Nachos Project Mechanics

Tim Brecht
Computer Science 4321/5421

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. Groups of one will *not* be allowed.

When you form a team, please send mail to brecht@cs.yorku.ca to register your team. Also be sure to let me know which account you'll be using because I'll need to have the disk quota increased for that account. 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 (where increments are in days), 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 and exams for other courses.

When you hand in an assignment, you must identify the following on the cover page of the assignment:

1 The course name and number.

2 The team members names and userids.

3 The date.

4 How late this assignment is.

5 How much of the total slip time you have left.

6 Where to find the source code for the assignment.

After you have used up your slip time, any assignment handed in late will be marked off 25% per day. There will be no extensions granted. DUE DATES AND TIMES ARE ENFORCED. REALLY. 12:30 means 12:30. 12:31 IS LATE.

The first step in doing the assignments is to make a copy of the directory on ariel, /cs/course/4321/code. 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 the directories '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
```

If you can please highlight this code in your assignments before you hand them in. You may wish to denote changes for different assignments differently. for example you may wish to used CHANGED1 for the first assignment, CHANGED2 for the second, etc.

In addition to your code, we also expect you to turn in a document (keep this in a file called README in the directory for the assignment) 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 document. If you like you can just use plain text - no fancy formatting is required (although certainly permitted) as long as it is easy to read and well written. The code should be printed using enscript. There is an example of how the LISTING.ps file was generated in the file code/Makefile under the target "listing:". Please print only the code relevant for the assignment.

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. A portion of your grade for each assignment will be determined by how well your test cases are designed. Hand in an "enscript -2rG" version of your TESTCASE file.

You can hand in assignments at the beginning of lectures. ONCE I START THE LECTURE THE ASSIGNMENT WILL BE CONSIDERED LATE!! To hand in a late assignment it is best to give it to me in person (so I know exactly when it was handed in). If for some reason you can't find me hand it in to the TA. Do not put assignments under my door.

The intent of the grading for the project is mainly 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. DO NOT implement some complex feature just because someone else in the class is implementing it. Complex features are not always good features. 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 roughly as follows: 40% design, 40% implementation and 20% testing. 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 get roughly 40% credit. The implementation portion of the grade considers whether you implemented your design, and provided documentation that the reader could understand. The testing portion considers the design of the test cases (the extent to which the tests show simply and elegantly that your implementation works or does not work). You will receive better marks if your testing shows that your implementation does not work than if your implementation does not work but your test cases to not show it. Test programs should demonstrate with very little output whether or not different aspects of the system work correctly.

NOTE: THE 2nd, 3rd and 4th ASSIGNMENTS ARE RELATIVELY OPEN ENDED AND CAN REQUIRE AN INCREDIBLE AMOUNT OF TIME TO COMPLETE. YOU ARE BETTER OFF IMPLEMENTING A SIMPLE SOLUTION (OR PART OF THE ASSIGNMENT) THAT WORKS IS DEBUGGED AND TESTED THAN A FANCY SOLUTION THAT DOES NOT WORK OR IS UNTESTED.

IT IS UP TO YOU TO MAKE DECISIONS ABOUT HOW TO TACKLE EACH PROBLEM. IT IS PROBABLY BEST TO BREAK EACH ASSIGNMENT UP INTO PHASES AND COMPLETE ONE PHASE AT A TIME.

For example, you can describe a well designed page replacement strategy in your documentation but begin by implementing a simple algorithm (e.g., random). Once you have the simple strategy working then implement the better algorithm. It is often difficult to debug both the algorithm and the mechanisms with which they are implemented. Build and debug the mechanisms using a simple algorithm first the replace it with the better algorithm.

Part of being a good software 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.

Unless otherwise stated, if an assignment has multiple parts, each part has equal weight in the grading.