CPSC/ECE 3220 – Fall 2017 – Exam 1 Name: ______________________

1. Give the definition (note: not the roles) for an operating system as stated in the textbook. (2 pts.)

<u>Referee / Illusionist / Glue.</u> Circle **only one** of R, I, or G. (1 pt. each)

2. R / I / G Virtual machine.
3. R / I / G Disk details such as sector size are hidden.
4. R / I / G Higher-level objects are provided, such as files.
5. R / I / G Resource allocation among users and applications.
6. R / I / G Files written by one application can be read by another.
7. R / I / G Prevent users from accessing each other's files without permission.

<u>Kernel mode / User mode.</u> Circle **one or both** of K and U, as applies. (2 pts. each)

8. K / U Valid to execute a store instruction in this mode.
9. K / U An exception occurs if the execution of a privileged instruction is attempted in this mode.
10. K / U In this mode, the effective address generated by load instruction is checked so that access is allowed only to a designated region of memory.

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

1)

2)

3)

4)

12. Identify the two major differences between a jump-to-subroutine instruction (i.e., procedure call instruction) and a syscall instruction (i.e., software interrupt or trap instruction). (2 pts.)

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

13. P / T Has an associated control block.
14. P / T Has an associated data segment.
15. P / T Has an associated PC (program counter).

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. (1 pt. each)

asynchronous I/O	green threads	OS kernel	resource
boot ROM	guest OS	portability	safety
busy waiting	hardware timer	privileged instruction	semaphore
concurrency	host OS	process	test-and-set instruction
condition variable	interrupt handler	program	thread
critical section	interrupt masking	protection	thread context switch
efficiency	liveness	publish (RCU)	thread create
false sharing	lock	race condition	throughput
finished list	MCS lock	readers/writers lock	virtualization
grace period (RCU)	memory barrier	reliability	waiting list

16. ____________________ The rate at which a group of tasks are completed.

17. ____________________ A sequence of code that operates on shared state.

18. ____________________ Multiple activities that can happen at the same time.

19. ____________________ A kernel procedure invoked when an interrupt occurs.

20. ____________________ Instruction available in kernel mode but not user mode.

21. ____________________ The ability to temporarily defer any hardware interrupts.

22. ____________________ A property of a system that does exactly what it is designed to do.

23. ____________________ The ability of software to work across multiple hardware platforms.

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

25. ____________________ When the behavior of a program relies on the interleaving of operations of different threads.

26. ____________________ An OS that provides the abstraction of a virtual machine, to run another OS as an application.

27. ____________________ Suspend execution of a currently-running thread and resume execution of some other thread.

28. ____________________ An efficient spinlock implementation where each waiting thread spins on a separate memory location.

29. ____________________ A thread spins in a loop waiting for a concurrent event to occur, consuming CPU cycles while it is waiting.

30. ____________________ A sequential stream of execution. Also a single execution sequence that represents a separately schedulable task.

31. ____________________ A synchronization variable that enables a thread to efficiently wait for a change to shared state protected by a lock.

32. ____________________ An instruction that atomically reads a value from memory to a register and writes the value 1 to that memory location.

33. ____________________ The execution of an application program with restricted rights - the abstraction for protection provided by the OS kernel.

34. ____________________ The set of threads that are complete but not yet de-allocated, e.g., because a join may read the return value from the TCB.

35. ____________________ The isolation of potentially misbehaving applications and users so that they do not corrupt other applications or the OS itself.

36. ____________________ The kernel is the lowest level of software running on the system, with full access to all of the capabilities of the hardware.

37. ____________________ Extra inter-processor communication required because a single cache entry contains portions of two different data structures with different sharing patterns.

38. ____________________ For a shared object protected by a read-copy-update lock, the time from when a new version of a shared object is published until the last reader of the old version is finished.

