Tracing and Debugging Nachos Programs

There are at least three ways to trace execution: (1) add *printf* (or *fprintf*) statements to the code, (2) use the *gdb* debugger or another debugger of your choosing, and (3) insert calls to the *DEBUG* function that Nachos provides.

Many people debug with *printfs* because any idiot can do it, whereas even smart people need to spend a few hours learning to use a debugger. However, investing those few hours will save you many more hours of debugging time that could better be spent watching TV or doing just about anything else. *Printfs* can be useful, but be aware that they do not always work right, because data is not always printed synchronously with the call to *printf* . Rather, *printf* buffers ("saves up") printed characters in memory, and writes the output only when it has accumulated enough to justify the cost of invoking the operating system's *write* system call to put the output on your screen. If your program crashes while characters are still in the buffer, then you may never see those messages print. If you use *printf* , it is good practice to follow every *printf* with a call to *fflush* to avoid this problem.

One of most important concepts in this course is the idea of a *thread* . Your Nachos programs will execute as multiple independent threads of control, each with a separate execution stack. When you trace the execution path of your program, it is helpful to keep track of the state of each thread and which procedures are on each thread's stack. You will notice that when one thread calls *SWITCH* , another thread starts running (this is called a *context switch*), and the first thing the new thread does is to return from *SWITCH* . Because ***gdb*** and other debuggers are not aware of the Nachos thread library, tracing across a call to *SWITCH* might be confusing sometimes.

The *Debug* Primitive

If you want to debug with print statements, the nachos *DEBUG* function (declared in ***threads/utility.h***) is your best bet. In fact, the Nachos code is already peppered with calls to the *DEBUG* function. You can see some of them by doing an ***fgrep DEBUG *h *cc*** in the ***threads*** subdirectory. These are basically print statements that keep quiet unless you want to hear what they have to say. By default, these statements have no effect at runtime. To see what is happening, you need to invoke ***nachos*** with a special command-line argument that activates the *DEBUG* statements you want to see.

See ***threads/main.cc*** for a specification of the flags arguments for ***nachos*** . The relevant one for *DEBUG* is ***-d*** . The ***-d*** flag followed by a space and a list of single-character debug flags (e.g., ***nachos -d ti***), enabling the nachos *DEBUG* statements matching any of the specified flags. For example, the ***t*** debug flag activates *DEBUG* statements relating to thread events. See ***threads/utility.h*** for a description of the meanings of the current debug flags.

For a quick peek at what's going on, run ***nachos -d ti*** to activate the *DEBUG* statements for thread and "interrupt" events. If you want to know more, add some more *DEBUG* statements. You are encouraged to sprinkle your code liberally with *DEBUG* statements, and to add new debug flag values of your own.

Miscellaneous Debugging Tips

The *ASSERT* macro, also declared in ***threads/utility.h*** , is extremely useful in debugging, particularly for concurrent code. Use *ASSERT* to indicate that certain conditions should be true at runtime. If the condition is not true (i.e., the expression evaluates to 0), then your program will print a message and crash right there before things get messed up further. *ASSERT* early and often! *ASSERTs* help to document your code as well as exposing bugs early.

Warning : Each Nachos thread is assigned a small, fixed-size execution stack (4K bytes by default). This

may cause bizarre problems (such as segmentation faults at strange lines of code) if you declare large data structures (e.g., *int buf[1000]*) to be automatic variables (local variables or procedure arguments). You will probably not notice this during the semester, but if you do, you may change the size of the stack by modifying the *#define* in ***threads/thread.h*** .

Defining New Command-Line Flags for Nachos

In addition to defining new debug flags as described in The Debug Primitive, it is easy to add your own command-line flags to Nachos. This allows you to initialize the value of a global variable of your choosing from the command line, in order to control the program's behavior at runtime. Directions for doing this are available on the course web site.