

Spring 2020

- Home
- Announcements
- Assignments
- Grades
- Syllabus
- Quizzes
- My Media
- Media Gallery
- Zoom

Quiz 3

Due May 28 at 11:59pm Points 13 Questions 9
Available May 28 at 8:30am - May 28 at 11:59pm about 15 hours
Time Limit 90 Minutes Allowed Attempts 2

Instructions

CS 444/544 Operating Systems II

Quiz III

Quiz Description

You have 90 minutes to answer the questions in this quiz. In order to receive the credit, you must answer the question by choosing the answer from the listing. We have an open-material policy on this quiz, so you may refer to slides, textbooks, your JOS source code, your note, or other online materials. However, we strictly prohibit chatting with other students or asking for help online (posting a question, etc.).

In case if you cannot understand any questions in the quiz, please find Yeongjin or any TAs for clarification, but please DO NOT ASK for checking if your answer is correct or not.

Lastly, only two attempts are allowed for this quiz.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	30 minutes	10 out of 13

Correct answers are hidden.

Score for this attempt: 10 out of 13
Submitted May 28 at 7:24pm
This attempt took 30 minutes.

Question 11 / 1 pts

1.[Trap handling in JOS Lab 3] In JOS Lab 3, we have a user-level program named as `soffint` that runs the instruction `int $14` in its execution. Although the code in the program attempted to generate the interrupt number #14 (page fault) via software interrupt, the CPU will generate a general protection fault (interrupt number #13).

In this regard, why does the CPU generate a general protection fault, not a page fault upon running `int $14`? Choose the best answer that describes this phenomenon.

☐

The software interrupt instruction, `int $X`, always generates an interrupt number `X-1`, so it generates the general protection fault (interrupt number #13) when we run `int $14`

☐

Whenever setup the interrupt descriptor table (IDT), we set the privilege of having a page fault as ring 0. So running of `int $14` from ring 3 is prohibited, thus generate a general protection fault.

☐

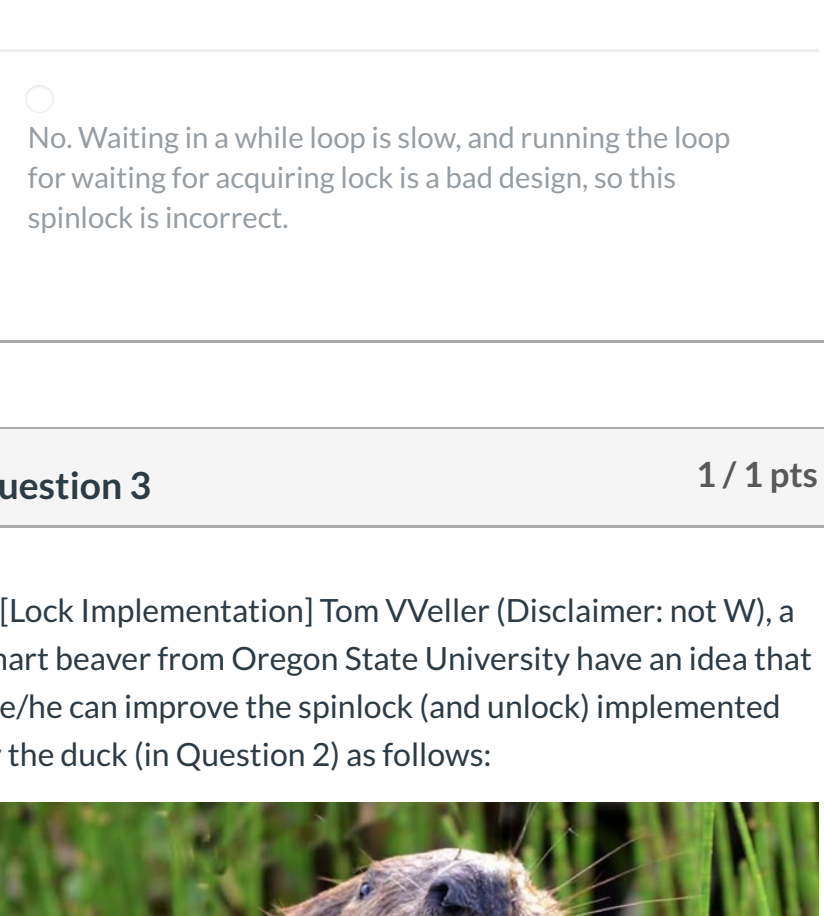
Generating a software interrupt #14 is not allowed by hardware in default, so it generates the general protection fault.

