UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE AND ENGINEERING

MIDTERM EXAMINATION, March, 2017
Third Year – Materials
ECE344H1 - Operating Systems
Calculator Type: 2
Exam Type: A
Examiner – D. Yuan

Please Read All Questions Carefully!
For each of the questions from 3 to 6, you will receive *20%* of the mark if you choose NOT to answer the question (i.e., leave it blank). But if you answer the question and your answer is completely wrong you will receive 0.

*There are **8** total numbered pages, **6** Questions.*
You have 60 minutes. Budget your time carefully!

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

Name: _____

Student ID: _____

Question 1 (20 marks): Multiple choices. There are one or more correct choices for each question. Mark all of them. For each question, you will get full marks only if your answer is 100% correct, otherwise you will get 0.

(1)(3 marks) multiple threads used in multi threading share:

(A) Program counter
(B) Heap
(C) File descriptors
(D) Call stack
(E) None of the above

(B,C)

(2) (3 marks) In Unix Shell implementation, exec() is used in

(A) parent process and overwrites child address space
(B) child process and overwrites parent address space
(C) child process and overwrites child address space
(D) parent process and overwrites parent address space

(C)

(3) (3 marks) fork() returns

(A) 0 in child process
(B) 1 in child process
(C) process id of child in parent process
(D) process id of parent in child process
(E) process id of child in child process
(F) process id of parent in parent process

(A, C)

(4) (3 marks) Which of the following is/are true:

(A) Starvation could occur on an OS with non-preemptive scheduling
(B) Starvation could occur on an OS with preemptive scheduling
(C) Deadlock could occur on an OS with non-preemptive scheduling
(D) Deadlock could occur on an OS with preemptive scheduling
(E) None of the above

(A,B, C, D)

(5) (3 marks) A process can go

(A) from ready state to running state
(B) from ready state to waiting state
(C) from waiting state to ready state
(D) None of the above

(A, C)

(6) (5 marks) Which of the following is/are true:

(A) A process may contain multiple threads, but each thread belongs to only one process
(B) The process ID of a zombie process can be reused by another process
(C) If the memory usage of process A is twice the size as process B, then the latency of fork() in process A is roughly twice as the latency of fork() in process B.
(D) Kernel-level threads of the same process do not share the same memory address space.
(E) User-level threads of the same process share the same memory address space
(F) None of the above

(A, E)

Question 2 (10 marks): Consider this program:

```
void main() {
  fork();
  int pid = fork();
  if (pid == 0)
    exec("/bin/ls"); // ls does NOT create any process
  else
    fork();
}
```

How many processes are created when we execute this program once in a Shell, including the first process created by the Shell? No explanation is needed.

6

Marking: There are cases where students answered "6 threads", and we gave partial marks. Otherwise it's binary.

Question 3 (20 marks): Assume that on a Unix OS, the PID of a newly created process will be its parent's PID + 1 (and in case this PID is already used, it will try parent's PID + 2, parents' PID +3, so on and so forth). A PID will be immediately available for reuse once the process terminated and its parent has reaped it. Assume that the Shell process has PID 200, and no process on this system has PID greater than 200.

Consider this program:

```
void main() {
  int pid = fork();
  if (pid == 0) {
    exec("dummy"); // dummy is a program that exits without any output
    printf("L1: %d, %d\n", getpid(), getppid());
  }
  else {
    waitpid(pid);
```

```
    int pid2 = fork();
    if (pid2 == 0) {
      printf("L2: %d, %d\n", getpid(), getppid());
    } else {
      waitpid(pid2);
      printf("L3: %d, %d\n", getpid(), getppid());
    }
  }
}
```

Note that getpid() is a system call that returns the PID of the calling process; getppid() returns the PID of the parent of the calling process.

(a) (3 marks) Now if you execute this program in a Shell, how many possible outputs can this program produce, assuming it finishes normally?

Ans: only one possible output.

(b) (12 marks) Now if you execute this program once in the Shell, write down the output below, one row per line of output. You don't have to fill all the rows. If multiple outputs are possible, write down only one.

