2022-2023 Van Eney '09 Fellows Program

Jeffrey Huang – Mr. Soden, Ms. Jackson

1 Introduction

1.1 The Big Question

Advancements in technology have always influenced how human civilizations survive, evolve, and triumph, but nothing compares to the computing revolution in the last century. Programmable silicon is now in everything, and computers are essentially a requirement to function in modern daily life. Driving all of these is a piece of software called the kernel.

We have come a long way. In the early days, you would have to manually assemble teams of workers to put holes in punch cards that would be fed into a room-sized machine. Now, you can code even on your iPhone using its touch screen display. The phone can then execute the program at speeds 100,000 times faster than the Apollo 13 computer. Slowly but surely, computer processes have been abstracted away, hiding what actually goes on, and encapsulating modern programming into high-level programming languages. Code has become very human-readable, coming to the point where it feels like you're writing English sentences rather than coding specific operations for the computer to function.

Abstraction levels in computer science detail how close a program is to the bare metal: in other words, how many steps it takes for a piece of high-level code to be transformed into manipulating the individual electrons that travel through the CPU. The lower the level, the closer it is to the hardware. Throughout my coding journey, I've always been intrigued by what exactly is going on when I run a program. I'm not satisfied with metaphors and symbolic explanations–I want to get at the heart of the computer: the kernel. It is the conductor in an orchestra of components and software; it's the bridge between the physical hardware you interact with and the code that powers your applications. One specific CPU architecture[1] is of interest. It's the one used by Intel and used in all of their processors, and is the most common processor in computers: x86. This leads me to my motivating question:

How are modern x86 operating systems built and how do they bridge the gap between hardware and software?

1.2 Motivating This Project

I started learning to program in 6th grade and have been passionately pursuing it in my spare time ever since. It's a constant process of discovering new fields in computer science and adapting to the rapidly changing landscape. It's impossible to learn everything. Even Bjarne Stroustrup, the creator of C++, one of the oldest and most popular programming languages still used, admits he only knows 70% of everything there is to know about the language that he created. I noticed recently that my interests and projects have brought me down to lower and lower levels of programming. In my most recent project, I wanted to learn more about how GUIs[2] work. With my experience using C++, a low-level language, I set out on porting a game to the TI-84+CE calculator. I was able to learn so much working in a limited development environment. I had to learn how things like memory architectures and LCD screens work. Even then, I was working with a toolchain, an abstraction that hides away even lower-level procedures. I find myself constantly trying to figure out what's really happening underneath those layers of abstraction.

In the cybersecurity competitions I participated in, I even got as close to manually inspecting machine code in order to reverse-engineer or even exploit its vulnerabilities. These experiences have taught me how to

[1]Specific instruction set used by processor, detailing what operations it can do
[2]Graphical User Interfaces

code in assembly, how the C compiler works, and other things that I have never thought about or investigated before.

I definitely have a lot to learn, but I consider my accomplishments in competitions and projects[3] have prepared me to undertake this project.

2 Goals

2.1 Plan

My plan is to write a functioning operating system from scratch, which can then be theoretically installed in a fully functioning custom-built calculator.

2.1.1 Phase 1

The first phase is writing the OS (Operating System). This includes two main parts: the bootloader and the kernel. The bootloader is the bridge between pressing the power button and having the kernel started. Modern operating systems usually use an abstracted and streamlined tool called UEFI, which hides away some of the environment implementations. I will be avoiding that, and I will start off with my custom bootloader. This will teach me more about what goes under the hood, and how exactly everything works together.

The second part is the kernel. It is to the CPU what the subconscious is to the brain. It handles the vital processes behind the scenes in order to create a fully usable and fluid experience on the computer. At the heart of every fancy operating system, there is a kernel managing everything. It handles low-level operations like managing memory, handling interrupts, setting up the file system, and interfacing with device drivers. I will also be including the necessary drivers in the kernel, programs that handle communication with other devices, such as the hard drive, screen, or keyboard. Here is a brief rundown of my OS specifications.

Here is a brief rundown of my OS specifications.

Kernel Specifications:

- Intel i386 architecture compilation target
- 32bit
- Written in C/C++ and assembly
- Paged memory
- POSIX.1 compliant[4]

Included Drivers:

- Keyboard driver
- Visual drivers
- LCD driver
- Serial driver
- USB driver
- HDD driver
- SSD driver

[3]Some are hosted on my GitHub

[4]Seehttps://www.opengroup.org/austin/papers/posix_faq.html

2.1.2 Phase 2

Phase 2 is the next step toward exploring low-level programming. If I get my operating system to a satisfactory level, I will be able to port and reduce it to actual hardware with an x86 microprocessor. This will act as a possible demonstration of what my operating system can do.

