Advanced Unix System Administration
Spring 2007
Homework 2

This assignment is due via email to <sluo+decal@ocf.berkeley.edu> by 11:59 PM on **Friday, March 2**. (I may be willing to extend this to the following Monday, if this seems to be needed.) All the files mentioned are in `~sluo/hw2-files` on the login server (`plague.ocf.berkeley.edu`); the ones which can be used off the login server are also in a tarball `hw2-files.tar.gz` available from the website. If you do not already have access to the login server, you need to email me about setting up an account.

These exercises are intended to give you some experience using `strace` and friends.

1. *Becoming a daemon.* In principle, this is just the job of getting into the background and out of the user's way. In practice, getting completely unentangled from the foreground is more difficult than you might expect.

   a. You've probably noticed by now that forking a process puts it in the background. Examine, compile and run `daemon1.c`, which does just that. Follow its execution using `strace`. What does the program do? What happens when you try to log out? Why? (Hint: look at the files `daemon1` has open.)

   b. Compare `daemon2.c` to the previous version. What has changed? Now compile and run it. Follow its execution using `strace` (or `truss`, or `ktrace`, or whatever equivalent your OS has). What does this program do? Does it solve the problems of the previous version? Look at the list of files it has open. Do you think we've successfully solved the problem?

   c. Try killing the parent's process group (`kill -TERM -[pgid]`, where the process group ID will be equal to the process ID of the parent). What happens? Have we successfully solved the problem?

   d. Compare `daemon3.c` to the previous version. Now what has changed? Compile and run it; follow its execution using `strace`. What does this program do? Does it solve the problems of the previous version? Look at the list of files it has open. Do you think we've successfully solved the problem?

   e. Set up OpenSSH's `sshd` to run as yourself on a high port. Follow its execution using `strace`. How does `sshd` get itself into the background? Does this have any advantages over what `daemon3` does? (If you don't have OpenSSH on your system, another system daemon will do; stick to traditional/well-established/well-written ones, as this is something that many programs get wrong.)

   f. Look up what sessions and session leaders are. Why might you want to fork twice when starting a daemon? Hint: this has something to do with the process's controlling terminal. (This information is surprisingly difficult to find,

and there seems to be a good deal of confusion out there as to what this actually means. The authoritative source here is the Single Unix Specification, the successor to POSIX, available for free reading and/or download after registration from the Open Group. Unfortunately, like many standards documents, it's written extremely tightly, and needs to be read very carefully to be interpreted correctly. If you find a more accessible source for this information, let me know! The Unix Programming FAQ may also be helpful here.)

g. *Optional, but recommended if you have time.* Try (at least some of) this on other OSes. What do you find?

2. *Tracing a running process. This exercise must be done on the login server.* Among the files for this week's assignment is `wrapper`, which forks off a child process to perform some tasks.

   a. Run this program and observe its behavior. Now try tracing it with `strace`. Does the program do the same thing? If not, why not?

   b. Attach to the child process using `strace`. Describe in detail what the program is doing or trying to do, and the errors it is getting.

   c. What files does the process have open? Identify the file descriptors that were referred to in the `strace` output.

   d. What did `wrapper`'s child process do after the call to `fork()`? (This might take a bit of thought, but you do have enough information to answer this question, assuming you already did the first three parts of the problem.)

   e. *Optional; might take a bit of thought.* What do you think `wrapper` does before the call to `fork()`? (You should be able to answer this question after our unit on security. Until then, don't worry too much if you can't.)