Timothy Holmes (username: THOLME15)

Attempt 2

Written: Nov 9, 2020 9:42 PM - Nov 9, 2020 9:43 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1 — 1 / 1 point

In a prethreaded concurrent server, a master thread accepts all connections, and puts the connection file descriptors in a shared buffer. A worker from a fixed set of worker threads then gets a file descriptor from the buffer, and serves it. Why wouldn't this work as is with a process-based (fork-based) server, even assuming that the buffer is properly shared between the processes using `mmap`?

A. Because memory shared by `mmap` cannot be modified; it is made read-only as soon as at is flagged as shared.

B. Because different processes have different file descriptor tables, so a master's file descriptor makes no sense to the worker.

C. Because it is not possible to create a pool of worker child processes.

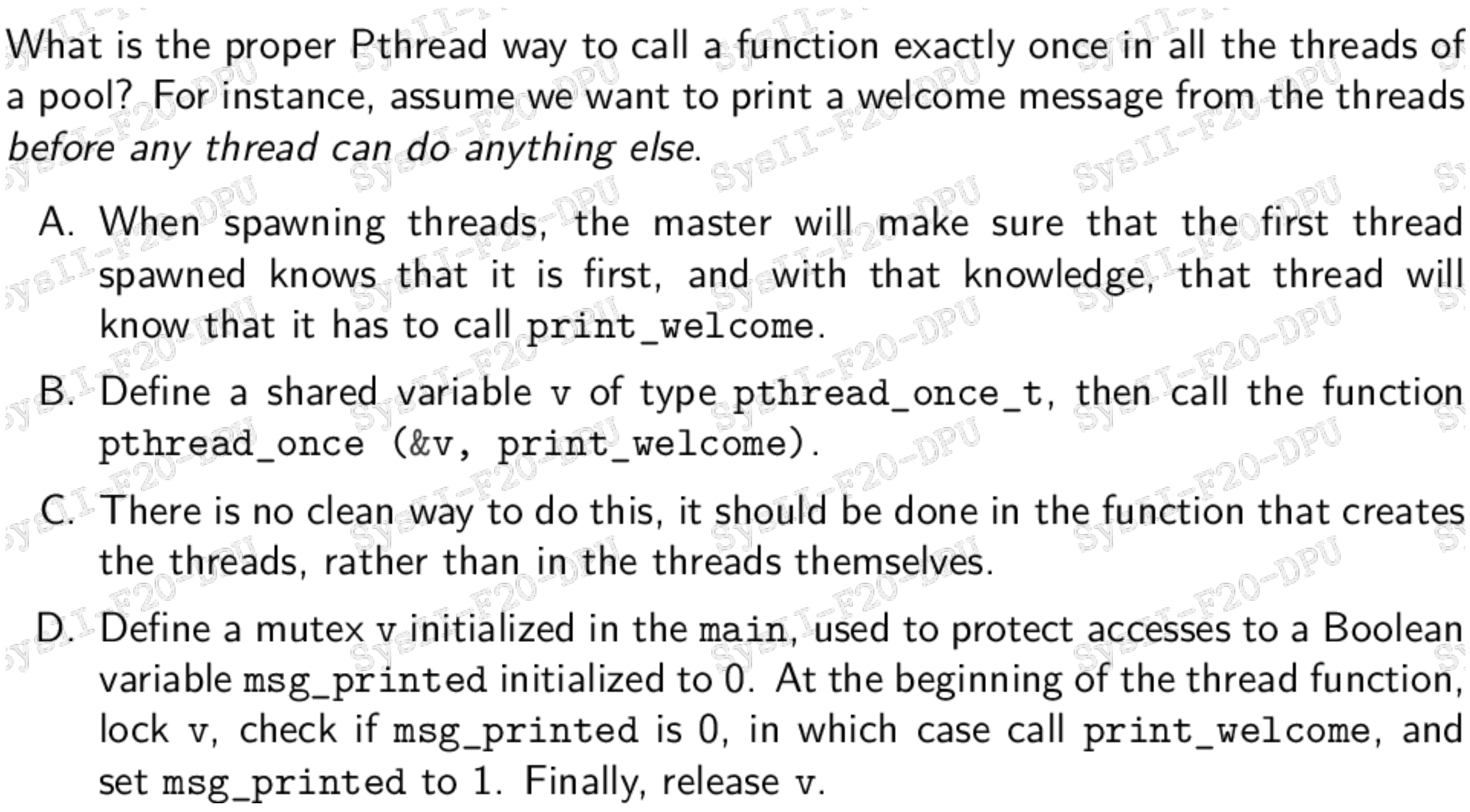
D. Because the buffer needs to be modified not only by the master, to put file descriptors in it, but also the workers, to remove items from it, and `mmap` only allows one process to write to shared memory.

