android only allows a single application to use the display at a time. While multiple applications can run simultaneously, only one draws on the screen. When an application is installed onto an Android device, a unique user ID is generated and assigned to the application. The application is always run using these unique credentials. We wrote a simple, 15-line patch to a core Android drawing library, `libsurfaceflinger.so`, which informs the kernel of the process ID of the application currently drawing on the screen. Students use the unique user ID associated with the process to easily assign the associated threads and processes to a single scheduling entity, the container. The container is used for scheduling so that the priority boost is applied to all of the threads and processes of the foreground application.

Students are asked to run Android using their new scheduler and consider what qualitative impact there is on system performance compared to the existing Linux scheduler. A correct solution implemented on the Google ADP1 booted the GUI 5–10 seconds faster than the standard Linux kernel, but initialized the network and cellular connections significantly slower. Students are also asked to reflect on this scheduler's impact on graphics-intensive games where process starvation can occur and overall game play can suffer.

The Android emulator provides a similarly satisfying experience for remote and distance learning students. The same modified drawing library is used in the emulator. A correct solution implemented on the emulator booted the GUI faster, and made the entire UI qualitatively more usable and responsive.

3.4 Virtual Memory

The fourth project explores memory management, a critical aspect of mobile device OSes, with a focus on virtual memory and paging Using a mobile device to investigate virtual memory and paging highlights real-world system constraints and provides a unique platform to investigate creative solutions in memory sharing and allocation.

Students write a new monitoring mechanism to track the working set of specified processes, and a new system call to extract the recorded data. The working set is defined as the set of pages accessed (read or write) by a process during some time period. To test this mechanism, students add a set of processes to the monitoring mechanism, and then write a user space program that invokes the new system call and displays usage information for each process.

Here we follow up the investigation of the zygote process seen in the first project as discussed in Section 3.1. This process loads several libraries and Java classes, pre-initializes Dalvik virtual machine state, and listens on a socket for connections from a client. To start a Java application, Android connects to the zygote socket and requests that the process fork. The child begins executing Java code at a particular method of a specified class, and the parent resumes listening on the socket. When the child forks, all memory mapped by the parent is shared copy-on-write with the child. This includes all loaded libraries and initialized Dalvik virtual machine state. This is drastically different from a traditional desktop or server where processes are spawned from a shell or init process that has little or nothing in common with the application being started.

Students use their working set monitor to investigate the cross-process shared memory of the zygote and its children. We define the set of pages originally mapped by the zygote as the, "Android working set." Students use their user space utility to calculate the intersection of the Android working set with the working set of each zygote child process. For simplicity, we assume that no process un-maps or re-maps a region of memory originally mapped by the zygote; this allows us to use the virtual addresses returned by our new system call without needing a more complicated virtual to physical address mapping. By investigating the unique implementation of a zygote process, students gain an appreciation for the memory constraints of a real device and can measure the effectiveness of Android's zygote solution for reducing memory usage.

The Android emulator provides the ability for remote or distance learning students to complete this assignment without a physical device. The assignment does not require the use of any physical device features.

3.5 File Systems

The final project focuses on file systems. Similar to virtual memory, file systems tend to be large and complex pieces of code, so we have students implement extensions to an existing file system code base. This assignment requires students to gain practical understanding of how the virtual file system (VFS) infrastructure is designed, which is the key file system abstraction layer that every file system designer needs to understand. In addition to the file abstraction, students are exposed to issues that arise in real-world systems, such as the endian-ness of permanent data storage, data consistency and reliability, and embedded data retrieval.

Students modify an existing disk-based file system to automatically include GPS information so that this information can be used by any application. We refer to this as the geo-tagged file system. All files and directories in a geo-tagged file system include embedded location information in the form of latitude and longitude values. Students write a new system call and user-level daemon to inform the kernel of the current device location, which is retrieved from the GPS sensor via the Android hardware abstraction library similar to the one used in the second project discussed in Section 3.2. Students then modify the `ext2` file system to retrieve the last known location data and update a file or directory every time it is created or modified. Students test the geo-tagged file system by writing a second system call to retrieve the embedded location data for a given pathname. This assignment brings together several topics from earlier homework assignments including system calls, synchronization, and sensor data management. As a more advanced challenge, students can implement VFS layer interfaces which would allow any disk-based file system to implement the geo-tagging.

While it is possible to use a GPS sensor with a desktop or laptop system, the integration of location services in a mobile device is much more engaging, and exposes the student to real-world issues of sensor data reliability and permanent storage. In addition to the actual device location returned by the hardware abstraction layer, we also ask students to store the relative "age" of the location data (the number of seconds since the location was last updated in the kernel). This provides a basic confidence metric that can be used when later retrieving a file's location.

Students discover the practicality of their solution as they create a file system image which contains at least three files with unique location data. Students can have fun visiting different places while keeping track of exactly where they were with a simple shell command on their phone such as: `echo "HERE" > `date +"%s"`.txt`.

