Review of Final Exam for COP 4610 Operating Systems - Fall 2025

Name: _____ **Student ID:** _____

Score: _____

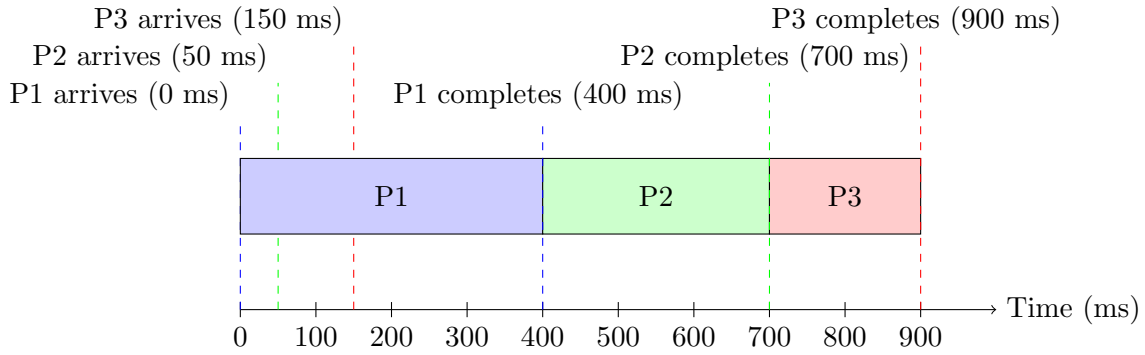
Assignment 7: CPU Scheduling

1. (1 point) Suppose we have three processes:

Process ID	Required CPU Time	Arrival Time
1	400 msec	0 msec
2	300 msec	50 msec
3	200 msec	150 msec

For the FIFO scheduling, what is the turnaround time of process 3? Answer: 750

FIFO Scheduling Diagram:



2. (1 point) For the FIFO scheduling, what is the wait time of process 1?

Answer: 0

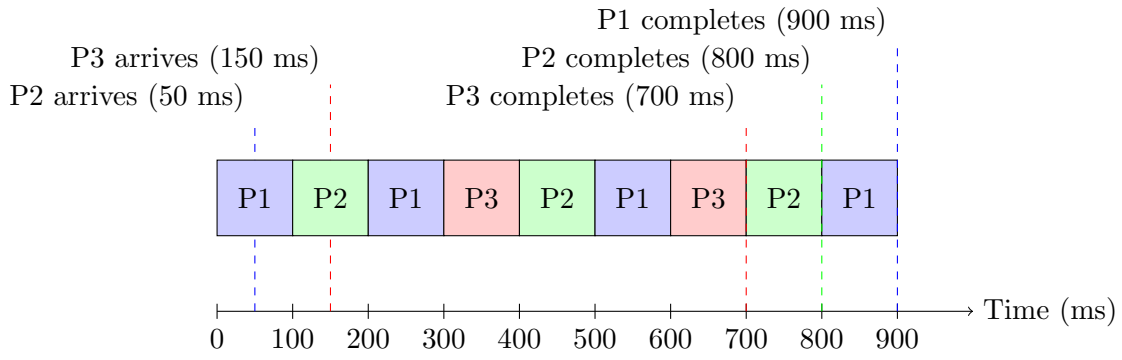
3. (1 point) For the FIFO scheduling, what is the wait time of process 3?

Answer: 550

4. (1 point) Assume the time slice of 100 msec and a zero context-switching cost. For the round-robin scheduling, what is the wait time of process 3?

Answer: 350

Round Robin Scheduling Diagram:



Here is the breakdown of the sequence:

- At 49 ms, Process 1 is running.

- At 50 ms, Process 2 arrives, so the RR queue becomes [2].
- At 100 ms, Process 1 finishes its time slice, and the RR queue becomes [2, 1].
- At 100 ms, Process 2 starts running, and the RR queue is [1].
- At 150 ms, Process 3 arrives, so the RR queue becomes [1, 3].
- At 200 ms, Process 2 finishes its time slice, and the RR queue is [1, 3, 2].
- At 200 ms, Process 1 starts running, and the RR queue is [3, 2].
- At 300 ms, Process 1 finishes its time slice, and Process 3 starts running, with the RR queue now being [2, 1].
- At 400 ms, Process 3 finishes its time slice, and Process 2 starts running, with the RR queue being [1, 3].
- At 500 ms, Process 2 finishes its time slice, and Process 1 starts running, with the RR queue being [3, 2].
- At 600 ms, Process 1 finishes its time slice, and Process 3 starts running, with the RR queue being [2, 1].
- At 700 ms, Process 3 finishes and terminates. Process 2 starts running, with the RR queue being [1].

Thus, the wait time for Process 3 is calculated as:

$$\text{Wait time for Process 3} = (300 - 150) + (600 - 400) = 350 \text{ ms}$$

5. (1 point) Assume the time slice of 100 msec and a zero context-switching cost. For the round-robin scheduling, what is the response time of process 2?

   Answer: 50

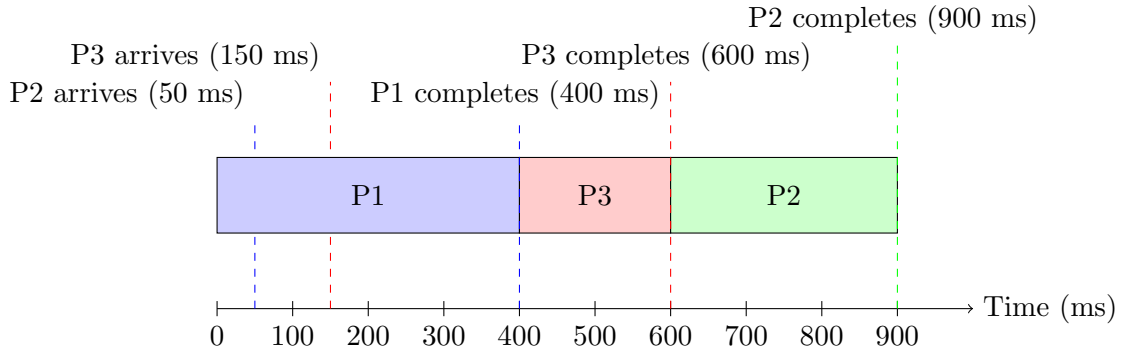
6. (1 point) Assume the time slice of 100 msec and a zero context-switching cost. For the round-robin scheduling, what is the turnaround time of process 1?

   Answer: 900

7. (1 point) For the SJF scheduling, what is the turnaround time of process 1?

   Answer: 400

   SJF Scheduling Diagram:



8. (1 point) For the SJF scheduling, what is the response time of process 2?

