CPS 536: Project Overview **Dr. Clark, Winter 1997**

The projects for this class will be based on the NACHOS instructional operating system. This system uses a simulated machine running as a Unix process. You will develop several enhancements to the operating system provided and then test your implementations on this simulated system.

The ground rules are as follows. You are to turn in *your own work.* That means, I want an original project turned in from every student. However, I encourage you to not work in isolation. You should discuss the goals and even the possible solutions to these problems with your classmates. You should work through C++ examples and discuss them with your peers. I also strongly suggest that you meet regularly with me to discuss your solutions before you try implementing them.

The first step in doing the assignments is to make a copy of the NACHOS files for yourself. Do this by logging into the Sun systems and typing the following commands from a shell window in your home directory. (Assume here that your prompt is $.)

```
$mkdir nachos
$cd nachos
$/home/rjc/pub/cps536/install.nachos
```

Your nachos directory will now contain source code for Nachos for each assignment, in directories *threads, userprog, vm, filesys, network*, plus code for the machine emulation, in the directory *machine*. For the assignments, you will need to make changes to files in *threads, userprog, vm, filesys, network* but there should be no need to modify the files in the *machine* directory.

Important! When you are doing the assignments, help me find the changes you have made. You must surround every piece of code you modify with:

```
#ifdef CHANGED
   put your changes here
#endif
```

In addition to your code, I also expect you to turn in a *README* file containing a written outline of what you did for the assignment, explaining and motivating your design choices. Of course, you should also document your code if you want me to understand what you did (a requirement for getting a good grade!). Some assignments have parts that ask you to explain something about your design; this explanation should also go in the README file.

I expect you to run test cases to demonstrate that your code works (or that it doesn't!); the results of these test cases should be put into a *TESTCASE* file (along with an explanation of what each test case is supposed to show). For the first assignment, I will give you a few test cases and you should create some more. In general, it is up to you to convince me that your code works. You should pick the minimal set of test cases that do this.

To hand in an assignment, you will use an online submit program. Make sure you follow the instructions for submitting each assignment.

Now, about the grading. My expectation is that with effort, most of you will get close to full credit on each assignment. The purpose of the project grading is to help me 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. You cannot 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, even if your implementation doesn't completely work, provided your design is logical and easy to understand. (In other words, don't pull an all-nighter just to fix the last remaining bug in your solution.)

The grading for the project will be as follows: 50% design, 50% implementation and testing. I have structured the grading in this way to encourage you to think through your solution before you start coding. If all you do is 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 will still get half credit. The implementation portion of the grade considers whether you implemented your design, ran reasonable test cases, and provided documentation that the reader could understand. Remember, a solution that works isn't necessarily the best 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.