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 must also prepare some simple demonstrations that show that your system behaves as required.

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.

Whether you work in a group or individually, you must apply to us for a Unix group identifier. **You must have a Unix group name to submit your work, even if you are working alone.** The group name will be needed by the markers to access your files when your assignments are complete. It can 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:

Cover sheet/marking guide

The cover sheet/marking guide for each assignment can be downloaded and printed from the course web page. Each assignment has its own cover sheet/marking guide.

Design document (3 pages maximum)

Your design document should include:

- A description of how you enhanced the Nachos operating system to satisfy the assignment requirements. For example, you might discuss significant new classes you added, explaining what they do, how they do it, and how they are used to implement the assignment requirements.
- A brief description of what works and what does not, clearly and explicitly indicating any parts of the design that have not been implemented.
- A discussion of strengths, weaknesses, and limitations of the design, and justification for the design choices you made.

It should **NOT** include:

- a restatement of the assignment or any portion of the assignment.
- sections of code
- code showing class definitions, or lists of function or method prototypes.

Your design document should be entirely self-contained. We may choose to review your code, but it should not be necessary for use to review your code or other documents to understand your design.

Testing document (3 pages maximum)

Your testing document should includes:

- brief overview of your testing strategy
- testing issues (what it is important to test)
- testing plan (how you arranged to address each of the testing issues)
- brief description of how to run the tests and a description of the output that is expected.

Your testing document, like your design document, should be self-contained.

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 cover sheet/marking guide, design document, and testing document must be submitted in hard copy form in the CS350 assignment boxes on the 3rd floor of MC. Your Nachos code and test programs must be submitted electronically. Instructions are included below. Do not submit hard copies of your code.

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.

An important aspect of the design document is justifying your design decisions. 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. We want to know how such things affected 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. If you wish, your document may refer to specific parts of the code for additional details. However, your document 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.

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 to explain the design for unimplemented features provided that it is clearly and explicitly stated which features were not implemented. .

4.2 Testing Document

You are required to provide a set of user test programs to demonstrate the functionality of your Nachos system. In addition, you must produce a testing document. This is a document that tells the markers what aspects of the system you have designed tests for.

As is the case for the design document, you may design tests and describe them in your testing document even if they are not implemented. However, your document must clearly identify which of the described tests, if any, are not implemented.

There is a hard limit of three pages for this document. Longer documents will be truncated at three pages.

Your testing document should clearly state which aspects of your system you are trying to test, and it should describe how each of these aspects is tested. To describe how some aspect of the design is tested, you will need to identify the test program (or programs) that accomplish the test, what that program actually does, how that program should be run to accomplish the test, and the output that should be expected. Ideally, the output itself will be self-explanatory, but if explanation is needed, it should be included.

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 testing plan should cover the main features of your design. You should also include some testing of error and boundary conditions.

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 some or all of your test programs. The system that we build from your submission will also be used during your demonstration.

Your code should be submitted on-line, using the `cs350_submit` shell script. This script is found in the cs350 `bin` directory. If you have set up your command path as described in the Nachos installation guidelines on the course web page, the cs350 `bin` directory should be in it. The script takes a single parameter, which is the unique numeric group identifier that was assigned to you when you registered your project group. For example, if you are in group 45 (`cs350_45`), you would submit your code using the command:

```
cs350_submit 45
```

When you run this command, you should be in the top level directory of the copy of Nachos that you wish to submit. The top level directory is the one that has `code` and `c++example` and `coff2noff` as subdirectories.

What `cs350_submit` does is first to check that you are a member of the group whose identifier you specified. This means that you *must* have a group identifier to submit your work, even if you are working alone. It then makes a copy of everything in the `code` subdirectory of your copy of Nachos - this will include all of the Nachos code, your test programs (in the `code/test` subdirectory), and the Makefiles (in the build subdirectories). All of this is packaged and compressed into a single file called `cs350asst.zip`. That file is

left in the directory in which you ran the `cs350_submit` program. The script then sends a message to the course account to indicate where this file is located. Note that `cs350_submit` need only be run once per group.

You may run `cs350_submit` more than once if you need to (although try to avoid it). Your 'official' submission will be the latest one we receive before the due date.

Do not remove the `cs350asst.zip` file until after your submission has been marked. The markers must retrieve the file in order to retrieve your code. Also, do not attempt to modify the `cs350asst.zip` file after you have submitted it. Part of the message sent to us by the `cs350_submit` script is a checksum for the file. If you try to change the file after it has been submitted, the markers will detect a checksum mismatch.

5 Demonstrations

You will be meeting with the TA assigned to mark your assignments to discuss and demonstrate your project. After the assignment due date, a TA will contact you and arrange a meeting time.

The purpose of these meetings is for you to demonstrate to the TA that your code works. You should demonstrate this by running some or all of by your tests and explaining the purpose and results of the tests to the TA. This will also be an opportunity for the TA to ask questions about your design and testing strategy. If there are known problems with your system, e.g., things that have not been implemented, they should be pointed out early in the demo.

Ideally, your demo should be interactive and flexible. Flexibility can be achieved by providing a command shell so that you or the TA can interactively launch various combinations or sequences of test programs and by parameterizing test programs so that they are relatively easily to customize during the demo. A flexible demo will allow you to address any concerns that the TA may have about your system, and will help to convince the TA that your system is robust.

6 Marking

For each assignment your mark will be based on your design, your testing, and your implementation. as outlined on the marking guide. The design portion of your mark will be determined by your design document. The testing portion of your mark will be determined by your testing document. This testing of the marks is awarded if you have understood what needs to be tested, and if you have done a good job designing and documenting effective test programs. The implementation portion of your mark will be based on how well your system (including the test programs) operates. If you have a good design on paper but the implementation was sloppy and/or has serious flaws you should receive a good grade for the design but a lower grade for the implementation. The implementation portion of your mark will be determined by your demonstration, and possibly by additional executions of your test programs by the markers. There will be *no* implementation marks for code that does not run or that cannot be demonstrated and 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.

Your design and testing documents are expected to be well-organized and clearly written. Your code is expected to be well-structured, commented and readable, in case we need to review it.

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.

The standard penalty for cheating is a grade of **-100%** on the assignment. Any such incidents will also be reported to the Associate Dean (Undergraduate Studies) of the student's faculty. This may lead to additional punishment.