# CMPS 111, Winter 2014
# Projects Overview

Here, you'll find information that applies across all of the projects. Project-specific information can be found in each project's home page. There will be four projects assigned this quarter, each one due about 2 weeks after it is assigned. Due dates are listed on the individual project descriptions on the Resources page.

- Project 1 : Writing a simple shell
- Project 2 : Scheduling
- Project 3 : Memory allocation
- Project 4 : File system

## Design & documentation

The assignments in this class do not typically require you to write large quantities of code. For most assignments, you need only write several hundred lines of code, if not fewer. However, the concepts covered in class and in the operating system are quite difficult for most people. As a result, deciding ***which*** lines of code to write is *very* difficult. This means that a good design is crucial to getting your code to work, and well-written documentation is necessary to help you and your group to understand what your code is doing. A sample design document is available on the Resources page.

The most important thing to do is ***write your design first and do it early***! Each assignment is about two weeks long; your design should be complete by the end of the first week. Doing the design first has many advantages:

- You understand the problem and the solution before writing code.
- You can discover issues with your design before wasting time writing code that you'll never use.
- You can get help from the course staff with the concepts without getting bogged down in complex code.
- You'll save debugging time by knowing exactly what you need to do.

## Doing your design early is the single most important factor for success in completing your programming projects!

## Using the MINIX 3.0 kernel

This quarter, we will be using the MINIX 3.0 kernel for our programming assignments. This is a complete (and working) operating system, and is well documented in the textbook *Operating Systems Design and Implementation (3rd edition)*. There's also lots of documentation at the MINIX web site.

The MINIX operating system comes with a complete set of tools, including C compiler and linker as well as debugger. We recommend you run MINIX as a virtual machine, using the hypervisor of your choice. Below are instructions for running MINIX using VirtualBox on any machines in the E105 lab or the UNIX timeshare (unix.ic.ucsc.edu). **The first lab sections will focus on getting MINIX up and running, so make sure to attend at least one!**

## Installing MINIX on an E105 machine or unix.ic

```
$ fs lq
```
***(must have at least 300MB remaining)***

```
$ wget http://users.soe.ucsc.edu/~danl/MINIX_cmps111w14.ova
$ VBoxManage import MINIX_cmps111w14.ova
```

Start the virtual machine and begin using MINIX via the console. Login and password are "root".
You may also use MINIX via ssh:

```
$ ssh -p2222 root@localhost
```

**Important! Before powering off virtual machine, make sure to shut down MINIX. To do so, use the '`poweroff`' command inside the console or your ssh session. Only after you have done this power off the virtual machine**

**Tip: To make ssh-related interaction easier, add the following to your `~/.ssh/config`:**

```
Host minix
    Hostname localhost
    Port 2222
    User root
```

## Moving stuff between host OS and MINIX

- One way to exchange files is to use `scp`. Details to follow in lab.
- A more convenient way may be to use `sshfs`, but this is not available in the lab or on the unix timeshare.

**Tip: Make a recursive copy of `/usr/src` on MINIX to have as a reference.**

## Debugging & testing

Although `gdb` is not available in MINIX, you can use the `mdb` debugger. Details to follow in lab.

## Handing in the project

Each project must be handed in using the `submit` system on `unix.ic`. For each project, hand in documentation, code, and any test files you created (scripts or data). More details will be given in lab.

### Design documentation

Your design documentation should describe the design of your project. This includes the data structures you're going to use, as well as the high-level pseudo-code for each high-level function. You **must** document functions that are "visible" to code you didn't write. You may also have "internal" functions; you don't *have* to document them. However, we suggest that you document major functions that aren't visible—it makes design simpler—but you don't have to include every single function you're going to implement in your design document. A sample design document is available on the Resources page.

Documentation must be plain ASCII text. Your final design document will be turned in as part of your full project, and will be the basis for the design grade for your project. Your design document should be called `DESIGN.` An example of one such document and the corresponding program is available on the Resources page.

### Project code

Please turn in *all* of the files you **modified** to build your project: source code, `Makefile`s, and any other scripts you may have used. In addition, please include a `README` file to explain anything unusual to the TA —testing procedures, etc. Your code should unpack properly in a "clean" MINIX distribution so it can be built. You can select individual files to include in your `tar` file by executing commands like this at the "root" of your source tree:

```
tar cf proj1.tar DESIGN README fs/file1.c mem/file2.c
```

This will include `file1.c` from the `fs` directory and `file2.c` from the `mem` directory along with `DESIGN` and `README` in the "main" directory. When the tar file is unpacked, the files will go back to their original locations.

### Turning in files

Please turn in exactly one file for each project: a ***compressed*** tar file containing your design document and all of your source code, test files, `Makefile`s, and anything else you need to run the code.

You can turn in the files by running this Unix command in your project directory:

```
submit cmps111-elm.f07 projn  projn.tar.gz
```

Note that you can compress a file by using a command like `gzip projn.tar`.

## Grading the projects

The intent of the grading for the project is *not* to differentiate among those students who do a careful design and implementation of the assignments. Rather, the grading helps us identify those students who (i) don't do the assignments or (ii) don't think carefully about the design, and therefore end up with a messy and over-complicated solution. Remember that you can't pass this course without at least making a serious attempt at each of the assignments. Further, the grading is skewed so that you will get substantial credit for a clean, logical, easy-to-understand design, even if your implementation doesn't completely work. This means that you should first strive to come up with a clean design of your project on paper. Also, don't try to add fancy features just because someone else is!

The grade for each project will be 40% for design and 60% implementation. We've structured the grading in this way to encourage you to think through your solution before you start coding. If all you do is to work out a detailed design for what you would do to address the assignment (and if the design would work!), but you write no code, you'll still get almost half of the credit for the assignment. The implementation portion of the grade considers whether you implemented your design, ran reasonable test cases, and provided documentation that the TA could understand. Part of being a good computer scientist is coming up with simple designs and easy-to-understand code; a solution that works isn't necessarily the best that you can do. Thus, part of the design and implementation grade will be based on whether your solution is elegant, simple, and easy to understand.

## Additional Resources

- The MINIX 3 web site
    - Command reference (man pages)
    - Programming in the MINIX 3 environment