CPSC/ECE 3220 – Summer 2017 – Exam 1 Name: __ (with suggested answers) __

1. The four generic actions that hardware performs in response to an interrupt are: (1.5 pts. each)

1) save PC and PSR
2) switch to kernel execution mode
3) disable or restrict further interrupts
4) load new PC from the IVT (interrupt vector table)

Process/Thread. Circle **one or both** of P or T, as applies. (2 pts. each)

2. [✓ P] / [✓ T] Has an associated control block.
3. P / [✓ T] Has an associated SP (stack pointer).
4. P / [✓ T] Has an associated PC (program counter).
5. [✓ P] / T Is created in response to the UNIX system call fork().
6. [✓ P] / T The abstraction for protection provided by the OS kernel.
7. P / [✓ T] A single execution sequence that represents a separately schedulable task.

True/False. Circle **only one** of T or F. (1.5 pts. each)

8. [✓ T] / F A return from interrupt instruction should be a privileged instruction.
9. T / [✓ F] To provide multiuser protection, hardware must have at least three execution modes.
10. [✓ T] / F When power is turned on, a processor will execute a bootstrap loader in kernel mode.
11. T / [✓ F] When a user attempts to execute a privileged instruction in user mode the CPU should stop.
12. [✓ T] / F The interrupt vector table should be held in kernel memory and not arbitrarily changed by users.
13. [✓ T] / F The OS should be invoked by a syscall or trap instruction rather than a normal jump to subroutine.
14. [✓ T] / F Users should not be allowed to write into the execution mode bit(s) in the processor status register.

Multiple Choice. Circle **only one** of the possible answers. (1.5 pts. each)

15. Store instructions are allowed:
   (a) in user mode only
   (b) in kernel mode only
   (c) in both modes without any memory protection
   [✓ (d) in both modes but with a bounds or other check on the effective address when in user mode]
   (e) in both modes but with a bounds or other check on the effective address when in kernel mode
   (f) in both modes but with a bounds or other check on the effective address regardless of user or kernel mode

16. What is the component that is responsible for loading the initial value in the program counter for an application program before it starts running?
   (a) linker
   [✓ (b) kernel]
   (c) compiler
   (d) bootstrap loader
   (e) none of the above

17. High processor utilization typically implies:

(a) high throughput
(b) high response time
(c) both
(d) neither

Word Bank. Write one of the words or terms from the following list into the blank appearing to the left of the appropriate definition. Note that there are more words and terms than definitions. (2 pts. each)

API	HAL	memory barrier	process	semaphore
boot ROM	handler	microkernel	program	starvation
busy waiting	IRET	monothlic kernel	race condition	synchronization
child process	kernel	operating system	resource	thread
concurrency	liveness	parent process	safety	upcall
critical section	lock	privacy	security	virtualization

18. _synchronization_ The coordination of access to shared data.

19. _critical section_ A sequence of code that operates on shared state.

20. __IRET__ A privileged instruction that restores the PC and PSR.

21. __handler__ A kernel procedure invoked when an interrupt occurs.

22. __API__ The system call interface provided by an OS to applications.

23. __safety__ The property of a program such that it never enters a bad state.

24. __resource__ A physical or virtual entity that can be assigned to a user or application.

25. __security__ A computer's operation cannot be compromised by a malicious attacker.

26. __starvation__ The lack of progress for one task, due to resources given to higher priority tasks.

27. __upcall__ An event, interrupt, or exception delivered by the kernel to a user-level process.

28. __virtualization__ Provide an application with the illusion of resources that are not physically present.

29. _monolithic kernel_ An OS design where most of the OS functionality is linked together inside the kernel.

30. __kernel__ Lowest level of software on a system, with full access to all the capabilities of the hardware.

31. _race condition_ When the behavior of a program relies on the interleaving of operations of different threads.

32. _operating system_ A layer of software that manages a computer's resources for its users and their applications.

33. The textbook describes the OS acting in three different roles. Identify at least two distinct actions, features, or services provided by the OS in the role of referee. (2 pts.)

[from question 1 at the end of chapter 1]
Various possible answers, e.g., scheduling, checking permissions

34. What is the difference between a guest OS and a host OS? (2 pts.)

[from question 2(j) at the end of chapter 1]
A guest OS runs in a virtual machine provided by a host OS, which runs on physical hardware and provides the abstraction of a virtual machine.

35. Can the lost update problem happen on a uniprocessor, or is it a potential problem only for multiprocessors? If only for multiprocessors, then explain why the problem cannot happen on a uniprocessor. If it can happen on either a uniprocessor or multiprocessor, show an example of the lost update problem on a uniprocessor. (5 pts.)

