Please print in pen:

Waterloo Student ID Number:

WatIAM/Quest Login UserID:

UNIVERSITY OF WATERLOO

Midterm Exam - Spring 2026 - ECE 350

1. Before you begin, make certain that you have one **2-sided booklet with 12 pages**. You have **90 minutes** to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question.

2. Please read all of the questions before starting the exam, as some of the questions are substantially more time consuming. Read each question carefully. Make your answers as concise as possible. **If there is something in a question that you believe is open to interpretation, then please write your interpretation and assumptions!**

3. All solutions must be placed in this booklet. If you need more space to complete an answer, you may be writing too much. However, if you need extra space, use the blank space on the last page of the exam clearly labeling the question and indicate that you have done so in the original question.

Good Luck!

Question	Points Assigned	Points Obtained
1	**70**	
2	**30**	
Total	**100**	

Q1. (70 points) True-False with explanation.

For each question:

- Circle your answer and write your explanation below each question.
- Explanations should not exceed 3 sentences.
- One point for correct true-false.
- One point for correct explanation.
- No points for any explanation if true-false is incorrect.

1. For every system call, the kernel validates the user-provided arguments before copying them from user-space memory into kernel memory.

   True False

False: *To avoid time-of-check-to-time-of-use attacks, the kernel must first copy arguments into kernel memory and then validate them. If it validated the data while it still lived in user space, a second user thread could change the values after the check but before use.*

2. Within the same process, one thread can access another thread's stack.

   True False

True: *All threads of a process share a single address space. Each thread's stack lives in that shared space, so a thread holding a pointer into another thread's stack can read or write it. Threads are isolated only by convention, not by hardware.*

3. Within the same process, different threads share the same heap.

   True False

True: *The heap is part of the process's single shared address space, so all threads allocate from and access the same heap (which is why heap allocation must be synchronized).*

4. Context switching between threads that belong to different processes has higher overhead than context switching between threads of the same process.

   True False

True: *Switching between processes requires changing the address space (e.g., loading a new page-table base and flushing or tagging the TLB). Threads in the same process share an address space, so that work is avoided.*

5. System calls allow a user process to directly access the kernel's memory.

   True False

False: *A system call lets a user process request a service from the kernel; it does not grant direct access to kernel memory. Control transfers to a controlled kernel entry point that runs kernel code, and the kernel mediates all access on the process's behalf.*

6. An interrupt handler is a kernel thread that runs at the highest priority.

   True False

False: *Interrupt handlers are not threads and are not scheduled by the scheduler. They run in interrupt context in response to a hardware signal and cannot block like a thread.*

7. Hyperthreading is the functionality that lets multiple threads, each with its own registers and program counter, share the functional units of a single pipeline.

   True False

True: *Hyperthreading (simultaneous multithreading) replicates per-thread architectural state such as registers and the program counter while the threads share the core's execution (functional) units, improving utilization of those units.*

8. An I/O request can be completed without ever being processed by a device driver.

   True False

True: *A request can be satisfied from a cache. For example, a read() whose data is already in the file-system buffer cache returns from the kernel without involving the device driver or the physical device.*

9. With POSIX threads, if the main thread calls exit(), all threads in the process terminate.

   True False

True: *exit() terminates the entire process and therefore all of its threads. (By contrast, pthread_exit() in the main thread would let the other threads keep running.)*

10. A thread's state can change directly from “ready” to “finished.”

    True False

True: *Although a thread normally runs before finishing, a ready thread can be terminated without running again, (e.g., if the process exits or the thread is cancelled while it is still in the ready queue.)*

11. The maximum number of virtual addresses depends on the processor/ISA's address width (the width of the memory bus).

    True False

True: *An n-bit virtual address can name 2^n distinct locations, so the size of the virtual address space is fixed by the virtual-address width.*

12. During the boot process the kernel is loaded into memory by the bootloader.

    True False

True: *The bootloader runs first and loads the kernel image into memory, then transfers control to it.*

13. The command-line arguments passed when a program is invoked are stored at lower addresses in the virtual address space than any variable allocated on the heap.

    True False

False: *Command-line arguments are placed near the top of the user stack, at high addresses — above the heap, so they sit at higher addresses than heap variables, not lower.*

14. All implementations of a mutex on a uniprocessor involve busy waiting.

    True False

False: *On a uniprocessor a mutex can be implemented by briefly disabling interrupts to make the critical section atomic, which involves no busy waiting at all.*

15. A simple test-and-set spinlock on a multiprocessor involves busy waiting.

    True False

True: *Any test-and-set-based lock spins on a shared variable. Even a blocking mutex must spin on a short internal guard (using test-and-set) to protect its own data structures, so some busy waiting cannot be eliminated when test-and-set is used.*

