
Due via email by 11:59pm on Friday 9 October 1998

Introduction

The goal of this project is for you to familiarize yourself with Nachos — both the operating system and the underlying emulated machine. This project consists mostly of reading and studying the Nachos code, and will comprise 5% of your final course grade. Later projects will be more programming-oriented, and will comprise a higher percentage of your grade.

Getting Started

To begin, review the material presented in Lecture 09 on Wednesday 23 September. If you haven't read the Nachos "Overview Paper" that's online (or Appendix A in the 4th edition of the text) yet, you should do so now.

Second, read the file "Project 1 — Getting an Early Start" on the class web page. Decide whether you will use aegis for the project, or some other platform, and copy the necessary files to your account. Compile Nachos to produce an executable program "**nachos**" in the **threads** directory, and make sure it runs and produces the expected output.

Reading the Nachos Source Code

Now start reading the Nachos source code. I suggest that you read the files in the order described below, and as you do so, read the corresponding sections in Thomas Narten's "A Road Map Through Nachos" and Archna Kalra's "Salsa — An Operating Systems Tutorial".

To begin, read through the following files. When you compiled Nachos into the **threads** directory, the Makefile turned on the **THREAD** switch. Notice what code in these files is included when they are compiled using the **THREAD** switch, and what code is omitted when they are not compiled with the **USER_PROGRAM**, **FILESYS**, and **NETWORK** switch. Notice what command line arguments you can give to Nachos, and what global data structures are created.

- **threads/main.cc**, **threads/threadtest.cc** — a simple test of the thread routines.
- **threads/system.h**, **threads/system.cc** — Nachos startup/shutdown routines.

Then read through the following files, and see how Nachos implements and schedules threads. Study the thread class, its private data, and its public member functions. Study the Scheduler class, and how it dispatches threads. Glance at the code for context switching, but don't read it in detail

- **threads/thread.h**, **threads/thread.cc** — thread data structures and thread operations such as thread fork, thread sleep and thread finish.
- **threads/scheduler.h**, **threads/scheduler.cc** — manages the list of threads that are ready to run.
- **threads/switch.h**, **threads/switch.s** — assembly language magic for starting up threads and context switching between them. *Don't worry if you don't understand these two files — you are not responsible for understanding them.*

Next, skim through the following files, so you will recognize the functions when you encounter them elsewhere. After reading about **DEBUG** statements, go back through the files above, and see which debugging options may be useful when working with threads.

- **threads/list.h**, **threads/list.cc** — generic list management.
- **threads/utility.h**, **threads/utility.cc** — some useful definitions and debugging routines.

Now that you've gotten an overview of the Nachos operating system, it's time to look at the emulated machine underneath. You don't have to read this code in great detail, as you aren't going to be modifying it, but you should familiarize yourself with it, because you can't really understand the operating system unless you understand the hardware that it runs on. All of these files are in the **machine** directory. For now, you should read the files listed below, but you can ignore the files that describe the emulated console, disk, and network.

- **machine/machine.h, machine/machine.cc** — emulates the part of the machine that executes user programs: main memory, processor registers, etc.
- **machine/mipsim.cc** — emulates the integer instruction set of a MIPS R2/3000 CPU.
- **machine/interrupt.h, machine/interrupt.cc** — manage enabling and disabling interrupts as part of the machine emulation.
- **machine/timer.h, machine/timer.cc** — emulate a clock that periodically causes an interrupt to occur.
- **machine/stats.h** — collect interesting statistics.

Tracing Through and Debugging Nachos Source Code

One of the goals of this project is to read and understand the thread system in Nachos. Besides just reading the source code, you should also trace the execution path (as described below) for the simple test case provided.

To trace through code in Nachos, there are three main approaches: (1) using the **gdb** debugger, (2) using *printf*, and (3) using the *DEBUG* function provided by Nachos. The debugger **gdb** usually works, and is often the best alternative, although tracking across a call to *switch* can be confusing. Adding calls to *printf* often works, but sometimes fails since *printf* does not always flush the stdout buffer as expected.

