Nachos Project Mechanics

Tom Anderson
Computer Science 162

The project is to be done in teams of two or three. The assignments will be the same for either size team. Groups of three will have less coding to do per person but will also have to manage the interactions between more people. We will *not* allow any groups of one this semester.

When you form a team, please send mail to c162-ra@po to register yourselves with the readers (Ryan Lo and Steve Huntsberry). Each reader will grade the same teams for the first three assignments; they will then swap and grade the other half of the teams for the last two assignments. Team members will all receive the same grade for each assignment.

We will use flexible slip dates for the assignments. Each team is given an automatic extension of 4 calendar days, which you can use on any assignment during the semester, in any increment, as long as the total amount of lateness did not add up to more than 4 days. For instance, you can hand in one assignment 4 days late, or each of four assignments 1 day late. This should let you schedule due dates around the due dates for other courses. When you hand in an assignment, you must identify at the top of the assignment, (i) how late this assignment is, and (ii) how much of the total slip time you have left.

After you have used up your slip time, any assignment handed in late will be marked off 10% per day. There will be no extensions granted.

The first step in doing the assignments is to make a copy of the directory on po, ~c162/nachos. This directory contains source code for Nachos for each assignment, in directories 'threads/userprog/vm/filesys/network', plus code for the machine emulation, in directory 'machine'. For the assignments, you will need to make changes to files in 'threads/userprog/vm/filesys/network', but you should not need to modify the files in the 'machine' directory. Also, so we can tell what 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, we 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. Keep this short and to the point! Of course, you should also document your code if you want the reader 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.

We 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). We are not going to tell you what test cases to run – it is up to you to convince us that your code works. You should pick the minimal set of test cases that do this.

To hand in an assignment, you simply send mail to c162-ra@po saying that you have completed the assignment and where to find the directory with the .cc, .h, README, and TESTCASE files for that assignment. From that point on, you must **not** modify any of the files in that directory. The best way to do this is to run the UNIX command, "chmod -w *.cc *.h". This will prevent vi and emacs from being able to write those files.

Because later assignments build on earlier ones, if you later find that you need to modify a frozen file, for instance, because of a bug, you should make a copy of the directory, and change the Makefile to use the files from that copy. You will also need to change the protections back with "chmod +w file.cc".

To help us with the grading for assignments 2-5, we will conduct 10-15 minute "design reviews" or interviews with each group, where you will be asked to explain and defend the design you chose for implementing each assignment. We will conduct the reviews the week before the assignment is due. The reviews will help determine your grade for the assignment; they are required.

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 – in other words, don't implement some complex feature just because someone else in the class is implementing it. My expectation is that most of you will get close to full credit on each assignment. Rather, the 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. We have 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 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. Part of being a good engineer is coming up with simple designs and easy to understand code; a solution which 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.