OS161 Project 2 Executive Summary

Noah Weiner

Collaborators I worked on this alone, although I found a lot of internet resources to do so (the Pearls in Life blog, and a couple OS161 GitHub repositories, although none of those were using the UW version with the process struct already implemented).

What I completed I implemented the fork, exit, getpid, and waitpid syscalls. I tested them with the onefork and widefork tests from UW, both of which ran successfully. The process struct I fleshed out with a PID, parent PID, exit lock, exit condition variable, exit flag, and exit code. The process table I implemented as a primitive array of process structs, with the PID acting as the index. For fork, I create the process using the out-of-the-box proc_create_runprogram function and then modified the resulting struct within sys_fork. When a process exits, its still running children are inherited by their grandparent, and exited children are destroyed. The process itself is only destroyed if its parent is the kernel process (essentially parentless).

What I didn't complete Unfortunately, testbin/forktest fails, but I think this is because it runs out of memory (or memory is never de-allocated). I would have liked to test out testbin/badcall/waitpid too (I doubt my code would have passed), but I never implemented argument passing.

What I learned I learned a lot more about how syscalls work in general. I was aware of waitpid, but didn't know that it had such a significant on exit logic - when a process is destroyed, how its exit code is handled, etc. I also learned a lot about development workflow. Using a lot of finer-grained git commits was very helpful, because it allowed me to revert my code back much more easily. I learned a hell of a lot about OS161's structure as well, and became much more familiar with process, thread, and address space functions as well as the structure of the MIPS trap handler.