CS350 : Operating Systems

General Assignment Information

1 Introduction

Assignments in CS350 are based on NachOS. NachOS is a workstation simulation, along with a simple operating system for the simulated workstation. The assignments require you to enhance the NachOS operating system. For each assignment, you will be given a set of general requirements describing the enhancements that must be made. You are to design, implement, test and document changes to NachOS that will satisfy the requirements. You will also be required to demonstrate that your system behaves as required by providing readable and comprehensive testing documentation.

2 Using NachOS

NachOS is available in the CSCF Unix environment. To use NachOS, you must first install it in your account. There is a shell script, called `install_nachos`, that will do this for you. Please read the NachOS installation instructions on the course web page and follow them carefully.

You may work on your implementation on machines outside of the CSCF environment. In particular, see the course web page for information about running NachOS on Linux. However, **to receive credit for your implementation work, your code must compile and execute correctly in the CSCF environment. It is your responsibility to ensure that it does so.**

The course web page also contains important information about using NachOS, about working in groups and sharing files in the CSCF environment, and about NachOS itself: how the machine works, how the operating system is organized, and what it is capable of doing. Please read it.

3 Project Groups

You may work on these assignments alone, or in groups of up to three students. You are responsible for completing the assignments whether you have partners or not. The requirements are the same in either case. If you choose to work with partners, they need not be in the same section as you.

If you want a partner and do not have one, you may wish to try posting a "partner wanted" message on the course newsgroup.

If you work in a group, you must apply to us for a Unix group identifier. Apart from being just an administrative requirement, this will also help members of your group to share files. See the course web page (under Working in Groups) for information about various ways of sharing files in a Unix environment.

Choosing your group and obtaining a Unix group name is Assignment 0. Follow the Assignment 0 instructions on the course web page.

4 What to Submit

For each assignment you are expected to submit the following items:

Design document (3 pages maximum)

Testing document (3 pages maximum)

It is not acceptable to "trade" testing document pages for design document pages, or vice versa. For example, it is not OK to submit a four page design document if your testing document is only two pages long. *Each* document has a three page limit.

Code

Your NachOS code and test programs. Do a `make distclean` in your NachOS build directory before

submitting your code. There is no point to including all of those `.o` files and executables in your submission, since we are going to rebuild your system anyways.

Your design and testing documents must be submitted in PDF format. There are a variety of ways to create PDF documents. Here are some options:

- If you prepare your document using LaTeX, you can create PDF directly by using `pdflatex` to compile your LaTeX source document.
- If you use OpenOffice to prepare your documents, you can export PDF versions of the document directly from the OpenOffice menus.
- You can create plain text documents using your favorite text editor. You can then use the `a2ps` command to convert your text files to Postscript. You can then use `ps2pdf` to convert the Postscript document to PDF.
- From many other applications, you can print your document to a file, which should produce a Postscript version of the document. You can then convert this to PDF using `ps2pdf`.

Please test your PDF documents to ensure that they can be read in the CSCF teaching environment. The program `acroread` (Adobe Acrobat Reader) can be used to read PDF documents.

All of these items (both documents and your code) are to be submitted electronically, using the `submit` program. More information about submitting the assignment can be found in Section 5

4.1 Design Document

There is a hard limit of three pages for this document. Use a readable font, at least 10 point. Longer documents submitted to us will simply be truncated after three pages.

Your design document should provide an overview of the changes you have made to NachOS to support the assignment requirements. Write your document for an audience that already understands operating systems in general, NachOS in particular, and the assignment requirements. Assume your readers will be asking *how?* and *why?*, and provide answers to these types of questions. Your document should explain how each of the assignment requirements were addressed in your system.

Your design document should discuss and justify design decisions that you made. Sometimes one is forced to make decisions to solve certain problems and other times one makes a design decision in anticipation of future extensions and demands on your code. Your document should identify the strengths, weaknesses and limitations of your design.

If your design does *not* address some of the requirements, those that are not supported should be noted explicitly. Finally, if your system implements features other than the required ones, your document should describe the extra features, and should explain how they were implemented.

Your design document should *not* include program code, class definitions, or lists of function or method prototypes. It should be self-contained. The markers should be able to determine whether your design addresses all of the assignment requirements without having your code in front of them. Your design document should *not* include a restatement of the assignment or any portion of the assignment.

You may find it helpful to have at least an outline of your design document prepared for your group before beginning heavy coding. This way your group members have a better idea of what their tasks are, and your group can coordinate its efforts more effectively. Also, if you write your design first in English, you may find it easier to implement it in C++, rather than the other way around.

NOTE: your group is responsible for ensuring that the design document matches the implementation and visa versa. Any description of features or designs that are implied to be implemented but are not actually implemented will be treated as a case of academic dishonesty It is acceptable (and encouraged) to explain the design for unimplemented features provided that it is clearly and explicitly stated which features were not implemented. .

4.2 Testing and the Testing Document

You are required to provide a set of user test programs to demonstrate the functionality of your NachOS system. Test programs are normally located in the `code/test` directory in the NachOS distribution. Your test programs should be submitted electronically along with the rest of your NachOS code.

The purpose of the tests is to demonstrate to the markers that your system satisfies the assignment requirements, and that it is robust. We do not expect 100% test coverage of your implementation but we do expect you to do a reasonable job of designing tests that will convince us that your implementation is correct. Your tests should cover the main features of your design. You should also include stress tests that demonstrate the stability of your system, e.g, by running concurrent processes that make a variety of system calls.

When you design and implement tests, remember their purpose, and remember your target audience - the markers. Clear, simple, and meaningful tests with succinct output are good. Verbose, overly long, or confusing tests are not good. These tests are the primary means by which you demonstrate your implementation to the markers. Doing a great job on implementing your system and a bad job designing tests is a mistake, since you may fail to get credit for your implementation efforts.

