Review of Final Exam for COP 4610 Operating Systems - Fall 2024

Name: _____ **Student ID:** _____

Score: _____

♠ **This question is likely to be on the exam.**

HW5

1. (5 points) ♠ When starting a program, what happens after each step to switch from the kernel level to the user level? Match each step on the left with its corresponding action on the right.

Step	Action
Create a process and initialize the address space → A	A. Load the program into the memory
Initialize the translation table → B	B. Set the HW pointer to the translation table
Load the program into the memory → C	C. Initialize the translation table
Set the CPU to user mode → D	D. Jump to the entry point of the program
Set the HW pointer to the translation table → E	E. Set the CPU to user mode

Instead of focusing on matching, we should focus on the logical flow of actions that take place during this transition.

1. Create a process and initialize the address space
   At this step, the operating system sets up the **process control block (PCB)** and allocates an **address space** for the new process. This is the foundational step for any program to start execution.
2. Load the program into memory
   After the address space is set up, the program's binary is loaded into the **code segment** and its associated data into the appropriate memory regions. This ensures the program is ready to run.
3. Initialize the translation table
   Once the program is loaded into memory, the OS sets up a **translation table** (e.g., page tables) that maps virtual addresses to physical addresses. This step is critical for memory isolation and management in a virtual memory system.
4. Set the HW pointer to the translation table
   After the translation table is initialized, the **hardware pointer** (like the `CR3` register on x86) is updated to reference this table. This ensures that subsequent memory access by the program uses the correct virtual-to-physical address mapping.
5. Set the CPU to user mode
   Once the memory is set up, the OS switches the **CPU mode** from **kernel mode to user mode**. This is necessary to allow the program to execute with limited privileges for security.
6. Jump to the entry point of the program
   Finally, the CPU jumps to the **entry point** of the program (usually the `main()` function or a similar starting point), and the program begins its execution.

2. (1 point) ♠ Suppose you have a 16-bit machine with a page size of 16B. Assume that any unsigned integer can be a potential memory address. Also, assume that the machine can support up to only 32KB of physical memory. ($1\text{K} = 2^{10}$).

   For paging-based memory allocation, how many page table entries (in decimals) do you need to map the virtual address space of a process, with pure paging alone?

   Answer: 4096

   - Parameter 1: Virtual address size (16-bit machine)
     A 16-bit address implies that the virtual address space is 2^{16} bytes, which equals $65,536$ bytes (or 64KB).

- Parameter 2: Page size (16 bytes)
  Each page in memory is 16 bytes. To calculate the total number of pages in the virtual address space, we divide the virtual address space by the page size:

$$\text{Number of pages in virtual address space} = \frac{\text{Virtual memory size}}{\text{Page size}} = \frac{65,536}{16} = 4,096.$$

- Parameter 3: Physical memory size (32KB)
  While the machine has 32KB of physical memory, this parameter is not relevant for determining the number of page table entries. The page table is responsible for mapping the entire virtual address space, regardless of the size of the physical memory.

3. (1 point) ♠ Suppose you have a 16-bit machine with a page size of 16B. Assume that any unsigned integer can be a potential memory address. Also, assume that the machine can support up to only 32KB of physical memory. (1K = 2^{10}).

   Translate the virtual address $0x0000$ to its physical address (base 16) via the segmented-paging scheme. Assume that a program has two segments and a page size of 16B.

   Segment Table:

Segment Number	Page Table Base	Page Table Bound
0	$0x8000$	$0x8$
1	$0x0000$	$0x8$

   Physical Memory Address $0x0000$**:**

Virtual Page Number	Physical Page Number
$0x000$	$0x500$
$0x001$	$0x504$
$0x002$	$0x508$
$0x003$	$0x50B$
...	

   Physical Memory Address $0x8000$**:**

Virtual Page Number	Physical Page Number
$0x000$	$0x100$
$0x001$	$0x104$
$0x002$	$0x108$
$0x003$	$0x10B$
...	

   Answer: 1,000

   Step-by-Step Process:

   1. Understand the structure of the virtual address:
      The virtual address is composed of three parts:

      Segment Number (1 bit) || Virtual Page Number (11 bits) || Page Offset (4 bits)

      - The machine uses a total of 16 bits for its address. - The Segment Number is the most significant bit (1 bit). - The Virtual Page Number is the next 11 bits. - The Page Offset is the least significant 4 bits.
      For $0x0000$ (in binary: 0000 0000 0000 0000):
      - Segment Number = 0 (most significant bit).
      - Virtual Page Number = $0x000$ (next 11 bits).

