## USER

1. Consider a direct-mapped cache memory. Which one of the following statements is true?
(a) The cache has 1 line per set.
(b) The cache has 1 word per block.
(c) The cache has 1 set per cache.
(d) None of the above.

## ASSISTANT

According to the context, a direct-mapped cache memory has exactly one line per set (E=1). Therefore, option (a) "The cache has 1 line per set" is true.

## USER

2. Which one of the following statements about cache memories is true:
(a) Larger caches are more susceptible to capacity misses than smaller caches.
(b) Caches with lower associativity are more susceptible to conflict misses than those with higher associativy.
(c) Caches with higher associativity are more susceptile to cold misses than those with lower associativey.
(d) None of the above

## ASSISTANT

Based on the context, it can be inferred that a larger cache tends to increase the hit time, not capacity misses. Therefore, option (a) "Larger caches are more susceptible to capacity misses than smaller caches" is false.

The text also explains why an L1 cache is smaller than an L2 cache and an L2 cache is smaller than an L3 cache, which suggests that larger caches do increase the hit time, but it does not specify whether they are more susceptible to capacity misses or not.

As for options (b) and (c), the text does not provide information about the relationship between associativity and conflict misses or cold misses. Therefore, I would ask for clarification on these points.

So, according to the rules, my answer is: None of the above (d).

## USER

3. Which one of the following is NOT contained in an ELF executable file?
(a) Machine code
(b) Global variables
(c) User stack
(d) Symbol table

## ASSISTANT

According to the context, an ELF relocatable object file contains the following sections: .text, .rodata, and

.data. However, there is no mention of a "User stack" section.

The text does mention that local C variables are maintained at run time on the stack and do not appear in either the .data or .bss sections. This suggests that the user stack is managed by the operating system or the runtime environment, rather than being part of the ELF file itself.

Therefore, my answer is: (c) User stack.

## USER

4. Assuming no errors, which one of the following statements about fork is true?
(a) Called once, returns once.
(b) Called once, returns twice.
(c) Called once, returns never.
(d) Called twice, returns once.
(e) None of the above.

## ASSISTANT

According to the context, when a program calls fork, it creates a new process by duplicating the calling process. The system call fork() returns once to the parent process and again to the child process.

Therefore, my answer is: (b) Called once, returns twice.

## USER

5. Assuming no errors, which one of the following statements about execve is true?
(a) Called once, returns once.
(b) Called once, returns twice.
(c) Called once, returns never.
(d) Called twice, returns once.
(e) None of the above.

## ASSISTANT

According to the context, when a program calls execve, it replaces the current process image with a new one. The system call execve() does not return; instead, it terminates the calling process.

Therefore, my answer is: (c) Called once, returns never.

## USER

6. Which one of the following statements about processes is false?
(a) The operating system kernel runs as its own separate process.
(b) Each process shares the CPU with other processes.
(c) Each process has its own private address space.
(d) The environment for a process is stored on the stack.

## ASSISTANT

Based on the context, it can be inferred that:

* (a) The operating system kernel runs as its own separate process: TRUE
* (b) Each process shares the CPU with other processes: FALSE; in fact, each process gets exclusive use of the CPU while it is executing.
* (c) Each process has its own private address space: TRUE
* (d) The environment for a process is stored on the stack: FALSE; actually, the environment is stored in a data structure maintained by the operating system.

Therefore, my answer is: (b) Each process shares the CPU with other processes.

## USER

7. What happens if the parent of a zombie child terminates?
(a) The zombie child becomes a wraith and is never reaped.
(b) The zombie child is reaped by the init process.
(c) The zombie child is reaped by the process with the nearest PID.
(d) None of the above.

## ASSISTANT

According to the context, when a parent process terminates, its zombie children are immediately reaped and their resources are released. Therefore, my answer is: (b) The zombie child is reaped by the init process.

## USER

8. Suppose that the kernel delivers two SIGCHLD signals to the parent while the parent is not scheduled. When the kernel finally schedules the parent, how many times will the SIGCHLD handler be called?
(a) None, because sending multiple signals will always crash the program.
(b) Exactly once, because signals are not queued.
(c) Exactly twice, because signals are queued.
(d) More than twice, depending on how the handler is installed.

## ASSISTANT

Based on the context, when the parent catches a SIGCHLD signal and executes its handler, it reaps all its children. Then, it becomes a zombie process itself. If another SIGCHLD signal arrives while the parent is not scheduled, it will still be handled when the parent is eventually scheduled again.

Therefore, my answer is: (b) Exactly once, because signals are not queued.

## USER

9. Which one of the following statements is NOT true of storage allocators?
(a) In the best case, coalescing with boundary tags is linear in the number of free blocks.

(b) Seglists typically approximate best fit search.
(c) Payloads must be aligned to some boundary.
(d) Explicit lists are typically faster than implicit lists.
(e) None of the above.

## ASSISTANT

According to the context, storage allocators are systems that manage memory allocation and deallocation. Coalescing with boundary tags is a technique used in storage allocators to merge adjacent free blocks into a single larger block.

Payloads being aligned to some boundary is also true because most operating systems require alignment of data structures (payloads) on specific boundaries such as word or double-word boundaries for optimal performance.

Seglists and implicit lists can approximate best-fit search, which means they try to find the smallest available block that satisfies the requested size. This approach can be used by storage allocators to efficiently manage memory allocation.

Finally, explicit lists are typically faster than implicit lists because they provide better locality of reference and avoid the need for costly searching and scanning.

Therefore, my answer is: (a) In the best case, coalescing with boundary tags is linear in the number of free blocks. This statement is actually true, not false.

## USER

10. Which one of the following addresses is 8-byte aligned?
(a) 1110110101110111
(b) 1110110101110100
(c) 1110110101110000
(d) 1110110101110110
(e) None of the above

## ASSISTANT

To determine if an address is 8-byte (64-bit) aligned, we can simply check if the last three bits are zero.

(a) 1110110101110111: Last three bits are 111, not zero.
(b) 1110110101110100: Last three bits are 100, not zero.
(c) 1110110101110000: Last three bits are 000, which is zero!
(d) 1110110101110110: Last three bits are 110, not zero.

Therefore, my answer is: (c) 1110110101110000. This address is 8-byte aligned because its last three bits are all zeros.