Assignment 4

No additional base code is needed beyond what you already have for Assignment #3. I have outlined the functions in syscalls.cpp that you need to implement below.

Syscalls.cpp

1. sysClose():
   This function will close a file descriptor and free the userspace buffer associated with that file descriptor (and any pages that go with it). Freeing a file descriptor entails zeroing its entry in the array in the task struct as well as clearing the open file table entry for that file (assuming the next available file descriptor for that task is > 2).
2. sysFree():
   If a process is in state PROC_ZOMBIE or PROC_KILLED, this function will free all the frames associated with that process, zero out the task struct for that pid, and zero the page directory and page table for that pid.
3. sysKill():
   This function kills a process (as long as the pid to kill is > 1). This will mark the task state as PROC_KILLED, it will close all files associated with that pid, and will traverse all processes running and change the ppid for those from the killed pid to pid 1 (as someone needs to be the parent of processes whose parents have been killed).
4. sysSwitch():
   Arg1 here will be the process to switch to. CurrentPid is the currently running process. This function will make the currently running process go to sleep (change the state to PROC_SLEEPING), change the state of the arg1 pid to PROC_RUNNING, and make a context switch to the arg1 pid. Be sure to appropriately update the RUNNING_PID_LOC value as this address in memory is used to allow a running process to know its own pid. (Consider using readValueFromMemLoc() and storeValueAtMemLoc() for these actions.)

These functions can be tested for functionality between the shell and the editor program as demonstrated in the lecture video.

Be sure to include validation of the values coming into the functions (e.g., are you attempting to kill a pid that doesn't exist? Are you attempting to switch to a pid that is in a state other than PROC_SLEEPING, etc?).

Additionally, be sure to use the constants (from constants.h) whenever possible. You should not be using a hard-coded value if there is a corresponding constant in constants.h, (but do not

modify, add to, or alter constants.h in any way). You cannot create new helper functions and you should not create any new global variables. **Be sure to follow a professional coding standard, including the naming of your variables, structures, etc. You should use descriptive names that are appropriate. Nonsensical or whimsical names will lose points. You must also use the custom types where I have supplied them (e.g., uint8_t, uint16_t, uint32_t) unless you need a signed typed or one I haven't supplied. You will lose points if you don't follow this convention.**

When you are happy with everything and have tested the functionality in the shell and editor as I showed in the lecture, upload the SUBMIT-ME.zip file to Canvas.

Additionally, you must attest and add the following honor statement to your Canvas submission (in the Canvas submission comments):

I, <state your name>, attest that this submission is my own work. I have not worked with anyone else on my implementation and have not used AI or other unpermitted sources to help me with my solution, in accordance with our definition of class cheating shown in Lecture 1.

Any submission that does not include this attestation will not receive any credit.