CS444/544 Operating Systems II Instructor: Yeongjin Jang

Name: OSU ID:

CS 444/544 Operating Systems II
Sample Quiz #2

You have 30 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

If you find any ambiguity in the questions, be sure to write down any assumptions you make. You do not have to list all of the assumptions.

In case if we cannot read your answer nor interpret your answer, we can't give you credit.

Write your name and OSU ID on this cover sheet, and make sure you have all pages with the quiz sheet package: Quiz #2 should have total 8 pages.

NO Internet access, NO communication with other students, and NO consulting to textbook, slides, laptop, etc.

Section	I	II	Total
Score			
Max Score	20	80	100

II. Multiple choices (20 pts, 4 pts each)

II-1. In x86, which of the following instruction runs atomically?

a) cmpxchg	b) pusha	c) lea	d) xchg	e) mov

II-2. In x86, which of the following instruction runs atomic test and test-and-set?

a) cmpxchg	b) int $0x30	c) lock cmpxchg	d) lock	e) xchg

II-3. Can we implement locks without using any of hardware atomic instructions?

a) YES	b) NO			

II-4. Which register is being used for storing ‘compare’ value when running the cmpxchg instruction?

a) CR3	b) EAX	c) EBX	d) ECX	e) ESP

II-5. Which of the following term is not relevant to data racing / thread synchronization?

a) LOCK	b) Intel TSX	c) LL/SC	d) Page Table	e) test-and-set

V. Locks (80 pts)

Beaver the Hacker implements a spin lock as follows.
Please answer the following questions regarding the implementation.

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

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

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

(See next pages)

1) (10 pts) The code what Beaver has written seems not optimal. So Duck has suggested the following implementation:

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

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

Is this implementation correct (for a spin lock)? Please provide your answer (correct or incorrect) and provide a justification for your answer (i.e., If correct, why can this implementation synchronize threads? If incorrect, why threads are not able be synchronized with these functions?)

2) (10 pts) spin_unlock() implemented by Beaver is slow. Can we use Duck's implementation (spin_unlock()) to replace Beaver's spin_unlock(), like the following?

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

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

Please provide your answer (yes or no) and provide a justification for your answer.

3) (20 pts) spin_lock() implemented by Beaver is also somewhat slow. Can you describe why Beaver's implementation using xchg() is slow and suggest a better implementation (please provide a pseudo-code for your implementation suggestion)?

4) (20 pts) Beaver is now encountered a case of deadlock while running the following two threads:

```
struct lock l1, l2;

Thread 1:

spin_lock(&l1);
spin_lock(&l2);

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

a) Can you describe a possible scenario of deadlock when Beaver runs these two threads?

b) Can you resolve the problem? Please provide your fix as pseudocode (or description).