CPS 536: Homework 0 — Dr. Clark, Winter 1997

Nachos Assignment #0: Getting Your Feet Wet

The goal of this assignment is to start learning your way around the Nachos system and introduce programming with threads. This is also a good time to ramp up on C/C++ skills and learn to use a debugger.

In this assignment, I give you part of a working thread system. Your job is to study the system and figure out how it works. For the next assignment you will enhance the thread system implementation and then use it to solve several synchronization problems.

The first thing you should do is play with the C++ examples. In the NACHOS code you installed is a directory called *c++example*. This directory contains three sample source programs as well as a short C++ tutorial written by Tom Anderson. (You can view this document by typing "ghostview c++.ps".) You should practice editing, building and using the debugger on these programs.

The next thing to do is start reading and figuring out the partial thread system that is already written for you. Look in the directory *code/threads* for the code that implements this system. This thread system implements thread fork, thread completion, along with semaphores for synchronization. Run the program *nachos* for a simple test of the sample code. Trace the execution path (by hand) for the simple test case provided.

When you trace the execution path, it is helpful to keep track of the state of each thread and which procedures are on each thread's execution stack.

The files for this assignment are:

main.cc, threadtest.cc — a simple test of our thread routines.

thread.h, thread.cc — thread data structures and thread operations such as thread fork, thread sleep and thread finish.

scheduler.h, scheduler.cc — manages the list of threads that are ready to run.

synch.h, synch.cc — synchronization routines: semaphores, locks, and condition variables.

list.h, list.cc — generic list management (LISP in C++).

synchlist.h, synchlist.cc — synchronized access to lists using locks and condition variables (useful as an example of the use of synchronization primitives).

system.h, system.cc — Nachos startup/shutdown routines.

utility.h, utility.cc — some useful definitions and debugging routines.

switch.h, switch.s — assembly language magic for starting up threads and context switching between them.

interrupt.h, interrupt.cc — manage enabling and disabling interrupts as part of the machine emulation.

timer.h, timer.cc — emulate a clock that periodically causes an interrupt to occur.

stats.h – collect interesting statistics.

Properly synchronized code should work no matter what order the scheduler chooses to run the threads on the ready list. In other words, we should be able to put a call to Thread::Yield (causing the scheduler to choose another thread to run) anywhere in your code where interrupts are enabled without changing the correctness of your code. You will be asked to write properly synchronized code as part of the later assignments, so understanding how to do this is crucial to being able to do the project.

To aid you in this, code linked in with Nachos will cause Thread::Yield to be called on your behalf in a repeatable but unpredictable way. Nachos code is repeatable in that if you call it repeatedly with the same arguments, it will do exactly the same thing each time. However, if you invoke *nachos -rs* #, with a different number each time, calls to Thread::Yield will be inserted at different places in the code. Try this several times with several different random numbers using the current test program. Make sure to run various test cases against your solutions to future problems.