   Answer: 550

   Do not confuse SJF (Shortest Job First) with SRJF, also called SRTF (Shortest Remaining Time First).

   A common misunderstanding is the following: at 50 msec, Process 1 has 350 msec remaining and Process 2 requires 300 msec, so Process 2 is shorter and should start running as soon as it arrives. Under this assumption, Process 2 would have a response time of 0. This reasoning actually follows the SRJF(SRTF) rule, where the scheduler always chooses the process with the shortest remaining CPU time and may preempt the running process.

   For non-preemptive SJF, once a process starts running it cannot be interrupted. The scheduler only makes a choice when the CPU becomes idle, and it uses the original total CPU burst times to decide which job is

shortest. In this example, Process 1 runs from 0 to 400 msec. At 400 msec, both Process 2 (300 msec) and Process 3 (200 msec) are waiting, so SJF selects Process 3 first and Process 2 runs last. As a result, the response time of Process 2 is 600 - 50 = 550 msec.

Execution:

- Process 1 runs from 0 to 400 msec.
- Process 3 (shorter) runs from 400 to 600 msec.
- Process 2 starts at 600 msec and finishes at 900 msec.

Response Time for Process 2:

$$\text{Turnaround time for Process 2} = 400\,\text{msec} + 200\,\text{msec} - 50\,\text{msec} = 550\,\text{msec}$$

9. (1 point) For the SJF scheduling, what is the turnaround time of process 2?

   Answer: 850

   For SJF scheduling, since it is non-preemptive, once a process starts running, it cannot be interrupted (unlike SRTF).

   Execution:

   - Process 1 runs from 0 to 400 msec.
   - Process 3 (shorter) runs from 400 to 600 msec.
   - Process 2 starts at 600 msec and finishes at 900 msec.

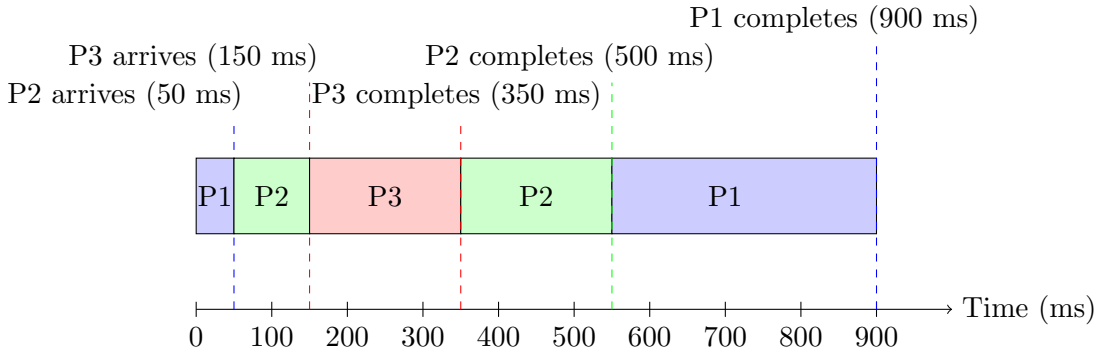
   Turnaround Time for Process 2:

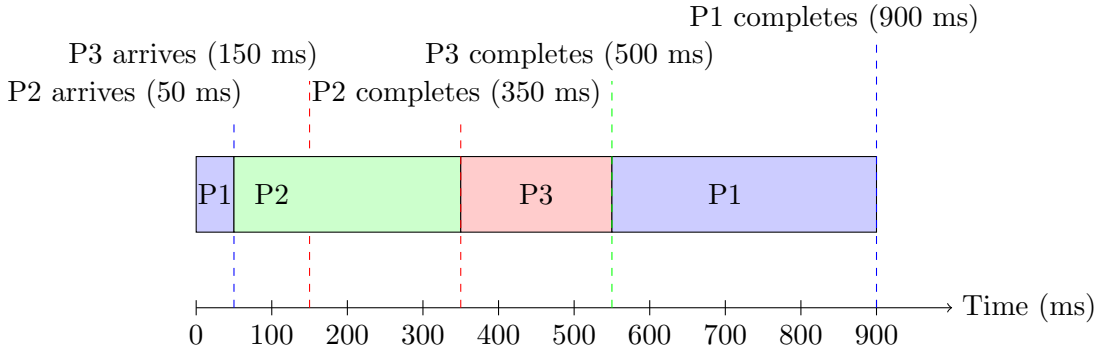
   $$\text{Turnaround time for Process 2} = 900\,\text{msec} - 50\,\text{msec} = 850\,\text{msec}$$

10. (1 point) For the SRTF scheduling, what is the wait time of process 3?

    Answer: 200 or 0

    SRTF Scheduling Diagram:





11. (1 point) For the SRTF scheduling, what is the wait time of process 1?

    Answer: 500

12. (1 point) For the SRTF scheduling, what is the wait time of process 2?

    Answer: 200 or 0

    As you work through the SRTF (Shortest Remaining Time First) scheduling problem, you might encounter a situation where both Process 2 and Process 3 have equal remaining time. The question then arises: which one should you select?

    In real-world scenarios, typically Process 2 would continue running since it's already in execution. This approach minimizes the cost of a context switch. However, in our exercises, we assume that the context switch cost is zero. Therefore, you can choose either process, and both answers are considered valid.

    This means that for some questions, there may be two correct answers, depending on which process you choose to continue.

13. (4 points) Threads A and B are morning routines. Both threads share variables 'location' and 'action'.

    location = bedroom

    action = sleeping

```
// Thread A
1. enter the washroom
2. location = washroom
3. action = taking a break
4. enter the dining room
5. location = dining room
6. action = eating
```

```
// Thread B
7. enter the washroom
8. location = washroom
9. action = taking a break
10. enter the dining room
11. location = dining room
12. action = eating
```

    Assume that context switches can occur at any time. Is it possible that the shared variables can reach a state where 'location = washroom' and 'action = eating'? Please identify a sequence or sequences of operations that will lead to the embarrassing state.

    - ☒ 1, 7-12, 2
    - ☒ 1-5, 7-8, 6
    - ☒ 1-6, 7-8
    - ☐ 7-8, 1-6
    - ☐ It is not possible.
    - ☐ 1, 7, 2, 8, 3, 9, 4, 10, 5, 11, 6, 12

14. (2 points) Assuming both threads are running in the same address space, sharing variables 'x' and 'y', what are the possible final value pairs of 'x' and 'y'? Select all that apply.

```
// Thread A
x = 1;
x = x + 1;
```

```
// Thread B
y = 1;
y = x + 1;
```

    - ☐ (3, 2)
    - ☐ (1, 2)
    - ☒ (2, 3)
    - ☒ (2, 2)

Assignment 8: Thread

1. (2 points) A person uses a computer with a second-generation Intel i7 processor that has 2 physical cores and 4 logical threads to run a program that performs intensive calculations and requires relatively little memory access. Initially, they run the program with 1 thread, and it takes 60 seconds to complete. Then, they run the program with:

    2 threads: it finishes in 30 seconds.

