CS350 **Operating Systems** Winter 2020

Assignment Zero

1 Introduction

The purpose of this assignment is to familiarize you with OS/161 and Sys/161.

OS/161: OS/161 is a simple operating system kernel, which is made available to you along with a small set of user-level libraries and programs that can be used for testing. The baseline OS/161 that we distribute to you has very limited functionality. Each of the CS350 assignments will ask you to improve OS/161 in some way to add additional functionality to the baseline.

Sys/161: Sys/161 is a machine simulator. It emulates the physical hardware on which OS/161 runs. Apart from floating point support and certain issues relating to cache management, it provides an accurate emulation of a server with a MIPS R3000 processor. You will use Sys/161 each time you want to run OS/161. However, you are neither expected nor permitted to make any changes to Sys/161.

Before you can complete this assignment, you will need to obtain and build a copy of OS/161. Sys/161 and the toolchain needed to build and debug OS/161 and its applications are pre-installed and ready to use in the `linux.student.cs` environment. If you are planning to work on your own machine, rather than in the `linux.student.cs` environment, you will also need to obtain and install both Sys/161 and the toolchain before you will be able to build, run, or debug OS/161 code.

2 Revision Control

OS/161 is a non-trivial program - the kernel alone consists of more than 30,000 lines of C code. Although it is not required, we urge you to use a revision control and source code management system, such as SVN or git, to manage your copy of OS/161. Feel free to use any system you are familiar with. If you are not familiar with these systems, any effort you devote now to learning one of them will pay itself back to you in this course and beyond. The course web page (see "Assignment Information") includes links to information about some of these systems. If you use a source code management system, you should first place your fresh copy of OS/161 under revision control, then create a working copy, and then proceed (using your working copy) to configure, build and modify OS/161 as described below.

3 Preliminaries

Before you start working on this assignment, you must complete the following steps:

1. Install a copy of OS/161 in your `linux.student.cs` account. If you plan to work on your own machine, install OS/161, Sys161, and the toolchain in your machine instead. For those working on the `linux.student.cs` machines, there are step-by-step installation instructions at
   `https://www.student.cs.uwaterloo.ca/~cs350/common/Install161.html`
   For those working on their own machines, there are instructions at
   `https://www.student.cs.uwaterloo.ca/~cs350/common/Install161NonCS.html`

2. Once you have a installed OS/161, you will need to learn how to to modify it, build it, run it, and debug it. There is a detailed guide to the use of OS/161 at
   `https://www.student.cs.uwaterloo.ca/~cs350/common/WorkingWith161.html`
   Read and understand this *before* you start working on this assignment.

4 Assignment Requirements

For this assignment, your are expected to make two minor changes to the OS/161 kernel:

4.1 Customize the Kernel Boot Output

When OS/161 boots, it produces output that looks similar to this:

```
sys161: System/161 release 1.99.06, compiled Sep  9 2013 23:13:03

OS/161 base system version 1.99.05
Copyright (c) 2000, 2001, 2002, 2003, 2004, 2005, 2008, 2009
   President and Fellows of Harvard College.  All rights reserved.

Put-your-group-name-here's system version 0 (ASST0 #17)

4916k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0

cpu0: MIPS r3000
OS/161 kernel [? for menu]:
```

Note the line that says "`Put-your-group-name-here's system` ...". Your first assignment is to change OS/161 such that the kernel identifies itself as your kernel when it boots. For example, if your name were Liberty Valance, your kernel should say "`Liberty Valance's system` ...".

Once you have done this, make sure that you can re-build and run OS/161 with your customized boot output.

4.2 Add a Kernel Menu Command

The OS/161 kernel includes a simple system that allows debugging messages to be displayed when the kernel runs. There can be different types of debug messages, and the kernel can be told to display only messages of certain types. For example, the file `kern/thread/thread.c` includes the statement

```
DEBUG(DB_THREADS,"Forking thread: %s\n",name);
```

This defines a debugging message of type `DB_THREADS`.

The debugging mechanism is implemented in the file `kern/include/lib.h`. This file also includes definitions of all of the pre-defined debugging message types, such as `DB_THREADS`. There is a kernel global variable, `dbflags`, which defines which types of debugging messages should be displayed when the kernel runs (see `kern/lib/kprintf.c`). In the baseline code, `dbflags` is set to zero, meaning that no debugging messages are displayed.

