Sarah Whelan
slw96
9/17/2015

EECS 338 Assignment 1

Question 1:
Find out the execution time, the username (cuserid()), and parent and child processes' real and effective user ids, and their group ids. Explain briefly the purposes of these ids.

The username from cuserid is “a username associated with the effective user ID of the process” (man page for cuserid). This is a human readable string that is a way for humans to more easily idenify a user from their id. Note that this is the username associated with the *effective* id so if for some reason this process had to run under a different id to obtain permissions then the username associated with the effective user is shown.

The real and effective user IDs are respectively who tried to run the process and who is effectively running the process.

“When a normal program is executed, the effective and real user ID of the process are set to the ID of the user executing the file. When a set ID program is executed the real user ID is set to the calling user and the effective user ID corresponds to the set ID bit on the file being executed.” (man page for getuid)

This indicates that under normal conditions these ids would be the same and a way o identify the user who called the process. However, there are some cases where the program may need the permissions of say the user that wrote the file and not the user running the file so the effective user ID is set to allow these case.

The real and effective group ids are the group ids of the real and effective user respectively. Groups can be used to provide levels of permissions to users at a more abstract level.

Question 2:
For both the parent process and the child processes, at the end of their execution, find and print the current time (ctime() and time() calls), (ctime provides the seconds elapsed since the Epoch, 1970-01-01 00:00:00 +0000 (UTC) (man page for ctime)) user CPU time, and system CPU time that they use. (Have your processes loop for a while in order to accumulate some nonzero time values. Otherwise, you may observe zero time values. Why?)

Here our program is simply doing a few calculations and printing results to standard output and as such doesn't take very long, in computational time, to run at all. In fact in many cases this program could execute so quickly that the internal time keeping records are not fine-grained enough to catch the difference. If the internal clock only measures in milli or mico seconds but a process takes nano seconds then the recorders won't see the difference. This is especially true for CPU time as it may take only a few CPU time slices but be waiting on other things (IO etc) and as such the user time is often longer than the CPU time. This looping is there to hopefully ensure the program takes up enough time for the record keeping mechanisms to notice and be able to report an actual non-zero time.