- ○ Answer A.
- ✓ ○ Answer B.
- ○ Answer C.
- ○ Answer D.

▷ View Feedback

Question 2

1 / 1 point



What is the proper Pthread way to call a function exactly once in all the threads of a pool? For instance, assume we want to print a welcome message from the threads *before any thread can do anything else*.

A. When spawning threads, the master will make sure that the first thread spawned knows that it is first, and with that knowledge, that thread will know that it has to call `print_welcome`.

B. Define a shared variable `v` of type `pthread_once_t`, then call the function `pthread_once (&v, print_welcome)`.

C. There is no clean way to do this, it should be done in the function that creates the threads, rather than in the threads themselves.

D. Define a mutex `v` initialized in the `main`, used to protect accesses to a Boolean variable `msg_printed` initialized to 0. At the beginning of the thread function, lock `v`, check if `msg_printed` is 0, in which case call `print_welcome`, and set `msg_printed` to 1. Finally, release `v`.

- ○ Answer A.
- ✓ ○ Answer B.
- ○ Answer C.
- ○ Answer D.

▷ View Feedback

Question 3

1 / 1 point

What is a reentrant function?

A. A function that protects its accesses to shared variables with mutexes.
B. A function that saves all its local variables in registers.
C. A function that can call itself.
D. A function that accesses no shared variables when called from multiple threads.

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

View Feedback

Question 4 1 / 1 point

What is lock-and-copy?

A. A technique to make some thread-unsafe functions safe by making their calls mutually exclusive, and making a private copy of their return values.
B. A technique that can turn some non-reentrant functions into reentrant functions.
C. A type of functions that are thread-safe by design.
D. A technique that relies on mutexes to make sure that a long read in memory is not interrupted by a write, leading to an inconsistent read.

- ✓ ◯ Answer A.
- ◯ Answer B.
- ◯ Answer C.

- Answer D.

View Feedback

Question 5 1 / 1 point

What is a race condition?

A. Any bug that occurs only under certain thread scheduling.

B. A scheduling quirk that leads the master thread to be executed much more than the other threads.

C. A race for acquiring a specific resource between two or more threads, that results in one thread to be locked waiting.

D. Any bug that occurs as a result of accessing a memory location without mutexes.

- [✓ Answer A.]
- Answer B.
- Answer C.
- Answer D.

View Feedback

Question 6 1 / 1 point

When are race conditions significantly more likely?

A. When the system has lots of RAM (more than 8GB).

B. When the CPU is single core.

C. When the CPU has multiple cores.

D. When the system has little RAM (less than 512MB).

- ○ Answer A.
- ○ Answer B.
- [✓ ○ Answer C.]
- ○ Answer D.

▷ View Feedback

Question 7 1 / 1 point

What is a deadlock?

A. A situation in which a process or a thread waits for a condition that will never be true.

B. A situation in which a process or thread crashes as a result of a seemingly unlikely scheduling.

C. A situation in which a process or thread waits for a resource to be available, but every time it is made available, another process or thread takes control of the resource.

D. A situation in which a thread waits for another thread to finish, but the other thread is waiting for a resource to be made available.

- [✓ ○ Answer A.]
- ○ Answer B.
- ○ Answer C.
- ○ Answer D.

▷ View Feedback

Question 8 1 / 1 point

What is the lowest level of parallelism among these parallel computing mechanisms?

