**Project Title**

**Your Name**

College of Information and Computer Sciences

msonderegger@umass.edu

5/21/23

**Advisor:** Tim Richards

**Second Committee Member:** Joe Chiu

**Research Type:** Thesis

1. Introduction

Operating systems are a fundamental part of modern computing but current courses are not properly preparing students to understand and work on them. As computers have become more ubiquitous and advances in virtualization have continued to abstract users further and further from the hardware they are using, operating system courses have followed suit, focusing on the concepts that are crucial to modern operating systems without showing how they are implemented in practice. While understanding concepts like virtual memory, concurrency, and file IO is important, operating system design is inseparable from hardware interfaces, so it is equally important to understand how operating system designers work with hardware to achieve these goals. By teaching concepts without practical implementation students are left confused and unprepared for operating system development, because while they have learned what an operating system does, they have not learned how it does it, which is the ostensible goal of an operating system class. My thesis aims to bridge this gap by creating a simple, modular, and extensible operating system based on the Xv6 educational operating system designed by MIT for the RISCV architecture. In conjunction with this I will be creating a course and preparing reference materials that will help students understand the RISCV architecture and instruction set. Together these two resources will let students build their own operating system from scratch that will can run on a bare metal RISCV chip. By doing this they will achieve a deeper understanding of what it means to design an operating system and the complexities that arise from working with real hardware interfaces. The main focus of my research will be to design the operating system, and design it in such a way that it can be easily configured. In order to do so I have chosen the RISCV architecture as the target for development. The advantages of using RISCV are its small base instruction set, its open instruction set architecture, and its simplicity (looking at you x86). To run my operating system I will be using QEMU, a popular emulation software that will allow me to run the operating system on any computer and easily use debugging tools like GDB to inspect what it is doing and determine the cause of bugs that arise in the development process. This operating system will be written in a combination of assembly and C. Understanding assembly is necessary to understanding the connection between software and hardware and any operating system course would be remiss if it did not teach students how to read and write assembly code. However, assembly is difficult to program in because it is prone to bugs and hard to understand. Therefore, the operating system will largely be written in a higher level language, namely C. C has many advantages over other similar languages like C++ and Rust because it is simple and it is easy to see how it compiles down to equivalent assembly. It is also very common and potential students will very likely have already been exposed to it before. In order to design this operating system so that it can be simple, understandable, and extensible, the following guidelines have been laid out. They may be subject to change as development continues.

1. Every function should have a comment. Every section in a function should have an explanation of what it’s doing.
2. Functions should be no longer than 60 lines of code.
3. A single file should have no more than 10 functions.
4. Directories will be used to group files with similar functions together.
5. Function names should avoid abbreviations where at all possible.
6. Braces will always be used with conditional and looping structures. If a statement is small it may be included on the same line with braces surrounding it.
7. Macro usage should be limited to #defines where at all possible. Macros must be used if there is a magic number (e.g. max number of processes, memory max, etc.)
8. A README shall exist and be maintained through out the development process.

Finally the operating system will be compiled through makefiles to simplify the distribution and compilation. If time allows I plan to research ways to simplify the development environment by creating a web based interface to QEMU and GDB. GDB already provides a programing interface through the GDB Machine Interface (GDB/MI), allowing a developer to programmatically inject commands into a running GDB process. There are numerous bindings for this API in various languages, particularly Python through, pygdbmi. Additionally there is an existing tool called gdbgui that creates a web interface to pygdbmi allowing a developer to easily read values and execute gdb commands from a browser of their choice. It runs locally so it is a perfect candidate for creating a simple development environment that a student could use. I plan to research how to connect gdbgui to a running QEMU process to allow easy debugging of an emulated process.

