John Neis CSCI 451, Homework 6

Case 1) Program locked up. Process still active, but no longer actually doing anything. Case 2) Multiple methods were used in attempts to crash a thread without crashing the rest of the process. None of these attempts were successful.

Case one is easier to analyze and figure out what occurred here. Essentially, due to the way that I created my locking algorithm, exiting the thread on the occurrence of ‘&’ left the mutex in a locked state, which forced the other two threads into a state of starvation. Since both of the other threads are in a while loop that will only exit when *pthread\_mutex\_trylock()* returns with a lock on the mutex, both threads are essentially locked into a *while(!0)* loop that will never end.

Case two would suggest that pthreads are not, in fact, individually killable. The first attempt in case two was to use the “kill -s 9 TID” command from within the thread. This caused bash to return an error message, claiming the thread ID is an illegal number. The next attempt had the thread raise a segmentation fault signal by using “raise(SIGSEGV)” from signals.h. This, however, had the unintended effect of crashing the whole process. Since the whole point is to only crash thread 2 and neither of the others, further attempts were made. Any signal from signals.h which causes a termination of a thread also caused the entire process to simply terminate. This seemed to be a dead end toward the point of this exercise, so other functions were investigated, such as pthread\_kill() and pthread\_cancel(). pthread\_kill seemed to exhibit the same behavior as raising signals. Pthread\_cancel didn’t seem correct, as it appears pthread\_cancel is designed to safely exit a thread, which is not the desired effect. Further, community guidance on this topic is also unhelpful, as it appears all efforts have been focused on safe and proper use of threads.