A. Out-of-order instruction processing.
B. Cloud computing.
C. Multicore processing.
D. Hyperthreading.

[✓ Answer A.]

Answer B.

Answer C.

Answer D.

View Feedback

Question 9

1 / 1 point

What is the relationship between deadlocks and race conditions?

A. These are two distinct concepts, with no relation.
B. Deadlocks often result from race conditions.
C. Race conditions often result from deadlocks.
D. They describe the same class of problems.

Answer A.

[✓ Answer B.]

Answer C.

Answer D.

View Feedback

Question 10 1 / 1 point

What is hyperthreading?

A. The ability of a multicore CPU to recognize that two threads come from the same process, and force their scheduling on a single core to reduce the cost of context switch.

B. The duplication of registers and operation queues within a core, to have multiple threads executing on a single core, sharing functional units.

C. The ability of compilers to optimize the order of assembly instructions to maximize the use of functional units.

D. The duplication of functional units in order to increase the size of the operation queues, and process more instructions in parallel.

- Answer A.
- ✓ Answer B.
- Answer C.
- Answer D.

View Feedback

Question 11 1 / 1 point

If one wants to optimize parallel computation, what should be avoided?

A. Accessing shared variables.

B. Relying on mutexes.

C. Creating an unbounded number of threads.

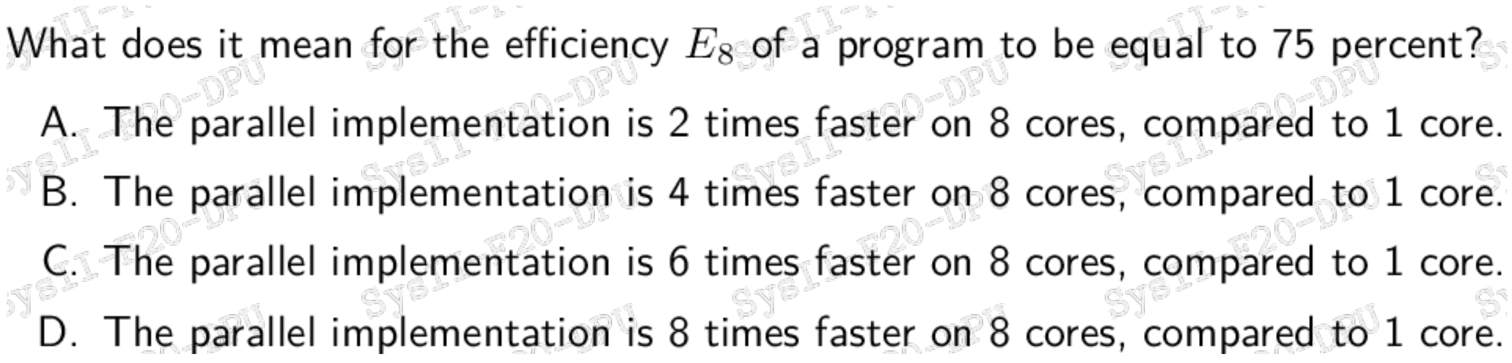
D. Two of A, B, C.

E. All of A, B, C.

- ○ Answer A.
- ○ Answer B.
- ○ Answer C.
- ○ Answer D.
- ✓ ○ Answer E.

View Feedback

Question 12 1 / 1 point


What does it mean for the efficiency E_8 of a program to be equal to 75 percent?

A. The parallel implementation is 2 times faster on 8 cores, compared to 1 core.
B. The parallel implementation is 4 times faster on 8 cores, compared to 1 core.
C. The parallel implementation is 6 times faster on 8 cores, compared to 1 core.
D. The parallel implementation is 8 times faster on 8 cores, compared to 1 core.

- ○ Answer A.
- ○ Answer B.
- ✓ ○ Answer C.
- ○ Answer D.

View Feedback

Question 13 1 / 1 point

In the parallel implementation of quicksort, we spawned a new thread for each recursive call. This led to great performance gains. Is it always the case for recursive functions?

A. Yes, this is a universal technique to get speedups from parallel computing.
B. No, it only works because the number of concurrent threads was bounded by a small number in practice.
C. Yes, but only for *tail* recursive functions.
D. No, it only works because the recursion depth is small (≤ 32).