Yes, a lost update can happen in both. It can happen in a uniprocessor because of thread switching:

```
              X starts at 0
Thread A                        Thread B
load X
          → thread switch →
                                load X
                                add 1
                                store X
          ←thread switch ←
add 1
store X
```

36. Assume the only external interrupt available on a computer system is a hardware timer, yet overlapping I/O with CPU execution is important for the OS design. An I/O operation is started by an instruction that returns immediately. An I/O event, such as an I/O operation completion or error, causes bits to be set in a device register associated with the I/O device. Suggest a design approach to allow the OS kernel to handle I/O completions and errors. (2 pts.)

[from question 14 at the end of chapter 2]
The key idea is to incorporate polling.

37. The bzero() function writes zero in each memory word in a specified memory region. If you make the function multithreaded, do you need a lock and/or condition variables? Explain your answer. (4 pts.)

[from Figure 4.7]
No. Independent parts of the memory region can be assigned to the different threads. Even if there is some overlap between the threads, all threads would be writing zeros.

38. The ability to disable and re-enable interrupts was sufficient to implement a lock object for a uniprocessor system. Why was an additional spinlock needed to implement a lock object for a multiprocessor system? (4 pts.)

[from section 5.7.3]
Disabling interrupts affects only the current processor; it does not affect the other processors in a multiprocessor system. Thus interrupt handlers and system calls can be running on the other processors and interfere with the update to the data structures of the lock.

39. Show the implementation of the spinlock acquire() method. (3 pts.)

[from Figure 5.16]

```
void acquire(){
        while( test_and_set( &value ) )
                ;                               // spin
}
```

40. What is wrong with this code for the spinlock release() method? (1 pt.)

```
void release(){
        value = 0;
}
```

[from Figure 5.16]
The release() function should have a memory barrier to make the change to value visible and prevent reordering.

Questions 41-42: Consider the implementation of a lock and a condition variable. (4 pts. total)

41. What are the state and synchronization variables associated with a lock object? (There is no need to give the methods.)

[from Figure 5.17]
integer value (to indicate free or busy)
spinlock
waiting queue

42. What are the state and synchronization variables associated with a condition variable object? (There is no need to give the methods.)

[from Figure 5.18]
waiting queue

Question 43-46: Consider the following function that can be invoked by "thread_create(&threads[i], &go, i);" As shown in the textbook, a single parameter can be passed to a thread; in this case, it is a single integer value. (1.5 pts. each)

```
void go( int n ){
        int k;
        for( k = 0; k < 4; k++ ){
                printf( "hello from thread %d\n", n );
        }
        thread_exit( 0 );
}
```

43. When the thread_create() function is complete, is the created thread in INIT, READY, or some other state?

[from discussion on p. 148 in chapter 4]
READY

44. When the thread_exit() function is complete, the thread that called thread_exit() may continue to exist in a FINISHED state. Why is the thread not immediately destroyed?

[from discussion on p. 152 in chapter 4]
The thread may have to wait for the return code to be read by a thread_join() call from the parent.

45. Is the variable k in the code above a per-thread variable or a shared variable that is accessible by other threads?

[from question 8 at end of chapter 4]
per-thread

46. Where does the compiler store the variable k?

[from question 8 at end of chapter 4]
on the user-mode stack for that thread

47. What synchronization actions are missing from the following code for the blocking bounded queue (BBQ) remove() method? (This method appears as the get() method in the slides, along with some other different variable names. Base your answer on either version of the method; the same actions are missing in each version.) (4 pts.)

```
// Figure 5.8 from textbook
int BBQ::remove(){
        int item;


        while( front == nextEmpty ){
                itemAdded.wait( & lock );
        }

        item = items[ front % MAX ];
        front++;



        return item;
}
```

```
// version of remove() that is given in the slides
get(){
        // implicit int item;


        while( front == tail ){
                empty.wait( lock );
        }

        item = buf[ front % MAX ];
        front++;



        return item;
}
```

[from Figure 5.8]
Locking is missing (i.e., there should be explicit calls to lock.acquire() and lock.release()). The signal to a thread possibly waiting to insert is also missing (i.e., remove() should have a call to itemRemoved.signal() after the update to items[] and front).

Extra Credit. (up to 5 pts.)

Before entering a "priority critical section", a thread calls a PriorityLock::enter(priority) method. When the thread exits the priority critical section, it calls PriorityLock::exit(). If several threads are waiting to enter a priority critical section, the one with the numerically highest priority should be the next one allowed in. Describe your general approach in implementing a PriorityLock object. Define the state and synchronization variables for a PriorityLock object and describe the purpose of each.

[from question 14 at end of chapter 5]
The key idea is to use a queue of condition variables as done for the FIFO Blocking Bounded Buffer in Figure 5.14, except that the queue for a PriorityLock would be ordered by priority.