The Android emulator provides an emulated GPS sensor, allowing remote and distance learning students to fully participate in the assignment. The emulated location is accessed using the same hardware abstraction layer library used on the real device, and can be updated using the emulator's debug interface. The Android development tools also provide a graphical utility which can update the emulator's location by reading `.kml` files generated in Google Earth. Thus remote and distance learning students can take a virtual trip using Google Earth and save files in their geo-tagged file system from each location they visit.

4. EXPERIENCES

We used Android as the homework project platform in an introductory course on operating systems at Columbia University in Fall 2010. This was the first time we taught the entire course using Android devices and Android-kernel projects. Despite our mistakes and mis-steps, the overall response from students was overwhelmingly positive. Approximately one hundred students enrolled in the course, and we asked all of them to complete evaluation surveys at the end of the course. Sixty percent of the students completed the survey. Figure 1 shows the results. Of those students who completed the survey, 80% said they enjoyed using Android in the course. Most students who completed the survey said Android was both helpful in learning OS concepts

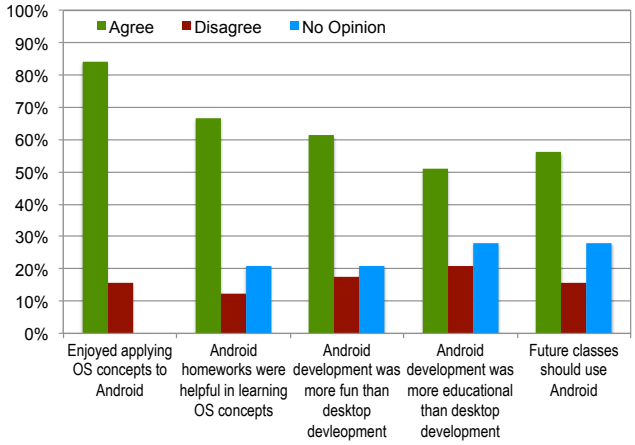

Figure 1: OS Course Survey Results

and more fun than traditional desktop development; students preferred Android to traditional desktop development by a ratio of 3 to 1. In addition, not more than 21% of the students actively disagreed with any given survey question.

Students were also asked to comment on what they enjoyed the most and what they enjoyed the least about the course. Students found the Android kernel projects fun, exciting, engaging and educational. They also appreciated the practical skills gained from their experience. When asked the question, "What did you like best about using Android for this course?" students responded with:

- "*The practicality of it,*"
- "*More fun and exciting,*"
- "*It's more like real work, not just homework,*"
- "*Increased curiosity about mobile platforms,*"
- "*Using a modern system,*"
- "*All sensor related assignments were fun.*"

Negative feedback was concentrated around the speed of the emulator and debugging of the embedded systems (emulator and mobile device). Students felt that the emulator was too slow, and that debugging an embedded kernel was overly complicated. Unfortunately, the speed of the emulator is directly correlated to the speed of the laptop on which it is run, and because the mobile device uses an ARM processor the instruction set must be emulated which is a slow process. In the future, as SMP support for ARM is integrated into QEMU, and as laptop processor speeds increase, it should be possible to speed up the emulator. Debugging an embedded system is an inherently complicated task, and simplifying the process is not easy. One approach to this problem is to create more explicit and detailed instructions on the use of Android and kernel level debugging techniques. such as the use of `/proc/last_kmsg` which stores the kernel log message buffer of the last booted kernel and can be used to diagnose the previous kernel crash. It also may be possible to provide hardware-modified devices that expose either a JTAG connection for low-level debugging, or a serial port for simplified kernel debugging. The Google ADP1, for example, exposes serial RX/TX pins on its *ExtUSB* connector.

Finally, although students had some difficulty with device debugging, the overall experience using Android as the

homework project platform in our introductory OS course was positive. Students not only learned the OS principles being taught, but also gained valuable real-world skills such as practical device debugging and embedded code development. We have already heard from students who were able to directly apply the skills they learned in this course to a professional job or internship opportunity.

5. RELATED WORK

In recent years, courses using the Linux kernel for OS programming projects have become increasingly popular [1, 6, 8, 11]. However, these approaches focus primarily on Linux desktop and server environments and provide no hands-on experience with mobile platforms. Lawson *et al* [9] describe a single programming project where students modify a Linux kernel designed to run on an iPod. However, this single project does not incorporate any mobile device specific pedagogy and does not provide a rigorous, project-based curriculum for understanding OS principles in the context of mobile platforms. The BabyOS [10] and embedded XINU [3] projects focus on embedded systems, but lack the real-world applicability of a production Linux kernel. Furthermore, because Android is based on the Linux kernel, our approach provides a straightforward transition path for courses that already use the Linux kernel to incorporate the mobile and embedded concepts embodied in our Android projects. Our approach provides a mapping of structured Linux kernel programming projects onto the Android platform where students gain insight into real-world systems and learn practical skills immediately applicable in today's job market.

