Classical algorithms for mutual exclusion

The program in the attachment consists of two threads. One repeatedly increments a shared variable s, the other one repeatedly decrements s. The value of the variable n specifies the number of repetitions in both the threads. The initial value of s is 0, hence the expected final value of s is also 0. The pthread library is used to start one of threads, but no means of mutual exclusion are used to prevent the threads from concurrent access to s. Note the use of symbols MUTEX_NONE, MUTEX_DEKKER and MUTEX_PETERSON in Makefile and concurrency.c (Makefile produces 3 executables which differ only in code fragments surrounded by corresponding #ifdef ... #endif).

a) In the designated places in the attached source code, implement Dekker’s and Peterson’s algorithms to prevent simultaneous access of the threads to s (do not use any additional library functions, do not change the rest of the source file). Count the number of the idle repetitions in the entry protocols (the number of executions of the busy waiting loops in Dekker’s and Peterson’s algorithms) in the variables c1 and c2 (c1 for one thread, c2 for the other thread). Make sure that there are 0 errors and 0 warnings reported by the compiler/linker.

b) Compare -DMUTEX_NONE, -DMUTEX_DEKKER, and -DMUTEX_PETERSON for $n=2^i$ for i=0, 1, …, 29, 30. Repeat the measurements 10 times for each i. Discuss the results. Also report the number of processors and the highest value of signed integer in your system (the latter can probably be found as the value of INT_MAX in /usr/include/limits.h).

c) Test all three programs thoroughly, look for an unexpected behaviour. Report what you found, explain it if you can.

Hand out a ZIP file hw5.zip containing source files and a report hw5.pdf.