☐

This is an error. Running `int $14` in `soffint` must generate interrupt number #14, a page fault, but because our JOS implementation is wrong, it generates the general protection fault, interrupt number #13.

Question 21 / 1 pts

2.[Lock Implementation] A super-smart duck from Oregon implemented a spinlock (and unlock) as follows:



```
struct lock {
    uint32_t lock_variable; // 0:
    available, 1: locked
};

void spin_lock(struct lock *l) {
    while(l->lock_variable) {
        l->lock_variable = 1;
    }
}

void spin_unlock(struct lock *l) {
    l->lock_variable = 0;
}
```

Question: is this spinlock implementation correct? Choose the best answer from below.

☐

Yes. This spinlock waits until the value of the lock variable becomes 0 in a while loop, and this is why it is called as a spinlock. After breaking from the while loop (if the value in the lock variable becomes 0), the first thread observes the value 0 and set the value as 1 in order to block other threads from acquiring the lock. Because all the other threads must wait in the while loop until the lock variable becomes 0 for the next time (if `spin_unlock` is called), so this lock is a correct lock and can create a critical section for mutex.

☐

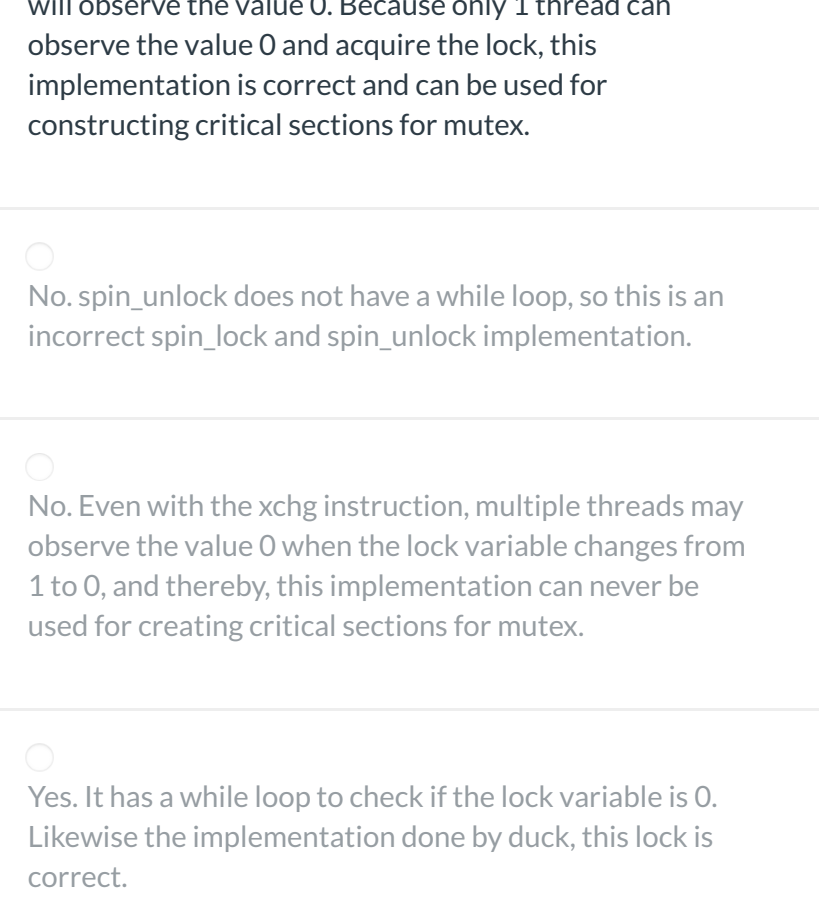
No. This spinlock will have two or more instructions for testing the lock variable if 0 and setting the lock variable as 1. Implemented in multiple instructions for the test-and-set would allow a race condition, and thereby, this spinlock cannot create critical section for mutex.

☐

No. Waiting in a while loop is slow, and running the loop for waiting for acquiring lock is a bad design, so this spinlock is incorrect.

