**Project Title**: ***Creating Own Operating System from Scratch***

**Names of the group members**: Max Coplan, Kirsten Wood

**Objectives and Motivations**:

**Motivations**: I have been a Linux user and developer for a number of years now. Ever since I started using Linux and Bash I have been fascinated by how they work, and the level to which the user can automate their workflow, and completely tailor their experience to however they want. From the kernel you run, to your compiler, to your bashrc, to your desktop environment, everything is under your control. I always enjoy learning about how Linux works internally. I want to try to make my own Linux distribution from scratch. Building Linux from scratch will teach me about all that makes Linux tick, how things work together and depend on each other. And how to use these parts of the operating system to customize it to my own tastes.

**Objectives**: Build my own Linux operating system from scratch. Starting with nothing but a blank SD card, slowly work my way up through an assembler, linker, compiler, libraries, tools, and the programs that make an operating system. When installing a regular Linux distribution, you often end up installing a lot of programs you probably would never use. And most of the programs that are installed you have little or no knowledge about how they work or how they interact and compliment the other services on your system. I want to have a bare minimal OS, in which I know what every component is and how it works. I want an operating system compiled entirely from source, allowing me to understand, configure, modify, and audit everything. In addition, this gives me extreme flexibility regarding how the system performs and what programs I need. This operating system will comply with various specifications that define modern day Linux distributions; it will be POSIX.1-2008 compliant, Filesystem Hierarchy Standard (FHS) version 3.0 compliant, and Linux Standard Base (LSB) version 5.0 compliant.

**Plan of work and timeline**:

Many of the ideas and methodologies behind this project have been laid out in the book [Linux From Scratch](http://www.linuxfromscratch.org/), which gives the readers ideas about how to construct an operating system, as well as lay out the structure of designing an operating system from scratch. The book gives a nice overview of the process, but the actual process of designing and creating my operating system will go far beyond the scope of the book: requiring research, troubleshooting, creativity, and problem solving. This project will no doubt be a large undertaking and will require a lot of time and effort to do effectively. I have wanted to pursue something like this for a while now, and when I signed up for this course and learned about the project, I saw this as a great opportunity to finally do it. This semester has been my lightest semester so far, which should allow me to devote the necessary time to complete my operating system.

I have been working on this project nearly since the moment you mentioned it in class. This time has been spent designing the OS, decided how it should work; coding, compiling, installing, and configuring the system; troubleshooting problems, learning how Linux, gcc, bash, all their tools, etc. work and how they relate to each other; sorting through issues such as circular dependencies, compile-time errors, dynamic linking references, filesystem misconfiguration, incompatible ARM code, cross-compiling dependencies, architecture incompatibilities, and much more.

In the last month I have spent over 60 hours working on this project, and it will require many more hours to come. I hope to lay out a rough timeline of how I’ve spent my time thus far, and estimate the required time needed in the future.

I started this project a little over a month ago with nothing but a blank SD card and a Raspberry Pi 4B. Given the Raspberry Pi’s ARM Cortex-A72 64-bit processor operating in dual ARMv7l 32-bit backwards-compatible mode, it presents a unique challenge trying to design this system. Most resources on the internet are for x86 or x86\_64 Intel processors, which has required me to do a lot of original innovation on my own.

With a blank SD card, you must have some way of formatting it and building the required software onto it until the point when you can boot from the SD card alone. To do this, you must have an existing Linux system capable of compiling and writing to the SD card. From this point forward this existing Linux distribution will be referred to as the “host”. This host will be used as a starting point to provide the necessary programs, including a compiler, linker, and shell, to build the new system.

**Hours 1-10:**

First we must create a new Linux native partition and file system. This is where the new LFS system will be compiled and installed. Then we must acquire or write our own packages and patches to build the new Linux system and store them on the new file system. Afterwards we must set up an appropriate working environment on the host. Then we will need to write, create, code, cross-compile, link, and configure the basic development toolchain which will be used to build the actual system. This is done to resolve circular dependencies; for example, to compile a compiler, you need a compiler.

The overall pattern in this process is:

1. Make a rudimentary, feature-poor, basic tool
2. Use this basic tool to create other tools (these tools are also basic)
3. Use your basic tool along with the basic tools made by the initial basic tool to **remake** that first tool
   1. This new tool is more powerful and more capable than the previous one
4. Use this new tool to remake the previous tools, as well as make new tools not previously possible
5. Use these new, better tools to remake the remade first tool even better than before
6. Repeat

**Hours 10-30:**

Each round of remaking tools is called a “pass”. In the first pass of the toolchain, I made a basic assembler, static linker, symbol editor, and memory tools (collectively called “**Binutils**”). I then made a very basic and limited version of the GNU C Compiler (**GCC**). This version of GCC was very limited and many people would not even recognize it as a real C compiler. For example, this version of GCC did not even have **Glibc**—the C standard library. Now you may ask, “What is C without the C standard library?” and the answer is “Not much.” This version of GCC was entirely statically linked (obviously, as there are no libraries yet on the system) and is only capable of doing completely static linking.

