SIENAcollege
Computer Science

Computer Science 330
Operating Systems
Siena College
Spring 2012

Lab 2: Programming Unix Processes and Threads in C

Due: 9:20 AM, Friday, February 3, 2012

This week, you will gain experience creating and managing processes and threads in our Unix environments.

You may work individually or with a partner on this lab assignment.

Running Class Examples

Run the class examples that used `fork` and POSIX threads in the following three environments to answer the questions that follow in a file `answers.txt`.

1. A Linux system in the lab.
2. The Linux system `appserver.sos.siena.edu`.
3. The FreeBSD system `winterstorm.teresco.org`.

Question 1: Show the output of `ps` on each system showing the both the parent and child processes of the `forking` program, along with the output of the program on each system. (2 points)

Question 2: Log into a Linux system in the lab other than the one where you are doing the work for this lab (in case this crashes the system), and run the `forkbomb` program. How many processes can you create before you either get an error message (or the system crashes)? We won't try this experiment now on the other systems, but I was able to create over 3000 processes on `winterstorm` before running into trouble and well over 23,000 (!) on `appserver`. (1 point)

Question 3: Draw a memory diagram showing all of the variables in existence in the `pthreadhello` example when both child threads are executing. (2 points)

A sample output of the `what_shared` program on the Mac as we saw in class is:

```
First, we create two threads to see better what context they share...
Set this_is_global=1000
Thread 1, pid 67582, addresses: &global: 10A8, &local: B5EFC
In Thread 1, incremented this_is_global=1001
Thread 2, pid 67582, addresses: &global: 10A8, &local: 280EFC
In Thread 2, incremented this_is_global=1002
After threads, this_is_global=1002

Now that the threads are done, let's call fork..
```

```
Before fork(), local_main=17, this_is_global=17
In parent, pid 67582: &global: 10A8, &local: 5FBFEF5C
In child, pid 67583: &global: 10A8, &local: 5FBFEF5C
Child set local_main=13, this_is_global=23
In parent, local_main=17, this_is_global=17
```

Question 4: Run the `what_shared` example in each of the environments and paste in your output. What important differences do you see in the output and what do these differences mean? (3 points)

On the Mac, we saw in class that the `proctree_threads` program could create a thread tree of height 10 without encountering any thread creation errors, but did encounter errors at height 11.

Question 5: What is the tallest tree you can create on each of the three environments before thread creation errors are reported? (1 point)

`fork()` Practice

Write the program for SG&G Programming Problem 3.14 on p. 134-135. Call your program `forkfib.c`. Include a `Makefile` that compiles this program into an executable called `forkfib`.

Notes:

- Do some error checking. If your program is run without a command-line argument, print a "Usage" line. If your program is given a negative length for the Fibonacci sequenace, print an appropriate error message.
- If you store the entire sequence in an array as you generate it, allocate the array of an appropriate size using `malloc(3)` Reminder to Java programmers: C has no garbage collection, so any memory you allocate with `malloc()` must be returned to the system with `free()`.
- If you store the values in the sequence with `int` values, you will notice they may overflow the storage capabilities of the data type. You can delay this somewhat by using `long` values. Even then, you will overflow the values with a relatively short sequence. In this case, also print an error message.

Adding POSIX Shared Memory

Read Section 3.5.1 of SG&G to learn about POSIX shared memory. Then write the program for Exercise 3.18 on p. 137-138 of SG&G. Call your program `forkfibshared.c` and include a `Makefile` that compiles this program into `forkfibshared`.

Once your program is working, add a call to `sleep(2)` (recall that "2" here refers to the manual section, not the argument to pass) to your program before you detach and free the shared memory segment. In a separate window, use the `ipcs` command to see that your shared memory segment is listed, and that it goes away when your program terminates. Save this output and include it in a file `ipcs.out` to be included in your submission.

Now Using POSIX Threads

Write the program for Exercise 4.17 on p. 168-169 of SG&G. Call your program `pthreadfib.c` and include a `Makefile` that compiles this program into `pthreadfib`.

Submission and Evaluation

This lab is graded out of 40 points.

To submit this lab, place all of the files that you are to turn in (and nothing else) into a directory, change to that directory, and create a "tar file" to submit using a command similar to:

```
tar cvf lab2.tar *.out *.txt *.c Makefile
```

This will create a file `lab2.tar` in your directory. Send this tar file as an attachment to *jteresco@siena.edu* by 9:20 AM, Friday, February 3, 2012.

Please include a meaningful subject line (something like "CS330 Lab 2 Submission") and use the exact filenames specified (for this lab and all semester) to make my job easier when gathering your submissions together for grading. You don't want to annoy your grader with misnamed or missing files just before he grades your assignment. Please do not include any additional files, such as emacs backup files, object files, or executable programs.

Grading Breakdown	
`answers.txt` responses and	9 points
`forkfib.c` correctness	6 points
`forkfib.c` efficiency, style, and elegance	2 points
`forkfib.c` documentation	2 points
`forkfibshared.c` correctness	6 points
`forkfibshared.c` efficiency, style, and elegance	2 points
`forkfibshared.c` documentation	2 points
`pthreadfib.c` correctness	6 points
`pthreadfib.c` efficiency, style, and elegance	2 points
`pthreadfib.c` documentation	2 points
`Makefile`(s) to build executables	1 point