4 threads: it completes in 18 seconds.

8 threads: it only improves slightly, finishing in 16 seconds.

At first, increasing the thread count improved the execution time significantly, but the performance gains diminished when increasing the number of threads from 4 to 8. What is the most likely reason for this?

☐ There is a bug in the program that causes poor performance with more than 4 threads.

☐ The processor's cache is too small to support 8 threads efficiently.

☒ The CPU has only 2 physical cores and 4 logical threads, limiting the performance improvement with 8 threads.

☐ The operating system cannot handle more than 4 threads.

2. (4 points) Suppose variables x and y are both initially set to 0. Two threads run concurrently without any synchronization mechanisms:

```
// Thread A
x = 2;
x = x * y;
x = y + 1;
```

```
// Thread B
y = 1;
y = y + 2;
y = y * 2;
```

After both threads have executed, which of the following results are possible? (Select all that apply.)

☒ x = 1, y = 6 ☒ x = 4, y = 6 ☐ x = 8, y = 6

☒ x = 7, y = 6 ☐ x = 3, y = 6 ☒ x = 2, y = 6

3. (2 points) Which of the following results occurs with the highest probability, given the same code from the previous question?

☐ x = 1, y = 6 ☒ x = 7, y = 6 ☐ x = 2, y = 6

☐ x = 8, y = 6 ☐ x = 4, y = 6 ☐ x = 3, y = 6

Explanation: There are 20 possible execution sequences when interleaving the instructions from both threads while maintaining the order within each thread. After analyzing all possible sequences:

x = 1, y = 6 occurs in 1 sequence.
x = 2, y = 6 occurs in 3 sequences.
x = 4, y = 6 occurs in 6 sequences.
x = 7, y = 6 occurs in 10 sequences.
Therefore, the result x = 7, y = 6 occurs with the highest probability.

4. (1 point) In multithreaded programming, race conditions are a common issue. For example, using 'x++' to increment a global variable 'x' may lead to incorrect results because the operation is not atomic (it involves load, compute, and store steps). Meanwhile, 'printf' is a more complex operation than 'x++'. In a multithreaded program where two threads print the characters 'a' and 'b' respectively, which of the following statements is correct?

☐ The program's output may contain unexpected or corrupted characters because 'printf' is a complex operation, and calling it from multiple threads may lead to unpredictable behavior.

☒ The program's output will contain only 'a' and 'b' without any errors or unexpected characters, because the operating system treats 'printf' as thread-safe and ensures there is no data corruption.

5. (4 points) Given the code `ins.c` at path `in-class-code/14_model/independent-stack/ins.c` (`https://github.com/FSU-COP4610-F25/in-class-code`). Compile and run it, then change `#define N` to 2, 4, 8, 16, 32, and 64. Observe the approximate stack size per thread.

☐ The per-thread stack size is fixed and cannot be changed.

☒ Each thread's stack size is roughly the same for all threads, typically about 8 MiB on Linux.

☐ As N increases, total virtual memory is fixed so each thread's stack size automatically becomes smaller.

☐ Each thread has its own stack, but stacks are shareable between threads.

6. (20 points) In `solution.json`, write two schedules: one with **finalSum = 5** and one with **finalSum = 2**.

Assumptions and format

- Single CPU; steps are serialized; total **27** steps from the read-modify-write pattern.
- Threads: `T1`, `T2`, `T3`.
- Actions (spell exactly): `load k`, `compute k->k+1`, `store k`.
- Each step is an object with fields: `time` (`1..27`), `thread`, `action`, `sum` (global sum after this step).
- Keep *spelling and capitalization* exactly as in the example; do not change anything else.

Answer:

Schedule with `finalSum = 5`


```
{
  "finalSum": 5,
  "steps": [
    { "time":  1, "thread": "T1", "action": "load 0",        "sum": 0 },
    { "time":  2, "thread": "T2", "action": "load 0",        "sum": 0 },
    { "time":  3, "thread": "T3", "action": "load 0",        "sum": 0 },
    { "time":  4, "thread": "T1", "action": "compute 0->1",  "sum": 0 },
    { "time":  5, "thread": "T2", "action": "compute 0->1",  "sum": 0 },
    { "time":  6, "thread": "T3", "action": "compute 0->1",  "sum": 0 },
    { "time":  7, "thread": "T1", "action": "store 1",       "sum": 1 },
    { "time":  8, "thread": "T2", "action": "store 1",       "sum": 1 },
    { "time":  9, "thread": "T3", "action": "store 1",       "sum": 1 },
    { "time": 10, "thread": "T1", "action": "load 1",        "sum": 1 },
    { "time": 11, "thread": "T2", "action": "load 1",        "sum": 1 },
    { "time": 12, "thread": "T1", "action": "compute 1->2",  "sum": 1 },
    { "time": 13, "thread": "T2", "action": "compute 1->2",  "sum": 1 },
    { "time": 14, "thread": "T1", "action": "store 2",       "sum": 2 },
    { "time": 15, "thread": "T2", "action": "store 2",       "sum": 2 },
    { "time": 16, "thread": "T3", "action": "load 2",        "sum": 2 },
    { "time": 17, "thread": "T3", "action": "compute 2->3",  "sum": 2 },
    { "time": 18, "thread": "T3", "action": "store 3",       "sum": 3 },
    { "time": 19, "thread": "T1", "action": "load 3",        "sum": 3 },
    { "time": 20, "thread": "T1", "action": "compute 3->4",  "sum": 3 },
    { "time": 21, "thread": "T1", "action": "store 4",       "sum": 4 },
    { "time": 22, "thread": "T2", "action": "load 4",        "sum": 4 },
    { "time": 23, "thread": "T3", "action": "load 4",        "sum": 4 },
    { "time": 24, "thread": "T2", "action": "compute 4->5",  "sum": 4 },
    { "time": 25, "thread": "T3", "action": "compute 4->5",  "sum": 4 },
    { "time": 26, "thread": "T2", "action": "store 5",       "sum": 5 },
    { "time": 27, "thread": "T3", "action": "store 5",       "sum": 5 }
  ]
}
```

