Operating Systems

CS 5/43201 Spring 2006
Department of Computer Science
Kent State University

Project Nachos

Suggested Reading: To do this first assignment, you will need to understand Nachos. Read the article "A Roadmap Through Nachos" by Tom Narten (it is in the class web site) very carefully. Also read the following source code files in the thread/ directory.

- main.cc threadtest.cc
- thread.h thread.cc
- scheduler.h, scheduler.cc
- synch.h, synch.cc
- list.h, list.cc
- synclist.h, synchlist.cc
- system.h, system.cc
- utility.h, utility.cc
- switch.h, swith.cc

You many also want to see the following files which implements the MIPS Virtual Machine:

- intterupt.h, interrupt.cc
- time.h, timer.cc
- stats.h
- console.h, console.cc
- disk.h, disk.cc

Preparation: The first thing you will need to do is to copy, install and test run a copy of Nachos in your directory. I have kept a ready to use copy of Nachos in my account. The link "**Begin here**.." in "**Project Page"** contains step by step instructions to get it. Make sure you can run the simple test program in Nachos by typing at the end of following those instructions:

```
% nachos
```

There is a function called *ThreadTest()* in file *threadtest.cc*. This is a sample user program and your starting point. This function is called by Nachos automatically for you from *main.cc*. In this assignment you will need to write many versions of this function. Inspect the file and the function, study and understand the output.

Your next step is to trace the execution path of this program. There are three ways to trace. (i) you may insert *printf()* statements inside, recompile and run it. (ii) you may use *gdb* debugger or (iii) you can use the built-in debug of Nachos. To use Nachos' debug run it with "-d" flag:

```
%nachos –d
```

If you browse you will see that Nachos code has already many DEBUG() (i.e. conditional *printf())* statements inserted in various routines. The "–d" flag activates them. For a complete listing of all these pre-inserted *printf()* statements in the source files in threads directory you may type:

```
%grep DEBUG *h *cc
```

If you run Nachos with “-d” flag, you will see that all DEBUG statements have ‘t’ flag in them. The files in machine directory similarly have “i” and “m” flags. You may choose to restrict the trace only to the thread or the machine routines by further specifying flags in following manners:

```
%nachos –d t
%nachos –d  ti
```

This assignment is about concurrently running threads. Concurrent threads voluntarily relinquish CPU by calling the function *thread::Yield().* when *Yield()* is called the calling function is temporarily suspended by Nachos scheduler and a second thread from the ready list is activated. If the concurrent threads are written correctly then the exact point where a thread yields does not matter. Conversely, one way of ensuring the correctness of a concurrent threaded system is to insert Yields() at different places and see how their running behavior changes. To help in such testing Nachos has provided a handy command line tool. It can insert random yields inside user threads on user’s behalf. You can invoke this feature by using –rs flag. Type:

```
%nachos –rs 100
```

Here the number 100 is the seed. It can be 200, 300 or any integer. Every time you invoke Nachos with the same seed the yields will be inserted at the same set of points. Experiment running Nachos with different seeds and try to understand any change is the output.

Road Map Questions For Nachos Threads

Here I have compiled a set of questions which will lead you through the Nachos system. You must know the correct answers for them to make sure that you have understood the Nachos System. These are not assignments and you do not need to turn in the answers. Instead, in about a week from now, most likely one un-annouced quiz will verify if you have solved these problems or not.

0. How many registers Nachos virtual machine has?
0. What are the physical locations of the *Stack Pointer* and *Current Program Counter* registers?
0. What are the preconditions for executing *machine::Run()?*
0. What is the name of the variable that remembers the ticks in Nachos? Which header file defines it?
0. List the three event points at which Nachos' time advances.
0. In Nachos, can you place an interrupt anywhere in time?
0. Will *SetLevel()* tick the clock if a process wants to:
   . Disable interrupt?
   . Enable interrupt from disabled state?
   . Re-enable interrupt when the interrupt is already enabled?
0. Why *OneTick()* checks SystemMode variable?
0. Why *StackAllocate* places a sentinel value at the top of allocated stack?
0. It is not physically possible for a thread to kill itself. Explain the mechanism how a Nachos' thread terminates.
0. Why *ThreadRoot()* and *Switch()* functions are implemented in Machine Language? Why there are so many versions of these two routines in *switch.s*?
0. Trace the routines invoked from the point an user process calls the routine *Fork().*
   I. Describe the exact point at which the new process starts running.
   I. What is the purpose of *ThreadRoot*?
   I. When exactly a child starts execution?
0. Trace the routines invoked from the point an user process calls the routine *Yield().*
   . Describe the exact point at which context switch has taken place.
   . Which thread removes the carcass of a terminating thread?
0. What is "Noff"? Convert a "coff" file into a "Noff" file and execute it in Nachos.