Ben Fisher

4/18

HW 4 Thread Safety

If these two snippets can both access the shared variable, x, they can both read and write on that variable. Both threads share the same copy of x, so what happens to one x will happen to the other instead of the presumed ideal situation where both threads have their own individual copy of x that does not affect the other. The interleaving construction from this example can lead to varied outcomes. For example, the code could run fully sequentially so x ends as 14 after being written to 13, rewritten to 130 from the line x*10, once again rewritten to 7, then finally rewritten one last time when it gets added to itself ending at 14. However, this is likely not the case depending on how the threads stack their processes. X could evaluate to 130 if the segments run concurrently but in opposite order, or it could write x to either 13 or 7 before any math is done on the variable leading to a final value of 260 or 140. The processes could jumble up even more where the math sections read different variable values as the other executes its function, so x could evaluate to 70 or 26. The interleaving threads access different parts of the threads at different times, which results in a mess of non-sequential reads and writes that makes the final value of x unpredictable.