Schedule with `finalSum = 2`


```
{
  "finalSum": 2,
  "steps": [
    { "time":  1, "thread": "T1", "action": "load 0",        "sum": 0 },
    { "time":  2, "thread": "T2", "action": "load 0",        "sum": 0 },
    { "time":  3, "thread": "T3", "action": "load 0",        "sum": 0 },
    { "time":  4, "thread": "T3", "action": "compute 0->1",  "sum": 0 },
    { "time":  5, "thread": "T3", "action": "store 1",       "sum": 1 },
    { "time":  6, "thread": "T3", "action": "load 1",        "sum": 1 },
    { "time":  7, "thread": "T3", "action": "compute 1->2",  "sum": 1 },
    { "time":  8, "thread": "T3", "action": "store 2",       "sum": 2 },
    { "time":  9, "thread": "T1", "action": "compute 0->1",  "sum": 2 },
    { "time": 10, "thread": "T1", "action": "store 1",       "sum": 1 },
    { "time": 11, "thread": "T3", "action": "load 1",        "sum": 1 },
    { "time": 12, "thread": "T3", "action": "compute 1->2",  "sum": 1 },
    { "time": 13, "thread": "T2", "action": "compute 0->1",  "sum": 1 },
    { "time": 14, "thread": "T2", "action": "store 1",       "sum": 1 },
    { "time": 15, "thread": "T1", "action": "load 1",        "sum": 1 },
    { "time": 16, "thread": "T1", "action": "compute 1->2",  "sum": 1 },
    { "time": 17, "thread": "T1", "action": "store 2",       "sum": 2 },
    { "time": 18, "thread": "T1", "action": "load 2",        "sum": 2 },
    { "time": 19, "thread": "T1", "action": "compute 2->3",  "sum": 2 },
    { "time": 20, "thread": "T1", "action": "store 3",       "sum": 3 },
    { "time": 21, "thread": "T2", "action": "load 3",        "sum": 3 },
    { "time": 22, "thread": "T2", "action": "compute 3->4",  "sum": 3 },
    { "time": 23, "thread": "T2", "action": "store 4",       "sum": 4 },
    { "time": 24, "thread": "T2", "action": "load 4",        "sum": 4 },
    { "time": 25, "thread": "T2", "action": "compute 4->5",  "sum": 4 },
    { "time": 26, "thread": "T2", "action": "store 5",       "sum": 5 },
    { "time": 27, "thread": "T3", "action": "store 2",       "sum": 2 }
  ]
}
```


Assignment 9: Mutual Exclusion

1. (6 points) Match the terms on the right with the definitions on the left.

Definition	Term
1. __A__ An all or nothing operation	A. atomic operation
2. __B__ Results depend on the timing of executions	B. race condition
3. __D__ Using atomic operations to ensure cooperation among threads	C. mutual exclusion
4. __C__ Ensuring one thread can do something without the interference of other threads	D. synchronization
5. __E__ A piece of code that only one thread can execute at a time	E. critical section
6. __F__ Consuming CPU time while waiting	F. busy waiting

2. (1 point) Consider a global variable ‘x‘ initialized to 0. Two threads are each tasked with incrementing ‘x‘ one million times. Four different implementations are proposed to perform the increment operation. After both threads have completed their execution, which implementation will consistently result in ‘x‘ being exactly 2,000,000?

   ☐ Using the assembly instruction ‘asm volatile(”addq $1, %0” : ”+m”(x));‘ to increment ‘x‘ (without any locking mechanism).

☒ Using the assembly instruction ‘asm volatile(”lock addq $1, %0” : ”+m”(x));‘ to increment ‘x‘ (with the ‘lock‘ prefix to ensure atomicity).

☐ Using ‘x++‘ to increment ‘x‘ in each thread.

☐ Implementing a simple lock by using a shared flag variable to protect ‘x++‘, where each thread checks the flag before proceeding with the increment (without proper atomic operations or memory barriers).

3. (1 point) Consider a global variable ‘x‘ initialized to 0. Two threads are each tasked with incrementing ‘x‘ one million times. Four different implementations are proposed to perform the increment operation. After both threads have completed their execution, which implementation will take the longest time to complete?

   ☐ Using the assembly instruction ‘asm volatile(”addq $1, %0” : ”+m”(x));‘ to increment ‘x‘ (without any locking mechanism).

   ☐ Implementing a simple lock by using a shared flag variable to protect ‘x++‘, where each thread checks the flag before proceeding with the increment (without proper atomic operations or memory barriers).

   ☒ Using the assembly instruction ‘asm volatile(”lock addq $1, %0” : ”+m”(x));‘ to increment ‘x‘ (with the ‘lock‘ prefix to ensure atomicity).

   ☐ Using ‘x++‘ to increment ‘x‘ in each thread.

4. (4 points) Two threads each increment a global variable x one million times. Four implementations without using any advanced locking mechanisms are proposed for the increment operation. Which implementation will take the shortest time? You can verify the result by editing our lecture code /in-class-code/15_exclusion/sum-atomic/sum.c (`https://github.com/FSU-COP4610-F25/in-class-code`)

   ☐ asm volatile(”lock addq $1, %0”)

   ☐ asm volatile(”addq $1, %0”)

   ☐ x++

   ☒ atomic_fetch_add(&x, 1)

5. (1 point) Why is the ”acquire_lock()” implementation shown below potentially problematic in a multi-core system?

```
int lock = 0;
void acquire_lock(int *lock) {
    while (*lock != 0) {}
    *lock = 1;
}
void release_lock(int *lock) {
    *lock = 0;
}

void *foo(void *arg) {
    acquire_lock(&lock);
    // Critical section: Do work here ...
    release_lock(&lock);
    return NULL;
}
```

   ☒ It allows race conditions because multiple threads can modify the lock without atomic operations.

   ☐ It uses too many atomic operations, which decreases performance.

   ☐ It prevents threads from releasing the lock once they acquire it.

   ☐ It always acquires the lock without checking if the lock is available.

6. (1 point) Which of the following statements about spinlocks is correct?

☐ Spinlocks are more efficient than mutexes in all scenarios.

☒ A spinlock uses busy waiting, meaning a thread repeatedly checks the lock without releasing the CPU.

☐ Spinlocks can be used to avoid busy waiting.

☐ A spinlock allows multiple threads to access the critical section simultaneously.

