1 Introduction

The purpose of this assignment is to get the Nachos code installed on your account, help you understand and approach the Nachos code base, and show you the basics of the gdb debugger. In subsequent assignments you will be modifying the Nachos code base, so it is important that you understand the code, up to certain extent. It is also CRUCIAL that you begin to use the debugger.

2 Installing Nachos

2.1. How to get a Unix lab account
By this time, you must have already got the mail from root@cs.virginia.edu Although the email says that the distribution should work on lava001 – lava010, currently, it is only working on lava002 and lava009, so do not waste your time on the other machines.

You should connect to the lava machines using ssh from the blue.unix cluster, or SecureCRT from a PC. The fully qualified names are lava002.cs.virginia.edu and lava009.cs.virginia.edu.

SecureCRT is a Windows terminal emulator which supports the ssh protocol, and can be obtained from ITC Software Central. It has already been installed on the PCs in the Olsson 001 lab.

2.2. Download Nachos:
Go to http://www.cs.virginia.edu/~son/cs414.f05/cs414-linux-nachos.tar.gz to download software package: cs414-linux-nachos.tar.gz

2.3. Copy Nachos to you home directory:
Use SecureFX to transfer files to your home directory on lava002 or lava009. For example: /home/userid/

2.4. Use following shell commands to compile nachos:
Unzip and un-tar the NACHOS distribution by typing:
1) gunzip cs414-linux-nachos.tar.gz
2) tar xvf cs414-linux-nachos.tar
This will create a directory called nachos-3.4 which contains the Nachos source tree as well as some documentation and a README file (which you should look over). Now go to the code directory inside the nachos-3.4 directory.
3) cd nachos-3.4/code
The makefile has been broken in a couple of minor ways. A part of your assignment is to fix the makefile to compile the Nachos kernel. After correcting the Makefile, compile nachos by typing:
4) gmake
This would take some time, make sure there are no errors during compilation. The result of executing gmake will be a nachos executable file in several of the subdirectories. Now go to the sub-directory threads:
4) cd threads
and execute nachos by typing
6) ./nachos
from within the threads directory.

2.5. Result:
It should produce the following output:
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 130, idle 0, system 130, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...

2.6. Halt program:
Go to the directory:
../nachos-3.4/code/userprog/
And type:
./nachos –x halt
You have to submit the output for this as part of the assignment.

3 Understanding the code and using the debugger

Nachos, like any other program, starts running in the main() procedure. In Nachos, main() can be found in the threads subdirectory. Take a look at threads/main.cc. The only function that is getting run for now is the ThreadTest function, which can be found in threadtest.cc. Look in threadtest.cc. It is the printf function in SimpleTest that produces the output when you run the nachos executable. ThreadTest is a simple example of a
concurrent program. There are two independent threads of control running “at the same
time” and accessing the same data.
You should already be able to compile Nachos and run the threads program. You should
also understand why nachos produces the output that it does - tracing through the code by
hand is a good way to accomplish this. 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. You will notice that when one thread calls SWITCH, another thread starts running,
and the first thing the new thread does is to return from SWITCH. This comment will
seem cryptic at this point, but you will understand threads once you understand why the
SWITCH that gets called is different from the SWITCH that returns. (Note: because gdb
does not understand threads, you will get bizarre results if you try to trace in gdb across a
call to SWITCH.)

Let’s work on understanding exactly what is happening as this program runs. In order to
do this, we will use the debugger to step through the program’s execution. In order to
understand what is going on, let’s run the nachos executable using our debugger, called
gdb. We will use this debugger to trace through the code for this simple test case. You
can run the debugger from the command line or from emacs. If you use the emacs text
editor, we recommend running it from emacs as you will be able to see the debugging
stepping through the text of your code. Here we will run the debugger from the command
line. General documentation for using gdb can be found here:
This site includes information on running gdb from emacs.
Start the debugger on the nachos executable from within the threads directory:
gdb ./nachos
You will see that you prompt changes to (gdb) to indicate that you are now running the
debugger. At this point, the debugger is ready to run the nachos executable, but it has not
yet started running it. If we were to run nachos right now, it would just run all the way
through just like before, and the debugger wouldn’t be very useful.
In order to utilize the power of the debugger, we need to get inside the code and then
pause so that we can see the program’s execution in slow motion. We can do this by
setting breakpoints.
The debugger will stop when it reaches a breakpoint and then you can step through the
code one instruction at a time and print out the value of variables in the program.
Set a breakpoint in SimpleThread:
(gdb) break SimpleThread
Then run the program:
(gdb) run
Now we can type “n” to get gdb to move on to the next command:
(gdb) n
Continue to step through until you see “*** thread 1 looped 3 times” printed. Print out
the stack for the current thread, and print the value of the variables “which” and “num”:
(gdb) where
(gdb) print num
(gdb) print which
This is just an example of how you can use gdb. We recommend that you use gdb for your assignments as trying to debug with printf statements or by reading through the code will take a very long time.

4 Note:

Note that the other test programs in the Nachos distribution will not run successfully until you have actually built a more fully featured kernel, to be done over the next few weeks.

5 Assignment:

A part of your assignment is to fix the makefile to compile the Nachos kernel and compile the “halt” test program. You also have to submit:

- **A brief description of the Thread Class.** One paragraph description of the services and interesting features of Nachos threads.
- **A brief description of the Semaphore Class.** One paragraph on how to use Nachos semaphores to synchronize threads.

6 What to Turn-In for This Homework

Print out the descriptions of the Thread and Semaphore classes and the transcript of the Nachos execution of “halt” program. Turn in the hard copy in cs414 class on October 4. Don’t forget to type your name at the top of the page.

7 Links

Here are some links that may help your progress:

1) The Nachos home page (http://www.cs.washington.edu/homes/tom/nachos/) has lots of useful info.
2) For an overview, see The Nachos Instructional Operating System. (http://www.people.virginia.edu/~vp9g/cs414/nachos.ps)