CS350 **Operating Systems** **Winter 2022**

A0 - Assignment Specification

1 Introduction

The objective 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.
Additional information about the Sys/161 simulator can be found at `https://student.cs.uwaterloo.ca/~cs350/common/sys161manual/`

Each CS350 assignment will be composed of a prelab and a progamming assignment component. The prelabs will require you to review background and related material, which will faciliate in completing the programming assignment. The programming assignment will be divided into a kernel side programming assignment based on OS/161 and userspace programming assignment in Linux. In this way, you will learn how to implement kernel functionality and how to implment userspace processes that leverage the Linux kernel functionalities.

Note, you will implement functionality in the OS/161 kernel and not in the Linux kernel. Implementing kernel-side functionality is non-trivial, therefore you will be working with OS/161 that is a smaller kernel built for educational purposes.

However, your userspace programming assginments will be based on the Linux kernel and the functionality offered by the Linux environment. There are two main benefits of this. First, if you have incorrect or incomplete implementation of OS/161 kernel side functions, you still have the opportunity to successfully complete the userspace programming assignment. Second, you have the opportunity to work closely with a widely used modern operating system.

The assignment specification will describe the requirements of each component and the instructions for submitting each component. In this assignment, prelab-A0 componenet will help you setup an environment in which you can build and run your copy of OS/161 and linux userspace processes. There are **no** submission requirements for prelab-A0. In the OS/161 kernel side of the the programming assignment, you are expected to make two minor changes to the OS/161 kernel. In the userspace programming assignment, you are expected to write a simple c program.

2 Prelab-A0 Requirements

In this prelab, you will briefly learn about revision control and Docker containers. You will use these tools to setup an environment for CS350 programming assginments. Familiarity with these tools will help you prepare for the course and beyond.

The OS/161 kernel alone consists of more than 30,000 lines of C code. It is non-trivial to build and manage OS/161. You will find that revision control will help you manage your OS/161 code and enable tracking changes and reverting code efficiently.

In this course, we will use Git for revision control. The course web page (see "Assignment Information") includes links to various resoruces on Git. You are recommended to follow the Git Tutorial to learn to use command line with git and to push and pull from local and remote repositories.

In particular, you should be comfortable with the material in chapters 8-10 of the Git Tutorial. Git is already available on the linux student environment. and as University of Waterloo students you have access to enterprise edition of gitlab at `https://gitlab.uwaterloo.ca/users/sign_in`.

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. However, students often find that working in the student environment is cumbersome and inconvenient. We have created a Docker container that provides Sys/161 and the toolchain required to build, run, debug and test your OS/161 and linux userspace programs.

To successfully complete prelab-A0:

Install a copy of OS/161 in your `linux.student.cs` account. Using the step-by-step installation instructions at `https://www.student.cs.uwaterloo.ca/~cs350/common/Install161.html`

Build and run OS/161 in your `linux.student.cs` account. Once you have installed OS/161, can use the guide at `https://www.student.cs.uwaterloo.ca/~cs350/common/WorkingWith161.html` to learn how to modify, build, run, and debug OS/161 code. Read and understand this *before* you start working on the programming assignment.

Place OS/161 source code under revision control using Git Use Git to place the fresh copy of OS/161 in the linux student environment under revision control. That is, initialize the `os161-1.99` directory with git.

Create remote GitLab repository for your OS/161 source code. You will host your CS350 programming assignments on a private and remote repository on GitLab. You will use this repository to move updates from local environment to the `linux.student` environment.

Create a clone of the OS/161 repository in your local environment Make sure you have git on your local machine. Use `git clone` to clone the OS/161 repository from the linux student environment or the remote GitLab repository to your local machine.

Install `cs350-container` on your local machine. We have a Docker container for CS350, which includes Sys/161 and the toolchain required to build, run, or debug OS/161 code. The `cs350-container` is available as a public repository on UW GitLab. You should clone the `cs350-container` repository and follow the instructions in the README.md file to install and run code through the `cs350-container`. You will find that the `cs350-container` gives you a similar enivornment to the linux student environment with the convenience of working on your local machine. You are encouraged, not required, to learn about Docker.