L2: 202, 201
L3: 201, 200

Question 4 (20 marks): TID lock

Bob proposes the following spin lock implementation. He thinks that by checking the thread id of the thread that acquires the lock he can ensure only a single thread ever holds the lock.

```
1. struct lock {
2.   int held;
3.   pid_t tid;
4. };
5.
6. void lock_acquire(struct lock *l) {
7. start:
```

```
8.   while (l->held);
9.   l->held = 1;
10.  l->tid = gettid(); // gettid returns the thread id of calling thread
11.  if (l->tid != gettid()) // retry if a different thread won
12.    goto start;
13. }
14. void lock_release(struct lock *l) {
15.   l->held = 0;
16. }
```

Is Bob right? If not, in the following table provide an interleaving, using only the line number of the program, of multiple threads that shows the problem. For example, the following table shows an interleaving where thread A first executes line 8 and 9, and thread B gets scheduled and gets stucked in the loop at line 8, and then thread A gets scheduled again and finishes the execution of lock_acquire().

Thread A	Thread B
8-9	
	8 (repeat N times)
10-13	

Write your answer below. You do not need to fill all the spaces. No need to further explain your answer.
Answer: No. Bob is not right.

Thread A	Thread B	Thread C (if necessary)
7-8		
	7-8	
9-11		
	9-11	
13		
	13	

Grading:
Yes/no: 4 marks

Question 5 (15 marks): Semaphore readers/writers
Consider the following code that is almost the same as we discussed in the lecture for reader/writer problem. (Recall the problem is that there can be multiple threads executing reader/writer functions, and you should allow multiple readers to read concurrently, but when a writer executes, no other reader or writer threads should execute.)

```
 1. int readcount = 0;
 2. Semaphore mutex = 1;
 3. Semaphore w_or_r = 1;
 4.
 5. writer {
 6.   P(w_or_r); // lock out readers
 7.   Write;
 8.   V(w_or_r); // up for grabs
 9. }
10.
11. reader {
12.   P(mutex);       // lock readcount
13.   readcount += 1; // one more reader
14.   if (readcount == 1) {
15.     V(mutex); // unlock readcount
16.     P(w_or_r); // synch w/ writers
17.   } else {
18.     V(mutex);   // unlock readcount
19.   }
20.   Read;
21.   P(mutex);      // lock readcount
22.   readcount -= 1; // one less reader
23.   if (readcount == 0)
14.     V(w_or_r); // up for grabs
15.   V(mutex);   // unlock readcount}
16. }
```

Does this code work? If not, in the following table provide an interleaving, using only the line number of the program, of multiple threads that shows the problem. (You do not need to fill all the spaces.)
No it doesn't work. (3 marks)

Thread A	Thread B	Thread C (if necessary)

11-15 (reader)		
	5-7 (writer)	
		11 - 14, 17-20 (read) ERROR! READ and WRITE at the same time

Question 6 (15 marks): Monitor: while or if?
Consider the following code that is almost the same as we discussed in the lecture for bounded buffer problem.

```
 1. Monitor bounded_buffer {
 2.   Resource buffer[2]; // 2 buffers available
 3.   Condition not_full; // space in buffer
 4.   Condition not_empty; // value in buffer
 5.
 6.   void put_resource (Resource R) {
 7.     while (buffer array is full)
 8.       wait(not_full);
 9.     Add R to buffer array;
10.     signal(not_empty); //wakeup a waiting thread, put it in ready queue
11.   }
12.
13.   Resource get_resource() {
14.     if (buffer array is empty)
15.        wait(not_empty);
16.     Get resource R from buffer array;
17.     signal(not_full); //wakeup a waiting thread, put it in ready queue
18.     return R;
19.   }
20. } // end monitor
```

The only change we made here from the lecture slides is that we replaced "while" to "if" on line 14. Does this code work? If not, in the following table provide an interleaving, using only the line number of the program, of multiple threads that shows the problem. (You do not need to fill all the spaces.) Assume that initially the buffers are empty.

Answer: No it doesn't work. (3 marks)

Thread A	Thread B	Thread C (if necessary)
13-15 (wait)		
	6-11 // wake up thread A	
		// a new consumer thread 14-19
16 ERROR: try to consume from empty buffers!		