Quiz Submissions - Week 1 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 5

Written: Sep 15, 2020 5:36 PM - Sep 15, 2020 5:39 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1 — 1 / 1 point

Which statement is *incorrect?*

A. SRAM is much more expansive than DRAM per bit.

B. SRAM is about 10 times faster than DRAM.

C. SRAM requires more transistors per bit than DRAM.

D. DRAM requires constant rewrite (*refresh*) of the data, but SRAM does not.

E. None of the above, they are all correct.

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

View Feedback

Question 2 — 1 / 1 point

Rank these storage types by read speed, from fastest to slowest.

A. DRAM, SRAM, SSD, rotating disk.

B. SRAM, DRAM, rotating disk, SSD.

C. SRAM, SSD, DRAM, rotating disk.

D. SRAM, DRAM, SSD, rotating disk.

E. DRAM, SSD, SRAM, rotating disk.

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

View Feedback

Question 3 **1 / 1 point**

ROM are used for (or in):

A. Solid state disks.

B. Firmware programs that are unlikely to change.

C. Disk caches.

D. All of the above.

E. None of the above.

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

Answer E.

View Feedback

Question 4

1 / 1 point

What do RAM and ROM stand for?

A. Random Access Memory and Random Order Memory.

B. Read Allowed Memory and Random Only Memory.

C. Random Access Memory and Read Only Memory.

D. Read Access Memory and Read Order Memory.

Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 5

1 / 1 point

Reading "$>$" as "consists of", which inclusion sequence is correct for common disks?

A. Disk $>$ surfaces $>$ platters $>$ tracks $>$ sectors.

B. Disk $>$ platters $>$ surfaces $>$ tracks $>$ sectors.

C. Disk $>$ platters $>$ tracks $>$ surfaces $>$ sectors.

D. Disk $>$ platters $>$ sectors $>$ surfaces $>$ tracks.

E. Disk $>$ surfaces $>$ platters $>$ sectors $>$ tracks.

Answer A.

✔ Answer B.

Answer C.

Answer D.

Answer E.

View Feedback

Question 6

1 / 1 point

What is *not* part of a common rotating disk drive?

A. A spindle.

B. A processor.

C. A mechanical read/write head.

D. Some DRAM.

E. Some ROM.

F. None of the above, they are all common on hard drives.

Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

✔ Answer F.

View Feedback

Question 7

1 / 1 point

The access time in a disk is the sum of:

A. seek time, execution time, transfer time.

B. seek time, loading time, execution time.

C. wait time, rotational latency, transfer time.

D. wait time, loading time, execution time.

E. seek time, rotational latency, tranfer time.

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

View Feedback

Question 8 **1 / 1 point**

When the CPU requests information from the disk, how does it wait for the information to arrive?

A. It probes the disk every N cycles to check if the information is ready.

B. It does not wait, it will receive a message when the information is ready in memory, and can do something else in the meantime.

C. It does not have to wait, since the disk is fast enough.

D. It stops executing instructions until the information is available.

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

View Feedback

Question 9

1 / 1 point

How does the CPU know in which surface/track/sector is the data it is searching for?

A. It precomputes a transaction table that is stored in the cache.

B. It does not; the disk provides an abstract view as a 1-dimensional array of sectors.

C. It asks an extra component on the CPU chip that is designed to find that information.

D. It does not; requests to the disk are made using file names.

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

View Feedback

Question 10

1 / 1 point

Locality is the idea that programs tend to use data and instructions with addresses near or equal to those they have used recently. What are the definitions of *temporal* and *spatial* locality?

A. Temporal: items that have been recently referenced tend to be re-referenced. Spatial: items with nearby addresses tend to be referenced close together in time.

B. Temporal: the less local, the more time a program needs. Spatial: the less local, the more memory a program needs.

C. Temporal: small chunks of program execute faster. Spatial: programs that use a small amount of space run faster.

D. Temporal: within small time windows, only few items are read from memory. Spatial: conversely, within a small memory window, only few instructions read that content.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 11 **1 / 1 point**

SSDs are made up of a set of EEPROMS. When writing a byte to a (functioning) SSD, what is the worst case situation?

