Project Overview

Nachos: Not Another Completely Heuristic Operating System

Tim Brecht
Computer Science 4321/5421
September 9, 1997

"I hear and I forget, I see and I remember, I do and I understand."
— Chinese proverb

The project for this course is to build an operating system from scratch. We believe that the best way for you to understand operating systems concepts is to build a real operating system and then to experiment with it. To help you get started, a very simple, but functional, operating system is provided called Nachos. Nachos was developed in the Computer Science Division of the University of California at Berkeley (see the paper by Christopher et. al.). Over the course of the semester, your job will be to improve the functionality and performance of Nachos; it is expected that you will eventually modify or replace much of what has already been written.

The project has five phases, corresponding to each of the major pieces of a modern operating system: thread management, multiprogramming, virtual memory, file systems, and networking. You will only do the first four phases in this course.

Some phases build on previous phases; for example, the file system uses thread management routines. As far as possible, the assignments are structured so that you will be able to finish the project even if all of the pieces are not working, but you will get more out of the project if you use your own software. However, all of the dependencies have not been eliminated. Each new assignment will use parts of previous assignments in much the same way that different parts of an operating system depend on each other. So you are likely to encounter difficulties when doing later assignments if you do a poor job on early assignments. Part of the charm of building operating systems is that you get to "use what you build" – if you do a good job in designing and implementing the early phases of the project, that will simplify your task in building later phases.

Part of the code provided is a software emulation of a network of MIPS-like workstations. For instance, the code emulates turning on and off interrupts, taking exceptions and page faults, as well as the actions of physical devices (e.g., a disk, a console, and a network controller).

It is important to realize that while Nachos is run on top of this emulation as a user program on UNIX, the code you write and most of the code provided are exactly the same as if Nachos were running on bare hardware. Nachos is run as a user program for convenience: so that gdb (the GNU debugger) can be used and so that multiple students can run Nachos at the same time on the same machine. These same reasons apply in industry – it is usually a good idea to test out system code in a simulated environment before running it on potentially flaky hardware.

In order to port Nachos to real hardware, we would have to replace the emulation code with a little bit of machine-dependent code and some physical machines. For example, in assignment 1, routines are provided to enable and disable interrupts; on real hardware, these functions are provided by the CPU. In assignment 4, the behavior of a physical disk is emulated; disk read and write requests instead go to a UNIX file and an interrupt is generated after some period of time. The details of how to make disk read and write requests varies tremendously from disk device to disk device; in practice, you would want to hide these details behind something like the Disk abstraction provided.

Unless you work for a really smart company, when you develop operating system software you usually cannot change the hardware to make your life easier. Thus, you are not permitted to change any of the emulation code, although you are permitted to change any of the Nachos code that runs on top of the emulation.

Nachos is coded in a subset of C++; a separate document covers the parts of C++ that you will need to know.

Finally, a student who used Nachos for an operating systems course at Berkeley had the following suggestions for doing well in the course, and since the same will likely be true for this course, they are included here:

Read the code that is handed out with the assignment first, until you pretty much understand it. *Start* with the ".h" files.

Don't code until you understand what you are doing. Design, design, design first. Only then split up what each of you will code.

Talk to as many other people as possible. CS is learned by talking to others, not by reading, or so it seems to me now.