Computer Science Thesis:

Writing an Operating System

Tyler Higley

Spring 2015

Advisor: Professor Ruehr

Institution: Willamette University

THOS (THOS Has Occasional Success)

An Operating System by Tyler James Higley

Abstract:

An operating system is the software that interfaces between users and the underlying hardware of the machine. It manages all of the computers resources and provides a platform on which other programs may operate. For my senior computer science thesis I wrote a simple minimally functional operating system for the x86 architecture using C and Assembly Language. The design and implementation, when possible, took primary inspiration from the Unix operating system. It has an inode based filesystem with a directory structure, interrupt driven user input, and command line interface. The OS in question, THOS, is integrated with the GRUB bootloader and is equipped with a builtin text editor.

Contents

1. Introduction

As a computer science student at Willamette University, the course I found most interesting was CS 353: Architecture and Compilers. In this course we dived into how computers work behind the scenes, from building logical circuits in a simulator, writing assemblers for and programs in an assembly language, to finally writing a compiler for a simple language. Although this class did much in the way of demystifying the low-level concepts unseen by many programmers, in fitting the class into a single semester, there were a number of concepts that there was simply not time to cover. One such area that had particular interest to me was operating systems. It was after taking this class that I decided that for my senior computer science thesis project, I wanted to create my own OS from scratch and, in the process, continue learning about the low-level concepts that happen behind the scenes all of the time.

2. Plan and Process

2.1 Original Vision

When I originally began this endeavor I was well aware that a large portion of this project would involve simply learning and planning. I knew that my expectations for what I could accomplish in a single semester would need to be flexible, however, when pressed, I came up with the following goals. At a minimum I wanted my OS to be console based, able to read and write files, and run a number of UNIX based commands. At the midrange level, I hoped to include a functional file system with features and directories. I also wanted to include builtin programs such as a vim-like text editor. Lastly, my stretch goal was to allow for user defined programs such as bash scripting.. Although I was able to do most of the things from my original plan, as the project wore on it became clear that there were many other challenges to tackle that are not immediately apparent to those unfamiliar with the concepts and implementation of operating systems. As time progressed, so did my understanding of the task I had ahead of me, and with this understanding came a change in both my expectations and focus.

2.2 The Evolution

Spring 2014:

I began tinkering with OS tutorials as early as the spring of 2014. It was during this time that I came to get some experience with x86 Assembly Language and was able to write a simple "Hello World" OS in 16 bit real mode.

Summer 2014:

As the summer began I had a full-time job but I continued to work on my early OS in my spare time. It took a long time to fully wrap my head around the concepts, however, slowly but surely I was able to write a bootloader that would load the rest of the operating system into memory, switch from 16 bit real mode to 32 bit protected mode, and jump to a simple OS kernel written in C.

Fall 2014:

In the fall, as part of the Senior Seminar Prospectus class, I began planning out the continuation of my project. It was at this point when I crafted the original goals mentioned above. I also collected a number of reference books such as *Operating Systems Design and Implementation 3rd Edition* by Andrew S Tanenbaum, *The Design of the UNIX Operating System* by Maurice J. Bach, and *The C Programming Language. 2nd Edition* by Brian W. Kernighan and Dennis M. Ritchie. Furthermore, I discovered the incredibly helpful OSDev Wiki which contains dozens of Tutorials and other various documentation pertaining to OS development. Although I was primarily in a preparation stage, I didn't stop working on my early OS.

Winter 2014:

At the beginning of winter break I went on vacation to Mexico. It was during that trip that I began working on the project that would truly become my operating system. Although I had accomplished a number of things already, I decided to start over from scratch since I had a better idea of how to lay everything out. I also made the decision not to use my own bootloader as I had been doing before. Instead, as recommended by a number of tutorials, I hoped to eventually take advantage of the GRUB bootloader. In the mean time I used an emulator that allowed me to test my kernel without the use of a bootloader. My primary goal for that vacation was to write a command line user interface that functioned like a UNIX terminal and that is exactly what I did.

Spring 2015:

Once I had returned to Willamette for my final semester it was time for things to get serious. I decided that for the first incarnation of the project, I needed three things: a user interface, a filesystem, and a process manager. Then I would create a shell program that could tie them all together. Since I already had the user interface, I started by working on the filesystem. I spend a few weeks writing a C program that more or less imitated the UNIX inode filesystem, at least on a minimal level. At this point, due to the large and complicated nature of implementing processes, a processing manager, and system calls when I didn't really have anything cohesive up and running yet, I elected to cut all of it out of my initial version. I spent the following two weeks integrating the filesystem into the user interface and added an assortment of basic commands (ls, rm, cat, chmod, wc). I also took the time to write a simple vi based text editor called te. Around March 5th I had what I was calling THOS: Version 0.1.

In what I will call phase 2 of the semester, I decided I needed to take a step back and do some research. I made a list of 9 things that I would like to ideally complete by the end of the project. The thought being that each part would add .1 to the version number and I could complete the semester with version 1.0. For the most part I was able to stick to this list however there were a few divergences. I decided to spend some time working on a couple tutorials courtesy of jamesmolloy.co.uk and wiki.osdev.org. Using these resources, I was able to set up both a global descriptor table and an interrupt descriptor table (the later of which allowed for interrupt driven user input) and I was able to fully integrate THOS with the GRUB bootloader. These tutorials were very helpful but they did pose some challenges. They were very old and often relied on tools that no longer exist. I ended up having to figure a lot of it out for myself. I went on to add a history, arrow key navigation for text input, and users with corresponding commands to add, remove, and change them. Around April 5th, I had what I would end up referring to as Version 0.6.