Here are a few guidelines for designing and implementing your tests:

- Be smart about your testing. If the test program is more than 3 or 4 pages long it is likely too long. If the output produced by program is more than a page or two, it is probably too verbose. Think about the number of different things that need to be tested and how much time it will take to look at the output (imagine that *you* are the marker).
- The source for the test programs should be clear, concise, and documented with comments about what the program does and about the expected return values for system calls.
- The test program should be bug free. Take some time when writing the test program to ensure that it is correct. Too often people waste a bunch of time trying to fix their system when in fact the system is fine but there is a bug in the test program (or they misunderstand how the test works or the return values that are expected).
- Output a simple message if the tests fail or succeed. Better yet only print output if unexpected results are obtained and when the program ends.
- Someone should be able to spend about 1 minute looking at the output of each test program to determine whether all of the tests in the program failed or succeeded.
- Use multiple test programs to test different aspects of your system.
- Test special cases. Ideally a production quality operating system should be completely bullet proof. No user program, even malicious ones, should be able to crash the system. In this course your system doesn't have to be that strong but there should not be GLARING holes.
- Check the return values of function and system calls. That is, check that the call has completed successfully. If it has not, then handle the error intelligently.

In addition to your test programs, you are expected to produce a testing document. **There is a hard limit of three pages for this document.** Longer documents will be truncated at three pages.

This document will be used as a guide by the markers when they are running your tests. It should include at least the following:

- A description of how to build and run your test programs. Keep it simple, e.g., arrange that your test programs can be built by running `make` in the `code/test` directory. If your tests are difficult or time consuming to initiate, provide scripts to drive the tests.
- A brief description of the purpose and methodology of each of your test programs. What does it test, and how does it test?
- If necessary, guidelines for interpreting the output of your tests. Keep in mind that the entire testing document must be very short. By making your test output clear and self-explanatory, you can minimize the amount of interpretive guidance that you need to provide in your testing document.

4.3 Code

You are required to submit a complete and self-contained copy of NachOS, modified as described in your design document to meet the assignment requirements. This should include both the NachOS code itself, as well as the user (test) programs you have written. We will build your submitted code, and then use your system to run your test programs.

5 Submitting Your Work

Your design and testing documents must be submitted in PDF format. Your design document must be in a PDF file named `design.pdf`. Your testing document must be in a PDF file named `testing.pdf`. Both of these files should be placed in the top level directory of the copy of NachOS that you plan to submit. The top level directory is the one that has `code` and `c++example` and `coff2noff`.

To submit your code, your design document, and your testing document, use the `submit` command. When you run this command, you should be in the top level directory of the copy of NachOS that you wish to submit. For example, to submit assignment one, you would use the command:

```
submit cs350 1 .
```

In this command, `cs350` is the course for which you are submitting, the `1` is the assignment number, and the "`.`" refers to the current directory. For more information on the use of the `submit` command, type `man submit` . Assuming that you have placed `design.pdf` and `testing.pdf` in the top level NachOS directory as required, this single call to the `submit` command will submit both your code and your documents.

Please do a `make distclean` in the NachOS build directory before submitting your code. This will remove binaries and similar compiler waste products (we will recompile your code on our own).

6 Marking

For each assignment your mark will be based on your design and your implementation. The cover sheet for each assignment summarizes the mark breakdown for that assignment.

The implementation portion of your mark will be determined by the execution of test programs - yours and ours. Your test programs are the primary means of evaluation. Therefore, it is very important for you to design and document tests that are meaningful and clear. We may design and run our own tests against your system, and some portion of the implementation marks will be determined by such tests.

There will be *no* implementation marks for code that does not run or that cannot be tested. This means that you should implement and test one part of the system at a time, rather than doing all the implementation first and leaving the testing until the end.

The design portion of your mark will be determined by your design document. You can receive design marks for designing a particular feature even if that feature has not been implemented. (Remember, however, that your design document should clearly identify design features that have not been implemented.)

Your design and testing documents are expected to be well-organized and clearly written. Your code, including your testing code, is expected to be well-structured, commented and readable.

6.1 Academic Dishonesty (a.k.a. cheating)

You are encouraged to discuss the course assignments with people outside of your group and to use the course newsgroup for such discussions. **Nevertheless, each group is expected to do its own detailed design, to prepare its own documentation and to do its own implementation and testing.** For example, it is okay to discuss why NachOS (as given to you) behaves in a certain way, or why you cannot get it to compile, or how to use its debug mode, or what a semaphore is, or the differences between two paging algorithms, or the problems that arise when a multi-threaded process is terminated. It is *not* okay to share the NachOS code that implements process termination or the design documentation that describes it. It is the responsibility of each group to ensure that its on-line code and documentation are protected from

general access. A good guideline is to leave pencils and paper (and their electronic equivalents) behind if you discuss the assignments with other groups.

Plagiarizing text or copying code from students who have taken this course during previous terms or from students at other universities is also academic dishonesty, and will be treated as such. Keep in mind that although the CS350 assignments are similar from term to term, there are often changes. Sometimes the changes are subtle. NachOS itself changes too. To discourage cheating, we archive student submissions from previous terms and use software to compare these to the current submissions from the current term.

Any incidents of cheating that we detect will be reported to the Associate Dean (Undergraduate Studies) of the student's faculty. The standard penalty for cheating is a grade of **-100%** on the assignment, if it is a first offense. For example, cheating on an assignment that is worth 10% of the final mark will lower your *maximum* final mark in the course from 100 to 80. Penalties for second offenses are generally much stiffer, e.g., suspension from the University.