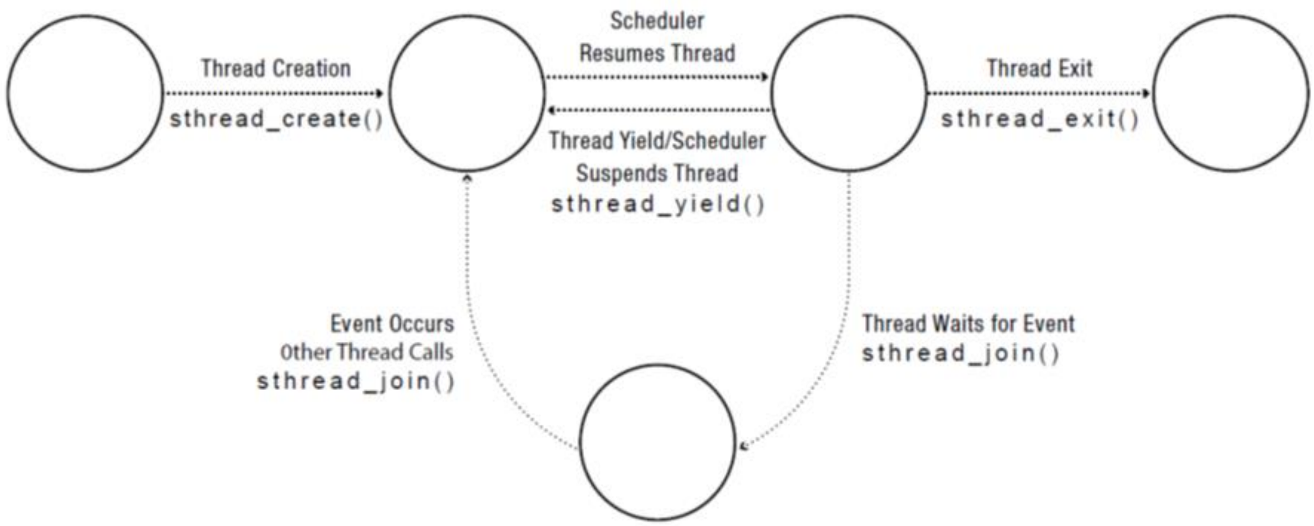
39. ____________________ A lock which allows multiple "reader" threads to access shared data concurrently provided they never modify the shared data, but still provides mutual exclusion whenever a "writer" thread is reading or modifying the shared data.

40. ____________________ An instruction that prevents the compiler and hardware from reordering memory accesses across the barrier - no accesses before the barrier are moved after the barrier and no accesses after the barrier are moved before the barrier.

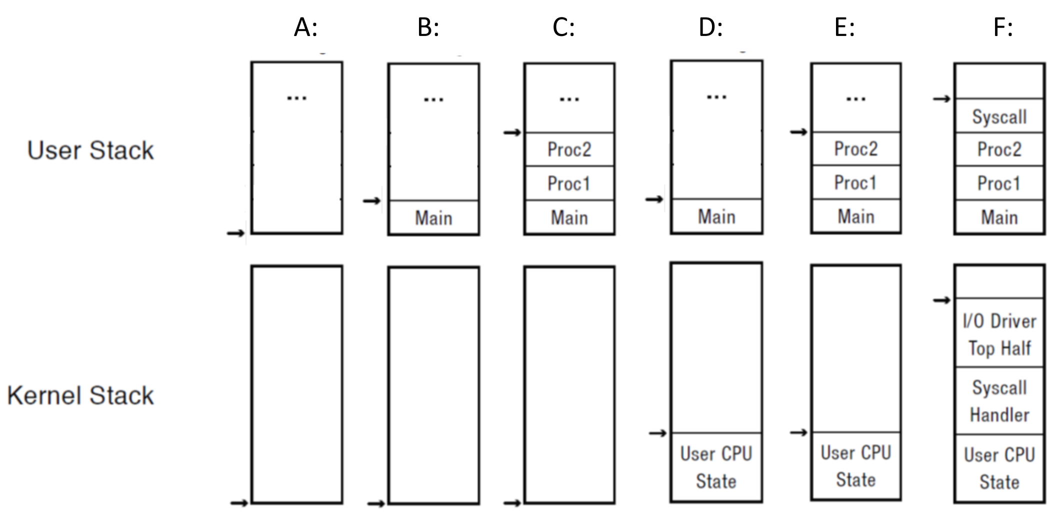
<u>True/False.</u> Circle **only one** of T or F. (1 pt. each)

41. T / F An OS kernel can use internal threads.
42. T / F wait() in UNIX is a condition variable wait operation.
43. T / F Each interrupt handler has its own process control block.
44. T / F Threads are more expensive for an OS to create than processes.
45. T / F UNIX and Linux are examples of the microkernel design approach.
46. T / F When power is turned on, a processor will execute a bootstrap loader in user mode.
47. T / F To provide multiuser protection, hardware must have at least two execution modes.
48. T / F An OS should never create more processes than the available number of processors.
49. T / F An OS provides a single, global waiting list for all the threads that are in the waiting state.
50. T / F When a user attempts to execute a privileged instruction in user mode the CPU should stop.
51. T / F On modern processors, all instructions are atomic. E.g., an x86 string move instruction is atomic.
52. T / F A command interpreter (or shell) is typically extended as a mini-language with control structures.
53. T / F An exception is an asynchronous interrupt, unrelated to the instruction currently being executed.
54. T / F A loadable device driver means that the kernel does not have to be recompiled to use the device.
55. T / F fork() in UNIX creates a new thread, which then executes the function that is passed as an argument.
56. T / F The interrupt vector table should be held in user memory so that users can assign exception handlers.
57. T / F In a typical modern OS, each schedulable unit of execution needs to have a different protection domain.
58. T / F When using fine-grain locking, all the locks should be stored together in a single array in kernel memory.
59. T / F Condition variables will remember how many previous signals and waits have been executed and therefore can be used just like general semaphores.
60. T / F Optimistic concurrency control using the compare and swap instruction allows all updates to succeed if the updates are being applied to different fields in a protected data structure. This approach gives the benefit of fine-grain locking without all the locks.