7. (1 point) What is the main benefit of using the ‘xchg‘ instruction in the ‘acquire_lock()‘ implementation shown below?

```
int lock = 0;
int xchg(int *addr, int newval) {
    int result;
    asm volatile (
    "lock xchg %0, %1"
    : "+m" (*addr), "=a" (result)
    : "1" (newval)
    : "cc"
    );
    return result;
}

void acquire_lock(int *lock) {
    while (xchg(lock, 1)) {}
}

void release_lock(int *lock) {
    xchg(lock, 0);
}

void *foo(void *arg) {
    acquire_lock(&lock);
    // Critical section: Do work here ...
    release_lock(&lock);
    return NULL;
}
```

☐ It allows multiple threads to enter the critical section at once.

☐ It ensures that the lock is released automatically after a fixed time.

☒ It performs the lock acquisition as an atomic operation, preventing race conditions.

☐ It allows the lock to be acquired without the use of hardware support.

8. (1 point) What does the ‘xchg‘ instruction do in the context of acquiring a lock?

☒ It swaps the values of the lock and the result atomically, ensuring no other thread can modify the lock during this operation.

☐ It checks if the lock is available and waits until it is free without swapping any values.

☐ It releases the lock by resetting the value to 0.

☐ It increments the value of the lock each time it is called.

9. (1 point) What happens to threads waiting for a mutex when the lock is already held by another thread?

☐ They continue busy-waiting until the lock is released.

☒ They enter a blocked state and the OS puts them to sleep until the lock becomes available.

☐ They are terminated by the OS to prevent deadlocks.

☐ They use atomic instructions to steal the lock from the current holder.

10. (1 point) What is a futex and how does it improve performance compared to a spinlock?

    ☐ A futex allows multiple threads to hold the lock at the same time.

    ☐ A futex is a special type of mutex that always uses the kernel to manage thread sleep and wake-up.

    ☒ A futex combines the advantages of spinlocks and mutexes by starting with spinning and escalating to a kernel-based mutex.

    ☐ A futex is a user-space only lock that prevents any kernel intervention.

11. (1 point) In a mutex-based system, how does the OS manage threads waiting for a lock?

    ☐ The OS puts all waiting threads in a busy-wait loop until the lock is released.

    ☐ The OS allows only the highest-priority thread to acquire the lock.

    ☒ The OS puts the threads into a blocked (sleep) state and wakes them up when the lock is released.

    ☐ The OS continually switches between all waiting threads, giving each a chance to acquire the lock.

12. (1 point) In a scenario where three threads (X, Y, and Z) are waiting for a mutex, what typically happens when the lock is released by Thread X?

    ☐ Both Thread Y and Thread Z are woken up, and they compete for the lock.

    ☐ Thread Z is woken up first if it has a higher priority than Thread Y.

    ☐ The OS randomly selects one of the waiting threads (Y or Z) to wake up.

    ☒ The OS wakes up Thread Y, following a first-come, first-served (FIFO) or priority-based policy.

13. (1 point) Why is the futex mechanism often more efficient than using pure spinlocks or pure mutexes?

    ☒ It uses a combination of spinning and sleeping, reducing busy-waiting and unnecessary context switches.

    ☐ It allows threads to share the lock when there is high contention.

    ☐ It automatically prioritizes the most important threads.

    ☐ It prevents context switches entirely.

Assignment 10: Synchronization and Bugs

1. (1 point) What is the primary goal of the Producer-Consumer problem?

    ☐ To ensure that consumers consume data faster than producers can produce.

    ☐ To avoid the need for condition variables or semaphores.

    ☒ To ensure synchronization between producers and consumers, so that the buffer doesn't overflow or underflow.

    ☐ To ensure that producers produce data as fast as possible, regardless of the buffer size.

2. (1 point) What is the major disadvantage of using a single condition variable in the Producer-Consumer problem?

    ☐ It allows both producers and consumers to operate simultaneously without synchronization.

    ☐ It improves performance by reducing the number of context switches.

    ☒ It can lead to deadlock if the same type of thread (producer or consumer) is repeatedly woken up.

    ☐ It makes the program easier to debug.

3. (1 point) Why does busy-waiting in the implementation of the Producer-Consumer problem shown below waste CPU resources?

```
void *Tproduce(void *arg) {
    while (1) {
        retry:
        pthread_mutex_lock(&lk);
        if (count == n) {
            pthread_mutex_unlock(&lk);
            goto retry;
        }
        count++;
        printf("O");
        pthread_mutex_unlock(&lk);
    }
    return NULL;
}

void *Tconsume(void *arg) {
    while (1) {
        retry:
        pthread_mutex_lock(&lk);
        if (count == 0) {
            pthread_mutex_unlock(&lk);
            goto retry;
        }
        count--;
        printf("X");
        pthread_mutex_unlock(&lk);
    }
    return NULL;
}
```

☐ Because it uses semaphores incorrectly.

☐ Because it only works on single-core systems.

☐ Because it doesn't allow threads to enter the critical section simultaneously.

☒ Because it repeatedly checks for a condition without yielding the CPU, even when conditions are not met.

4. (1 point) How do condition variables solve the busy-waiting problem in the Producer-Consumer problem?

☐ They guarantee that producers and consumers work in parallel.

☐ They eliminate the need for synchronization entirely.

☒ They allow threads to wait until a condition is met without consuming CPU resources.

☐ They prevent multiple threads from entering the critical section.

5. (1 point) In the following scenario with a buffer size of n=1:

The buffer is initially empty, and both consumers (C1, C2) are sleeping. Producer P2 is sleeping because the buffer was previously full. Producer P1 produces an item, filling the buffer, and signals a 'buffer change' event. The signal wakes up P2, but P2 finds the buffer still full, so P2 goes back to sleep without signaling. The OS schedules P1 again, but P1 also finds the buffer full and goes back to sleep without signaling.

After P1 is rescheduled by the OS and goes back to sleep without signaling, which thread will the OS likely schedule next, and can this thread signal the waiting consumers (C1, C2)? Why or why not?

☐ P2 will be scheduled and will signal the waiting consumers (C1, C2) even if it cannot proceed itself.

☐ One of the consumers (C1 or C2) will be scheduled, and it will signal the producers after consuming the item.

☒ The OS will not signal any thread, leading to deadlock because both producers and consumers are stuck waiting for signals.

☐ P2 will be scheduled again, but it cannot signal consumers because the buffer is still full.

Explaination:

The OS will not signal any thread, leading to deadlock because both producers and consumers are stuck waiting for signals.

In this scenario with a buffer size of 1, which starts out empty, here's the sequence of events:

P1 produces an item, fills the buffer, and signals a "buffer change." However, the signal incorrectly wakes up another producer (P2) instead of a consumer. In a typical producer-consumer problem, when a producer fills the buffer, it should signal a consumer, not another producer.

P2 wakes up due to the signal from P1 but finds the buffer still full, since P1 just filled it.

