2020/2021: Sistemas Operativos (*Operating Systems*) - 2º MIEIC

22.feb.2021

Exercise Sheet 2

Processes, threads and communication

1. Differentiate program, process and thread.
2. Consider the information the operating system keeps about the activities it manages.
   a. Present 5 relevant data items that differentiate a process from another.
   b. Do the same for threads.
   c. Present the data from a. and b. in a comparative table: one row for each item; one column for process and another for thread.
3. Use the `ps` command, perhaps with the options `a u x`, to observe the list of processes that are "registered" in the system at a given moment.
   a. Take a note of the different states that are visible and try to interpret their meaning with... "`man ps`"!
   b. With the same command (`ps`), eventually with new options, trace the family tree of the process that corresponds to `ps`, up to the "initial", root process.
   c. With the `pstree` utility, identify a similar tree (for `pstree` process). (Clue: there is an option that shows the PIDs.)
4. The Process state transition Diagram presented in the lectures' slides shows the 3 main states: *Running*, *Blocked* and *Ready*. Update the diagram to accomodate 2 more possible states: *New* and *Terminated*.
5. Write a program that does the following:
   - the initial process creates a child process;
   - after `fork()` the child process writes the word "`Hello`" to standard out;
   - after `fork()` the parent (initial) process writes "`World!`" to standard out, guaranteedly to appear on screen after "`Hello`".

   Could you give a similar guarantee, now having the parent write "`World:`" before the child writes "`Hello!`"?
6. Present an advantage and a disadvantage of having the architecture of a program based on processes compared with basing it on threads.
7. The use of signals in the communication of processes is often considered with some apprehension by application designers. Clarify the reason for this apprehension, presenting an illustrative, problematic situation.
8. Study the sample program "`ssignal`", that is in the "Code examples" web area of the course unit.
   a. Use it as basis for another program that satisfies the request in the last paragraph of the above problem 5 (the parent process writes "`World:`" after `fork()` and before the child writes "`Hello!`").
   b. Try to confirm the correction this new program, by making sure that, after `fork()`, the parent process starts executing after the child process (program variant no.1) and (variant no. 2) the parent starts before the child. For each of these tests, use a mere `sleep()`.
9. Study the example program "`sthread`", which shows, in a simplified way, the creation and termination of threads.
   a. Change the program in order to pass as argument to each new thread its

creation number (represented by the local variable `i`) and so that the threads end with a termination value of `i*i`, which the main thread will read.

b. Try to see the threads with the `ps` utility (perhaps called with the rigth option). If necessary, keep them running for a few seconds with `sleep()`.

10. Write a program that displays the sequence "`Operating Systems`" on the screen, where one thread displays "`Operating` " and another thread displays "`Systems`". The program should have a single function, executed by both threads, which takes as arguments the string to be written on the screen and its order of presentation.
11. Differentiate, by (at least) two aspects, the use of (*normal*, *anonymous*) *pipes* for interprocess communication relative to using *named pipes* (*FIFOs*).
12. One process is expected to send color information to another process, each color being one of the 7 of the rainbow; the other process should answer whether it already had it available in its private palette or not. Explain how this could be achieved (and under what circumstances) with each of the following inter-communication techniques:
    i. named pipes
    ii. (normal) pipes
    iii. signals
    iv. (cannot be done!)
13. Study the code fragments related to pipes presented in the lectures' slides and use them to write programs where `fork()` is used to create a new process, and:
    a. Program 1 uses *pipes*; the child receives "`Operating` " from the parent, concatenates it with "`Systems`" and writes the composed string to the standard output;
    b. Program 2 has almost the same specification as Program1: the exception being that it will use *named pipes* instead;
    c. Program 3 has the same specification as Program1 and, in addition, the parent also concatenates its string ("`Operating` ") with "`Systems`" received from the child, after which it writes the composed string to the standard output;
    d. Program 4 has almost the same specification as Program3, the exception being that it will use *named pipes* instead.
14. ... (additional exercises on Moodle for people with free time)