Question 31 / 1 pts

3.[Lock Implementation] Tom VVeller (Disclaimer: not WU), a smart beaver from Oregon State University have an idea that she/he can improve the spinlock (and unlock) implemented by the duck (in Question 2) as follows:



```
void spin_lock(struct lock *l) {
    while(xchg(&l->lock_variable, 1)
!= 0);
}

void spin_unlock(struct lock *l) {
    xchg(&l->lock_variable, 0);
}

uint32_t xchg(volatile uint32_t
*addr, uint32_t newval) {
    uint32_t result;
    asm volatile("lock; xchgl %0, %1"
: "+m" (addr), "=a" (result)
: "1" (newval)
: "cc");
    return result;
}
```

Question: is this lock implementation correct? Choose the best answer from below.

☐

Yes. The `xchg` instruction in the x86 architecture swaps the value supplied to the function with the value in the lock variable at the same time, atomically, meaning that running the swap as a single instruction, without having any interference by other threads. So returning 0 from the function `xchg` means that the lock is free at a moment, and because the instruction is atomic, only one thread will observe the value 0. Because only 1 thread can observe the value 0 and acquire the lock, this implementation is correct and can be used for constructing critical sections for mutex.

☐

No. `spin_unlock` does not have a while loop, so this is an incorrect `spin_lock` and `spin_unlock` implementation.

☐

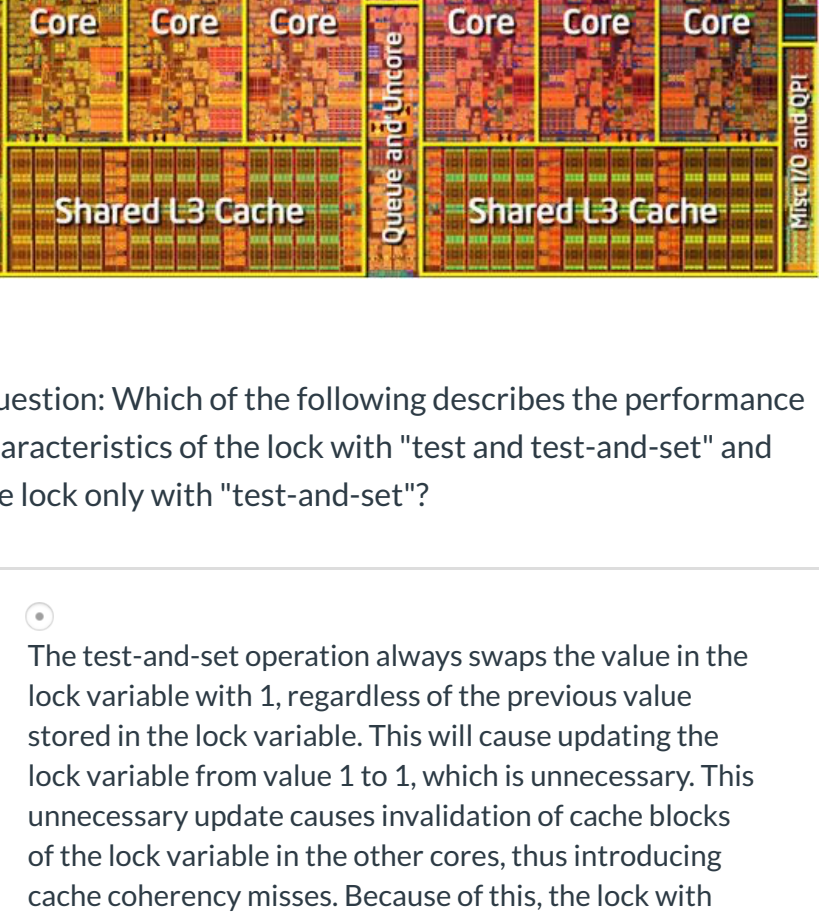
No. Even with the `xchg` instruction, multiple threads may observe the value 0 when the lock variable changes from 1 to 0, and thereby, this implementation can never be used for creating critical sections for mutex.

☐

Yes. It has a while loop to check if the lock variable is 0. Likewise the implementation done by duck, this lock is correct.

Question 41 / 1 pts

4.[Lock Implementation] Ben lee (Disclaimer: not L), who studies computer architecture at Oregon State University, has heard that a lock implementation with "test and test-and-set" is much more efficient than the implementation using only "test-and-set" atomic instruction.



