CS 4/53201 # Project #3 / Homework #4 **Operating Systems**

Homework # 4 portion due in class at 1:10pm on Monday 7 December 1998

Project #3 portion due via email by 11:59pm on Friday 11 December 1998

Introduction — Understanding and Extending the Nachos File System

In this assignment, you are given a version of Nachos that supports a simple file system. Your job is to study that file system, compare portions of it to material discussed in class, and make some minor extensions to it. As this assignment counts as both Homework #4 and Project #3, it will comprise 15% of your final course grade.

Installing and Testing a Version of Nachos That Supports the File System

When you compiled Nachos in the **threads** directory for Projects 1 and 2, only part of Nachos was compiled. For this project, you will need to compile a larger, more complete version of Nachos. You'll have to copy over some more Nachos files, so before you do so, you might want to clean up your account a bit. First, "*cd*" to the directory containing your **Makefile** and **threads** directory, and type "*make clean*" to remove all your old **.o** files in the **threads** directory. While you're at it, you might want to go into the **threads** directory and remove any old editor backup files and core files as well.

For Project 1, you copied all of the files in the **threads** directory, and some of the files in the **machine** directory, from **~walker/pub/nachos–3.4–hp/code** (assuming you used aegis or intrepid). For this project, you must copy ***all*** of the files in the **bin**, **filesys**, **machine**, **test**, **threads**, and **userprog** directories to your account. When you finish, edit the **Makefile** to comment out the 2 lines that compile into the **threads** directory, and uncomment the 2 lines that compile into the **filesys** directory. Now type "*make*", and you'll get a new executable **nachos** file — but this time, it will be placed in the **filesys** directory, rather than the **threads** directory.

Using the Nachos File System

Until now, the versions of Nachos that you have been compiling have used a "stub" file system, which actually just called the UNIX file system to work with files. For this assignment, the procedure described above compiles a new version of Nachos that implements a file system on top of an emulated disk.

This new version of Nachos creates a UNIX file named "DISK", and uses that file to emulate a physical disk. The Nachos kernel can use this new file system, but user programs can not (adding system calls that allow user programs to use the file system is left as a programming exercise). For testing purposes, you can supply command line options to successive executions of Nachos, and watch what happens (the file "DISK" will stay in your **filesys** directory, and *will accumulate the effect of successive executions of Nachos*). Some of the more interesting options are the following:

- –f format the emulated Nachos disk
- –cp copy a file from UNIX to the Nachos disk
- –p print the contents of a file on the Nachos disk
- –r remove (delete) a file from the Nachos disk
- –l list the contents of the Nachos disk directory
- –D display (print) the contents of the entire Nachos file system

Note that it would probably be more realistic to support system calls that access the file system, so that user programs such as "shell", "cp", "cat", etc. would perform these functions, rather than repeated executions of Nachos.

After you compile this version of Nachos, you might want to play around with it to see how it works. For example, try something like the following:

```
nachos –f
nachos –l
nachos –cp ../test/halt.c halt.c
nachos –l
```

```
nachos –p halt.c
nachos -cp ../test/halt halt
nachos –l
nachos –r halt
nachos –l
nachos –D
```

Preparing for the Assignment

The first step in this project is to prepare for the assigned problems by reading some of the source code, as described below.

To begin, read through the following files, and notice how they change when they are compiled using the FILESYS and FILESYS_NEEDED switches. Notice the command line options defined in each file.

- **threads/main.cc** — most of the command line interpretation for Nachos.
- **threads/system.h**, **threads/system.cc** — Nachos startup/shutdown routines.

Second, read through the following files to see how the file system is used and implemented.

- **filesys/fstest.cc** — a simple test case for the file system.
- **filesys/filesys.h**, **filesys/filesys.cc** — the top-level interface to the file system.
- **filesys/directory.h**, **filesys/directory.cc** — implements the disk directory.
- **filesys/filehdr.h**, **filesys/filehdr.cc** — manages the file headers (Nachos' version of disk descriptors / inodes).
- **filesys/openfile.h**, **filesys/openfile.cc** — translates file reads and writes to disk block reads and writes.

Next, skim through the following files to get an idea of how Nachos emulates the underlying disk. Note that this emulation is not part of the operating system — it's a simulation of the underlying hardware. You do not have to read these files in detail.

- **machine/disk.h**, **machine/disk.cc** — emulates a physical disk by sending requests to read and write disk blocks to a single UNIX file (named "DISK"), and then generating an interrupt after some period of time. These details vary tremendously from disk device to disk device, so in practice you would still want to hide them behind an abstraction like this one.

