Homework – Chapter 6

Name: __Justin Weigle_____

1. Give the name and definitions for the three properties required for a solution to critical section

Mutual Exclusion - only one process in critical section at a time
Progress - no waiting forever to decide who gets to go into their critical section next
Bounded Waiting - when a process requests to go into its critical section, there is a limit on the number of other processes that are allowed to enter their critical sections before said process must be allowed to go into its critical section

2. Describe some strategies for finding race conditions in source code.

The easiest way is to surround the questionable source code in a mutex lock and then slowly, piece by piece, move the code out of the mutex lock until the race condition shows back up.

3. Give a scenario where there is an apparent race condition in code but it isn't something that requires a critical section to fix.

Race conditions are not fixed with critical sections, they are caused by critical sections. Which means any scenario is valid here.

4. Given that some race conditions aren't important, why is it unlikely that automated tools will ever be able to identify them?

Using threads printing out of the order expected as an example: how should an automated tool identify such a thing? Should it read the text printed and decipher whether it was in the correct order? How does it know which order the output should be printed in?

There are too many factors in race conditions such as the example above. An automated tool would have to be quite extensive in order to fit everyone's needs for identifying race conditions.

5. Why don't we use peterson's solution for mutual exclusion any more?

Peterson's solution is not really compatible with modern computer architectures because, in order to improve performance, processors and/or compilers may reorder read and write operations that have no dependencies. This means that, like the example in the book demonstrates, a flag could be set out of expected order in another thread that causes multiple threads to enter their critical sections at the same time.

6. Explain the role of hardware in supporting mutual exclusion.

Computer architectures provide instructions that can force changes to memory to be propagated to all other processors. These are known as memory barriers/fences.

Another such thing provided is hardware instructions meant to operate atomically, such as test_and_set() and compare_and_swap(). On Intel x86 architectures, a lock is used to lock the bus while the destination operand of a compare_and_swap() instruction is being updated.

7. Prove that if semaphore's wait() and signal() operations aren't done atomically then mutual exclusion may be violated.

Given 2 processes P1 and P2, the following occurs while S = 1:

P1 → executes wait() and finds S = 1
P2 → executes wait() and finds S = 1
P1 → decrements S and enters critical section
P2 → decrements S and enters critical section

Both P1 and P2 were able to enter their critical sections at the same time, and S is left at 0 rather than the intended value of -1

8. Busy-wait a.k.a. "spinlock" was explained as a "tight loop" that waits for the resource to become available. Given that semaphores can efficiently pause a process while its waiting, why are these still used?

Spinlocks are still used because of multi-core environments. In a multicore environment, it would be necessary to disable interrupts on every core. Otherwise, instructions from different processes on different processors could be interleaved with the "locked" instructions.