After the OS/161 kernel boots, is displays a command prompt and waits for a command, like this:

```
OS/161 kernel [? for menu]:
```

If you type `?`, you should get a list of available commands and sub-menus, one of which is the operations sub-menu. **For this assignment, your second requirement is to add a new command to the kernel's operations sub-menu**. The new command, which must be called `dth`, should enable the output

of debugging messages of type DB_THREADS. (If such messages are already enabled, the command should have no effect). Thus, any kernel commands that are run after dth should run with DB_THREADS debugging messages enabled.

To do this, you will need to understand how the debug message mechanism works and how the kernel menu system works. The latter is implemented in the file kern/startup/menu.c. You should be able to complete this assignment by changing only that single file.

To test your new kernel option, we will use it to run one or more of the kernel's built-in thread tests with DB_THREADS debugging enabled using your new dth command. The kernel has several simple thread tests (e.g., tt1, tt2, tt3) that can be run from the kernel menu prompt.

For example, without DB_THREADS debugging enabled, thread test 2 (tt2) produces output like this:

```
OS/161 kernel [? for menu]: tt2
Starting thread test 2...
0123456701235674
Thread test 2 done.
Operation took 0.662769000 seconds
```

However, if DB_THREADS debugging has been enabled by running your new dth command, this test should instead produce output similar to this:

```
OS/161 kernel [? for menu]: tt2
Starting thread test 2...
Forking thread: threadtest0
F0orking t0hread: threadtest1
F1orking t1hread: threadtest2
F2orking t2hread: threadtest3
F3orking th3read: threadtest4
F4orking th4read: threadtest5
F5orking t5hread: threadtest6
F6orking t6hread: threadtest7
77
Thread test 2 done.
Operation took 0.717793640 seconds
```

Notice the messages produced by the DEBUG statement from threads.c.

5 Submitting Your Work

To submit your work, you must use the cs350_submit program in the linux.student.cs computing environment. If you are working on your own machine, you must first copy your modified OS/161 code to your account in the linux.student.cs environment, and then submit it from there.

Important! You must use cs350_submit, not submit, to submit your work for CS350.

Assuming that you followed the standard OS/161 setup instructions, your OS/161 source code will be located in $HOME/cs350-os161/os161-1.99. You should run cs350_submit from the $HOME/cs350-os161 directory, like this:

```
% cd $HOME/cs350-os161
% /u/cs350/bin/cs350_submit 0
```

The 0 argument to the cs350_submit command indicates that you are submitting assignment zero. cs350_submit packages up your OS/161 kernel code and then submits it to the course account using the regular submit command. This assignment only briefly summarizes what cs350_submit does. You can (and should) learn more on-line by reading https://www.student.cs.uwaterloo.ca/~cs350/common/SubmitAndCheck.html

When you run the cs350_submit command, you should see output that looks something like this:

```
@cpu20.student[116]% cs350_submit 0
This is /u/cs350/bin/cs350_submit for CS350 assignment 0.
/u/cs350/bin/cs350_submit: submitting .... to CS350 course account
############ OUTPUT FROM submit ###################
  |
  output from the regular submit command here
  |
############ END OF OUTPUT FROM submit ###################
/u/cs350/bin/cs350_submit: submission is complete!
    This submission *replaces* any previous submissions that
    you may have made for assignment 0.
```

Look carefully at the output from `cs350_submit`. If it looks like this:

```
/u/cs350/bin/cs350_submit IS ABORTING!  No files are being submitted to the course account.
```

then there was some kind of problem with your submission. Hopefully, the output from `cs350_submit` will give you a pretty good idea of what went wrong. You will need to correct the problems and try `cs350_submit` again. It is a good idea to run the `cs350_submit` command like this:

```
/u/cs350/bin/cs350_submit 0 | tee submitlog.txt
```

This will run the `cs350_submit` command and also save a copy of all of the output into a file called `submitlog.txt`, which you can inspect if there are problems. This is handy when there is more than a screen full of output.

You may submit multiple times. Each submission completely replaces any previous submissions that you may have made for this assignment.