UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

MIDTERM EXAMINATION, March, 2023
Third Year – Materials
ECE344H1 - Operating Systems
Calculator Type: 4
Exam Type: A
Examiner – D. Yuan

Duration: 55 minutes

*There are **8** total numbered pages, **6** Questions.* Read All Questions Carefully!

The teaching staff will not answer any questions about the exam. If any questions appear unclear or ambiguous, make any assumptions you need, state them, and answer the question that way. If you believe there is an error, state what the error is, fix it, and respond as if fixed.

For question 4-6: you receive 20% of the mark of each (sub-)question if you leave the answer blank.

Please put your FULL NAME and Student ID on THIS page only.

Name: _____

Student ID: _____

	Total Marks	Marks Received
Question 1	15	
Question 2	12	
Question 3	16	
Question 4	16	
Question 5	16	
Question 6	25	
Total	100	

Assume a single core CPU is used in this exam.

Q1 (15 marks): Multiple Choices (choose all correct answers)

(1) Assume we have a C program that has a global variable `A` with initial value being 0. If two threads each execute `A++`, what are the possible values of `A` after both threads complete?

(a) 0
(b) 1
(c) 2
(d) 3

b, c

(2) Which of the following scenarios will trigger an event?

(a) Divide by zero.
(b) Null pointer dereference.
(c) Execute test-and-set instruction.
(d) User presses a key on the keyboard.
(e) Modify PC (Program Counter).
(f) Modify SP (Stack Pointer).

a, b, d,

(3) If a kernel-level thread executes "cin >> variable_a" (C++ input operation), which of the following will happen?

(A) The status of all threads in the same process will be changed from Running to Ready
(B) Only the status of this thread will be changed from Running to Ready
(C) The status of all threads in the same process will be changed from Running to Waiting
(D) Only the status of this thread will be changed from Running to Waiting
(E) The status of all threads in the same process will be changed from Ready to Waiting
(F) Only the status of this thread will be changed from Ready to Waiting
(G) The status of this thread will not be changed

D,

(4) Which of the following are true about processes and threads?

(A) There can be multiple concurrent processes running the same program.
(B) A process can have multiple threads.
(C) A thread is more lightweight than a process.
(D) There can be multiple processes that are in running state at the same time.
(E) There can be multiple processes that are in ready state at the same time.
(F) Different threads do not share stacks, but they share the same heap.

A, B, C, E, F

(5) Which of the following are true about an OS?

(a) Some parts of the OS have to be written in assembly (instead of the C language).
(b) A web browser is an example of an OS.
(c) If an OS crashes, it will bring down all the processes running on top of it.
(d) Modern OSes require hardware support.

a, c, d

Question 2 (12 marks): OS161

Consider the following implementation of semaphore's `P()` and `V()` operations in OS161. Does it work? If not, fix it. No need to explain your answer.

```
P(sem) {
  spl = splhigh();
  if (sem->count==0) {
    thread_sleep(sem);
  }
  sem->count--;
  splx(spl);
}
```

```
V(sem) {
  spl = splhigh();
  sem->count++;
  thread_wakeup(sem);
  splx(spl);
}
```

change "if" to "while" in P().

Question 3 (16 marks): Events.

What are the minimal (i.e., absolutely necessary) actions the CPU needs to take when an event (i.e., exception or interrupt) happens? We list 4 operations below. For the first 3 operations, choose one correct answer from the choices given.

(1) Turn ___ interrupt
- (a) on
- (b) off

(b)

(2) Save the ___
- (a) PC (Program Counter).
- (b) SP (Stack Pointer).
- (c) General purpose registers.
- (d) All of the above.
- (e) This step is not necessary.

(a)

(3) Switch to ___
- (f) User mode.
- (g) Kernel mode.
- (h) Could either be User mode or Kernel mode.
- (i) This step is not necessary.

(g)

(4) Set the PC to ___**InterruptDescriptorTable[N]**____ (fill in this blank; assume the event number is N)

Question 4 (16 marks): UNIX Shell

Consider a buggy UNIX shell code like the following:

```
1 shell (..) {
2   while (1) {
3     char *cmd = read_command();
4     int child_pid = fork();
5     if (child_pid != 0) {
6       Manipulate STDIN/OUT/ERR file descriptors for pipes, redirection, etc.
7       exec(cmd);
8       panic("exec failed");
9     } else {
10      wait();
11    }
12  }
13 }
```

Recall that wait() will suspend the calling process (i.e., puts it to sleep) until any one of its child processes terminate. Now answer the following questions:

(a) What happens if the user executes "ls" on this shell? (1-2 sentences.)