- Page Offset = $0x0$ (last 4 bits).

2. Locate the segment table entry:
   Using the **Segment Number** 0, look up the corresponding entry in the segment table:

   $$\text{Page Table Base} = 0x8000, \quad \text{Page Table Bound} = 0x8.$$

   This tells us that the page table for this segment starts at physical address $0x8000$.

3. Use the Virtual Page Number (VPN) to find the physical page:
   The Virtual Page Number $0x000$ is used to index the page table starting at $0x8000$. Since the **Virtual Page Number** is 0, the first entry of the page table contains:

   $$\text{Physical Page Number (PPN)} = 0x100.$$

4. Combine the PPN and Page Offset to compute the physical address:
   The final physical address is computed by combining the **Physical Page Number (PPN)** with the **Page Offset**:

   $$\text{Physical Address} = \text{PPN} \parallel \text{Page Offset}.$$

   Substituting the values:

   $$\text{Physical Address} = 0x100 \parallel 0x0 = 0x1000.$$

Key Points:

- Virtual Address Structure: The virtual address is split into **Segment Number (1 bit)**, **Virtual Page Number (11 bits)**, and **Page Offset (4 bits)**.
- Page Table Address: The **Segment Number** determines the base address of the page table in physical memory. For $0x0000$, it points to $0x8000$.
- Physical Memory Access: The segment and page tables determine the mapping, and the final address combines the physical page number with the page offset.

4. (1 point) ♠ Suppose you have a 16-bit machine with a page size of 16B. Assume that any unsigned integer can be a potential memory address. Also, assume that the machine can support up to only 32KB of physical memory. ($1\text{K} = 2^{10}$).

   Translate the virtual address $0x0003$ to its physical address (base 16) via the segmented-paging scheme. Assume that a program has two segments and a page size of 16B.

   Segment Table:

Segment Number	Page Table Base	Page Table Bound
0	$0x8000$	$0x8$
1	$0x0000$	$0x8$

   Physical Memory Address $0x0000$**:**

Virtual Page Number	Physical Page Number
$0x000$	$0x500$
$0x001$	$0x504$
$0x002$	$0x508$
$0x003$	$0x50B$
...	

   Physical Memory Address $0x8000$**:**

Virtual Page Number	Physical Page Number
$0x000$	$0x100$
$0x001$	$0x104$
$0x002$	$0x108$
$0x003$	$0x10B$
...	

Answer: 1,003

Step-by-Step Process:

1. Understand the structure of the virtual address:
   In segmented-paging, the virtual address is divided into three parts:

   $$\text{Segment Number (1 bit)} \parallel \text{Virtual Page Number (11 bits)} \parallel \text{Page Offset (4 bits)}.$$

   For the 16-bit virtual address $0x0003$ (in binary: 0000 0000 0000 0011):
   - Segment Number: The most significant bit is 0, indicating **Segment 0**.
   - Virtual Page Number: The next 11 bits are $0x000$.
   - Page Offset: The least significant 4 bits are $0x3$.
2. Locate the segment table entry:
   Using the **Segment Number** 0, we find the corresponding entry in the segment table:

   $$\text{Page Table Base} = 0x8000, \quad \text{Page Table Bound} = 0x8.$$

   This tells us that the page table for this segment starts at physical address $0x8000$.
3. Use the Virtual Page Number (VPN) to find the physical page:
   The Virtual Page Number $0x000$ is used to index the page table starting at $0x8000$. Since the Virtual Page Number is 0, the first entry of the page table contains:

   $$\text{Physical Page Number (PPN)} = 0x100.$$

4. Combine the PPN and Page Offset to compute the physical address:
   The final physical address is computed by combining the **Physical Page Number (PPN)** with the **Page Offset**:

   $$\text{Physical Address} = \text{PPN} \parallel \text{Page Offset}.$$

   Substituting the values:

   $$\text{Physical Address} = 0x100 \parallel 0x3 = 0x1003.$$

5. (1 point) ♠ Suppose you have a 16-bit machine with a page size of 16B. Assume that any unsigned integer can be a potential memory address. Also, assume that the machine can support up to only 32KB of physical memory. ($1\text{K} = 2^{10}$).

   Translate the virtual address $0x8020$ to its physical address (base 16) via the segmented-paging scheme. Assume that a program has two segments and a page size of 16B.

   Segment Table:

Segment Number	Page Table Base	Page Table Bound
0	$0x8000$	$0x8$
1	$0x0000$	$0x8$

   Physical Memory Address $0x0000$**:**

Virtual Page Number	Physical Page Number
$0x000$	$0x500$
$0x001$	$0x504$
$0x002$	$0x508$
$0x003$	$0x50B$
...	