A. A whole *page*, i.e., a *part* of an EEPROM, has to be temporarily erased.

B. A whole *block*, i.e., a set of pages corresponding to a whole EEPROM, has to be temporarily erased.

C. Multiple blocks have to be temporarily erased.

D. The whole SSD has to be temporarily erased.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 12

1 / 1 point

Consider a `for` loop that accesses completely random positions in a large array.

A. The loop still exhibits *some* temporal locality, since it accesses the same instructions at each iteration.

B. The loop still exhibits *some* spatial locality, since the random accesses are all done on the same array.

C. Both A and B.

D. The loop has neither temporal nor spatial locality.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 13

1 / 1 point

A *hit* happens when the data requested is in a given cache. Otherwise, a *miss* occurs. What are the different types of miss?

A. block, line, set.

B. front, back, sideways.

C. register, cache, main memory.

D. cold, conflict, capacity.

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

View Feedback

Question 14 **1 / 1 point**

The main idea behind the memory hierarchy is:

A. Exploit multi-threading to access different data in a given level in parallel.

B. Exploit fast Internet access to backup data instantly to the cloud.

C. Exploit locality to have storage that costs as much as the cheap storage, but performs as well as the expensive storage.

D. Exploit low-level parallelism to access different levels of the hierarchy at the same time.

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

View Feedback

Question 15

1 / 1 point

A line in a cache contains at least these info:

A. A dirty bit, the set number, and a part of the lower level memory.

B. A map between some cached data and their actual address in memory.

C. The cache number, its type, and the data stored.

D. A valid bit, the tag, and the set of bytes cached.

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

View Feedback

Question 16

1 / 1 point

The typical implementation of caches depends on three parameters S, E, B. What do these letters correspond to, respectively?

A. Number of lines, number of elements, binary size.

B. Number of sets, number of edges, number of blocks.

C. Number of sets, number of lines, block size.

D. Number of caches, number of subcaches, byte size.

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

View Feedback

Question 17

1 / 1 point

A cache where each set has only one line is called:

A. A 0-way set associative cache.

B. A linear cache.

C. A direct mapped cache.

D. Such a cache cannot exist.

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

View Feedback

Question 18

1 / 1 point

How is an address in memory translated to a line in a cache?

A. The cache stores a map between the memory addresses that it has seen, and the lines in the cache.

B. The cache is always as big as the whole memory, so memory addresses of the memory are also valid in the cache.

C. Some bits of the address are interpreted as an index in a list of *sets*, then some other bits correspond to a *tag* that is searched among the lines of that set.

D. By hashing the (say) 64-bit address to a (say) 8-bit address, and using this as the cache address.

- Answer A.
- Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 19

1 / 1 point

Since memory is organized as a hierarchy that duplicates information, multiple copies of data exist (in fact, if some DRAM data is in the L1 cache, it is also in all the caches in between). In the event the CPU writes at some address that *is not* in a given cache (miss):

A. The data is written straight to memory, without using the cache.

B. The data is loaded into the cache, then modified in the cache.

C. A mix of A and B, depending on policy.

D. None of these options: caches are read-only.

Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 20

1 / 1 point

Since memory is organized as a hierarchy that duplicates information, multiple copies of data exist (in fact, if some DRAM data is in the L1 cache, it is also in all the caches in between). In the event the CPU writes at some address that *is* in a given cache (hit):

A. The data is immediately written to the main memory.

B. The data is overwritten in the cache, and when the line is flushed, it is written back to memory.

C. A mix of A and B, depending on policy.

D. None of these options: caches are read-only.

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

View Feedback

Question 21 **1 / 1 point**

In a typical multi-core CPU, how are on-chip caches shared?