16. Immediately after a thread calls thread_yield(), another thread always starts running.

    True False

False: *If the ready queue is empty, the scheduler simply reschedules the same thread, so no other thread runs.*

17. A thread can always enforce mutual exclusion by disabling interrupts on the processor it is running on.

    True False

False: *Disabling interrupts only stops preemption on the local processor. On a multiprocessor, threads on other processors keep running and can enter the critical section, so it does not guarantee mutual exclusion.*

18. Without atomic read-modify-write instructions, mutual exclusion cannot be implemented on multiprocessors.

    True False

False: *Software-only algorithms such as Peterson's algorithm achieve mutual exclusion using ordinary loads and stores. (On real hardware they also require memory ordering, e.g., fences, but no atomic read-modify-write instruction is strictly necessary.)*

19. A thread created by the thread_create() system call is guaranteed to run.

    True False

False: *The new thread is only placed on the ready queue. If the process terminates first (for example, its initial thread exits), all of its threads are destroyed, and the new thread may never run.*

20. A user-space thread running inside a mutex-protected critical section cannot be context-switched.

    True False

False: *A context switch is still possible. A mutex usable by user-space threads must not disable interrupts, so the thread can be preempted (e.g., by a timer interrupt) while holding the lock.*

21. On a uniprocessor, first-come-first-served (FCFS) scheduling minimizes average response time when all tasks have equal length.

    True False

True: *When all tasks are the same length, reordering them yields no benefit, and FCFS coincides with shortest-job-first, which minimizes average response time. (FCFS is work-conserving, so it never idles unnecessarily.)*

22. For a workload on a uniprocessor, scheduler A achieves lower average response time than scheduler B. Therefore, scheduler A must have higher throughput than scheduler B on that workload.

    True False

False: *Throughput and average response time are not directly related. For a fixed batch of jobs, throughput depends on when the last job completes (the makespan), which can be identical for two schedulers even though their average response times differ.*

23. Interrupt handlers can use spinlocks.

    True False

True: *Interrupt handlers may use spinlocks because acquiring a spinlock does not block the thread. (They must not use blocking locks, which could deadlock since a handler cannot sleep.)*

24. A thread can wait on multiple condition variables simultaneously.

    True False

False: *Once a thread calls wait() on the first condition variable it is suspended, so it cannot reach a second wait() call. A thread can wait on only one condition variable at a time.*

25. For a set of tasks with fixed execution times on a uniprocessor with zero context-switch overhead, the throughput of a work-conserving scheduler could be worse than that of a non-work-conserving scheduler.

    True False

False: *A work-conserving scheduler never leaves the CPU idle while runnable work exists, so it finishes the same total work no later than a non-work-conserving scheduler. Therefore, its throughput cannot be worse.*

26. On a uniprocessor, a user-space library can implement its own mutex so that locking the mutex does not require a mode switch into the kernel.

    True False

True: *Using an atomic read-modify-write instruction, an uncontended lock can be acquired entirely in user space; control only needs to enter the kernel when the thread must block.*

27. Assuming there is no context-switching overhead, under round-robin (RR) scheduling, changing the time quantum does not affect the response time of the longest task.

    True False

False: *If the time quantum is larger than a task's length, RR degenerates toward FCFS, so the order of the tasks (and hence the longest task's response time) depends on the quantum. Changing the quantum can therefore change the longest task's response time.*

28. For a fixed number of processes on a uniprocessor, improving throughput will improve the response time of at least one process.

    True False

True: *Suppose response times stayed the same or increased for every task. Then the completion time of the last-finishing task would not improve, so throughput (work completed per unit time) could not improve, a contradiction. Hence improving throughput must improve at least one task's response time.*

29. Lottery scheduling becomes RR when all tasks have the same number of tickets.

    True False

False: *Even with equal tickets, lottery scheduling still picks the next task by a random draw, whereas RR cycles through tasks deterministically. Equal expected CPU share does not make the two schedules identical.*

30. To improve average response time with lottery scheduling, we can give each job a number of tickets proportional to its runtime.

    True False

False: *To lower average response time, short jobs should be favored, so they should receive more tickets, not fewer. Giving tickets in proportion to runtime favors long jobs and worsens average response time.*

31. Assuming there is no context-switching overhead, first-come-first-served (FCFS) scheduling is always better than RR for average response time.

    True False

False: *Consider one very long job that arrives first plus several short jobs. FCFS makes all the short jobs wait behind the long one, while RR with a short quantum lets the short jobs finish quickly, achieving a lower average response time. So FCFS is not always better.*

32. In stride scheduling, a task with a lower stride is given more CPU time than a task with a higher stride.

    True False

True: *Stride is inversely proportional to a task's ticket count, and the task with the smallest pass value runs next. A lower stride means the task's pass advances slowly, so it is selected more often and receives more CPU time.*

33. In lottery scheduling, the ratio of the response times of two tasks with the same number of tickets and the same length, arriving at the same time, is close to one.

