ICS-OS: A Kernel Programming Approach to Teaching Operating System Concepts[*]

Joseph Anthony C. Hermocilla
Institute of Computer Science
College of Arts and Sciences
University of the Philippines Los Baños
College 4031, Laguna, Philippines
jachermocilla@uplb.edu.ph

ABSTRACT

Traditional approaches to teaching operating systems require students to develop simulations and user space applications. An alternative is to let them modify parts of an actual operating system and see their programs run at kernel space. However, this is difficult to achieve using modern real-world operating systems partly because of the complex and large source code base. This paper presents ICS-OS and the experiences and results of using it for teaching an undergraduate operating systems course. ICS-OS is based on the DEX-OS kernel which has a smaller source code base compared to mainstream operating systems, making it ideal for instruction. The students were able to demonstrate a deeper understanding of how a real operating system works by their succesful implementation of projects to enhance and extend ICS-OS.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design ; K.3.2 [**Computer and Information Science Education**]: Computer Science Education

Keywords

Operating systems, computer science education, kernel

1. INTRODUCTION

Operating systems is a core knowledge area in computer science education emphasized by the ACM and IEEE review task force for the computer science curriculum. Traditional approaches to teaching operating systems to undergraduates, as in the case at the author's institution, do not involve programming the components of an actual operating system that can run on real hardware. Instead, simulations are used or user space application development are done. Students, however, are more interested in writing code that runs at the kernel space, internal to the operating system itself. They want to concretize the abstract concepts in operating systems through kernel code. To achieve this, either the students can build an operating system from scratch[5] or modify an existing one.

Building an operating system from scratch is not the best option since a course is usually offered for a semester and there is not enough time to finish. In addition, the prerequisite knowledge needed to make an operating system may have not been acquired by the students yet. To write an operating system from scratch, one has to have knowledge of the processor architecture, assembly language, data structures, algorithms, and low-level C programming.

Modifying an actual operating system that runs on real hardware is a more viable alternative. However, the choice of the operating system to use is still an issue. In the past, several instructional operating systems have been proposed and developed. The next section briefly reviews some of them to highlight their strengths and weaknesses.

Two possible criteria for choosing the operating system to use are completeness and size of the source code base. An instructional operating system that does not implement high level abstractions like process management, memory management, and filesystems will unlikely be a good choice because of the missing features. On the other hand, an operating system with several thousands of lines of code and a complicated source directory structure will confuse students and will take more time to understand. Thus there should be a right balance between completeness and code size.

Recent developments in hardware emulation and virtualization have also made it easier to work with real-world operating systems. Testing a kernel need not require a reboot of the development machine for testing. Unnecessary boostrapping is no longer needed since the test machine is a software application running on the development machine itself.

The delivery of an operating systems course is usually through a lecture and a laboratory component. A popular textbook used by instructors in the lecture is the dinosaur book by Silberschatz and Galvin [10]. Typical laboratory activities involves learning to use a Unix-based operating system, developing simulations for different process scheduling algo-

[*]ICS-OS is an open source project hosted at http://code.google.com/p/ics-os/

7th National Conference on I.T. Education (NCITE 2009)
Capitol University, Cagayan de Oro City, 21-23 October 2009

rithms, understanding the fork() and exec() system calls, programming using user level threads, and implementing interprocess communication. All of these however are in user space and do not involve writing kernel code.

This paper presents ICS-OS and the experiences and results of using it for teaching an undergraduate operating systems course, specifically in the laboratory, at the Institute of Computer Science, University of the Philippines Los Baños. The students who took the course are in their third year and have completed the data structures and assembly language prerequisite courses.

2. RELATED WORK

Several instructional operating systems have been proposed and developed in the past[4]. This section briefly describes the popular ones to highlight their suitability for instruction. Mainstream operating systems such as Linux and BSD were not considered because they are too complicated already for use in teaching.

2.1 Minix

Minix[11] has been around for several years already and has been the running example in a textbook[12]. It is based on Unix, POSIX compliant, and runs on real hardware. Since its initial release, Minix has grown in size in terms of source code as well as in complexity which makes it difficult to use for teaching, given the advanced features like networking support.