1. Significance

Operating systems are undoubtedly a crucial part of modern computing. Without them we would be unable to multitask, access files, and connect to the network. In fact recent developments in supervisor and hypervisor technology can rightly be viewed as operating system development as well, since they provide the same kind of abstractions to operating systems themselves. These developments have enabled the rise of virtualization and cloud computing that has defined the past decade of computing developments. In short, the modern computing world would be impossible without operating systems. Clearly there is a need for intelligent and competent operating system designers to maintain the operating systems we rely on for daily life and to develop the operating systems that will make tomorrow possible. Systems classes do exist and are part of most modern computer science curriculums, however they frequently prefer to teach operating systems concepts from a high level without allowing students to actually interact with and develop their own operating system. This is quite understandable as the importance of quick and reliable operating systems has led to extensive research and modern operating systems are indeed complex. The current Linux code base has over 28 million lines of code (citation needed). This level of complexity is daunting to professors and students alike. It would be impossible to teach all of the Linux operating system in a single course (or even ten!) and so modern operating system classes have moved to teaching operating systems from a high level, focusing on concepts, like virtual memory, process management, file IO, without teaching how these are actually achieved in practice. This is a significant failing. While those concepts are undoubtedly important and ought to be taught, on their own they are not enough. Operating systems are fundamentally grounded in hardware techniques and understanding how they work in concert to achieve those concepts is just as vital to understanding how an operating system works and how it is developed. Indeed for some students, hiding trap tables or page tables within a black box of scheduling algorithms and memory mapping, makes it harder to understand. This research seeks to bridge this gap by creating an easy to teach course and an easy to modify operating system and that will introduce students to operating system design principles through practice rather than theory and teach operating systems from the bottom up. This is significant because while educational operating systems like Xv6 do exist, they are often hard to understand because have been written by operating system experts, they have few comments, and the comments present are unhelpful. This research advances the field in two crucial aspects. First and foremost it will improve the quality of instruction for students at the University of Massachusetts Amherst and beyond by providing them with a high quality education and real world experience in operating system design. This will ensure that there are competent developers who are capable of creating innovations in the operating system design world and putting it into practice in the real world. Additionally it will provide an important starting place for further research into RISCV operating system design. RISCV is a new and still developing CPU ISA that continues to see more and more adoption in the embedded world but relatively less adoption in the traditional computing world. This is in part due to the fact that there is a lack of high quality operating systems designed for the RISCV ISA. By innovating and improving on an existing RISCV operating system this research will encourage adoption of an ISA that allows for both energy efficient and high speed computation. Additionally, by creating an easy to use development environment this research will aid the future development of RISCV operating systems as well.

1. Background

*What have other researchers already written/published on your topic? (2-7 pages)*

* Demonstrate your understanding of the primary literature you covered in 499Y.
* Consult your research committee chair about additional key literature needed for your Honors Thesis/Project manuscript.
* Consult your advisor about what citations style to use and use it consistently.
* If you have a significant Preliminary Findings section below, this section may be on the shorter side of the page range. If you do not have preliminary findings, this section should be 5-7 pages.

1. Methodology

*How are you conducting your research? (2-3 pages)*

* Present a detailed description of the methodology or tools that you will use to perform research in the area that you have selected. For example, if you will be building a software system, what is the system architecture, what languages, frameworks, or APIs will you use? If you will be working on theory, what key prior results and techniques will you be building on? If you are doing experimental work, describe experimental designs.
* Describe any specialized training you have received or will receive to conduct your research (e.g., lab safety certification or human/animal testing), and indicate who provided the training.

1. Preliminary Findings

*What have you accomplished already? (length may vary)*

* Describe any preliminary research findings from 499Y. Describe what you have accomplished so far.
* In some cases, you might have already completed much of the research described in the proposal, in which case this section could be quite long. The text from this section can be reused for the final thesis document.
* In other cases, 499Y may have only involved a literature review and additional learning already reviewed in Section 3, in which case this section should state that there are no preliminary research findings at this time.

1. Evaluation

*How will your work be reviewed and graded? (1-2 pages)*

* Break your research into a sequence of milestones that can be evaluated individually (you should strongly avoid setting up a situation where successful completion of your project hinges on achieving a single monolithic goal).
* Describe the work to be completed in each milestone in detail.
* Specify how each of your individual milestones will be evaluated.
* Specify how your overall research will be evaluated depending on how many of your milestones are achieved.
* Describe how your committee will provide feedback regarding your progress.
* If you are registering for an Honors Project, be specific about the artifact that you will produce in addition to the Project manuscript (for most students, this will just be CD/DVD with a copy of your code).

This research will be broken into the following milestones:

1. Completing the instruction set listing on the RISCV wiki
2. Completing the reference pages for the RISCV wiki, providing a short and concise summary of the RISCV specification
3. Completing the Xv6 reference pages, providing a short and concise summary of all functions and their role in the operating system overall
4. Reorganizing the Xv6 code into more sensible modules. Additionally making the organization of the codebase more apparent by renaming the functions to remove abbreviations and using prefix name spacing.
5. Commenting the Xv6 source code fully
6. Creating a development environment to build Xv6, run it with QEMU, and debug it with GDB.
7. Modularizing and editing the Xv6 source code such that it is possible to build it:
   * With or without virtual memory
   * With or without concurrency
   * With a custom scheduler
8. Creating a course plan with slides, exams, and projects
9. Creating a web based development environment, possibly based on gdbgui
10. Making substantive changes to Xv6