P2 cannot proceed with production, so it goes back to sleep without signaling.

The OS reschedules P1, but P1 also finds the buffer still full and cannot proceed with production, so it also goes back to sleep without signaling.

****

Why does OS reschedule P1 without any signaling?

The OS scheduler selects threads from the Ready state to run on the CPU.

If a thread is blocked, it cannot be scheduled until it's unblocked.

C1, C2, P2 are sleeping (blocked).

P1 sends a signal that mistakenly wakes up P2, but P1 does not go to sleep and is still at ready state.

****

At this point, both producers and consumers are waiting:

The producers (P1 and P2) are waiting for a signal that the buffer is no longer full.

The consumers (C1 and C2) are waiting for a signal that the buffer has data.

Since no thread is currently signaling, the system is in deadlock. Therefore, OS will not signal any thread, resulting in deadlock because both producers and consumers are stuck waiting for signals.

6. (1 point) Why does using two condition variables (not_full and not_empty) prevent deadlock in the Producer-Consumer problem?

   ☒ Because producers signal only consumers and vice versa, ensuring that progress is always made by at least one thread type.

   ☐ Because the buffer size is always kept constant.

   ☐ Because it prevents producers from producing more items than consumers can consume.

   ☐ Because the OS automatically ensures that only one type of thread runs at a time.

7. (1 point) In the Dining Philosophers problem, what would lead to a deadlock if all philosophers follow the same sequence of actions?

   ☒ Philosophers grab the right chopstick before the left one at the same time.

   ☐ Philosophers grab the left chopstick before the right one.

   ☐ Philosophers only eat after thinking.

   ☐ Philosophers start eating before they grab chopsticks.

8. (1 point) In the following scenario, how many threads are required to cause a deadlock?

```
void os_run() {
    spin_lock(&list_lock);
    spin_lock(&xxx);
    spin_unlock(&xxx);
}
void on_interrupt() {
    spin_lock(&list_lock);
    spin_unlock(&list_lock);
}
```

☒ 1 ☐ 2 ☐ 5 ☐ 3

9. (1 point) In the context of deadlock, what role does the operating system play in thread synchronization?

☐ The OS manages thread scheduling and CPU time allocation, but does not manage thread synchronization or signal passing.

☐ The OS automatically handles all synchronization between threads, eliminating the need for condition variables.

☒ The OS automatically signals threads waiting for a condition to be met.

☐ The OS ensures that no thread can enter the critical section if another thread is already there.

10. (1 point) Which of the following deadlock prevention techniques involves making sure no thread waits while holding a resource?

☒ No waiting

☐ Circular wait prevention

☐ Infinite resources

☐ Banker's Algorithm

11. (1 point) What is the main difference between a semaphore and a condition variable?

☐ Semaphores are always used with mutexes, whereas condition variables do not require mutexes.

☒ Semaphores allow multiple threads to access the critical section simultaneously, whereas condition variables typically allow only one thread at a time.

☐ Condition variables are used in user space, whereas semaphores are only used in kernel space.

☐ Condition variables automatically track resource availability, whereas semaphores do not.

12. (1 point) What does the 'P' operation in semaphores do?

☒ Decreases the semaphore's value by 1 and may block the thread if the value is negative.

☐ It checks if the buffer is full and puts the thread in a busy waiting loop.

☐ Puts the thread to sleep until the semaphore's value becomes positive.

☐ Increases the semaphore's value by 1 and signals a thread to continue.

13. (1 point) In the Producer-Consumer problem, what is the main advantage of using semaphores without mutexes when dealing with simple operations?

☐ It simplifies the code, making it easier to implement complex buffer operations.

☐ It allows threads to produce and consume items even when the buffer is full or empty.

☐ It prevents deadlocks from occurring.

☒ It reduces overhead by eliminating the need for locking and unlocking, allowing more threads to operate concurrently.

14. (1 point) In which scenario is using condition variables preferable over semaphores?

☐ When multiple threads need to access the critical section simultaneously.

☒ When the synchronization logic becomes more complex, and you need finer control over waiting and signaling.

☐ When you want to avoid using any locking mechanisms in your program.

☐ When you have a simple, static buffer that does not require complex operations.

15. (1 point) Which of the following is NOT a condition necessary for a deadlock to occur?

☒ Preemptable resources

☐ Circular wait

☐ No preemption

☐ Wait while holding

16. (1 point) What is a practical approach to defensive programming?

☐ Ignoring potential edge cases

☒ Using assertions to validate program conditions

☐ Focusing only on speed optimizations

☐ Implementing all features before testing

17. (1 point) What is the primary focus of High-Performance Computing (HPC)?

☐ Handling system calls efficiently.

☒ Task decomposition and parallel computation.

☐ Managing thread context switching.

☐ Simplifying user interaction with web pages.

18. (1 point) Which technology is commonly used in High-Performance Computing for parallel processing?

☐ JavaScript.

☐ Goroutines.

☒ OpenMP.

☐ Promise.

19. (1 point) What is the main challenge in breaking down computational tasks in HPC?

☐ Managing shared memory access.

☐ Handling network requests asynchronously.

☐ Developing user-friendly interfaces.

☒ Ensuring that computation graphs are easy to parallelize.

20. (1 point) What is the primary focus in concurrent programming within data centers?

☐ Managing graphical user interfaces.

☒ Handling system calls efficiently and serving massive requests.

☐ Task decomposition for parallel processing.

☐ Using promises for asynchronous operations.

21. (1 point) What is a key reason why coroutines do not require context switching?

    ☒ They are scheduled in user space without kernel involvement.

    ☐ They rely on OS-level thread management.

    ☐ They automatically handle all system calls.

    ☐ They are only used in HPC environments.

22. (1 point) What is the main benefit of using Goroutines in Go programming for data centers?

    ☐ Goroutines allow sequential task execution.

    ☒ Goroutines combine the lightweight nature of coroutines and the power of threads.

    ☐ Goroutines do not require memory management.

    ☐ Goroutines are faster than JavaScript promises.

23. (1 point) What is the main benefit of using Goroutines in Go programming for data centers?

    ☐ Goroutines allow sequential task execution.

    ☒ Goroutines combine the lightweight nature of coroutines and the power of threads.

    ☐ Goroutines do not require memory management.

    ☐ Goroutines are faster than JavaScript promises.

24. (1 point) Which of the following best describes the benefit of using asynchronous callbacks in JavaScript?

    ☐ Each asynchronous operation blocks the execution until it is completed, ensuring tasks are performed in sequence.

    ☐ Asynchronous operations are limited to network requests and cannot be used for other types of tasks.