2.2 Nachos

Nachos[6] is instructional software for teaching undergraduate, and potentially graduate, level operating systems courses. The Nachos kernel and hardware simulator runs as Unix processes implemented in C++. A success story in using Nachos for teaching is described by Gary [8]. The fact that Nachos runs as Unix processes limits its appeal because it cannot run on real hardware and is dependent on the host system.

2.3 GeekOS

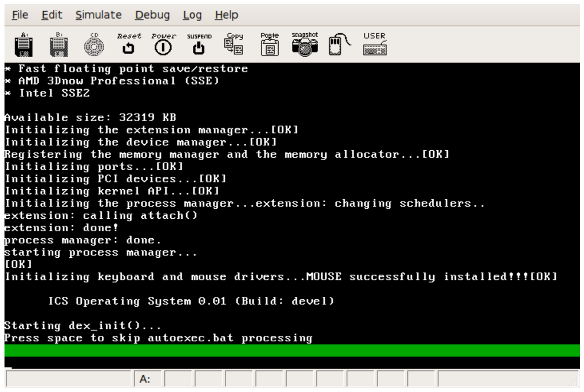
GeekOS[9] is an instructional operating system that runs on real hardware. The design goals for GeekOS are simplicity, realism, and understandability. Its main features include interrupt handling, heap memory allocator, time-sliced kernel threads with static priority scheduling, mutexes and condition variables, user mode with segmentation-based memory protection, and device drivers for keyboard and VGA.

2.4 DEX-OS

DEX-OS[7, 2] is an educational operating system based on an aspect-oriented approach to address cross-cutting concerns in operating systems design and implementation. The main components of DEX-OS are memory manager, process manager, device manager, and virtual file system. DEX-OS runs on the Intel 386 platform in 32-bit protected mode.

3. ICS-OS OVERVIEW

Similar to the Linux philosophy, ICS-OS can be considered as a *distribution* that is based on the DEX-OS kernel. ICS-OS provides an environment for learning operating systems



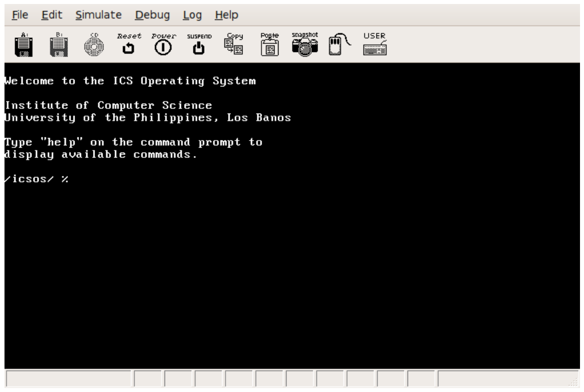
```
* Fast floating point save/restore
* AMD 3Dnow Professional (SSE)
* Intel SSE2

Available size: 32319 KB
Initializing the extension manager...[OK]
Initializing the device manager...[OK]
Registering the memory manager and the memory allocator...[OK]
Initializing ports...[OK]
Initializing PCI devices...[OK]
Initializing kernel API...[OK]
Initializing the process manager...extension: changing schedulers..
extension: calling attach()
extension: done!
process manager: done.
starting process manager...
[OK]
Initializing keyboard and mouse drivers...MOUSE successfully installed!!![OK]

      ICS Operating System 0.01 (Build: devel)

Starting dex_init()...
Press space to skip autoexec.bat processing
```

Figure 1: Booting ICS-OS.



```
Welcome to the ICS Operating System

Institute of Computer Science
University of the Philippines, Los Banos

Type "help" on the command prompt to
display available commands.

/icsos/ %
```

Figure 2: ICS-OS shell.

by kernel programming. A set of tools and utilities are packaged with ICS-OS to make it easy for students to write, modify, and test kernel code.