To submit CS350 programming assignments. Once you have completed your pogramming assignment and tested it locally using the testing and evaluation scripts in the `cs350-container`, you are ready to submit your programming assignment. To submit your OS/161 and userspace programming assignments, you can follow the following steps, to efficiently push updates to `linux.student` environment.

1. Push local updates to OS/161 source code from local environemnt to the remote repository on GitLab.
2. Pull updates to OS/161 from remote repository in GitLab to `linux.student` environement.
3. Use the `cs350_submit` script in the `linux.student` environment to submit your programming assignments. The `cs350_submit` command is described in detail in Section 6.

2.1 Exemption from Prelab A0

If the local machine you have is not stable or if the local machine you have has Windows Operating System that is older than Windows 10.Then you are exempt from completing the Prelab A0. In this case, you are expected to work primarily in the `linux.student` environment. It is possible to work in the local environment and use your GitLab account for hosting a remote repository with your OS/161 code. You can use the remote repository to push or pull updates between local and `linux.student` environments.

Preliminaries

In the case of an exemption, 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 using the 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`. However, there is limited support from CS350 staff for local machines.
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.
3. Place OS/161 source code under revision control using Git. Use Git to place the fresh copy of OS/161 in the linux student environemnt under revision control.

3 A0: OS/161 Kernel Side Programming Requirements

For A0 kernel side programming assignment, you are required to make two minor changes to the OS/161 kernel.

3.1 Customize the OS/161 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 was 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.

3.2 Add a Kernel Menu Command in OS/161

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 are required to add a new command to OS/161 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`.

4 A0: `userspace` Programming Requirements

For A0 `userspace` programming assignment, you are required write two simple programs in C programming language for linux environment.

4.1 Generate random numbers

Write a program, in a file called `make_numbers.c`, which generates `n` random numbers within the range `lo` and `hi`, inclusive. The program parameters `n`, `lo`, and `hi` are to be passed to the program at runtime. You are required to output the `n` randomly generated numbers in a file, called `log.txt`. The format of the file is illustrated below:

```
4
34
54
789
-987
```

The first number in the file, should be the number of randomly generated numbers, that is `n`. After that, every randomly generated number should be on a line by itself. There are no requirements on the size of `n`, `lo`, `hi`.

4.2 Sort numbers

Write a program, in a file called `sort.c`, which opens a file `log.txt`. The first number in the file, should be the number of lines in the file, after the first line. It represents a list of randomly generated numbers.

The format of the `log.txt` file is

```
4
34
54
789
-987
```

You are required to use an efficient sorting algorithm to sort the numbers in ~~non-increasing~~ non-decreasing order and write them to a file called, `sorted.txt`. For the numbers in the example `log.txt` files shown above. The `sorted.txt` file will look like this:

```
4
-987
34
54
789
```

The first line in `sorted.txt` should be the number of sorted numbers, in the file, followed by the numbers in non-increasing order, one on each line.

Your program should output the time taken to sort the numbers. You should use the UNIX system call `gettimeofday` for time measurements. Design the code so that the overhead for time measurements are negligible.

You are required to create userspace programs that are robust and verbose. A robust program will efficiently handle errors. You will need to use the `gcc` compiler to compile your program.