The parent becomes ls, prints the content of current directory, and terminates. The child hangs forever.

(b) How can we fix the bug? (No explanation needed.)

line 5 change to: if (child_pid == 0)

Question 5 (12 marks): Atomic Instruction

Besides test-and-set, another atomic instruction that is commonly supported by CPUs is compare-and-swap. The pseudocode below illustrates how it works:

```
bool compare_and_swap (bool* addr, bool oldval, bool newval) {
    bool old_reg_val = *addr;
    if (old_reg_val == oldval)
        *addr = newval;
    return old_reg_val;
}
```

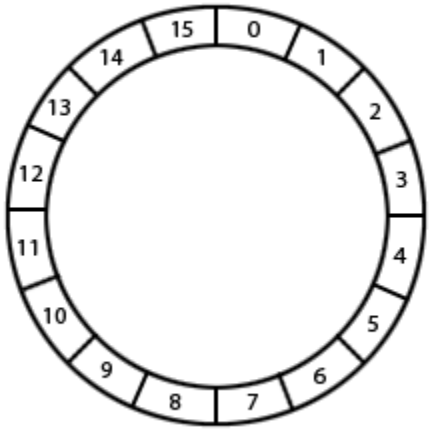
It compares the current value at `addr` with `oldval`, and if they're the same, it further sets `*addr` to `newval`. Regardless of the comparison result, it will return the old value stored at `addr`.

Similar to test-and-set, the underlying hardware will guarantee that these steps are performed atomically.

Now, how do you implement lock_acquire and release using compare_and_swap? *Note the only hardware support you can only use is compare_and_swap(). Both spin-lock and sleep-based semantics are acceptable.* Write your code below:

```
struct lock {
    bool held = false; // initial value sets to false;
};

void lock_acquire (struct lock* lock) {
  // write your code here, no more than 3 lines.

  while (compare_and_swap(&(lock->held), false, true);




}

void lock_release (struct lock* lock) {
    // write your code here, no more than 3 lines

   lock->held = false;



}
```

Question 6 (24 marks): Synchronization.
There are sixteen ECE profs meeting around a round table, each with a microphone in front of them. However, when a prof is speaking, the adjacent microphones must be turned off to avoid sound resonance.



This is essentially a synchronization problem since each prof must get control of three microphones before speaking. For example, prof #1 must get control of microphone #0, #1, and #2 before speaking.

Suppose we have an array of locks, each representing a microphone, and they have been properly initialized. Jacky, a student from ECE344, proposes the following solution:

```
lock_t* microphone[16];

void get_microphones_to_speak(int num) {
      // Assume num is always in the range of [0, 15]
      int left = (num - 1 + 16) % 16;
      int right = (num + 1) % 16;

      // Get control of adjacent microphones and my own
      lock(microphone[left]);
      lock(microphone[num]);
      lock(microphone[right]);

      speak();

      // Release microphones
      unlock(microphone[right]);
      unlock(microphone[num]);
      unlock(microphone[left]);
}
```

a. What is the problem with this program, assume there are 16 threads running `get_microphones_to_speak()`, each with a different `num` (within the range of [0, 15]) that represents different prof. Describe a concrete interleaving that shows the problem.

Could deadlock. Assume there is a context switch right after each thread acquired the left lock: microphone[left]. In that case,

b. Fix the problem with the simplest fix. Do not create any additional global or static variables.

```
void get_microphones_to_speak(int num) {
    // Assume num is always in the range of [0, 15]
    // Add your implementations
    // The idea is to grab the lock in the numerical order as their lock
number

      int ordered_locks[3];
      ordered_locks[0] = (num - 1 + 16) % 16;
      ordered_locks[1] = num;
      ordered_locks[2] = (num + 1) % 16;
      sort (ordered_locks); // Sort the locks
      for (int i = 0; i < 3; i++)
          lock(microphone[ordered_locks[i]]);

     speak();

      for (int i = 2; i >= 0; i--)
          unlock(microphone[ordered_locks[i]]);



}
```

This page is left blank intentionally.