    True False

False: *Lottery scheduling is randomized, so over short runs the two equally weighted tasks can finish at quite different times; the ratio of their response times need not be close to one. Lottery scheduling can be unfair, especially to short tasks.*

34. Given a fixed tie-breaking rule, weighted max-min-fair scheduling and stride scheduling produce the same schedule when tickets are assigned in proportion to weights and the stride-scheduling time quantum equals the target latency in weighted max-min fairness.

    True False

False: *Both policies converge to the same proportional-fair allocation over time, but they do not produce the same moment-to-moment schedule: weighted max-min fairness effectively gives each task its own time quantum, so it interleaves tasks differently than stride scheduling does.*

35. Lottery scheduling can prevent CPU starvation by giving at least one ticket to each task.

    True False

True: *As long as every task holds at least one ticket, it has a nonzero probability of winning each draw, so (probabilistically) it will eventually be scheduled and cannot starve forever.*

Q2 (30 points) Multiple Select

1. (3 points) Select the thread attributes that require hardware support to be saved safely during a user-to-kernel mode switch in x86 systems.

    ☒ Program counter (PC)
    ☒ EFLAGS register
    ☐ File descriptors
    ☐ General-purpose registers
    ☐ Memory map
    ☒ Stack pointer (SP)

2. (2 points) Which of the following may bring the processor from kernel mode to user mode? (select all that apply)

    ☒ A new process or thread begins executing
    ☒ A user-level upcall
    ☐ An interrupt, exception, or system call
    ☒ A process/thread context switch that resumes a user thread

3. (3 points) Select all correct statements about the virtual address-space layout of a typical C program.

☒ The stack grows from higher addresses toward lower addresses.
☐ The heap starts at address zero.
☐ Arrays are always stored on the heap.
☒ The stack and heap grow in opposite directions.
☐ The space between the stack and heap is always kept empty.
☒ Local variables of a function are stored on the stack.

4. (2 points) A thread that calls which of the following could continue running without being context-switched? (select all that apply)

☒ pthread_create()
☒ pthread_yield()
☐ pthread_exit()
☒ pthread_join()

5. (2 points) On a processor that does not support interrupts, which of the following cannot happen or work properly? (select all that apply)

☐ Programmed I/O
☒ Direct memory access (DMA)
☐ Voluntary mode switch
☒ Involuntary mode switch

6. (2 points) What are the disadvantages of protecting a critical section by disabling interrupts? (select all that apply)

☒ User-level code cannot use it.
☒ It can delay the handling of time-critical events.
☐ It wastes CPU cycles.
☒ It does not work well on multiprocessor systems.

7. (2 points) Which of the following statements about semaphores are INCORRECT? (select all that apply)

☒ A semaphore can be initialized to any value between −2^31 and 2^31 − 1.
☒ Semaphores cannot be used by interrupt handlers.
☐ A condition variable can be implemented with a single semaphore.
☐ A mutex can be implemented with a single semaphore.

8. (3 points) Which of the following statements about Mesa monitors are correct? (select all that apply)

☒ A thread that calls signal() keeps the mutex locked and does not yield the processor.
☐ A thread should call wait() inside an if statement.
☒ A thread that wakes up inside wait() is suspended again before returning if another thread holds the mutex.
☒ The thread that returns from wait() holds the mutex.
☐ A thread placed on the ready queue by a signal always runs immediately after the signaler is context-switched.
☒ The mutex is released after the thread calling wait() is placed on the waiting queue.

9. (3 points) Which of the following are advantages of FCFS over RR? (assume nonzero context-switching overhead; select all that apply)

☒ It achieves better or equal throughput.
☒ It achieves lower or equal context-switching overhead.
☒ It achieves better cache reuse by running each job to completion.

10. (2 points) Two threads run simultaneously with the program orders shown below. Which of the following global execution orders are consistent with sequential consistency? (select all that apply)

Thread A (CPU 1)	Thread B (CPU 2)
Load1	Load4
Load2	Store1
Load3	Load5

☒ Load1, Load2, Load3, Load4, Store1, Load5
☒ Load1, Load4, Load2, Store1, Load3, Load5
☒ Load4, Store1, Load5, Load1, Load2, Load3
☐ Load1, Store1, Load2, Load3, Load4, Load5

11. (6 points) Consider the single-threaded processes in the table below. Select all correct options. (For RR, at the end of each quantum the running process is placed at the back of the ready queue first, and only then are any newly arrived processes appended.)

Process	Arrival time	Burst time (s)
P1	0	2
P2	1	6
P3	4	1
P4	7	4
P5	8	3

☐ Average response time of FCFS = 5.2.
☒ Average response time of SRTF = 4.8.
☐ Average response time of RR with quantum 1s = 6.4.