The final debugging option, which is particularly useful when working with threads, is to use the Nachos *DEBUG* function, which is declared in **threads/utility.h**. The command line options to Nachos are specified in **threads/main.cc** and **threads/system.cc**; if you look at those files you will see that the command line option for debugging is “-d”, which should be followed by a flag to tell Nachos which type of debugging messages to print (these flags are defined in **threads/utility.h**). To look at the various debugging statements that are included in the thread system in Nachos, execute the command “*grep DEBUG *h *cc*” in the **threads** directory — as you can see, all of the those debugging statements have the “t” flag. In the **machine** directory, the debugging statements have “i” and “m” flags. Putting all this together, you might want to run Nachos as “*nachos -d t*”, “*nachos -d i*”, or “*nachos -d ti*” to see what your code is doing while working with threads. If you need more information, add more debugging statements (add your own debugging flag), or use the Nachos *ASSERT* function.

Overview of the Problems

The problems given below are intended to test your knowledge of the Nachos source code as you read through it in the order described above. They do not ask about everything in the code, but if you read a piece of code and then can answer the corresponding questions easily, you should be well prepared with a basic overview of Nachos for future projects. However, if you go through the code as quickly as possible, merely searching for the answer to these questions instead of trying to understand the code, you may encounter difficulties later.

Write your answers to these questions in a text file named **p1.answers**. When you finish, you will email the file **p1.answers** to the TA as explained later.

The Problems

- (40 points) These questions are concerned with the Nachos operating system.
 - When you use “g++” to compile nachos, one of the command line arguments is “-DTHREADS”. What does this argument do?
 - If you type “*nachos -d t*” to enable debugging of threads, where in the code is the “-d” processed? Where is the “t” processed?
 - Where and how is the main thread created?
 - The function *Thread::Fork* forks a new thread, and calls the specified function, passing it a single argument. What do you do if you want to pass multiple arguments to that function?
 - In function *Thread::Fork*, interrupts are disabled before calling *Schedule::ReadyToRun*. The comment says this function assumes interrupts are disabled. Why would this be a reasonable assumption?
 - If a thread calls *Thread::Yield*, what happens to that thread?
 - What functions add a thread to the ready list and remove a thread from the ready list?
 - At the end of function *Schedule::Run*, what variable points to the currently-running thread?
- (30 points) These questions are concerned with the emulated machine that runs underneath the Nachos operating system.

- a. Of the 40 general-purpose registers in the CPU, which one is used as the Program Counter?
 - b. Where and how is the main memory for the machine created?
 - c. What function fetches, decodes, and executes instructions, and where is it defined?
 - d. For project 1, the emulated machine pieces referred to in the above questions don't actually exist. Explain.
 - e. What variables keep track of "time", and where are they defined?
 - f. When and how does the emulated machine check for interrupts? What does it do if it finds one?
3. (30 points) Run nachos without and with the "*-d t*" command line option, and explain the output from Nachos (i.e., why and how it is switching between threads). Now add the statement "*if (which == 0)*" before the statement "*currentThread->Yield();*" in the *SimpleThread* function, defined in **threads/threadtest.cc**. Explain this output as well, and why it is different from the output without this statement.

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 class web page, and may be extended if necessary as the project deadline approaches; see the class 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 class web page under the project assignment. Thus, you might want to check the class web 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 the file **p1.answers** to the TA for grading by typing the following commands in the **threads** directory (replace "Your Name Here" with your own name):

```
elm -s "Project 1 for Your Name" ltong@mcs.kent.edu <p1.answers
```

The TA will read her email on Saturday and Sunday morning, and send you an email telling you whether or not she received your submission correctly.

Important warning — once you submit your file, **DON'T TOUCH IT AGAIN** — if your email didn't reach the TA, or something happens, the TA may need to ask you to resubmit your file. 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 file 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).

The project is due at 11:59pm on Friday 9 October 1998. For a discussion of my late policy, see the class syllabus. However, you should probably plan on starting early, ending on time, and then spending the weekend resting or working on your other classes, instead of trying to perfect a late project.