Physical Memory Address $0x8000$**:**

Virtual Page Number	Physical Page Number
$0x000$	$0x100$
$0x001$	$0x104$
$0x002$	$0x108$
$0x003$	$0x10B$
...	

Answer: 5,080

Step-by-Step Process:

1. Understand the structure of the virtual address:
   In segmented-paging, the virtual address is divided into three parts:

   $$\text{Segment Number (1 bit)} \parallel \text{Virtual Page Number (11 bits)} \parallel \text{Page Offset (4 bits)}.$$

   For the 16-bit virtual address $0x8020$ (in binary: 1000 0000 0010 0000):
   - Segment Number: The most significant bit is 1, indicating **Segment 1**.
   - Virtual Page Number: The next 11 bits are $0x002$.
   - Page Offset: The least significant 4 bits are $0x0$.
2. Locate the segment table entry:
   Using the **Segment Number** 1, we find the corresponding entry in the segment table:

   $$\text{Page Table Base} = 0x0000, \quad \text{Page Table Bound} = 0x8.$$

   This tells us that the page table for this segment starts at physical address $0x0000$.
3. Use the Virtual Page Number (VPN) to find the physical page:
   The Virtual Page Number $0x002$ is used to index the page table starting at $0x0000$. From the provided page table:

   $$\text{Virtual Page Number (VPN)} = 0x002 \implies \text{Physical Page Number (PPN)} = 0x508.$$

4. Combine the PPN and Page Offset to compute the physical address:
   The final physical address is computed by combining the **Physical Page Number (PPN)** with the **Page Offset**:

   $$\text{Physical Address} = \text{PPN} \parallel \text{Page Offset}.$$

   Substituting the values:

   $$\text{Physical Address} = 0x508 \parallel 0x0 = 0x5080.$$

HW6

1. (8 points) Match the terms on the right with the definitions on the left.

Definition	Term
A Storing copies of data at places that can be accessed more quickly	A. caching
B Recently referenced locations are more likely to be referenced soon	B. temporal locality
C Reference locations tend to be clustered	C. spatial locality
D Leaving behind cache content with no localities	D. cache pollution
E Data brought into the cache for the first time	E. compulsory misses
F Caused by the limited size of a cache	F. capacity misses
G Immediately propagates update through various levels of caching	G. write-through caching
H Delaying the propagation until the cached item is replaced	H. write-back caching

2. (1 point) ♠ What is the effective access time of memory (in decimals) through L1 and L2 caches for the following hardware characteristics?

	Access Time	**Cache Hit Rate**
L1 Cache	1 clock cycle	75% or $\frac{3}{4}$
L2 Cache	3 clock cycles	75% or $\frac{3}{4}$
Memory	4 clock cycles	100% or 1

Answer: 2

Step-by-Step Calculation:

1. Understand the concept of Effective Access Time (EAT):
   The EAT represents the average time required to access memory, considering both cache hits and cache misses at each level of the memory hierarchy. It is computed as:

   $$\text{EAT} = (\text{Hit Rate of L1}) \times (\text{Access Time of L1}) + (\text{Miss Rate of L1}) \times (\text{EAT of L2}).$$

2. Calculate the EAT for L2 cache:
   For L2 cache, the **Hit Rate** is 75% (or $\frac{3}{4}$), and the **Miss Rate** is 25% (or $\frac{1}{4}$). The memory access time for a miss in L2 is 4 clock cycles. Thus:

   $$\text{EAT of L2} = (\text{Hit Rate of L2}) \times (\text{Access Time of L2}) + (\text{Miss Rate of L2}) \times (\text{Memory Access Time}).$$

   Substituting the values:

   $$\text{EAT of L2} = \left(\frac{3}{4} \times 3\right) + \left(\frac{1}{4} \times 4\right).$$

   Simplify:

   $$\text{EAT of L2} = \frac{13}{4} \text{ clock cycles}.$$

3. Calculate the EAT for L1 cache:
   For L1 cache, the **Hit Rate** is 75% (or $\frac{3}{4}$), and the **Miss Rate** is 25% (or $\frac{1}{4}$). The EAT of L2 (calculated above) is used for L1 misses. Thus:

   $$\text{EAT} = (\text{Hit Rate of L1}) \times (\text{Access Time of L1}) + (\text{Miss Rate of L1}) \times (\text{EAT of L2}).$$

   Substituting the values:

   $$\text{EAT} = \left(\frac{3}{4} \times 1\right) + \left(\frac{1}{4} \times \frac{13}{4}\right).$$

   Simplify:

   $$\text{EAT} = \frac{25}{16} \text{ clock cycles}.$$

   Round to the nearest integer:

   $$\text{Effective Access Time} = 2 \text{ clock cycles}.$$

HW7

1. (5 points) Match the terms on the right with the definitions on the left.