    ☒ Asynchronous operations immediately return when started, and a callback function is called when the operation completes, allowing the program to continue executing other tasks without waiting for the asynchronous operation to finish.

    ☐ Asynchronous callbacks are slower than traditional synchronous programming due to the overhead of managing multiple threads.

25. (1 point) Which technology primarily powers asynchronous event-driven programming in Web 2.0?

    ☒ Promise.

    ☐ Goroutines.

    ☐ OpenMP.

    ☐ MPI.

26. (1 point) What is the main advantage of using promises in JavaScript?

    ☒ They prevent callback hell by improving code readability.

    ☐ They allow direct parallelism.

    ☐ They use hardware-level atomic instructions.

    ☐ They are better suited for numerical computation.

Assignment 11: File System

1. (1 point) Which of the following storage types retains data even when the power is turned off?

☐ SRAM ☐ DRAM ☒ SSD ☐ Cache memory

2. (1 point) What is the primary goal of crash consistency in a file system?

   ☐ Automatically repairing disk hardware after a crash.

   ☒ Ensuring the file system remains in a consistent state before and after a crash.

   ☐ Preventing performance degradation caused by multiple write operations.

   ☐ Improving the random read and write performance of the disk.

3. (1 point) What is the fundamental method to ensure crash consistency in storage systems?

   ☐ File System Checking

   ☒ Journaling

   ☐ RAID

   ☐ Non-Volatile Memory

4. (1 point) When an operating system needs to read a byte of data from a block device (such as a hard drive), which process is correct?

   ☐ The operating system directly reads a single byte of data from the hard drive to the CPU.

   ☒ The operating system reads the entire block containing the byte from the hard drive, loads it into memory via DMA, and then retrieves the needed byte from memory to the CPU.

   ☐ The operating system reads a block of data from the hard drive to memory via DMA, then transfers the entire block to the CPU, where the required byte is extracted.

   ☐ The operating system uses DMA to transfer a single byte from the hard drive directly to the CPU.

5. (1 point) In a modern operating system, data on storage devices is typically accessed in which of the following units?

   ☐ Bit ☐ Byte ☒ Block ☐ Page

6. (1 point) By increasing the physical area of a NAND chip to expand SSD capacity, how does the read/write speed typically change?

   ☒ It increases ☐ It stays the same

   ☐ It decreases ☐ It increases initially, then decreases

   Explanation: If you ask this question to ChatGPT, it may answer: "By increasing the physical area of a NAND chip, you typically increase its capacity, but the speed does not change."

   However, in real products you often find that higher-capacity chips are actually faster, because they are designed with more internal parallelism. For example, when the capacity increases, the manufacturer may widen the data path or external bus from 32 bits to 64 bits. With a 64-bit bus instead of a 32-bit bus, the chip can transfer more data per cycle, so its effective read speed (and often write speed) increases.

7. (1 point) In a modern operating system, how are read and write requests to storage devices typically scheduled?

   ☐ The operating system uses classic algorithms, such as the elevator algorithm, to schedule storage requests.

   ☐ The operating system employs more advanced versions of these algorithms for scheduling.

   ☒ The operating system delegates the scheduling process entirely to the storage device's internal controller.

☐ The operating system requires the user to set up the appropriate scheduling algorithm for each device.

8. (1 point) The operating system has read a byte of data from the hard drive. Now it needs to update this byte and write it back to the hard drive. What is the correct process?

   ☒ The operating system updates the byte in memory, then uses DMA to write the entire block containing that byte back to the hard drive.

   ☐ The operating system updates the byte in memory, then uses DMA to write that byte directly to the hard drive.

   ☐ The operating system locates the byte on the hard drive and directly modifies it.

   ☐ The operating system needs to update all bytes in the entire block before writing it back to the hard drive.

9. (1 point) Which of the following best describes the concept of a virtual drive?

   ☐ A virtual drive represents the entire file system as a single entity.

   ☒ Each individual file in the file system can be treated as a virtual drive.

   ☐ A virtual drive is created when a file or resource is logically separated from the physical hardware.

   ☐ Virtual drives are mainly used for managing large storage devices.

10. (1 point) What does the process of the command "mount" represent?

    ☐ Mounting creates a new virtual drive.

    ☒ Mounting connects one virtual drive to another virtual drive, creating a layered structure.

    ☐ Mounting duplicates the data of a virtual drive onto another virtual drive.

    ☐ Mounting converts a physical drive for storage into a virtual drive.

11. (1 point) What is the primary role of namespace management in a file system?

    ☐ Providing random read and write functionality.

    ☒ Organizing virtual drives into a hierarchical structure.

    ☐ Allocating file storage space.

    ☐ Managing file read and write offsets.

12. (1 point) What is the main reason for having hard links and symbolic links in a file system?

    ☐ Hard links provide a way to create exact duplicates of a file, while symbolic links are used to reference files across file systems or directories.

    ☒ Hard links are designed to allow multiple filenames for the same file data on the same file system, while symbolic links provide a flexible way to reference files, even if they are moved or deleted.

    ☐ Hard links are faster to access because they point directly to the file's data blocks, while symbolic links are slower because they require resolving a path.

    ☐ Hard links are used exclusively for directories, while symbolic links are used for files.

13. (1 point) A file system includes the following features for files: Alias (A secondary name for the file) and Shortcut (A path-based reference to the file). How should these features be represented in terms of links?

    ☒ Alias is a hard link, and shortcut is a symbolic link.

    ☐ Alias is a symbolic link, and shortcut is a hard link.

    ☐ Both alias and shortcut are hard links.

    ☐ Both alias and shortcut are symbolic links.

14. (1 point) Which type of link can be used across file systems?

☐ Hard Link. ☒ Symbolic Link. ☐ File Descriptor. ☐ Directory Link.

15. (1 point) In a RAID 5 system consisting of four disks, data blocks and parity blocks are distributed as follows:

- Disk 1: Data blocks A1, B1, C1; Parity block Dp
- Disk 2: Data blocks A2, B2; Parity block Cp; Data block D1
- Disk 3: Data block A3; Parity block Bp; Data blocks C2, D2
- Disk 4: Parity block Ap; Data blocks B3, C3, D3

When the RAID 5 system needs to update data block A1, it must perform a series of operations to ensure data consistency and correct parity. These operations involve reading and writing data and parity blocks and performing calculations using bitwise XOR operations.

Steps Options:

1) Write the updated data block A1' (A1 prime) to Disk 1.
2) Calculate Delta_A1 = A1' XOR A1.
3) Read the old data block A1 from Disk 1.
4) Calculate the new parity block Ap' = Ap XOR Delta_A1.
5) Read the old parity block Ap from Disk 4.
6) Write the updated parity block Ap' to Disk 4.