These milestones will be evaluated according to the following criteria:

1. All of the RISCV instruction set listing pages included on the landing page are fully and and correctly completed. The page is largely free of typos and other grammatical mistakes.
2. There are summaries for the following features the RISCV ISA provides outside of the instruction set listing. The pages are largely free of typos and other grammatical mistakes.
   * All CSSR registers that handle interrupt masking, interrupt delegation, execution mode, trap vectors, and virtual memory. Extra CSSRs may be included.
3. Every function and file should have a page describing it’s purpose and any important notes about the functionality or behavior. The pages are largely free of typos and other grammatical mistakes.
4. This is largely a qualitative measurement so the grading will be subjective. Consideration shall be given to demonstrated effort to move files around and rewrite code to ensure that a single file has no more than 10 functions and the functions are grouped sensibly and legibly.
5. Again comment quality is a subjective measure, however checking that every function has a header comment describing it’s purpose and comments explaining all non obvious sections. Consideration should be given to the accuracy and thoroughness of the comments.
6. This one is fairly simple. I should demonstrate that I have a makefile or script that is capable of setting all of these systems up to run.
7. It should be possible to build a version of Xv6 with any of those possibilities. This can be given partial credit based on how many of the possibilities are possible.
8. I will work closely with Professor Richards to propose a curriculum that will teach students how operating systems work from the ground up and incorporate the operating system I have created into several projects that involve adding functionality to the operating system. I will provide sample solutions for all projects assigned.
9. Similar to 6 I should be able to demonstrate that I have a makefile or script that is capable of setting the system up to allow for debugging.
10. This is a stretch goal and so it is relatively more vague. I should make a contribution that includes at least 3 more functions and no less than 100 lines of code. I should be able to demonstrate it working

Achieving up to milestone 5 will earn a C. This is due to the fact the work is important to the course but not necessarily very hard, as it is mostly copying and pasting into tables. Achieving milestone 7 will earn a B. This work is harder as it involves writing code and a thorough understanding of how all the pieces of the operating system fit together. Achieving up to 9 will earn an A. After revising the existing Xv6 codebase, creating the course is the next goal of this thesis. Fully completing this stage will require many hours of both programming and careful distillation of the concepts that are crucial to modern operating system development. This is where I will demonstrate the knowledge I have gained through my research. As previously stated milestone 10 is a stretch goal and will require significant work. I hope to be able to complete this and truly make a UMass specific operating system, however as it is not necessary to my goal I do not think my grade should be tied to it.

1. Communication

*What are the expectations about meetings with your committee chair and other committee member(s)? Be specific. (1 paragraph)*

* How often are you meeting with your committee chair? How often will you meet with the full committee?
* What are your committee chair’s expectations of such meetings? What are your committee members’ expectations about meeting with you?
* What time commitment (number of hours of work per week) is expected to be applied to your research between meetings with your committee chair?

1. Timeline

* MM/DD/YY: Milestone 1
* MM/DD/YY: Milestone 2
* MM/DD/YY: Milestone 3
* …
* MM/DD/YY: 1st Draft of thesis submitted to advisor   
   (No later than 2 weeks before the last day of classes)
* MM/DD/YY: 2nd Draft of Thesis submitted to HPD   
   (No later than 1 week before the last day of classes)
* MM/DD/YY: Oral Defense (No Later than last day of classes)
* MM/DD/YY: Final Submission to CHC (No Later than last day of classes)

1. References

You should include references for all of the key readings from your 499Y semester plan.

A total of 5+ references is expected

* Use the reference and citation style used in your research sub-area.
* “The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213”, Editors Andrew Waterman and Krste Asanovic, RISC-V Foundation, December 2019. Retrieved December 12, 2022, from <https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>.
* “The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203”, Editors Andrew Waterman, Krste Asanovic, and John Hauser, RISC-V International, December 2021. Retrieved December 12, 2022, from <https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf>.
* “xv6: a simple, Unix-like teaching operating system”, Russ Cox, Frans Kaashoek, and Robert Morris, MIT, August 2021. Retrieved December 12, 2022, from <https://pdos.csail.mit.edu/6.S081/2020/xv6/book-riscv-rev1.pdf>.
* “Safe Systems Programming in Rust”, Ralf Jung, et al., Communications of the ACM, April 2021. Retrieved December 12, 2022, from <https://dl.acm.org/doi/10.1145/3418295>.
* “Teaching Operating Systems: Just Enough Abstraction”, Philip Machanik, Rhodes University, July 2016. Retrieved December 12, 2022, from <http://dx.doi.org/10.1007/978-3-319-47680-3_10>.