Question: Which of the following describes the performance characteristics of the lock with "test and test-and-set" and the lock only with "test-and-set"?

☐

The test-and-set operation always swaps the value in the lock variable with 1, regardless of the previous value stored in the lock variable. This will cause updating the lock variable from value 1 to 1, which is unnecessary. This unnecessary update causes invalidation of cache blocks of the lock variable in the other cores, thus introducing cache coherency misses. Because of this, the lock with "test-and-set" would suffer many L1 cache misses, while the lock with "test and test-and-set" does not. So "test and test-and-set" is faster.

☐

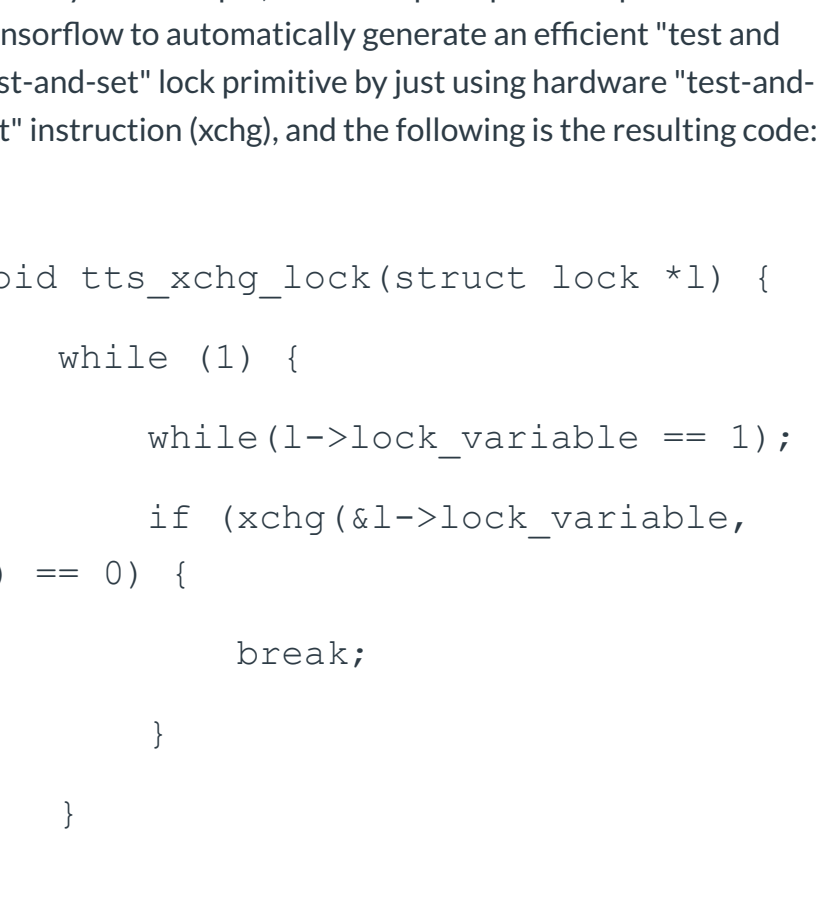
Ben heard about an incorrect information. Because "test and test-and-set" involves an additional "test" step on top of "test-and-set", so "test and test-and-set" is slower than "test-and-set". So it is not efficient at all.

☐

Historically, "test and test-and-set" is believed more efficient than "test-and-set" because the former would suffer less cache misses. So in an old CPU implementation, what Ben heard is correct: "test and test-and-set" is more efficient than "test-and-set". However, because modern CPUs such as Intel x86 are highly optimized for its performance (to win from the battle versus AMD), this is not true for nowadays. Both "test and test-and-set" and "test-and-set" does not incur many cache coherence misses.

Incorrect Question 50 / 1 pts

5.[Lock Implementation] Alan Fem (Disclaimer: not rn), who studies Artificial Intelligence, heard that Intel processors do not have the true "test and test-and-set" instruction. The instruction `cmpxchg` (compare-and-exchange) should work as "test and test-and-set", however, due to the complexity in designing memory bus, they implemented it as always writing the value in the memory bus regardless of the result of the comparison (but they do not update the value if the comparison fails).