To see how far I can take this, I will continue my research into the parts needed to create a fully functioning calculator. Like the TI-84+CE, it can not only do simple calculations, but it can also run programs, graph complex equations, store notes, do simulations, and much more. This requires not only a fairly powerful microprocessor but also some basic hardware.

One of my main goals for this phase is to see if I can make a version cheaper than the TI-84+CE but still have more capabilities in both hardware and software. The price of the name brand is over-inflated due to it being the most popular name brand on the market. My version of this calculator will be far less locked down, allowing people to not only learn to program but also to run complex calculations at faster speeds than the slow 48MHz of the TI-84's ez80 processor.

The working list of parts and tools includes:

- Board with x86 microprocessor
- Breadboards
- 9-Pin Keypad matrix
- 320x240 LCD Screen (Resolution not final)
- Other I/O[5] components

2.2 Schedule

In order to get 40% of my project done during the summer, I will have to strictly and realistically partition my time in order to get everything set up and prepared. Not only will I report weekly updates on my progress, but I will keep up with any and all timeline requirements set by the Fellows Committee. My work will come in two phases, but not necessarily will I have a specified point at which I stop work on Phase 1 and start Phase 2. I will first start off creating the operating system (Phase 1), and as it approaches a usable state, I will begin work on Phase 2, the hardware assembly. However, through this project, I will regularly switch between the two phases in case I need to add necessary features or fix bugs or errors. Making revisions is an inevitable part of the process and I will be working as hard as I can to bring this project to completion

3 Possible Mentors

I am currently considering Mr. Baraty to be a mentor. He is Severn's computer science teacher and has extensive knowledge of DIY hardware. Since he works in the Graw Innovation Center, he can help me learn how to use the different tools and assemble my hardware.

4 Why Fellows

Access to resources like mentors and professionals is also one of the reasons why I want to pursue this as a Fellows project. I can't stress enough how invaluable of an opportunity this is to learn and grow. This is an ideal Fellows project for many reasons. As opposed to an ISP, this ambitious project will take more than three months to complete and will greatly benefit from mentorship. Not only does it create something tangible, a custom piece of hardware, but it will also create something permanent: the code that will live on GitHub in a repository. The features I add to my operating system and calculator are measurable qualities that will allow me to be more on course with my end goal. This is also a project that no other Fellow has done before. It will be an example of a computer science project built by someone who has been interested in this

[5] Input/Output

field for a long time and made his Fellows project his crowning achievement before he enters college. I was always inspired by the Fellows presentations when I was an underclassman and knew it was a powerful way to demonstrate what a student can accomplish at Severn. This project will not only set an example for what the Fellows program can produce but through my beginning-of-the-year and end-of-the-year presentations it can inspire other students to follow this path.

5 Applications Beyond Severn

Beyond the Severn Fellows program, I hope to make an influence on both the school and me. I feel that there is a great lack of computer science in Severn. I want to help the culture surrounding this fascinating and amazing subject by creating something that no other Fellow has done before. My goal is that this will inspire younger people to pursue their own questions and interests in computer science and to have something exist at Severn long after I graduate.

This Fellows project will also serve as a personal learning experience. Not only will it be another project to be proud of, but it will prepare me for my further studies in computer science. I want to study computer science in college, and I feel that the culmination of mentorship and self-determination will make me a better thinker, researcher, and learner.

6 Qualifications

During my time at Severn, I have always sought the most rigorous courses in every subject to challenge my learning and thinking abilities. Not only that, but I have also made personal commitments to compete and learn at some of the highest levels. Over the course of three years, I have been practicing for the USACO and Codeforces competitions. These test your ability to think quickly and create programming solutions to solve problems in the most efficient way possible. Studying algorithms and data structures has fundamentally changed how the way I think about code and has given me experience in learning deeply about one particular subject. I also spent time in marathon-style cybersecurity competitions during my sophomore and junior years. Cyberstart America was the first real venture into cybersecurity. Over the course of a few months, I was constantly learning and attempting the problems in each challenge. With each problem, I had to think creatively and learn something new in order to solve it. The dedication and motivation to learn about something I'm passionate about are what led me to qualify for the national competition and score in the top 3% of students. I have also completed other long-term projects both team-based and personal coding projects, most of which are hosted on GitHub. These vary in terms of time spent, but my dedication to exploring new projects and ideas has led me to be an excellent problem solver, organizer, and ultimately someone who can bring my ideas to completion.