

COMP 530 INTRODUCTION TO OPERATING SYSTEMS

Fall 2009
Kevin Jeffay

Homework 6, October 28

Due: November 4

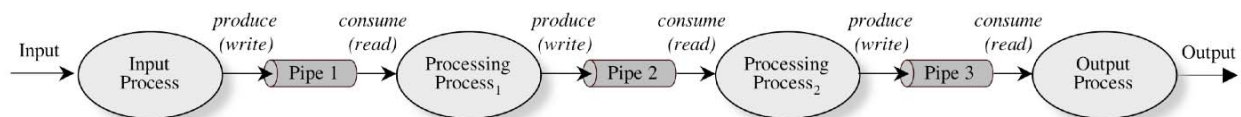
The “UNIX Warmup” Revisited Revisited: Producer/Consumer Interaction with Message Passing

Homework 4 was an exercise in using lightweight threads and shared memory within a Unix address space to re-implement a solution to the string processing problem from Homework 1. In this assignment you will use Unix processes instead of threads and message passing instead of shared memory to develop yet another solution to the Homework 1 string processing problem.

Specifically, in this exercise you will re-implement your multithreaded C program from Homework 4 to (1) use Unix processes instead of ST threads, and (2) use the Unix message passing facility known as “pipes” for inter-process communication rather than shared memory. Thus, in this assignment you will *not* be using the ST package. This will require you to eliminate your bounded buffers and replace them with a Unix pipe. From a coding standpoint, this solution should be the simplest one yet!

In more detail, you’ll keep the pipelined structure of Homework 4 but fork a process for the processing that was previously in each thread. In addition, each bounded buffer from your Homework 4 solution will logically be replaced with a Unix pipe. There should be one-to-one relationship between processes in your Homework 6 solution and threads in your Homework 4 solution as well as between pipes in your Homework 6 solution and bounded buffers in your Homework 4 solution.

The result should be the new processing pipeline illustrated below. Inside each of your new Unix processes, the deposit/remove operations on a bounded buffer will be replaced by read/write operation on a pipe. The read/write operations on a pipe are the Unix equivalent of the send/receive message passing primitives we studied in class.



The producer/consumer relationships between threads should be exactly as it was in Homework 4. Each pair of communicating producer/consumer threads will become a pair of communicating producer/consumer processes. The i^{th} process in the pipeline, P_i , reads data that is generated by process P_{i-1} , and writes data for use by process P_{i+1} . Thus, process P_i simultaneously acts as a consumer with respect to process P_{i-1} (with process P_{i-1} playing the role of process P_i 's producer), and process P_i acts as a producer with respect to process P_{i+1} (with process P_{i+1} playing the role of process P_i 's consumer).

All we are doing here is replacing shared memory based communication and synchronization mechanisms with mechanisms based on buffered message passing. If you implemented Homework 4 correctly, the application logic in your threads should require only very minimal (if any) changes to work inside a process with pipes. Note that you are to use pipes to directly synchronize your producer/consumer processes. You need not (and should not) implement the “buffermanager” structure we discussed in class.

Once again, from the standpoint of the user, your program should behave *exactly* as a correct solution to Homework 1 does. That is, the program will read from standard input and write 80 character lines to standard output changing carriage returns to spaces, and pairs of asterisks to caret characters. Your program will be tested the same way your homework 1 program was tested.

The purpose of this assignment is to gain experience using Unix inter-process communication mechanisms.

Grading

“Submit” your program electronically for grading by emailing jeffay@cs.unc.edu when the program is ready for grading. Programs whose files are dated after 2:00pm on November 4 will be considered late. You will submit one file for this homework assignment. Your main program should be in a file called *HW6.c*.

The program should be neatly formatted (*i.e.*, easy to read) and well documented. For this program, approximately 40% of your grade for a program will be for correctness and 60% for “programming style.”

Extra Credit

For extra credit, write a variation of your Homework 6 program wherein each process is now written as a separate C program that reads from stdin and writes to stdout. Solve the Homework 1 string processing problem by executing your separate programs from the shell and connecting them via pipes from the command line of the shell.

If you attempt the extra credit, name your programs *HW6x.c* where “x” is an integer that indicates the position of the program in the pipeline. And please note that, as always, you should absolutely not even think about the extra credit portion of the assignment until after you have a *perfectly* executing version of Homework 6.