Quiz 3

截止日期 11月29日 23:59 **分數** 13 **問題** 9
可用 11月19日 8:30 - 11月29日 23:59 11 天 **時間限制** 無 **允許的嘗試** 3

說明

CS 444/544 Operating Systems II

Quiz III

Quiz Description

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.

Only, three attempts are allowed for this quiz.

再次參加測驗

嘗試記錄

	嘗試	時間	分數
最新的	嘗試 1	15 分鐘	得分：13；總分：13

正確答案已隱藏。

此嘗試的分數： 得分：**13**；總分：13
已提交11月22日 15:33
此嘗試持續 15 分鐘。

問題 1 **1 / 1 分數**

1. [Trap handling in JOS Lab 3] In JOS Lab 3, we have a user-level program named as softint 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 generates a general protection fault, not a page fault upon running int $14? Choose the best answer that describes this phenomenon.

- Generating a software interrupt #14 is not allowed by hardware in default, so it generates the general projection fault.
- This is an error. Running int $14 in softint must generate interrupt number #14, a page fault, but because our JOS implementation is wrong, it generates the general protection fault, interrupt number #13.
- **When we 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.**
- The software interrupt instruction, int $X, always generates an interrupt number X-1, so it generates the general projection fault (interrupt number #13) when we run int $14

問題 2

1 / 1 分數

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.

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.

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 will set the value as 1 in order to block other threads from acquiring the lock. Because all the other thread 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. 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.

問題 3

1 / 1 分數

3. [Lock Implementation] Tom VVeller (Disclaimer: not W), 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. It has a while loop to check if the lock variable is 0. Likewise the implementation done by duck, this lock is correct.

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.

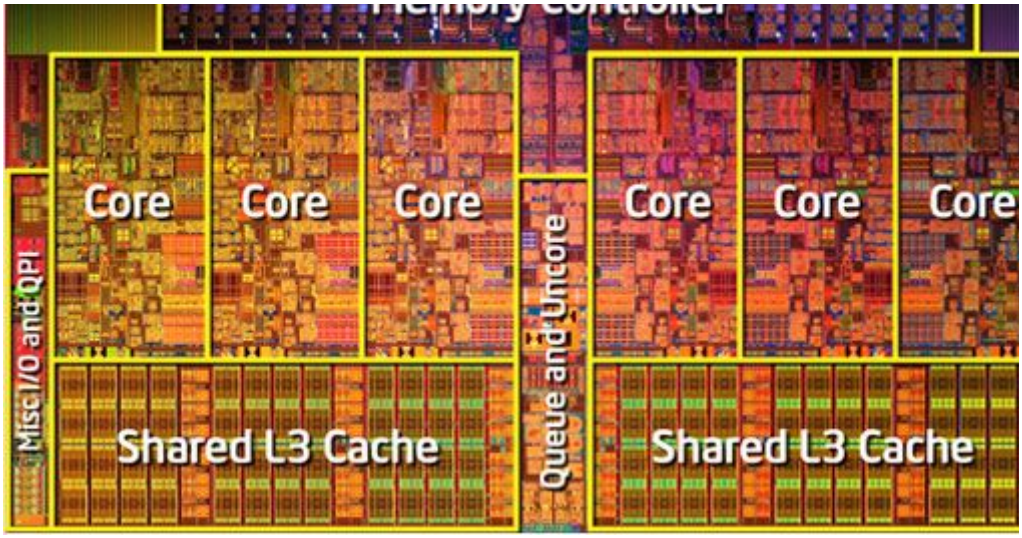
Yes. The xchg instruction in the x86 architecture swaps the value supplied to the function with the value in the lock variable at the same time, atomically, meaning that running the swap as a single instruction, without having any interference by other threads. So returning 0 from the function xchg means 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.

問題 4

1 / 1 分數

4. [Lock implementation] Ben Iee (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"?

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.

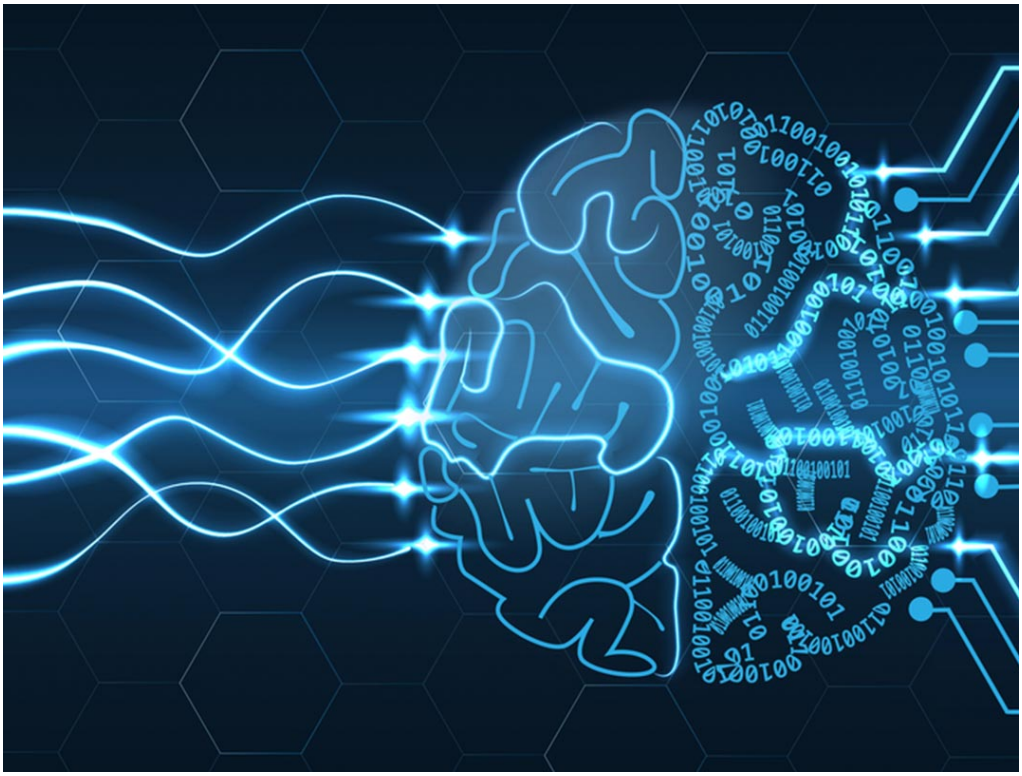
Ben heard about an incorrect information. Because "test and test-and-set" involves an additional "test" step on top of "test-and-set", so "test and test-and-set" is slower than "test-and-set". So it is not efficient at all.

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.

問題 5

1 / 1 分數

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" lock primitive by just using hardware "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 spin_unlock for unlocking the lock acquired by tts_xchg_lock will render this lock implementation incorrect.

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.

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

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.

問題 6

1 / 1 分數

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) ?

- No
- Yes

問題 7

1 / 1 分數

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

問題 8

1 / 1 分數

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

問題 9

5 / 5 分數

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

[選擇]

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

[選擇]

(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 3 [選擇]

(Select one) In JOS, which value the fork() returns to the child environment if the function have been executed successfully? 0

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

回答 1：

False

回答 2：

False

回答 3：

False

回答 4：

0

回答 5：

Trapframe

測驗分數： 得分：**13**；總分：13