Timothy Holmes (username: THOLME15)

Attempt 1

Written: Oct 28, 2020 10:22 PM - Oct 28, 2020 10:31 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1 — 0 / 1 point

In an event-based server, how many different processes/threads are run at any given time?

A. 1.

B. 2.

C. As many as the event poll size.

D. As many as there are clients.

➡ ◯ Answer A.

◯ Answer B.

◯ Answer C.

✖ ◯ Answer D.

▷ View Feedback

Question 2 — 0 / 1 point

In an event-based server, the server probes all the connections to check if some data is available. How does it do that?

A. Using a signal handler that is triggered when new data arrives on an open file descriptor.

B. Using a virtual file descriptor that aggregates all the file descriptors that the process wants to read from.

C. Using some polling system calls that can wait for an event to arrive on a set of file descriptors.

D. Using non-blocking sockets, that is, sockets created using the option `O_NONBLOCK`. These sockets return from `read` immediately if no data is available.

✗ ○ Answer A.

○ Answer B.

➡ ○ Answer C.

○ Answer D.

▷ View Feedback

Question 3 — 1 / 1 point

In POSIX threads, how is a thread identified?

A. By both a *thread* ID and the ID of the process in which it lives.

B. By a *thread* ID that is unique system-wide.

C. Threads do not have identifiers.

✓ ○ Answer A.

○ Answer B.

○ Answer C.

View Feedback

Question 4 0 / 1 point

At any given time, a process state can be described by the current value of the CPU registers, its memory (or more precisely, the virtual memory structures), and the process-specific info in the kernel (e.g., descriptor table). In a process that has two threads, what information is common to the threads?

A. The memory and the kernel info.

B. The memory only.

C. The kernel context only.

D. All of the info.

➡ ◯ Answer A.

◯ Answer B.

◯ Answer C.

✖ ◯ Answer D.

View Feedback

Question 5 0 / 1 point

Threads are created using `pthread_create()`. What is the function used to reap a thread?

A. `pthread_wait()`.

B. `pthread_reap()`.

C. `pthread_join()`.

D. Threads are not reaped.

Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 6

0 / 1 point

What are common points between threads and processes?

A. The costs of creating threads and processes are the same.

B. The ways memory from the callee thread/process is shared with new threads/processes (created using `pthread_create` and `fork`) are the same.

C. They have separate control flow.

D. They run concurrently and are context switched.

E. Two of A, B, C, D.

F. Three of A, B, C, D.

G. All of A, B, C, D.

Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

Answer F.

Answer G.

View Feedback

Question 7 1 / 1 point

When creating a new thread, the fourth argument of `pthread_create()` is passed as an argument to the thread main function. What is the type of that argument?

A. `void`.

B. `int`.

C. `unsigned long`.

D. `void *`.

- Answer A.
- Answer B.
- Answer C.
- ✔ Answer D.

View Feedback

Question 8 0 / 1 point

What is detached mode?

A. A running mode for processes in which they can't reap the threads they spawn.

B. A running mode for threads in which they don't need (and can't) be reaped.

C. A running mode for threads in which they can't reap other threads.

D. A running mode for processes in which they don't need (and can't) be reaped.

- Answer A.
- ➡ Answer B.

✖ ◯ Answer C.

◯ Answer D.

▷ View Feedback

Question 9 0 / 1 point

Why is it hard, in practice, to identify which variables are shared between threads?

A. Because any thread can access the whole memory space of the process, including the stacks of the other threads.

B. Because we don't have control over the scheduling of threads, as this is the kernel's job.

C. Because threads don't share the same memory space.

D. More than one of A, B, C.

➡ ◯ Answer A.

◯ Answer B.

◯ Answer C.

✖ ◯ Answer D.

▷ View Feedback

Question 10 0 / 1 point

Assume we want to pass a few parameters when creating a handful of threads. We bundle these arguments into a `struct` that contains, for instance, a thread number. Instead of passing the address of the structure directly in the fourth argument of `pthread_open()`, we first make a copy, using `malloc`, and pass this copy, that the new thread frees as soon as it has processed its arguments. Why didn't we pass the original structure in the first place?

A. Because the new thread would access a value that is located on the main thread's stack, resulting in a memory fault.

B. Because the original structure could be manipulated by the main thread before it is read by the new thread.

C. Because it would be less efficient.

D. More than one of A, B, C.

✗ ○ Answer A.

➡ ○ Answer B.

○ Answer C.

○ Answer D.

▷ View Feedback

Question 11 **1 / 1 point**

What is the difference between a global variable and a local static variable?

A. The number of instances of each variable in threads: global variables appear exist once for all the threads, local static variables have one instance per thread.

B. Where the variable is allocated: global variables are allocated in the `data` or `bss` segment, while local static variables are allocated on the stack.

C. The scope of the variable: global variables can be used everywhere, while local static variables are only accessible within the scope of their definition.

D. More than one of A, B, C.

- ◯ Answer A.
- ◯ Answer B.
- ✔ ◯ Answer C.
- ◯ Answer D.

▷ View Feedback

Question 12 **0 / 1 point**

What is the role of the `volatile` keyword?

A. It is used in function definitions. It indicates to the compiler that the function will have a side effect.

B. It is used in variable definition. It forces all references to the variable to be made from memory, rather than storing it in a register.

C. It is used in variable definition. It asks the compiler to deallocate the variable as soon as it detects it is not referenced anymore, to free up space.

