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).

Dekker's and Peterson's algorithms to prevent simultaneous writes to the shared variable s. The variables c1 and c2 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.

a) 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. Run the measurements with -O0 and -O3 in CFLAGS in Makefile. Report the number of processors in your system, 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).

b) Watch for any unexpected behaviour of the programs. If there is some, give a plausible explanation. Propose a correction to the source code which does not use additional system calls. Test your proposal.

Hand out a ZIP file hw4.zip containing source files and a report hw4.pdf.