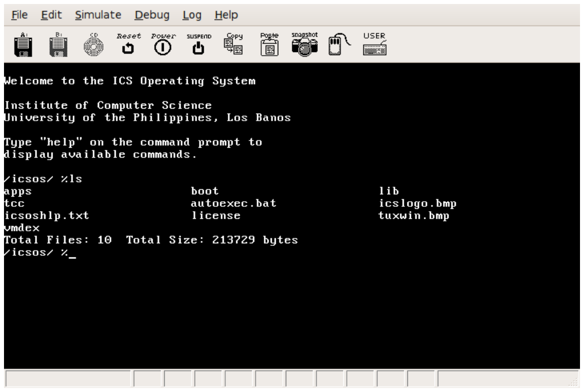
ICS-OS is generally divided into two main components, the kernel and user applications. The kernel is loaded at bootup using GRUB as the bootloader. Figure 1 shows ICS-OS booting. Essentially, the kernel stays in the main memory of the computer until shutdown. Since GRUB supports compressed kernel images, the size of the kernel binary is further reduced, correspondingly the distribution size. ICS-OS can thus fit in a single floppy disk. User applications, on the other hand, are loaded by the user manually through the shell or by initialization scripts. The shell in ICS-OS is implemented as part of the kernel (Figure 2).

These two components provide a diverse area for experimentation by the students. They can work on the kernel or develop user applications. A software development kit and a minimal C standard library is provided for programming applications. Figure 3 shows the *ls* command being executed by the shell.

4. TEACHING WITH ICS-OS

4.1 Development Environment

A convenient environment for operating system kernel development is essential. The compilers, assemblers, build tools, disk utilities, and emulators should be readily available on


File Edit Simulate Debug Log Help
A: B: CD Reset Power SUSPEND Copy Paste snapshot USER

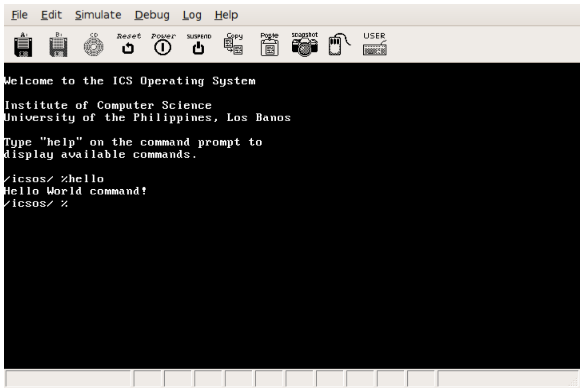
```
Welcome to the ICS Operating System

Institute of Computer Science
University of the Philippines, Los Banos

Type "help" on the command prompt to
display available commands.

/icsos/ %ls
apps                        boot                        lib
tcc                         autoexec.bat                icslogo.bmp
icsoshlp.txt                license                     tuxwin.bmp
vmdex
Total Files: 10  Total Size: 213729 bytes
/icsos/ %_
```

Figure 3: Executing the internal *ls* command.


File Edit Simulate Debug Log Help
A: B: CD Reset Power SUSPEND Copy Paste snapshot USER

```
Welcome to the ICS Operating System

Institute of Computer Science
University of the Philippines, Los Banos

Type "help" on the command prompt to
display available commands.

/icsos/ %hello
Hello World command!
/icsos/ %
```

Figure 4: Executing the newly created *hello* internal command.

the development machine. A Linux system can provide all these tools. Thus, the laboratory computers were reformatted and Ubuntu was installed. The following additional packages were also added to complete the setup.

- Make
- GCC/TCC
- NASM
- Bochs
- VirtualBox
- Subversion (optional)

4.2 Student Activities

After the development environment was ready, students were given activities to familiarize themselves with the source code of ICS-OS. ICS-OS was written in the C programming language and some assembly language.

4.2.1 Bootloader and Intel 386 Protected Mode

This activity, although not directly related to ICS-OS source code, was conducted in order for the students to understand how a PC boots, and eventually loads an operating system kernel. They were asked to develop a simple bootloader using assembly language. The exercise was then extended so that the CPU is switched into the protected mode instead of the default real mode. This is because modern operating systems run in protected mode, as with the case of ICS-OS.

