

# SYSTEMS II — PROJECT 1

## The beginnings of a kernel

### 1 Overview

Now that we have a BIOS that can load a kernel, we need to start the kernel itself. It must “take control” of processor, load a *process*, and jump to it, prepared to handle the possible interrupts that will occur. More specifically, our kernel must establish a *trap table* full of pointers to *interrupt handlers*. One of the interrupt handlers, the *system call handler*, must provide the capability for a process to intentionally vector to the kernel for a particular task.

### 2 Getting started

Do the following to prepare yourself to work on this project:

1. **Login** to the CS workstations or one of our servers (`vega.cs/remus/romulus`).
2. **Create and change into** a directory for this project. You could name it something clever, such as `project-1`, or `through-these-cheese-trees-freezy-breeze-blew`.<sup>1</sup>
3. **Copy** my source code file that will serve as a skeleton for your kernel:

```
$ cp ~sfkaplan/public/COSC-261/project-1/kernel.asm .
```

4. **Edit** your new `kernel.asm` file with the editor of your choice, fulfilling the requirements in the next section, 3.

### 3 Your assignment

Augment your kernel to give it the following characteristics:

- *Perform an initialization sequence:*
  1. **Initialize a trap table:** When the kernel begins execution, it should create and set the entries of a *trap table*. (See Chapter 4 of the k-system documentation for details on each interrupt.)
  2. **Set the trap base register:** Set the processor’s TBR with the base address of the trap table.
  3. **Load the first program:** Assume that the system has at least three ROMs, where the ROM’s after the BIOS and kernel contain *user programs*. Find the third ROM—that is, the first user program—and load its contents into a free portion of main memory.
  4. **Execute the program:** Jump from the kernel to the beginning of the loaded user program. This program must execute in *user mode* (in contrast to the *supervisor mode* in which the kernel executes). [Hint: See the `JUMPM` instruction in the k-system documentation.]

---

<sup>1</sup>Whence?

- *Handle all possible interrupts:* Each interrupt should trigger a printed message to the console, indicating what just happened. For now, the kernel should `HALT` after printing this message (since any program that triggers, say, a *divide by zero* interrupt cannot be resumed).

[Hint: Note the importance of incremental development here. Don't start with the printing of messages, because that's complex code. Have each interrupt set some register with some "magic constant" that will show you, should things execute correctly, which interrupt handler was reached. Once that mechanism is in place and works, you can add the printed messages. Always think in terms of small steps that can be written, tested, and debugged before progressing.]

- *Implement the `EXIT` system call:* The `SYSCALL` interrupt should be handled by code that examines some memory location (registers count as *memory locations*) for a *system call code*. Assign some constant to be the code for the `EXIT` syscall. The syscall handler should examine this location, determine if an `EXIT` was requested, and then do it. That is, end the program's execution by freeing its memory, and then...
- *Execute a sequence of user programs:* After one user program properly exits, find the *next* ROM that holds a user program, and then load and execute that one.

## 4 How to submit your work

Use the CS submission systems to submit your work. Specifically, you will need to submit your `kernel.asm` file. You may use either of two ways, from the servers and/or workstations, to use the submission system:

1. **Web-based:** Open a browser,<sup>2</sup> and point it at:

`https://www.cs.amherst.edu/submit`

2. **Command-line based:** From a shell running on any of the servers (`remus/romulus/vega.cs`), or on any of the workstations in Seeley Mudd 007, you can submit your work via the following command:

```
$ ~lamcgeoch/submit kernel.asm
```

You must, of course, be within whichever directory contains your BIOS code. Once you issue this command, you will be prompted to select the correct course and assignment for which you are submitting work; those choices should, I hope, be self-evident.

This assignment is due at **11:59 pm on Sunday, February 28<sup>th</sup>**.

---

<sup>2</sup>Be sure to open it from within the server or the workstation! If you're using your own laptop or a public Windows/Mac machine, don't open the browser from within there, since it won't have access to the files on the server/workstations.