○ Answer A.
✓ ○ Answer B.
○ Answer C.
○ Answer D.

View Feedback

Question 14 — 1 / 1 point

Why is the notion of *absolute* speedup a better metric for the benefits of parallelism than *relative* speedup?

A. Because it compares the parallel implementation to the sequential implementation.
B. Because it compares the parallel implementation to the ideal asymptotic complexity.
C. Because it compares a certain number of cores to just one core.
D. Because it doesn't make the speedup relative to the number of cores.

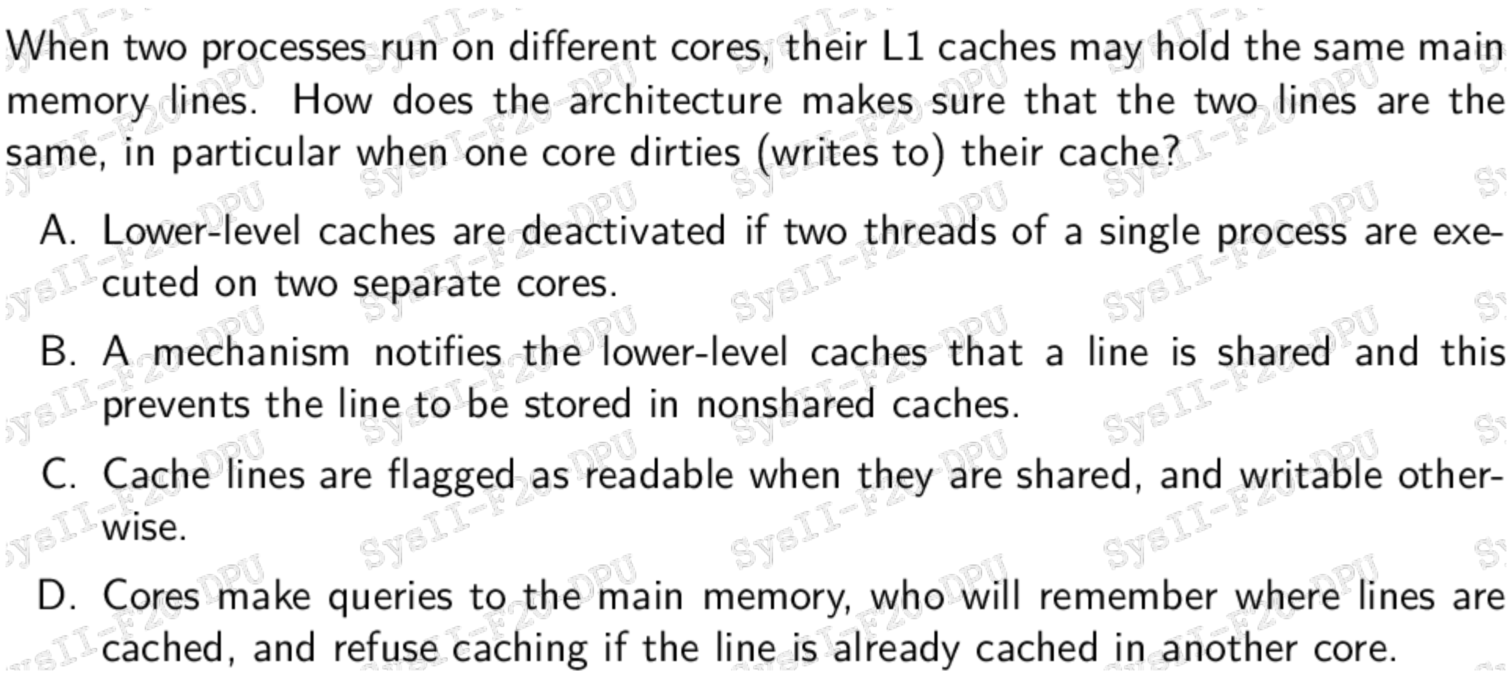
✓ ○ Answer A.
○ Answer B.

- Answer C.
- Answer D.