Various pedagogical OSes have also been developed [4, 13, 14], but none of them offer students practical experience or insight into modern mobile platforms. Atkin *et al* [2] developed a pedagogical OS focused on portability and mobility. However, all programming projects are done at user level in a simulator, and do not offer the engaging, real-world experience of using a popular, mobile computing platform such as Android on real mobile devices.

Several institutions offer courses in mobile application development [16] . These courses focus on mobile application APIs, and do not teach OS concepts. They offer no insight into the lower-level software infrastructure of mobile platforms on which applications run. In contrast, our method of using Android to teach OS provides students with a real understanding of how things work under the covers as embodied by the unique OS environment created by the Linux kernel running on a mobile device.

6. CONCLUSIONS

We have developed a series of hands-on Android Linux kernel programming projects designed to immerse students in a mobile computing environment while simultaneously teaching core OS principles. The projects progressively introduce OS principles and mobile computing, and implicitly teach students about real-world embedded device development. We use key aspects of modern mobile devices, such as orientation sensors, to enrich the hands-on experience and increase student engagement.

We created an Android virtual lab where both on campus and remote students complete Android Linux kernel programming projects using an emulator or mobile device. We also leverage a distributed version control system and live demonstration infrastructure for homework design, distribution, submission, and grading. Our experience with 100 students using Android to teach OS demonstrates that students enjoy using mobile devices while learning OS principles, and appreciate the practical skills they gain.

7. ACKNOWLEDGEMENTS

This work was supported in part by NSF grants CNS-1018355, CNS-0914845, and CNS-0905246, and a Google Research Award.

8. REFERENCES

[1] C. L. Anderson and M. Nguyen. A Survey of Contemporary Instructional Operating Systems for use in Undergraduate Courses. *Journal of Computing Sciences in Colleges*, 21:183–190, October 2005.

[2] B. Atkin and E. G. Sirer. PortOS: An Educational Operating System for the Post-PC Environment. In *Proceedings of the 33rd ACM Technical Symposium on Computer Science Education*, SIGCSE '02, pages 116–120, New York, NY, USA, 2002. ACM.

[3] D. Brylow. An Experimental Laboratory Environment for Teaching Embedded Operating Systems. In *Proceedings of the 39th ACM Technical Symposium on Computer Science Education*, SIGCSE '08, pages 192–196, New York, NY, USA, 2008. ACM.

[4] R. Cox, C. Frey, X. Yu, N. Zeldovich, and A. Clements. Xv6 – A Simple Unix-like Teaching Operating System. `http://pdos.csail.mit.edu/6.828/xv6/`.

[5] Deloitte Development, LLC. Deloitte Predictions for the Technology, Media and Telecommunications Sector, 2011. `http://www.deloitte.com/us/telecompredictions2011`.

[6] R. Hess and P. Paulson. Linux Kernel Projects for an Undergraduate Operating Systems Course. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE'10, pages 485–489, New York, NY, USA, 2010. ACM.

[7] O. Laadan, J. Nieh, and N. Viennot. Teaching Operating Systems Using Virtual Appliances and Distributed Version Control. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE'10, pages 480–484, March 2010.

[8] O. Laadan, J. Nieh, and N. Viennot. Structured Linux Kernel Projects for Teaching Operating Systems Concepts. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE'11, pages 287–292, March 2011.

[9] B. Lawson and L. Barnett. Using iPodLinux in an Introductory OS Course. In *Proceedings of the 39th ACM Technical Symposium on Computer Science Education*, SIGCSE '08, pages 182–186, New York, NY, USA, 2008. ACM.

[10] H. Liu, X. Chen, and Y. Gong. BabyOS: A Fresh Start. In *Proceedings of the 38th ACM Technical Symposium on Computer Science Education*, SIGCSE '07, pages 566–570, New York, NY, USA, 2007. ACM.

[11] J. Nieh and C. Vaill. Experiences Teaching Operating Systems Using Virtual Platforms and Linux. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education*, SIGCSE '05, pages 520–524, New York, NY, USA, 2005. ACM.

[12] OpenIntents. SensorSimulator – openintents – Sensor Simulator for simulating sensor data in real time. – Make Android applications work together. – Google Project Hosting. `http://code.google.com/p/openintents/wiki/SensorSimulator`.

[13] B. Pfaff, A. Romano, and G. Back. The Pintos Instructional Operating System Kernel. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 453–457, New York, NY, USA, 2009. ACM.

[14] A. S. Tanenbaum. A UNIX Clone with Source Code for Operating Systems Courses. *SIGOPS Operating Systems Review*, 21:20–29, January 1987.

[15] Wikipedia. Mobile Device – Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Handheld_device`.

[16] E. Woyke. iPhone and Android Apps 101. `http://www.forbes.com/2008/11/11/mobile-apps-colleges-tech-wire-cx_ew_1111mobileapps.html`.