Definition		Term
A	Allowing pages that are referenced actively to be loaded into memory	A. demand paging
B	A referenced page is not in memory	B. page fault
C	Adding more memory results in more page faults	C. Belady's anomaly
D	When the memory is overcommitted	D. thrashing
E	Pages that are referenced within the past T seconds	E. working set

2. (1 point) ♠ For the following reference stream, what is the content of page 2 at the end of the reference stream under the FIFO policy?

   A process makes references to 4 pages: A, E, R, and N.
   Reference stream: `E A R N R E A R N E A R`
   Physical memory size: 3 pages.

Reference Stream:

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1												
2												
3												

☐ A ☐ E ☐ N ☒ R

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1	E	E	E	N	N	N	N	R	R	R	A	A
2		A	A	A	A	E	E	E	N	N	N	R
3			R	R	R	R	A	A	A	E	E	E

3. (1 point) ♠ For the following reference stream, what is the content of page 2 at the end of the reference stream under the MIN (Minimum Page Replacement) policy?

Reference Stream:

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1												
2												
3												

☐ R ☒ E ☐ A ☐ N

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1	E	E	E	E	E	E	A	A	A	A	A	A
2		A	A	N	N	N	N	N	N	E	E	E
3			R	R	R	R	R	R	R	R	R	R

4. (1 point) ♠ For the following reference stream, what is the content of page 2 at the end of the reference stream under the LRU (Least Recently Used) policy?

Reference Stream:

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1												
2												
3												

☐ N ☐ A ☐ E ☒ R

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1	E	E	E	N	N	N	A	A	A	E	E	E
2		A	A	A	A	E	E	E	N	N	N	R
3			R	R	R	R	R	R	R	R	A	A

5. (1 point) ♠ For the following reference stream, what is the content of page 2 at the end of the reference stream under the LFU (Least Frequently Used) policy?

Reference Stream:

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1												
2												
3												

☐ R ☐ E ☐ N ☒ A

Memory Page	E	A	R	N	R	E	A	R	N	E	A	R
1	E	E	E	N	N	N	A	A	A	E	E	E
2		A	A	A	A	E	E	E	N	N	A	A
3			R	R	R	R	R	R	R	R	R	R

HW8

1. (8 points) ♠ Match the terms on the right with the definitions on the left.

Definition	**Term**
__A__ Converting between serial bit stream and a block of bytes	A. device controller
__B__ An OS component that hides the complexity of an I/O device	B. device driver
__C__ Making no distinction between device addresses and memory addresses	C. memory-mapped I/O
__D__ A CPU repeatedly checks the status of a device	D. polling
__E__ A device controller notifies the corresponding driver when the device is available	E. interrupt-driven I/Os
__F__ Using an additional controller to perform data movements	F. direct memory access
__G__ A keyboard is an example of...	G. character device
__H__ A disk is an example of...	H. block device

2. (9 points) ♠ When we run `gcc HelloWorld.c && ./a.out` in the Shell, the operating system goes through the following steps. Please arrange the following steps in the correct order.

   A. The Shell creates a child process to run `gcc`: The Shell calls `fork` to create a child process and then runs `gcc HelloWorld.c` in the child process.

   B. The compiler compiles `HelloWorld.c` into an object file: The compiler translates the source code into a machine-code object file (e.g., `HelloWorld.o`).

   C. The Shell checks the compilation result: If `gcc` succeeds (returns 0), the Shell continues to execute `./a.out`; otherwise, the operation terminates.

   D. The linker generates the executable file: The linker combines the object file with required libraries, producing an executable file (by default, `a.out`).

   E. The child process calls `execve`: In the child process, the `execve` system call replaces the process image with that of the `a.out` executable file.

   F. The Shell creates another child process to run `./a.out`: The Shell calls `fork` again to create a new child process, and in the child process, it executes `./a.out`.

G. The program begins executing: The `a.out` program runs in the child process, with its own memory space and execution context.

H. The child process exits and returns a status: After execution, the child process terminates, returning its exit status to the parent process.

I. The program outputs its result: The `a.out` program completes and prints "Hello, World!" to the terminal.

Answer: ABDCFEGIH

3. (1 point) In the question of `gcc HelloWorld.c && ./a.out`, the Shell calls `fork` multiple times to create child processes. Who is the parent process for these child processes?