The next step was the build the C standard library. **Glibc** was compiled by the toolchain programs built in the first pass. Again, this version of Glibc is very limited and many may not even recognize it as being Glibc. But with Glibc in hand a second pass of the toolchain was built; including new assemblers, linkers, Binutils, and GCC. This time, the toolchain was dynamically linked using the second pass toolchain. Afterwards, we build many other tools using this second pass toolchain. The most notable from a user-perspective are **Bash** and the GNU **Coreutils**. After this, the rest of the project no longer depends at all on the host system, except for the running kernel.

**Hours 30-60:**

With these new tools in-hand, the full Linux operating system can be built. We **chroot** to enter our constructed virtual environment on the SD card and start a shell whose root directory is the root of our SD card. Essentially this is similar to rebooting and instructing the kernel to mount our operating system as the root partition. It is not quite a full reboot, but instead uses **chroot** because our system is not quite ready for native booting.

From here I made nearly the entirety of what looks like a Linux distribution. Following the previous pattern, I remade the **Binutils** and **GCC** for a **third pass**, compiled the **Linux kernel API headers**, remade Glibc, the preprocessors, Coreutils, **Bash**, and much more; creating the frontends for various **networking** interfaces, set up the **filesystem**, created **users** and groups, defined the **permissions** model, added support for robust dynamic **linking**, and much more.

This section has taken the majority of the time so far and has required tons of effort in not just creating and configuring the system, but extensive troubleshooting and problem solving for many of the things that have gone wrong. One particularly traumatizing example was after the third pass of GCC things seemed to be working fine for a while until suddenly many of the tools I had made with GCC stopped working in various ways. It was not clear the cause of the issue (and it did not occur to me at the time that what all these tools had in common was that they were made with that version of GCC) as they stopped working in different ways, and took me nearly 3 hours to determine the cause of it: I had incorrectly set up and configured the dynamic linker in GCC. This means that all the tools I had made with the third pass of GCC, and all the tools made with those tools, were all linking to the wrong libraries and were useless. At the time I had been using this third pass of the toolchain for almost 8 hours. Meaning I had to go back, delete everything I had done for that time, figure out how to correctly do dynamic linking, and do everything over again. A lot of wasted time indeed.

**Today (hours 60-64):**

At the time of writing the most recent thing I have done is the compilation, installation, and configuration of the eudev package of tools. This package contains programs for the dynamic creation of device nodes.

**Future: hours 64-70:**

This operating system is not done yet, and there is still much work to be done. The next big section to be done is designing the **init** and boot system of the OS. When the computer boots, the first user-space program to be launched is **init**, that sets up basic programs such as **login** and runs a script called **rc**. A big design choice I needed to make when creating this operating system is whether to use the current and more widely-used **systemd** init system or the older and more modular **System V** init system. I ended up choosing the **System V** boot process given its simplicity and modularity, and I believe I will learn more about the Linux boot and **init** process from using System V over systemd.

**Hours 70-90**

After installing and configuring **init** and other basic system configuration, the **kernel** and **boot loader** will need to be set up. So far I have found compiling and getting GCC to work has been the most challenging part. However, I suspect that actually configuring, compiling, installing, and setting up my own custom kernel will end up being the most difficult part of this process. If this section follows the patterns of my previous tools, it will take about 7 or 8 hours to get *something* up and running, and another 12 or 13 hours after that for troubleshooting and redoing it correctly.

Once this is completed I will have a **functional, complete**\*, **bootable Linux distribution** with which I can continue my curiosity in operating systems; and continue developing, tweaking, refining, and updating to suit my needs as time goes on.

\*see the next section on evaluation methodology for how I define “complete”

**Expected results and evaluation methodology**: It is difficult to determine when you are “done” making an operating system. Should you just be able to boot it? Must it have some level of networking capability? Should you be able to do certain kinds of programming on it? Must it have a graphical desktop environment? Of course, with a project like this it is never really “over” forever. There are always more things you can do with it. This is where the common OS standards become useful. Perhaps a system that complies with POSIX, FHS, and LSB can be considered a “full” operating system. Linux From Scratch agrees with that standard but adds an additional requirement to have a “full” LFS operating system: it must be able to remake itself. To make your own Linux system you must have an existing Linux system. So, once your LFS environment has reached the point you could make another OS from it is when the authors say you have made a minimal Linux distribution.

To summarize: The completed Linux operating system will be POSIX, FHS, and LSB compliant and will have all the necessary tools and structures in place to make another Linux operating system from scratch from it.

**Deliverables:** A POSIX, FHS, and LSB compliant operating system that will have all the necessary tools and structures in place to make another Linux operating system from scratch from it.

**References**:

[POSIX](https://en.wikipedia.org/wiki/POSIX)

[FHS](https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard)

[Linux From Scratch](http://www.linuxfromscratch.org/lfs/index.html)

[LSB](https://en.wikipedia.org/wiki/Linux_Standard_Base)