Derek Trom

[Derek.trom@und.edu](mailto:Derek.trom@und.edu)

CSCI451 HW6

12 Oct 2020

Case 1 when the pthread\_exit(); was called resulted in the program getting stuck in my thread function as the & symbol was read from hw5-2.in. The reason this occurred was definitely in the way I designed it and would be expected for this type of unexpected exit in my design. This occurred where I called it because when the exit was called no unlock was able to be called on either the main thread mutex lock that I put in place. The state was also not able to transition which was another barrier I put in place for the program so that the threads would not be running over each other when trying to read the file. The unlock also did not occur for thread 3. This led to a deadlock in the program as the mutexes were never unlocked.

Case 2 when trying to cause the thread to automatically die. If the system call “kill -s 9 PID” was used to kill the thread this also signaled the main thread to exit. This seems to be a safety net of pthread and kills all processes. However I found a workaround to just kill the child thread before any unlocking of mutexes via a signal handler as well as using the pthread\_kill() command. If I used the pthread\_kill(PID, int SIG) command on the running thread I was able to signal a SIGSEGV to the signal() call in main. The signal() call caught the segmentation fault signal and was able to keep the main thread alive. This called my signal handler and I was able to “ignore” the child being killed thus leading to deadlock in the thread functions because the main thread was not forced to quit because of a segmentation fault