Arrange the above steps in the correct order to properly update data block A1 and maintain parity consistency in the RAID 5 system.

☐ 5, 2, 4, 3, 1, 6 ☒ 3, 5, 2, 4, 1, 6
☐ 1, 2, 3, 4, 5, 6 ☐ 6, 5, 4, 3, 2, 1

We need to understand how to distribute the data.

16. (1 point) Based on the above RAID 5 system, when the RAID 5 system needs to update both data blocks A1 and A2, certain steps can be executed as follows:

1) Read old data blocks A1 and A2 from Disk 1 and Disk 2, respectively.
2) Read old parity block Ap from Disk 4.
3) Compute data differences: Delta_A1 = A1' XOR A1, Delta_A2 = A2' XOR A2.
4) Compute new parity block: Ap' = Ap XOR Delta_A1 XOR Delta_A2.
5) Write new data blocks A1' and A2' to Disk 1 and Disk 2, respectively.
6) Write new parity block Ap' to Disk 4.

Arrange the above steps in the correct order to properly update data blocks A1 and A2 while maintaining parity consistency in the RAID 5 system.

☒ 1, 2, 3, 4, 5, 6 ☐ 6, 5, 4, 3, 2, 1
☐ 3, 4, 1, 2, 6, 5 ☐ 1, 2, 3, 5, 6, 4

17. (1 point) Based on the above RAID 5 system, when the RAID 5 system needs to update both data blocks A1 and B2, certain steps can be executed as follows:

1) Read old data blocks A1 and B2 from Disk 1 and Disk 2, respectively.
2) Read old parity blocks Ap and Bp from Disk 4 and Disk 3, respectively.
3) Compute data differences: Delta_A1 = A1' XOR A1, Delta_B2 = B2' XOR B2.

4) Compute new parity blocks: Ap' = Ap XOR Delta_A1, Bp' = Bp XOR Delta_B2.
5) Write new data blocks A1' and B2' to Disk 1 and Disk 2, respectively.
6) Write new parity blocks Ap' and Bp' to Disk 4 and Disk 3, respectively.

Arrange the above steps in the correct order to properly update data blocks A1 and B2 while maintaining parity consistency in the RAID 5 system.

☒ 1, 2, 3, 4, 5, 6
☐ 6, 5, 4, 3, 2, 1
☐ 3, 4, 1, 2, 6, 5
☐ 1, 2, 3, 5, 6, 4

We need to understand how to recover the data.

18. (1 point) In the RAID 5 system described earlier, multiple steps can be executed in parallel due to the distribution of data and parity blocks across all disks. For example, when updating data blocks A1 and B2, it is possible to perform parallel reads from Disk 1, Disk 2, Disk 3, and Disk 4 to read A1, B2, A parity, and B parity simultaneously. This parallelism takes full advantage of RAID 5's distributed parity design. If the data were stored using RAID 4 (with a dedicated parity drive), which of the following statements is correct regarding parallelism when updating data blocks such as A1, A1 and A2, or A1 and B2?

☐ For RAID 4, all steps for updating A1, A1 and A2, or A1 and B2 are identical to RAID 5, and the level of parallelism is the same in both RAID 4 and RAID 5.

☐ The parallelism of parity computation depends on the parallelism of read and write operations. In RAID 4, since it cannot read A parity and B parity in parallel, it cannot compute parity in parallel. However, in RAID 5, it can read A parity and B parity in parallel, so it can compute parity in parallel.

☒ For RAID 4 and RAID 5, the parallelism for updating A1 and A1 and A2 is the same, but the parallelism for updating A1 and B2 differs. This is because RAID 4 stores all parity information on a single drive, which requires sequential access to read and write parity blocks.

☐ RAID 4 allows more parallelism than RAID 5 because the dedicated parity drive simplifies the read and write operations.

We need to understand the advantages of RAID 5, particularly how it enables parallel reading, computing, and writing operations.

19. (1 point) Why does journaling provide "all-or-nothing" guarantees for data consistency in the event of a crash?

☐ Journaling writes data directly to the file system, bypassing any intermediate logging steps, ensuring all operations are applied atomically.

☒ Journaling ensures that all changes are first written to a persistent log (journal), and only after the changes are successfully logged is the actual data updated. This way, incomplete or crashed transactions can either be rolled back or replayed completely.

☐ Journaling ensures consistency by skipping failed transactions entirely, without ever recording partial changes in the journal.

☐ Journaling provides "all-or-nothing" guarantees by mirroring data changes to redundant storage in real time to prevent crashes from affecting data integrity.

Assignment 12: I/O Device

1. (1 point) How does an operating system abstract various I/O devices?

☒ A set of registers and protocols

☐ Various electrical signals

☐ Various interfaces, such as serial and parallel interfaces

☐ Character Devices and Block Devices

2. (1 point) From the CPU's perspective, all types of devices can be abstracted as a single kind of device. What is this device?

   ☐ Interface ☒ BUS ☐ DMA ☐ Controller

3. (1 point) How does an operating system abstract interaction with various I/O devices?

   ☐ The operating system uses system calls like read, write, and ioctl to manipulate device registers through the bus.

   ☐ The operating system uses system calls like read, write, and ioctl to implement device protocols on the bus.

   ☐ The operating system uses system calls like read, write, and ioctl to access the hardware's electrical signals through interfaces.

   ☒ The operating system uses system calls like read, write, and ioctl to communicate with the driver, which then manages devices through their registers and protocols over the bus.

4. (1 point) Where is the device driver located in a computer system?

   ☐ In user space
   ☐ In the device controller
   ☒ In kernel space
   ☐ In the DMA controller

5. (1 point) When we run `ls /dev`, we see many devices listed. Are all of these devices real?

   ☐ Yes, all devices are real; each listed device corresponds to an actual physical device.

   ☐ No, some devices are simulated by the device controller.

   ☒ No, some devices are simulated by the device driver.

   ☐ No, some devices are simulated by the DMA controller.

6. (1 point) There are over 300 system calls in the operating system. Which system call is associated with the most kernel code modules?

   ☐ read ☐ write ☐ open ☒ ioctl

7. (1 point) Which of the following is NOT a typical consideration when designing a Linux device driver?

   ☐ Defining the purpose and main functions of the driver (e.g., read, write, and ioctl operations)

   ☐ Registering the device with a major and minor number

   ☒ Setting up the device's hardware configuration directly within user space

   ☐ Implementing core functions that handle device-specific operations