View Feedback

Question 15

1 / 1 point



When two processes run on different cores, their L1 caches may hold the same main memory lines. How does the architecture makes sure that the two lines are the same, in particular when one core dirties (writes to) their cache?

A. Lower-level caches are deactivated if two threads of a single process are executed on two separate cores.

B. A mechanism notifies the lower-level caches that a line is shared and this prevents the line to be stored in nonshared caches.

C. Cache lines are flagged as readable when they are shared, and writable otherwise.

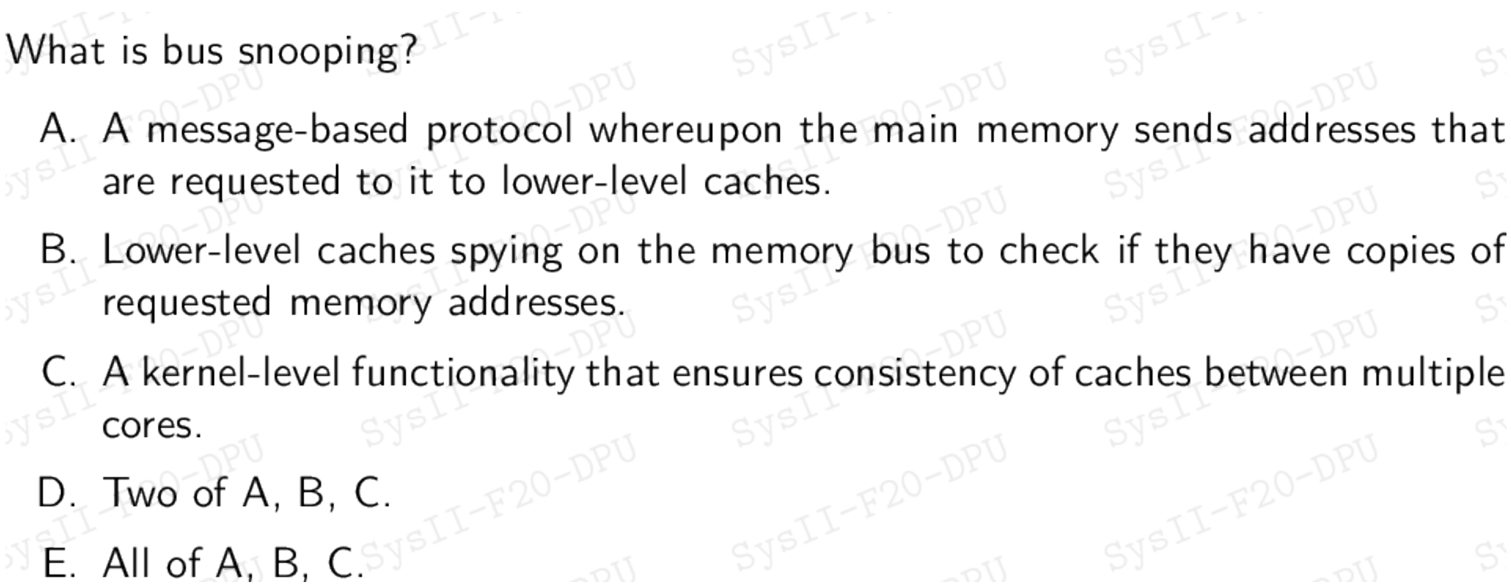
D. Cores make queries to the main memory, who will remember where lines are cached, and refuse caching if the line is already cached in another core.

- Answer A.
- Answer B.
- ✓ Answer C.
- Answer D.

View Feedback

Question 16

1 / 1 point



What is bus snooping?

A. A message-based protocol whereupon the main memory sends addresses that are requested to it to lower-level caches.

B. Lower-level caches spying on the memory bus to check if they have copies of requested memory addresses.

C. A kernel-level functionality that ensures consistency of caches between multiple cores.

D. Two of A, B, C.

E. All of A, B, C.

- ○ Answer A.
- ✓ ○ Answer B.
- ○ Answer C.
- ○ Answer D.
- ○ Answer E.

▷ View Feedback

Attempt Score:16 / 16

Overall Grade (highest attempt)**:**16 / 16

Done