The Nachos File System

The Nachos file system is very similar to a simplified version of UNIX. It supports commands to create and remove files, and to open and close files. Directories are stored on disk, and contain file names and pointers to file headers. File headers specify the length of the file, and the sectors (blocks) where it is stored on the disk. All of the file headers are stored on the disk, although some of them are also loaded into memory when necessary for reasons of efficiency (see the FetchFrom and WriteBack functions for directories and file headers).

Most of the functions provided by Nachos, in particular the FetchFrom and WriteBack functions, depend on the fact that the on-disk representation of the various classes is the same as the in-memory representation of that class. This means that if you change one of the classes, you must reformat the emulated Nachos disk, or else delete it and create a new one. If you don't do this, you will probably get mysterious and unpredictable errors.

For a good overview of the Nachos file system implementation, see "The File System" in Archna Kalra's "Salsa — An Operating Systems Tutorial", and Sections 5 and 6.5 in Thomas Narten's "A Road Map Through Nachos", both available on the OS class home page.

Hints on Debugging

First, read through the debugging hints in the previous assignments. In addition to the Nachos debugging options discussed there, the file system supports two new options: "*–d f*" prints out information about the file system, and "*–d d*" prints out information about the emulated disk.

Adding New Files to Nachos

If you need to add a new file to Nachos, it's very easy to do so. However, the Nachos **Makefile** structure is pretty complicated, so if may take you a while to figure out how to do this. For example, suppose you want to add the files **filesys/blat.h** and **filesys/blat.cc**. To do this, edit the file **Makefile.common**, and update the definitions of FILESYS_H, FILESYS_C, and FILESYS_O. That's all you have to do! Then, the next time you

run "*make*", the **Makefile** in the **filesys** directory will be updated automatically, and your new files will be compiled and lined into Nachos.

Identifying Your Changes

So that the TA and I can easily identify which code you have changed or added, surround all changes and additions in your code by comments in the following form:

```
// PROJECT 3 CHANGES START HERE

<your changed code goes here>

// PROJECT 3 CHANGES END HERE
```

Use your own judgment about how much code to surround in a single comment.

The Problems

1. Homework #4 — due in class at 1:10pm on Monday 7 December 1998.
   (This part counts as Homework #4 — 5% of your course grade,
   and also counts as 1/3 of your Project #3 grade — 2.5% of your course grade)

   Write 3 pages of typed text (single-spaced, either ASCII text or 10-point word-processor output) describing how the Nachos file system is implemented and how its file and directory data structures compare to those described in Lectures 30 and 31. You can include figures if you like, but they must fit within the 3-page limit. Do not include additional text; part of the goal here is to write as clear a description as possible in a limited amount of space—a very "real world" experience. Also, be certain this description is in entirely your own words, as plagiarism will not be tolerated.

2. (50 points) Project #3 — due via email by 11:59pm on Friday 11 December 1998.
   (This part counts as 2/3 of your Project #3 grade — 7.5% of your course grade)

   Problem to be assigned later…

Where to Get Help

Help is available from Prof. Walker and from the TA (Ms. Linlin Tong):

- For questions on what the assignment is asking, please contact Prof. Walker.
- For questions on Nachos, please contact either the TA or Prof. Walker.
- For help with your code or debugging, please contact the TA.

Our office hours are on the course syllabus, and may be extended if necessary as the project deadline approaches; see the course web page for any announcements of extended office hours.

Also, if there are corrections or amplifications to this project, or if someone asks a question and we feel the answer may be relevant to other people, that information will be posted on the course home page under the project assignment. Thus, you might want to check the course home page periodically until the project due date to avoid getting bogged down in some problem to which a solution has been announced.

Cooperation versus Cheating

See the class syllabus, and contact me if you have any questions. For this project, you are allowed to *study the Nachos source code* with your friends, but you are ***not*** allowed to work with anyone else to actually *solve the problems*, and you are certainly ***not*** allowed to *copy anyone else's solution.*

Submitting Your Project

When you finish, submit all files that you modified (including your **p3.problem2** file) to the TA for grading, in the same manner that you submitted your files for Project 2.

Important warning — once you submit your files, **DON'T TOUCH THEM AGAIN** — if your email didn't reach the TA, or something happens, the TA may need to ask you to resubmit your files. However, before she lets you do so, she will ask you to log on in her presence, and she will check the modification dates on your files to make sure that they haven't been modified after the due date (if they have been, you will be assessed the appropriate late penalties).