Alan thought that the method that always writes to the memory bus is stupid, so she/he pull up his/her powerful TensorFlow to automatically generate an efficient "test and test-and-set" instruction (xchg), and the following is the resulting code:

```
void tts_xchg_lock(struct lock *l) {
    while (1) {
        while(l->lock_variable == 1);
        if (xchg(&l->lock_variable,
1) == 0) {
            break;
        }
    }
}
```

(and she/he will use the same `spin_unlock` from the previous question).

Question: Is this implementation correct? Choose the best answer from the below.

☐

No. The implementation of `tts_xchg_lock` is correct, however, this implementation must use "test and test-and-set" when it runs the unlock operation. Re-using `tts_xchg_lock` will render this lock implementation incorrect.

☐

Yes. Although the implementation does not use an atomic operation (`xchg`) when it runs the "test" step (while), because the "test-and-set" (`xchg`) step runs atomically, only 1 thread can observe the value 0 from the lock variable, thus letting only 1 thread breaking the outer while loop. Additionally, because the inner while loop that checks if the lock variable is 1 prevents updating the lock variable from value 1 to 1, this implementation may remove cache coherence misses, meaning that it can run more efficiently than "test-and-set" spinlock that Tom implemented.

☐

Yes. TensorFlow always generates the best result as AlphaGo does.

☐

No. This implementation does not use an atomic operation (either `xchg` or `lock cmpxchg`) when it runs the "test" step (while). This will make the entire "test and test-and-set" operation non-atomic, and thereby, two or more thread may break the while loop. Therefore, this implementation cannot create critical sections for mutex.

Question 61 / 1 pts

6.[Deadlock] Using spinlock, a beaver implemented a critical section between two threads as follows:

```
struct lock l1, l2;

Thread 1:

spin_lock(&l1);
spin_lock(&l2);
...
spin_unlock(&l2);
spin_unlock(&l1);

Thread 2:
spin_lock(&l2);
spin_lock(&l1);
...
spin_unlock(&l1);
spin_unlock(&l2);
```

Question: can this implementation cause deadlock (assuming no infinite loop in the critical section)?

☒

Yes

☐

No

Question 71 / 1 pts

7.[Deadlock] Using spinlock, a beaver implemented a critical section between two threads as follows:

```
struct lock l1, l2;

Thread 1:

spin_lock(&l1);
spin_lock(&l2);
...
spin_unlock(&l2);
spin_unlock(&l1);

Thread 2:
spin_lock(&l1);
spin_lock(&l2);
...
spin_unlock(&l2);
spin_unlock(&l1);
```

Question: can this implementation cause deadlock (assuming no infinite loop in the critical section)?

☐

No

☒

Yes

Incorrect Question 80 / 1 pts

8.[Deadlock] Using spinlock, a beaver implemented a critical section between two threads as follows:

```
struct lock l1, l2, meta;

Thread 1:

spin_lock(&meta);
spin_lock(&l1);
spin_lock(&l2);
spin_unlock(&meta);
...
spin_unlock(&l2);
spin_unlock(&l1);

Thread 2:
spin_lock(&meta);
spin_lock(&l2);
spin_lock(&l1);
spin_unlock(&meta);
...
spin_unlock(&l2);
spin_unlock(&l1);
```

Question: can this implementation cause deadlock (assuming no infinite loop in the critical section)?

☐

No

☒

Yes

Partial Question 94 / 5 pts

9. Answer each of the following questions.

(True/False) When JOS bootstrapping processor wakes up other application processors for the first time, those processors running in the protected mode enabled with paging because the bootstrapping processor has already enabled these two modes:

(True/False) JOS does not require the global kernel lock if we assign a separate kernel stack per each processor

(True/False) One way of creating a mutually exclusive critical section is to disable system-wide interrupt using 'cli' as lock and then enable the interrupt back using 'sti' as unlock after finishing the execution in the critical section. By using these two instructions, one can implement a critical section for mutex in Ring 0

(Select one) In JOS, which value the fork(2) returns to the child environment if the function have been executed successfully?

(Select one) Which of the following stores the information about the reason of a page fault?

Answer 1:
True

Answer 2:
False

Answer 3:
False

Answer 4:
0

Answer 5:
Trapframe

Quiz Score: 10 out of 13

Last Attempt Details:

Time:	30 minutes
Current Score:	10 out of 13
Kept Score:	10 out of 13

1 More Attempt available

Take the Quiz Again

(Will keep the highest of all your scores)