61. In the following thread state diagram, label the states (i.e., write the labels inside the circles) with the state names: Finished, Init, Ready, Running, and Waiting. (5 pts.)



For each description of a thread in questions 62-65, write the letter of the matching stack diagram, A-F. (1 pt. each)



62. ________ Newly created thread.

63. ________ Running thread that is executing inside Proc2().

64. ________ Waiting thread that has made a system call for input/output.

65. ________ Ready thread that will resume executing Proc2() when next dispatched.

66. A shared counter starts with value 0. Ignoring any compiler or hardware instruction reordering, what is the set of possible final values for the counter when the following three threads are run without mutual exclusion? (3 pts.)

Thread 1	Thread 2	Thread 3
int local;	int local;	int local;
T1S1: local = shared_counter;	T2S1: local = shared_counter;	T3S1: local = shared_counter;
T1S2: local = local + 1;	T2S2: local = local + 1;	T3S2: local = local + 1;
T1S3: shared_counter = local;	T2S3: shared_counter = local;	T3S3: shared_counter = local;

67. Assume you have a library that provides multiprocessor queueing locks (with data type “Lock”) and condition variables (with data type “CV”). Add the appropriate declarations and code to what is shown below (these are the same three threads from question 66) so that the shared data structure is updated with mutual exclusion. (5 pts.)

```
// global data structures

int shared_counter = 0;
```

// thread 1 – increment counter	// thread 2 – increment counter	// thread 3 – increment counter
int local;	int local;	int local;
local = shared_counter;	local = shared_counter;	local = shared_counter;
local = local + 1;	local = local + 1;	local = local + 1;
shared_counter = local;	shared_counter = local;	shared_counter = local;
thread_exit();	thread_exit();	thread_exit();

68. What synchronization actions are missing or performed incorrectly in 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 either missing or incorrect in each version.) (3 pts.)

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

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

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

        itemRemoved.signal();

        return item;
}
```

OR

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

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

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

        full.signal( lock );

        return item;
}
```

69. Other than owner, what are the three synchronization variables for a multiprocessor queueing lock? (3 pts.)

70. What is the synchronization variable for a condition variable? (1 pt.)

71. Below is a list of methods for the Queue data type, the declaration of the Scheduler class from Figure 5.17, and the code for the wait method for the condition variable data type CV from Figure 5.18. Please fill in the code for the signal method for the condition variable data type CV. If you do not know C++, then please use C-like pseudo code for your answer. (3 pts.)

```
// Queue methods
        void add( Thread *thread );
        Thread *remove();
        bool notEmpty();

// …

Class Scheduler{
  private:
        Queue readyList;
        SpinLock schedulerSpinLock;
  public:
        void suspend( Lock *lock );
        void makeReady( Thread *thread );
}

// …

// Monitor lock is held by the current thread, which is identified by Thread *myTCB
void CV::wait( Lock *lock ){
        assert( lock.isHeld() );
        waiting.add( myTCB );
        // switch to new thread and release lock
        scheduler.suspend( &lock );
        // thread will later resume here when called with scheduler.makeReady()
        lock->acquire();
}
```

// YOU FILL IN THE NECESSARY CODE:

```
// Monitor lock is held by the current thread
void CV::signal() {

}
```

Extra Credit Questions. (Up to 2 pts. each.)

XC-1. Why did the Intel x86 **popf** instruction prevent the transparent virtualization of the (old) x86 architecture?

XC-2. Suppose you have to implement an OS on hardware that supports exceptions and traps but does not have interrupts. Can you devise a satisfactory substitute for interrupts? If so, explain how, If not, explain why not.

XC-3. Can UNIX fork() return an error? Why or why not?

XC-4. Give the textbook's definition of a thread-local variable.

XC-5. Why does the textbook say that Peterson's algorithm is not guaranteed to work with modern compilers or hardware?