4.2.2 Download, build, and test

The first activity was to download the source code of ICS-OS[3]. The source code is distributed in a tar.gz file and via Subversion. After extracting, students built and tested ICS-OS using the commands below.

```
(1) $make
(2) $sudo make install
(3) $bochs -q
```

The first command creates the kernel binary in ELF format. The second command builds the distribution floppy image using the loopback device for mounting, and thus requires administrator privileges, so the *sudo* command is used. The third command starts Bochs[1] which loads ICS-OS (see Figure 1).

4.2.3 Add a new shell command

The best way to understand how ICS-OS works internally is to start with the shell since it contains the commands the users can use to access the operating system services. Thus, in the next activity, students modified the kernel, specifically the shell, by adding an internal hello shell command. The students were directed to navigate to the console source code (kernel/console/console.c). They were asked to modify the function console_execute() by adding the following simple code fragment.

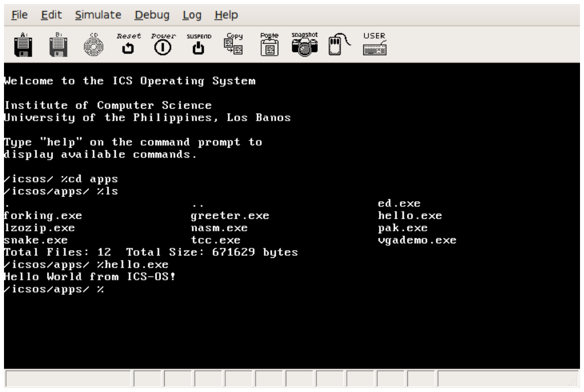
```
/*--START--*/
if (strcmp(u,"hello")==0)
{
  printf("Hello World command!\n");
}
else
/*--END--*/
```

The result of this activity is shown in Figure 4.

4.2.4 Develop user applications

Application development in ICS-OS is done via a software development kit. The externally callable functions and system calls are implemented in the file sdk/tccsdk.c. User programs are linked against this file to generate the executable code that is compatible and runnable within ICS-OS. To ease the development process, a Makefile template was created to perform the build automatically. As an example, the Makefile for the hello.exe application is shown below.

```
CC=gcc
ICSOS_ROOT=../..
SDK=../../sdk
CFLAGS=-nostdlib -fno-builtin -static
```


File Edit Simulate Debug Log Help
A: B: CD Reset Power SUSPEND Copy Paste snapshot USER

```
Welcome to the ICS Operating System

Institute of Computer Science
University of the Philippines, Los Banos

Type "help" on the command prompt to
display available commands.

/icsos/ %cd apps
/icsos/apps/ %ls
.                        ..                       ed.exe
forking.exe              greeter.exe              hello.exe
lzozip.exe               nasm.exe                 pak.exe
snake.exe                tcc.exe                  vgademo.exe
Total Files: 12  Total Size: 671629 bytes
/icsos/apps/ %hello.exe
Hello World from ICS-OS!
/icsos/apps/ %
```

Figure 5: Executing the *hello.exe* application inside ICS-OS.

```
LIBS=$(SDK)/tccsdk.c $(SDK)/libtcc1.c $(SDK)/crt1.c
EXE=hello.exe
$(EXE): hello.c
        $(CC) $(CFLAGS) -o${EXE} hello.c $(LIBS)
install: $(EXE)
        cp $(EXE) $(ICSOS_ROOT)/apps
uninstall:
        rm $(ICSOS_ROOT)/apps/$(EXE)
clean:
        rm $(EXE)
```

For the activity, the students were asked to develop the hello.exe application, which is not part of the shell but can still be executed within ICS-OS. First they created the hello/ folder relative to the contrib/ directory and copied the sample Makefile in it. The Makefile for hello.exe is shown above. Then, the hello.c source file, which contains the application logic, was coded inside the hello/ folder. The application was built and installed using the commands below.

