EECS 211
Advanced System Software
Winter 2006

Assignment 5

Posted: February 28, 2006
Due: March 14, 2006

Topic: Multi-programming in Nachos

Instructions:

The goal of this fifth assignment is to extend the support for user programs to actual multi-programming in the Nachos kernel. This assignment follows the second task of “Nachos Assignment 2” described in the file **`doc/userprog.ps`** of the Nachos installation. The instructions below assume that you read **`doc/userprog.ps`** in parallel.

Task 1: Complete the necessary support for user programs

This assignment builds upon the previous assignment (we will work again in the **`userprog`** directory). Thus, it is essential that the tasks outlined in Assignment 4 are properly completed before you continue!

You have two options, choose one of the following:
(a) Leave the previous assignment as is
(b) Re-submit the previous assignment

Deliverable 1: (10 extra points, or re-evaluation of Assignment 4)

For option (a), submit a text file **`task1.txt`** stating that you choose option (a). This will earn you 10 extra points (and your score for Assignment 4 is final).

For option (b), submit all deliverables for Assignment 4 for re-grading. This is your chance to build the necessary code for this assignment, and make up for any lost points at the same time. Note, however, that you cannot earn any extra credit this way.

Task 2: Implement multi-programming with time-slicing

See item 2 in **`doc/userprog.ps`**, but please note that we will not do the part of the assignment that covers response time measuring and analysis (ignore the second paragraph of item 2).

There are a couple of tasks necessary to accomplish true multi-programming.

First, you will have to add support for the system calls SC_Exec and SC_Join so that new processes can be properly started and properly terminated. At the same time, you will need to adjust the SC_Exit system call such that it cleans up the current process and joins with its parent (who will be waiting in SC_Join). Note that the function **`StartProcess()`** in **`progtest.cc`** will be helpful in implementing the SC_Exec system call.

Second, you will need to modify/extend the memory allocation for new processes so that multiple processes can be loaded at the same time (into different pages!). Critical files for this task are **`addrspace.h`** and **`addrspace.cc`**, as well as **`bitmap.h`** and **`bitmap.cc`**. Note that you may also need to adjust your code for copying memory between user- and kernel-land due to the new memory layout.

Third, you will need to instantiate a timer interrupt to preempt running processes so that they Yield() to other processes running at the same time.

Deliverable 2: (30 points)

a) Text file **`task2.txt`** with a brief description of your implementation and a list of files that you have modified.
b) All modified source files, including **`exception.cc`**, **`addrspace.h`** and **`addrspace.cc`**.

Task 3: Validate your implementation using simple test programs

To test your multi-programming environment, we will re-use some of the previous user-programs, as well as create a few new user programs as additional test cases. We will also utilize the provided shell (source file **`test/shell.c`**) as environment to run the other programs in.

As first test case, start Nachos with the given shell as root process. Then, use the shell process to start the following user programs and run them in order:

(a) Program **`HelloWorld.c`**:
should print the string HelloWorld to the console and then cleanly exit
(b) Program **`Name.c`**:
should ask the user for her/his name and then print it backwards
(c) Program **`WriteToStdin.c`**:
tries to write a character '**`x`**' to the standard input stream
(d) Program **`halt.c`**:
provided program to halt the Nachos system

Note that after each user program terminates, the shell program should resume and ask for the next program to run (except for **`halt`** which should terminate the entire system). Note also that even the "bad" **`WriteToStdin`** program should not influence the proper further execution of the shell.

As a second test case, create the following new user-programs:

(a) Program **`count.c`**:
counts from 0 to 9 and prints each value line by line to the standard output stream

(b) Program **`alpha.c`**:
enumerates the alphabet from 'a' to 'z' and prints each letter line by line to the standard output stream

(c) Program **`par.c`**:
starts the two programs **`count`** and **`alpha`** in parallel, waits for the completion of both, and then terminates by halting the machine.

Now, run Nachos with the **`par`** program and see if the two programs **`count`** and **`alpha`** actually execute concurrently. Note that the concurrency will depend on your timer settings and it may be necessary to insert additional instructions (i.e. counting loops) into the programs to actually observe "interleaved" output.

Deliverable 3: (30 points)

For each of the user programs used above, submit its source file (e.g. **`par.c`**). Submit also two log files **`shell.log`** and **`par.log`** showing that the two scenarios listed above successfully run on your extended Nachos kernel.

Submission instructions:

To submit your homework, send an email with subject "EECS 211 HW 5" to the course instructor at doemer@uci.edu. Please include the deliverables listed above as attachments.

To ensure proper credit, be sure to send your email before the deadline:
March 14, 2006, 11:59pm.

--
Rainer Doemer (ET 444C, x4-9007, doemer@uci.edu)