Advanced Unix System Administration
Fall 2008
Homework 3

This assignment is due via email to <sluo+decal@ocf.berkeley.edu> by 11:59 PM on **Monday, October 27**.

1. *A login session.* This problem requires root access to a Unix system; you may use your homework 1 VMs for this.

   a. Trace an SSH login. What UID does the login process initially run as? Must this be the case, and why? Identify the parts of the output that show when the process changes user and group IDs, and when your login shell is invoked.

   b. Look up what a POSIX "session" is, and what a "session leader" is. Give two reasons why it's important that a new session be created for a user login. (Hint: one has to do with an important feature people expect from their shells; another has to do with what happens if, say, the SSH daemon dies or is killed.) Identify in the trace output where the session corresponding to your login shell is created. Identify which process ends up being the session leader for the login session once the login shell is invoked. (If you're having trouble finding these definitions, try looking in the Single Unix Specification, available from the Open Group. I encourage you to look for other resources first, though, since, like all standards documents, the writing is extremely dense.)

   c. Look up what a "controlling terminal" is. Why is it important that a user login has a controlling terminal? Identify in the trace output when the login process acquires your controlling terminal, and give the name of the device file which corresponds to it.

   d. Outline the steps needed for a running process (such as `sshd` or `login`) to create a normal login session for a user. What is the latest point at which you could effectively set up resource limits for a login session, and why?

2. *Use of `mmap()`.* Compile the two programs `wcl.c` and `wcl-mmap.c`. These two programs count the number of newlines in a file, reading the whole file into memory first (similar to what `wc -l` does, but in a dumber way). As the name suggests, `wcl-mmap` uses `mmap()`, while `wcl` uses `read()`. Run the programs on a variety of text files large and small (a copy of Project Gutenberg's edition of *War and Peace* is included in the tarball). It's suggested you do this problem in a controlled environment, such as your homework 1 VM.

a. Does one run faster than the other? If so, why? Make sure to trash the file cache in between runs (the memory hog examples from previous homeworks will come in handy); why does this make a difference?

b. *Do not do this on production machines.* Compile and run `malloc4.c`, which is like the previous `mallocX.c` memory examples, but continues to run until interrupted by a signal. With `malloc4` running (and hogging memory), try `wcl` and `wcl-mmap` again on reasonably large files (greater than the amount of free memory available on the system; `cat`ing *War and Peace* together a few times should do it). Which one works? Why?

c. *Optional; might take a bit of investigation.* Why would one *not* want to use `mmap()` for file access? (Why do the vast majority of programs use `read()`, `write()`, and friends?)

3. *Hard disk fragmentation and performance.*

a. Modern file systems designed for rotational media work very hard to prevent fragmentation, using similar strategies as memory allocators (allocating in widely spaced blocks, for example). Still, it's possible to induce fragmentation on such systems; how might you do this?

b. *Do not do this on production machines.* Try out your suggested disk fragmentation scheme (you will likely need a system to which you have root access), and run read benchmarks on large files that are and aren't fragmented. (Timing a `cat > /dev/null` works well here.) Make sure you trash the file cache in between runs – otherwise you won't see any consistent effects.

c. *Optional.* Try this out on solid-state media (a USB flash drive or SD card, for example). Do you see similar performance effects? Given this information, how would a scheme such as Windows Vista's ReadyBoost work?