A0-userspace: format of input and hints

The following information and hints can be used to format the input:

```
n : Must be a positive int, must be less than INT\_MAX
hi : Must be an integer, less than or equal to INT\_MAX and
      greater than or equal to INT_MIN
low : Must be an integer, less than or equal INT\_MAX and
       greater than or equal to INT_MIN

hi >= lo.

If hi == lo then we should expect the random number generator to only
produce "n" numbers of hi/lo.

The time requirement should be an integer in microseconds.

On an input error the program should exit with a value of 1.
Otherwise the program exits with a value of 0.

You can check exit codes after executing a command by typing: `echo $?`

Hints:

All numbers in the two sections of A0-userspace are to be considered to be int.

This program is trickier than it seems because the bounds make this difficult.
Since,  rand() has a maximum of RAND_MAX, while we are asking for INT_MIN
and INT_MAX

There are two ways of going about this:

Multiple rand() calls and composing an integer through bit operations.
The use of "/dev/urandom" to generate random bits for you.
As a general hint, once you have a random number you can mathematically
bound it to your hi and lo variables using:

random = lo + modulo(random, hi - lo + 1)

Where modulo function is that MATHEMATICAL MODULO.
A good reminder is that the modulo operator in C/C++ is not the mathematical one.

For example:

-9 mod 10 = 1

But in C

(int)(-9) % 10 = -9.

Userspace refers to Linux userspace, not OS161 userspace.
Your programming assignment should be in the directory structure as specified by the assignment.

Robustness and verbosity are typical programming practices:
```

```
As defined by Wikipedia, robustness is the ability of a computer system to cope
with errors during execution and cope with erroneous input.

Verbose: provide details of what went wrong.
```

5 Testing Your Programming Assignments

The expected flow of control when working on an assignment in CS350 is as follows:

1. You are working on your local machine and using `cs350-container` to build, run, test and debug OS/161 and linux programs.
2. You push updates to remote repository `my-cs350-repo`.
3. You pull updates from remote repository to `linux.student` environment.
4. You finally submit your programming assignment from the `linux.student` environment.

You can test your code and verify that is works correctly before you submit it for grading in the `cs350-container`, we have provided the `/assignments` folder that will hold all the public testing and evaluation scripts used for each assignment. You can use these scripts to run and verify that your code works as expected before you submit it. Instructions about the script are found in the README.md file in the `cs350-container` repository.

We are running auto-grading scripts using public and private tests, so once you submit your code, you will receive feedback and scores for the different components of the assignment. You can submit multiple times, however, each submission completely replaces any previous submissions that you have made for the same assignment.

6 Submitting Your Work

To submit your work, you must use the `cs350_submit` program in the `linux.student.cs` computing environment.

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

Note the usage for `cs350_submit` command is as follows

```
% usage: cs350_submit <assign_dir> <assign_num_type>
```

The `assign_dir` is the path to the root folder of the programming assignment. For the A0-kernel side programming assignment, the `assign_dir` is the path to your `os161-1.99` folder. For the `userspace` programming assignment, the `assign_dir` is the root directory for the `userspace` programming assignment. For A0, the `userspace` programming assignment root directory should contain both `make_number.c` and `sort.c` and the output files `log.txt` and `sorted.txt`

The `assign_num_type` for the kernel side is `ASST0`.

The `assign_num_type` for the `userspace` is `ASSTUSER0`.

Therefore, to run the `cs350_submit` command for submitting the A0-`userspace` programming assignment described in Section 4, the command will look like this:

```
% cs350_submit cs350-student/a0 ASSTUSER0
```

The argument `assign_dir` in the `cs350_submit` command, packages up your OS/161 kernel code or `userspace` program, respectively, and 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, for your kernel A0-kernel side programming assignment, you should see output that looks something like this:

```
@cpu20.student[116]% cs350_submit cs350-student/cs350-os161/os161-1.99/ ASST0
Please wait as we grade your assignment
Section Name                Marks Recieved   Out Of          Comments
Added Name                  0.0              2               Name not changed
Added Kernel Menu           0.0              3               dth command not found or failed
A cumulative log can be found here -  ASST0.log
We will always take the highest grade from all submissions as the final grade
Note: All submissions are stored and persisted and checked for plagerism after the due date
```

Look carefully at the output from `cs350_submit`. It is a good idea to run the `cs350_submit` command like this:

```
cs350_submit cs350-student/a0 ASST0 | 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.