THOS: A simple educational computer operating system

Tyler James Higley

December 2014

Willamette University, Salem OR, 97301

Abstract

The goal of this project is to write from scratch a simple minimally functional Operating System for the x86 architecture using C and Assembly Language. The design and implementation will take primary inspiration from Unix as well as similar minimal/educational Operating Systems projects such as Minix and Xinu. In the process I will attempt to educate myself in the field of Operating Systems and develop a closer understanding of what goes on in the layers between the hardware and user applications. If successful this project should not only be at least minimally usable for simple tasks, but also fill the role of an educational tool for others wishing to learn about Operating System concepts, design, and implementation.

Introduction

There are few programs that can rival the complexity of an Operating System. It is a piece of software that no modern computer can realistically function without. Whether it be the casual user checking their email, the scientific researcher graphing the results of thousands of lines of processed data, or the expert programmer compiling a piece of complicated software, everyone makes use of their Operating System.

Although there are a number of open-source OS distributions with which one could freely inspect and study the source code, the overwhelming complexity of these modern OS's make comprehension very challenging for novices to the subject.

It is my intention during the remaining year of my college education to personally write from scratch (more or less) a minimally functional command line operating system. In doing so I hope

to educate myself about relevant topics and techniques in OS development, establish a closer relationship and understanding with the inner workings of an Operating System as well as the underlying architecture they run on. My OS will be written for the x86 architecture set and will be in many ways based on Unix. Furthermore, one of my principal goals with this project is is to produce code that is not only functional, but simple and well documented such that others interested in the subject could learn from it.

Goals and Implementation

At a minimum I would like my operating system to be console based, able to read and write files to memory, and run short predefined programs such as basic Unix commands.

As a mid-range goal, I hope to include a functional file system with features such as permissions and navigation. In addition, I would also like to include the functionality for simple applications, such as a vim-like text editor, to take control of the screen. This contrasting with programs that run without interrupting the console.

Lastly, if everything runs smoothly, it would like to allow for user defined programs to be run. This could take a form similar to bash scripts, essentially executing a preset list of commands, or I could write some kind of simple compiler and programming language so that a user could create and run their own short and simple programs that are actually compiled on my Operating System.

Since this project will involve significant understanding of the topic, I will be taking a significant amount of time to focus on research and education. I have purchased a number OS textbooks such as Operating System Concepts 8th Edition by Abraham Silberschatz, Operating Systems in Depth by Thomas W. Doeppner, and Operating Systems Design and Implementation 3rd Edition by Andrew S Tanenbaum. The first edition of Tanenbaum's book was Linus Torvalds original inspiration behind the Operating System Linux. Tanenbaum is also the creator of another small educational OS called Minix which will also serve as a principal resource. There is also a

website called OSDev.org that serves as a community for individuals interested in OS creation. Additionally I will be taking a look at brokenthorn.com's OS Development Series, OSDever.net, Nick Blundell's PDF Writing a Simple Operating System, as well as other simple Operating Systems like Xinu, MikeOS, Xv6, KOS, SOS, NachOS and BlueFire OS.

The actual implementation will be done in x86 Assembly language using NASM and C using GCC. I will be primarily testing using QEMU (Quick EMUlator) but may eventually also use VirtualBox.

I am well aware that there is no way for me to realistically implement every aspect of a fully functional OS in a single semester but I do intend to write something that can stand on its own. The main components I will be focusing on are the User Interface, the Kernel, and the File System. Although my initial scope is limited, I intend to continue work on this project after the conclusion of my senior year. I hope to continue to add and improve until the project is fully functional

As for this semester, here is a list of what I hope to accomplish in more or less chronological order.

- Bootstrap the Operating System into memory
- Implement Function User Interface and Shell
- Implement necessary System Calls
- Write basic Unix commands
- Implement Inode based File System
- Write a memory manager
- Reading and Writing to memory
- Write a simple text editor
- Allow for Bash-like script (Thrash/Trash script)
- Create a simple language and write a compiler
- Allow user to run their own programs

Aside from the the obvious challenge of implementation, I anticipate it being challenging rapidly becoming a expert and learning enough to make this project a success. I will also need to be mindful of what features and components I can/should leave out and use my resources effectively.

I have already spent a lot of time working with x86 Assembly on some of the foundational aspects of the OS development by following various bootloader tutorials. As of now I have implemented a bootloader that bootstraps the OS, switches from 16 bit real mode to 32 bit protected mode and runs a bare C kernel. I have also figured out how to access keyboard scancodes and interpret them. Next I want to get the command line terminal and shell operational, but first I plan on taking some time to read various books and online material and get a better idea of how it all fits together. I also want to take a few steps back and relook at everything I have done so far.

Sources

Blundell, Nick. *Writing a Simple Operating System — from Scratch*. N.d.

Comer, Douglas E. *Operating System Design: The Xinu Approach*. Englewood Cliffs, NJ: Prentice-Hall Intern., 1984. Print.

Doeppner, Thomas W. *Operating Systems in Depth*. N.p.: John Wiley & Sons, 2011. Print.

Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1988. Print.

Silberschatz, Abraham. *Operating System Concepts*. 8th ed. Palatino: John Wiley & Sons, 2009. Print.

Tanenbaum, Andrew S. *Modern Operating Systems*. 2nd ed. New Jersey: Prentice Hall, 2001. Print.

Tanenbaum, Andrew S., and Albert S. Woodhull. *Operating Systems: Design and Implementation*. 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 1997. Print.

"Kryos." *Onyxkernel*. N.p., n.d. Web. 17 Nov. 2014. <https://code.google.com/p/onyxkernel/>.

"Operating System Development Series." *BrokenThorn Entertainment*. N.p., n.d. Web. 17 Nov. 2014. <http://www.brokenthorn.com/Resources/OSDevIndex.html>.

OSDev Wiki. N.p., n.d. Web. 17 Nov. 2014. <http://wiki.osdev.org/>.

"Write Your Own Operating System." *Joel Gompert*. N.p., n.d. Web. 17 Nov. 2014. <http://joelgompert.com/OS/TableOfContents.htm>.