D. It is used in function definitions. It asks the compiler to never *inline* the function (a function is *inlined* if calls to it are replaced by the actual body of the function, in the resulting assembly).

✖ ◯ Answer A.

➡ ◯ Answer B.

◯ Answer C.

◯ Answer D.

▷ View Feedback

Question 13 0 / 1 point

Why isn't incrementing a value in memory thread-safe?

A. Because it consists of three phases: fetch the value in a register, increment the register, then write it back to memory. A thread can be interrupted in the middle, and, when resumed, have an outdated value in the register.

B. Because accessing a value in memory can cause a page fault exception, and during the loading of the page in memory, another thread could be executed.

C. Because accessing a value in memory can cause a page fault exception, and block all the other threads until the page is loaded in memory, resulting in poor performances.

➡ ◯ Answer A.

◯ Answer B.

✖ ◯ Answer C.

▷ View Feedback

Question 14 0 / 1 point

Semaphores are used in particular for mutual exclusion, but what is a semaphore, in terms of type? In other words, what is `sem_t`?

A. A structure.

B. A `char`.

C. An integer.

D. A pointer.

✗ ◯ Answer A.

◯ Answer B.

➡ ◯ Answer C.

◯ Answer D.

▷ View Feedback

Question 15 1 / 1 point

The two operations on a semaphore are traditionally called `P` and `V`. What are their roles?

A. `P(s)` suspends execution until `s` is nonzero, then decrements it. `V(s)` increments `s`.

B. `V(s)` tests if `s` is zero, and increments it if it is, decrements it otherwise. `P(s)` zeroes out `s`.

C. `V(s)` suspends execution until `s` is nonzero, then decrements it. `P(s)` increments `s`.

D. `P(s)` tests if `s` is zero, and increments it if it is, decrements it otherwise. `V(s)` zeroes out `s`.

✓ ◯ Answer A.

◯ Answer B.

Answer C.

Answer D.

View Feedback

Question 16 0 / 1 point

Why are `P` and `V` implemented using system calls, rather than just normal memory manipulations?

A. Because they reference a variable shared across threads.

B. To ensure atomicity of their operations.

C. To avoid page faults.

D. To make them faster.

Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 17 1 / 1 point

In the course, we use `P` and `V`, and these are implemented using (actually, just wrappers around) some functions implemented in the libc. Which ones?

A. `P` is a wrapper around `sem_wait`, and `V` is a wrapper around `sem_post`.

B. `V` is a wrapper around `sem_post`, and `P` is a wrapper around `sem_init`.

C. `P` is a wrapper around `sem_post`, and `V` is a wrapper around `sem_wait`.

D. `V` is a wrapper around `sem_init`, and `P` is a wrapper around `sem_post`.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 18 1 / 1 point

Why is `volatile` (on global variables) considered harmful in multithreaded applications?

A. Because it is useless in practice, since library calls will prevent global variables to be put in registers.

B. Because it prevents a lot of optimizations by the compiler.

C. Because it provides a false sense of atomicity.

D. Two of A, B, C.

E. All of A, B, C.

Answer A.

Answer B.

Answer C.

Answer D.

✔ Answer E.

View Feedback

Question 19 1 / 1 point

In the theory of semaphores, what is a mutex?

A. A semaphore taking only values 0 and 1 used for mutual exclusion (`P` locks the mutex, `V` releases it).

B. Mutexes are different from semaphores, and semaphores cannot implement mutexes.

C. A design pattern that uses semaphores to avoid concurrent execution of critical portions of code.

D. A type of semaphore implemented using a single bit.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 20 0 / 1 point

In the producer-consumer problem, a pool of producers puts data in a fixed-size buffer, and a pool of consumer reads and removes data from the buffer. What is the problem that semaphores solves?

A. How to wait for data to arrive.
B. How to schedule producers and consumers.
C. Concurrent access to the buffer, so that it is in a consistent state at all time.
D. How to wait for free space to be available in the buffer.
E. Two of A, B, C, D.
F. Three of A, B, C, D.
G. All of A, B, C, D.

- ◯ Answer A.
- ◯ Answer B.
- ◯ Answer C.
- ◯ Answer D.
- ◯ Answer E.
- ➡ ◯ Answer F.
- ✖ ◯ Answer G.

▷ View Feedback

Question 21 **0 / 1 point**

What are the semaphores used solving the producer-consumer problem?

A. Just one mutex for accessing the shared buffer.

B. Two counting semaphores to signal production and consumption.

C. One mutex for modifying the buffer, and one counting semaphore to signal both production and consumption.

D. One mutex for modifying the buffer, and two counting semaphores to signal production and consumption.

- ◯ Answer A.
- ◯ Answer B.
- ✖ ◯ Answer C.
- ➡ ◯ Answer D.

▷ View Feedback

Question 22 1 / 1 point

In the readers-writers problem, some reader threads want to read a shared object, and some writer threads want to write to it. This could be solved using a single mutex, ensuring that no two threads have access to the object at the same time. Why is this very much suboptimal?

A. Because multiple writers should be allowed to write to the object at the same time.

B. Because one writer and one reader should be allowed to access the object at the same time.

C. Because multiple readers should be allowed to read the object at the same time.

- ◯ Answer A.

Answer B.

✔ Answer C.

View Feedback

Attempt Score:8 / 22

Overall Grade (highest attempt)**:**8 / 22

Done