As the end of the semester and the final due date were quickly approaching, I had to once again make some final decisions as to what was most crucial to my target Version 1.0. I went through a lot of my old progress to polish things up and fix a number of little bugs. I also went ahead and added a few more commands, but the biggest thing that I felt like my project was really missing was directories in the filesystem. I did some research into how UNIX implemented directories and made a plan of my own for their implementation. I dived into it and was able to finish their implementation before May 5th. Now after long last and hard work, I present you with my Computer Science Senior Seminar Thesis project: THOS Version 1.0.

3. My Operating System (THOS)

3.1 Overview

THOS can best be described as a simple minimally functional x86 Operating System with an inode based filesystem with a functional directory structure, interrupt driven keyboard input, and command line terminal user interface with UNIX like commands that allow users to interface directly with the OS and the files it contains. The OS supports 18 basic commands and 2 additional built-in programs. Although THOS can do many things, it is still a work in progress and has a number of limitations (see 4.2 Future Plans).

3.2 Builtin Commands

Below is a list of all 18 basic commands THOS supports as well as their syntax and a brief description.

ls — Lists all files in the current directory. The -l tag will include permissions, owners, and file size with the name of each file.

ls [-l]

cat — Prints the contents of a specified file to the screen.

cat <file name>

rm — Deletes the specified file from the current directory. Requires user ownership or root to run.

rm <file name>

wc — Reports the line count, word count, and char count of the specified file.

wr <file name>

history — Lists the previous 20 commands that have been run. User can also use the up arrow to access past commands.

history

chmod — Changes the permissions of a specified file based on 3 digit number (ex. 754 -> rwxr-xr—). Requires user ownership or root to run.

chmod <new permissions> <file name>

chown — Changes the owner of a specified file. Requires ownership or root to run.

chown <user name> <file name>

su — Changes the current user to specified user.

su <user name>

adduser — Creates a new user with given name.

adduser <user name>

deluser — Removes user with given name.

deluser <user name>

listus — Lists all user accounts.

listus

echo — Prints to the screen the given space separated set of strings.

echo <string>+

passwd — Allows user to set their password.

passwd

pwd — Prints the working directory path to the screen.

pwd

cd — Changes the current directory to specified directory inside of the current directory.

cd <directory name>

mkdir — Creates a new directory with the given name inside of the current directory.

mkdir <new directory name>

rmdir — Deletes directory with the given name from the current directory.

rmdir <directory name>

mv — Moves a specified file from the current directory to a specified directory inside of the current directory.

mv <file name> <directory name>

3.3 Other features

In addition to the various basic commands, THOS has two built-in programs, te and pong.

TE — is a simple text editor based on vi. The user may create a new file or edit an existing file. All four arrow keys may be used to navigate the file. The program enforces file permissions so it can only be run if the current user has read permissions and can only save the file if the user has write permissions.

te <new file name/file name>

PONG — is simple implementation of the arcade game of the same name. Although pong is not equipped with an abundance of features, it is meant to serve as an example for what could be created in the future.

pong

Aside from the built-in programs, THOS has a few more features of note. First of which is its integration into the GRUB bootloader. This allows for a bit more reliability and consistency in the OS as a whole. Also of note is the Terminal user interface which scrolls text, once the screen is full, may be cleared by typing "clear", displays the current user and directory to the user, and supports tab autocompletion for files within the current directory.

4. Conclusion

4.1 Lessons Learned

During the implementation of this project I gained significant exposure to both C and x86 Assembly language. I had to follow along with considerably old tutorials, and figure out how to improvise when the instructions were wrong, left things out, or used tools that were no longer available. I had to explore how filesystems work and how directories are implemented. I had to understand how the visual terminal seen on the screen was represented behind the screens. I had to read books and blogs and make judgments as to what I could feasibly accomplish. I learned to use the hardware emulation software QEMU, the assembler NASM, the compiler GCC, and the linker LD. I had to write bash scripts to run, build, and analyze the project. But most of all, I learned just how large an complicated even a simple operating system can be.

4.2 Future Plans

As one could imagine, I did not implement every aspect of a modern day operating system and even the things that I did implement have a number of limitations. However, I do not intend Version 1.0 to be the final incarnation of THOS. I plan to continue to work on and improve my projects for fun in my spare time. Below is a list of limitations and missing features I would one day like to overcome.

Commands — There are a number of additional commands that it would be nice to include. Also commands should handle paths to files.

cp, diff, more, ff, grep, date, cal, kill, ps, chgrp, addgrp, rmgrp

Text editor editable size — Currently, the text editor is limited to the size of the screen. I would like to be able to allow users to scroll through larger files.

Thrash Script — It would be useful if users could write files similar to bash scripts that could be run to execute multiple commands.

Filesystem Indirect/Double-Indirect blocks — Although I have implemented the indoor based filesystem, I did not account for indirect or double-indirect blocks. These would allow for larger sized files.

Permanent filesystem — Although you can read and write to the filesystem, it is really just a data structure in a running program. If the computer was shut off, all changes would be lost.

Pipe standard output — Currently all programs print their output to the screen. By implementing pipes, the result of one command could become the input of the next.

Process Manager and System Calls — This would require a very significant overhaul of the OS. Currently all command interface directly with the OS. It would be ideal if individual programs used System Calls to to interface with it.

4.3 Final Thoughts

All in all I am very pleased with the results of my senior thesis project. In the end I was able to do most of the things I originally set out to do and I learned a lot in the process. Furthermore, it has vastly expanded my interest in the area of operating systems and I look forward to learning more about them in the future. Hopefully my project is such that if someone else is interested in going down the same path I have, they can use my source code and documentation as a guide or reference in their own project. As my senior thesis course and my undergraduate education as a whole comes to a close, I have had to put a cap on THOS and present it to the world. Although one chapter is ending, I truly believe this is not the end but merely the beginning. Just as we all do, THOS will continue to evolve.