☐ Fork ☐ Execve ☒ The Shell ☐ HelloWorld.o

Answer: C

4. (1 point) In the question of `gcc HelloWorld.c && ./a.out`, where is the child process created to run `./a.out`, and where is its process image replaced with that of the `a.out` executable file?

☒ In the memory area allocated for the operating system kernel (kernel space)

☐ In the memory area allocated for application programs (user space)

☐ On the hard drive

☐ In the CPU

Answer: A

5. (1 point) In the question of `gcc HelloWorld.c && ./a.out`, where does the child process run when executing `./a.out`?

☐ In the memory area allocated for the operating system kernel (kernel space)

☒ In the memory area allocated for application programs (user space)

☐ On the hard drive

☐ In the CPU

Answer: B

6. (1 point) In the operating system's memory management, what is a "free list"?

☐ A list that records all allocated memory blocks

☐ A list that stores all memory blocks currently in use

☒ A list that records available memory blocks, used to allocate unused blocks to processes or threads when needed

☐ A list that stores information about all processes in the operating system

Answer: C

7. (1 point) When the operating system allocates memory to threads, this allocation process is not atomic; it is completed in several steps. If multiple threads share the same memory space without locking the free list, what might happen?

☐ Each thread would request and release memory in sequence

☒ Multiple threads might allocate the same memory block simultaneously, resulting in a race condition

☐ The operating system would automatically expand memory space to meet all threads' needs

☐ The operating system would automatically ensure each thread receives a different memory block

Answer: B

8. (1 point) When locks are applied to the free list, what might be a downside of this approach?

   ☐ Faster memory allocation

   ☒ Decreased allocation efficiency

   ☐ Reduced memory space

   ☐ Increased list capacity

   Answer: B

9. (1 point) In a multithreaded environment, to improve memory allocation efficiency and reduce the need for frequent locking of the free list, the SLAB allocator employs which of the following strategies?

   ☒ Each thread maintains its own SLAB for each object size, allowing it to allocate memory from its local SLAB without locking; locks are only needed when accessing the global list for new SLABs.

   ☐ Each thread has a separate memory block, with all memory operations occurring in the thread's local memory, eliminating the need for any locking.

   ☐ A single global SLAB list is maintained, requiring all threads to lock the global free list for every allocation and deallocation to ensure thread safety.

   ☐ The global SLAB list is split into smaller, page-level lists (List Sharding), with locks on each small list to reduce contention, but threads still need to acquire the lock on the respective small list for any memory operation.

   Answer: A

HW9

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

8. (1 point) Which of the following storage types retains data even when the power is turned off?

   ☐ SRAM ☐ DRAM ☒ SSD ☐ Cache memory

9. (1 point) ♠ By increasing the number of platters to expand HDD capacity, how does the read/write speed change?

   ☒ It increases
   ☐ It decreases
   ☐ It stays the same
   ☐ It increases initially, then decreases

10. (1 point) By increasing the platter area to expand HDD capacity, how does the read/write speed change?

    ☐ It increases
    ☐ It stays the same
    ☒ It decreases
    ☐ It increases initially, then decreases

11. (1 point) By increasing the density of pits on a CD to expand its capacity, how does the read/write speed change?

    ☒ It increases
    ☐ It stays the same
    ☐ It decreases
    ☐ It increases initially, then decreases

12. (1 point) ♠ By increasing the physical area of a CD to expand its capacity, how does the read/write speed change?

    ☐ It increases
    ☐ It stays the same
    ☒ It decreases
    ☐ It increases initially, then decreases

13. (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

14. (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.

15. (1 point) In a modern operating system, data on storage devices is typically accessed in which of the following units?

☐ Bit ☐ Byte ☒ Block ☐ Page

HW10

1. (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.

2. (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.

3. (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.

4. (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.

5. (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.

6. (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.

7. (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.

8. (1 point) ♠ Which type of link can be used across file systems?

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

9. (1 point) ♠ When a parent process creates a child process using `fork()`, how are file descriptors and file offsets handled between the parent and the child processes?

   ☐ File descriptors are independent, but file offsets are shared.

   ☐ File descriptors and file offsets are both independent.

   ☒ File descriptors and file offsets are both shared via the same file table entry.

   ☐ File descriptors are shared, but file offsets are independent.

10. (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.

11. (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

12. (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.

13. (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.

14. (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.

15. (1 point) What is the fundamental method to ensure crash consistency in storage systems?

    ☐ File System Checking

    ☒ Journaling

    ☐ RAID

    ☐ Non-Volatile Memory

16. (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.