EECS 111 - System Software

Spring 2010

Assignment 2

Posted: Monday, April 5, 2010

Due: Tuesday, April 20, 2010, 12pm (noon)

A. Discussion: Processes

The goal of these exercises is to review and clarify the understanding of essential aspects covered in recent lectures.

- Process creation: Exercises 3.3, 3.9, 3.10, and 3.13
- Context switch: Exercise 3.7

These topics will be discussed at the beginning of this week's discussion session.

B. Project: Parallel Processes, Inter-Process Communication

The goal of this project assignment is to speed-up the program developed in the previous assignment by using parallel processes that utilize the multiple CPUs available on the EECS department servers.

Step 1: Setup

- We will use the very same setup as in the previous project. Please refer to the instructions of Assignment 1 for details.
- For this project, we will extend the code written previously. To get started, create a copy of your previous source code and name it **`fibo2.c`**.

Step 2: Create two parallel child processes

To enable parallel execution, we will create two child processes which both perform a part of the required Fibonacci computation. When both child processes have completed, the parent process will combine and report the results.

Specifically, the n-th Fibonacci-number is the sum of Fibonacci(n-1) and Fibonacci(n-2). Thus, to share the work, we will let child process 1 compute Fibonacci(n-1), and child process 2 will compute Fibonacci(n-2). This way, the only dependence between the two tasks is the addition of the two results, which can be performed easily by the parent after both child processes have terminated.

In your program, create each child process by using the **`fork()`** system call and separate the control flow of the child and parent processes by examining the returned process identifier. After the child has done its job, it should cleanly terminate its process. The two processes for the children will be very similar. The parent process, on the other hand, will simply wait for the two children to complete their work by using the **`wait()`** system call, and can then add the two results.

The example code shown in Figure 3.10 in the text book (page 113) is very helpful for the proper process creation and termination. You may also want to consult the Solaris manual pages for the **`fork()`**and **`wait()`** system calls (both are in section 2 of the manual). Be sure to add proper error checking, reporting, and handling to your code.

Step 3: Use inter-process communication via shared-memory

In order for the child processes to report their results to the parent process, we will need to implement some inter-process communication. For this project, we will use shared memory for this purpose.

The example code shown in Figure 3.16 in the text book (page 125) is a nice example of shared-memory communication implemented via the POSIX API which we will use for this assignment. Again, please consult the respective Solaris manual pages for a detailed description of the shared memory functions available on our servers, and be sure to use proper error checking, reporting, and handling in your code.

Specifically for our parallel Fibonacci program, we need to communicate two integer values (one result from each child) to the parent process. To do this, allocate 8 bytes (i.e. **`sizeof(int[2]`**) of shared memory *before* the child processes are created. This way, the children will automatically have access to the same shared memory of the parent and can copy their results into the respective slot of this shared integer array. After the child processes have terminated, the parent process can retrieve the results from the shared memory, release the shared memory again, combine and print the final result, and finally terminate itself.

Step 4: Test and compare your parallel implementation

When developing parallel programs, careful planning and proper identification of parallel tasks are essential. In your program, print descriptive messages on the screen when each process starts and ends.

When running the program, your execution log should look similar to this:

```
malibu.eecs.uci.edu % fibo2 42
fibo2: computing fibonacci(42)...
fibo2: child1 computing fibonacci(41)...
fibo2: child2 computing fibonacci(40)...
```

```
fibo2: child2 computed fibonacci(40) = 102334155
fibo2: child1 computed fibonacci(41) = 165580141
fibo2: fibonacci(42) = 267914296
malibu.eecs.uci.edu %
```

To compare your parallel implementation with the previous assignment, use again the **`/usr/bin/time`** command to measure the user, system, and elapsed execution times, and compute the 40th, 41st, 42nd, and 43rd Fibonacci numbers with your new **`fibo2`** program.

List your measured times in a table (note the server name(s) you are using) and compare them against the times measured in the previous assignment. Briefly explain why the times come out this way. Do the times match your expectation? How do the *user* and elapsed (*real*) times compare? Why?

What is different from the naïve expectation that your program will be twice as fast as before? Why?

Deliverables:

1. Statement: "I have read the Section on Academic Honesty in the UCI Catalogue of Classes (available online at http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0) and submit this work accordingly."
2. Source file: **`fibo2.c`**, execution log **`fibo2.log`**, table with comparison of execution times, and brief interpretation of the results (3-4 paragraphs) [100 points].

Submission instructions:

To submit your homework, send the deliverables in an email with subject "EECS111 HW2" to the course instructor at doemer@uci.edu.

To ensure proper credit, be sure to send your email before the deadline: Tuesday, April 20, 2010, at 12:00pm (noon).

--

Rainer Doemer (EH 3217, x4-9007, doemer@uci.edu)