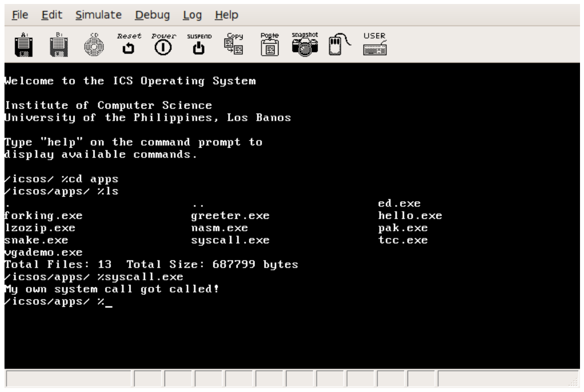
```
(1) $make
(2) $make install
```

The *install* target in the Makefile will simply copy the executable to the apps/ folder. It is only after the distribution floppy image is created that the application will be available inside ICS-OS for execution (see subsection 4.2.2). Figure 5 shows the output of hello.exe.

4.2.5 Add a new system call

The last guided activity for the students was to add a system call. System calls are the primary mechanism to access the services provided by an operating system. The students were directed to open kernel/dexapi/dex32API.c and add the following function definition.

```
int my_syscall(){
   printf("My own system call got called!\n");
   return 0;
}
```


File Edit Simulate Debug Log Help
A: B: CD Reset Power SUSPEND Copy Paste snapshot USER

```
Welcome to the ICS Operating System

Institute of Computer Science
University of the Philippines, Los Banos

Type "help" on the command prompt to
display available commands.

/icsos/ %cd apps
/icsos/apps/ %ls
.                        ..                       ed.exe
forking.exe              greeter.exe              hello.exe
lzozip.exe               nasm.exe                 pak.exe
snake.exe                syscall.exe              tcc.exe
vgademo.exe
Total Files: 13  Total Size: 687799 bytes
/icsos/apps/ %syscall.exe
My own system call got called!
/icsos/apps/ %_
```

Figure 6: Executing the *syscall.exe* application inside ICS-OS.

Then, the function api_init() was modified by adding a call to api_addsystemcall(). The system call number chosen for my_syscall(), in this example, is 0x9F.

```
api_addsystemcall(0x9F,my_syscall,0,0);
```

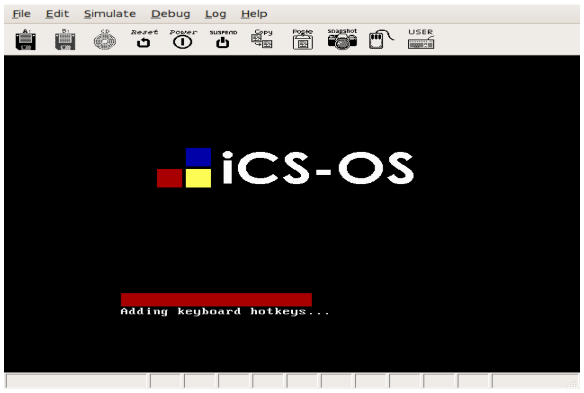
A user application, syscall.exe with code shown below, was developed to test the new system call directly by invoking it through the number. Figure 6 shows the result of the execution.

```
int main()
{
  dexsdk_systemcall(0x9F,0,0,0,0,0);
  return 0;
}
```

4.2.6 Student Projects

The previous activities were designed to familiarize the students with the code base of ICS-OS. To fully demonstrate their understanding of the concepts, they were asked to submit project proposals to extend or enhance ICS-OS. The project proposals may be system programs or application programs that can run inside ICS-OS. The following list some of the projects proposed and implemented by the students.

- Simple graphical user interface (see Figure 7)
- Standalone shells with advanced features, such as history
- Text editor
- Archiving utilities
- Help system (similar to man in Unix)
- Time/Date utilities
- File search utilities
- Disk utilities



Adding keyboard hotkeys...

Figure 7: A VGA based splash screen for ICS-OS.