A. All caches are shared across all cores.

B. No caches are shared across cores, cores use the main memory to share data.

C. The L1 and L2 caches are shared, but there is one unshared L3 cache per core.

D. There are L1 and L2 caches for each core, and a bigger L3 cache shared across cores.

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

View Feedback

Question 22

1 / 1 point

The three important metrics to evaluate cache performance are:

A. read throughput, write throughput, latency.

B. refresh rate, hit rate, miss rate.

C. miss rate, miss time, hit penalty.

D. miss rate, hit time, miss penalty.

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

View Feedback

Question 23

1 / 1 point

Blocking is a technique that can improve temporal locality of code. But what is blocking?

A. Reworking a computation so that instead of working on scalars (e.g., ints), it works on sets of scalars that can fit in the cache.

B. Stopping the computation after a certain number of cycles, and restarting it at a different position in memory.

C. Stopping the computation for a certain number of cycle, so that the cache is passively flushed.

D. Carefully putting limits on the memory addresses accessible, and then relaxing these limits one by one.

- ✔ Answer A.

- Answer B.
- Answer C.
- Answer D.

View Feedback

Question 24

1 / 1 point

Consider a piece of code that has some nested loops, and some code in the innermost loop. This code matches closely a given simple specification in English. Then rearranging the order of the loops:

A. Can drastically improve spatial locality.

B. Has no impact on spatial locality.

C. Probably cannot improve spatial locality: since the English description was simple, the matching code has the best spatial locality.

- ✔ Answer A.
- Answer B.
- Answer C.

View Feedback

Attempt Score:24 / 24

Overall Grade (highest attempt)**:**24 / 24

Done

Quiz Submissions - Week 2 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 2

Written: Sep 21, 2020 10:46 PM - Sep 21, 2020 10:59 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1 — 1 / 1 point

Why do we use linkers in C/C++, rather than concatenating all the sources into a single file and compiling it into an executable?

A. The reason is purely historical.

B. This leads to faster programs.

C. This leads to programs that use less memory.

D. For modularity (small source files, libraries) and efficiency (compilation time, libraries take up less space).

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

View Feedback

Question 2 — 1 / 1 point

What are the two steps of the linking process?

A. 1. Symbol resolution, 2. Relocation

B. 1. Symbol minification, 2. Expansion

C. 1. Desymbolization, 2. Resymbolization

D. 1. Symbol relocation, 2. Resolution

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 3

1 / 1 point

What is the name of the section in an ELF file that contains the code?

A. `.text` B. `.code` C. `.rodata` D. `.bss` E. `.data`

✔ Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

View Feedback

Question 4

1 / 1 point

Which statement is incorrect?

A. ELF is a format for object files.

B. No ELF file can be linked.

C. ELF can be used for relocatable, executable, and shared object files.

D. There are ELF files that are not executable.

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

View Feedback

Question 5

1 / 1 point

Will the object file corresponding to the following C file be huge (more than 100MB)?

```
int x[100 * 1024 * 1024] = {1};
```

A. There is no way to know, it depends on the CPU architecture.

B. Yes, since space for `x` will be allocated in the object file, in the `.data` section.

C. No, it will be tiny, since `x` will be "stored" in the `.bss` section.

D. This is not a valid C file, it cannot compile to an object file.

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

View Feedback

Question 6

1 / 1 point

Will the object file corresponding to the following C file be huge (more than 100MB)?

```
int x[100 * 1024 * 1024];
```

A. No, it will be tiny, since `x` will be "stored" in the `.bss` section.

B. This is not a valid C file, it cannot compile to an object file.

C. Yes, since space for `x` will be allocated in the object file, in the `.data` section.

D. There is no way to know, it depends on the CPU architecture.

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

View Feedback

Question 7

1 / 1 point

What is the difference between a local non-static and a local static C variable, from the linker's point of view?

A. The linker treats them in the same way.

B. The linker knows nothing about static local variables, but can see the non-static local variables in the object file (`.bss` or `.data`).

C. The linker doesn't know about local variables, whether they be static or not.

D. The linker knows nothing about non-static local variables, but the static local variables are in the object file (`.bss` or `.data`).

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

View Feedback

Question 8

1 / 1 point

What is relocation?

A. The rewriting of parts of the assembly code of relocatable object files so that pointers and calls have the right addresses in the executable file.