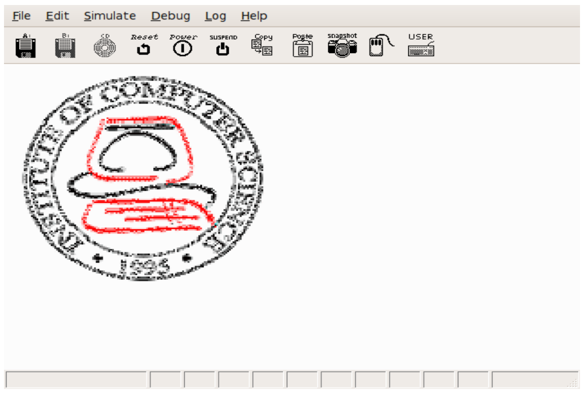


Figure 8: Displaying a bitmap.

- File splitters
- Enhanced process viewers
- Image viewers (see Figure 8)
- Simple interpreters
- Dictionary/Spelling utilities

5. RESULTS AND OBSERVATIONS

Overall, the students were able to appreciate how an operating system works with the use of ICS-OS. Given a properly setup development environment, documentation, and instructor guidance, kernel programming can be made easier for students to enable a deeper understanding of the abstract concepts.

However, during the course was offered, the following observations were noted. First, not all students are fluent in C programming. Some students have never worked on a relatively large source code base and were lost easily in the directory structure of ICS-OS. Second, during the project proposal stage, students proposed ambitious projects that eventually did not get implemented. Lastly, during the implementation phase of the projects, some of the common functions needed by the students for their programs are not yet implemented. One such function is scanf(). Thus, students need to find alternative functions, such as gets(), to get input from the user or implement the missing functions on their own.

The use of ICS-OS in teaching operating systems concepts, however, has achieved the following objectives for the students.

- They learned how an operating system works from the bare metal (real hardware/emulator).
- They were able to see the big picture on how a computer system works.
- They were able to understand the separation of kernel space and user space.
- They were able to understand how programs are loaded and how processes are created and executed.
- They were able to understand how system calls work and the importance and advantages of using application programming interfaces and software development kits.

6. CONCLUSION

In this paper, the author presented ICS-OS as an instructional operating system for teaching operating system concepts to undergraduate students through kernel programming. A high success rate in the projects indicates that a kernel programming approach is better in making students understand how a real operating system works compared to just programming simulations and user space applications.

7. ACKNOWLEDGMENTS

The author would like to thank Joseph Emmanuel DL. Dayo for contributing DEX-OS to the open source community and the students of CMSC 125, Operating Systems, during the Second Semester AY 2008-2009 who took the course using ICS-OS.

8. REFERENCES

[1] Bochs. `http://bochs.sourceforge.net/`.
[2] DEX Extensible Operating System. `http://sourceforge.net/projects/dex-os/`.
[3] ICS-OS: An Instructional Operating System. `http://code.google.com/p/ics-os/`.
[4] C. L. Anderson and M. Nguyen. A survey of contemporary instructional operating systems for use in undergraduate courses. *J. Comput. Small Coll.*, 21(1):183–190, 2005.
[5] M. D. Black. Build an operating system from scratch: a project for an introductory operating systems course. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 448–452, New York, NY, USA, 2009. ACM.
[6] W. A. Christopher, S. J. Procter, and T. E. Anderson. The Nachos Instructional Operating System. Technical report, Berkeley, CA, USA, 1993.
[7] J. E. Dayo and C. L. Khan. DEX-OS: An Aspect-Oriented Approach in Developing an Educational Extensible Operating System for the IBM PC and Compatibles. In *Proceedings of the 4th*

Philippine Computing Science Congress. Computing Society of the Philippines, 2004.

[8] J. E. Gary. Using Nachos in an upper division operating systems course. *J. Comput. Small Coll.*, 18(2):337–345, 2002.

[9] D. Hovemeyer, J. K. Hollingsworth, and B. Bhattacharjee. Running on the bare metal with GeekOS. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 315–319, New York, NY, USA, 2004. ACM.

[10] A. Silberschatz and P. B. Galvin. *Operating System Concepts (5th Edition)*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[11] A. S. Tanenbaum. A UNIX clone with source code for operating systems courses. *SIGOPS Oper. Syst. Rev.*, 21(1):20–29, 1987.

[12] A. S. Tanenbaum and A. S. Woodhull. *Operating Systems Design and Implementation (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.