B. The optimization of the positioning of data is the `.data` and `.bss` sections.

C. The internal use of libraries, in particular shared, to relocate code in shared memory.

D. The choice of order in which relocatable object files are merged into an executable files.

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

View Feedback

Question 9

1 / 1 point

Consider the statements `extern int x;` and `int x;`, at the global scope. Then:

A. These have the same effect.

B. If one file contains the first, and another file the second, they cannot be linked together.

C. Only the second one can trigger the creation of `x` in the `.bss` section.

D. The second is equivalent to `static int x;`

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

View Feedback

Question 10

1 / 1 point

Is it possible that an executable of less that 50KB, compiled from a valid C file, not using any nonstandard library, would fail to reach the *beginning* of the `main` function?

A. Yes, if the `.data` section is too big.

B. No, `main` *will* be reached.

C. Yes, if the `.code` section needs too much memory.

D. Yes, if the `.bss` section requires too much memory.

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

View Feedback

Question 11

1 / 1 point

I have a file `foo.c` that contains just:

```
int main () {
  return 0;
}
```

When I compile it to an object file (`gcc -c foo.c`), and call `readelf -a foo.o`, the output of `readelf` is 77 lines long. When I compile this to an executable (`gcc foo.o`), and call `readelf -a a.out`, the output of readelf is 257 lines long. Why?

A. `foo.o` has been linked with some system libraries to start the program.

B. This is because of the `LD_PRELOAD` environment variable.

C. gcc by default adds a lot of debug info when compiling to an executable.

D. The ELF format forces that executable objects be bigger than relocatable objects.

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

View Feedback

Question 12

1 / 1 point

What are the two main types of libraries?

A. Static and position-independent.
B. Static and shared.
C. Relocatable and dynamic.
D. Archived and non-archived.

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

View Feedback

Question 13 **1 / 1 point**

What is a `.so` file?

A. The Linux equivalent of a ZIP file.
B. An executable object file, but not in ELF.
C. One big object file.
D. A collection of object files.

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

View Feedback

Question 14

1 / 1 point

What is an archive (.a file)?

A. One big object file.

B. The Linux equivalent of a DLL.

C. An executable object file, but not in ELF.

D. A collection of object files.

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

View Feedback

Question 15

1 / 1 point

Assume a program is being linked with a static library. Then:

A. The whole library is added to the program.

B. Nothing happens: the linker is not aware of static libraries.

C. Only the relevant object files of the library are added to the program.

D. The linker checks if an equivalent dynamic library is available, and use it if that's the case.

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

View Feedback

Question 16

1 / 1 point

Assume a program is being linked with a dynamic library. Then:

A. Nothing happens: the linker is not aware of dynamic libraries.

B. Only relocation and symbol table info from the library are included in the program.

C. The linker translates the `.so` file into a `.a` file, and extracts the relevant parts of the library to put them in the program.

D. The whole code of the library is included in the program.

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

View Feedback

Question 17

1 / 1 point

Dynamic linking can happen at load-time or at run-time. What decides this?

A. Load-time: the *linker* (`ld`) is called when the program is executed, and resolves the undefined symbols. Run-time: the *programmer* calls the `ld` program using system calls, and asks for more libraries to be loaded.

B. Load-time: the *kernel* resolves unresolved symbols in the program, and automatically finds the libraries that provide them when starting it. Run-time: if the program calls an undefined function, the *kernel* tries to find the symbol in any library in `/usr/lib`, and loads it.

C. Load-time: the *linker* (`ld`) equips the program with a loader that silently loads, at startup, the dynamic libraries that were put on the command line. Run-time: the *programmer* uses functions (e.g., `dlsym`) to explicitly link with more libraries.

- Answer A.
- Answer B.
- ✔ Answer C.

View Feedback

Question 18

1 / 1 point

What is interpositioning?

A. A technique offered only by `ld`.

B. Having a library intercept calls to some functions silently.

C. A technique used solely for debugging.

D. A hacking technique that is illegal on non-UNIX machines.

- Answer A.
- ✔ Answer B.
- Answer C.

Answer D.

View Feedback

Question 19

1 / 1 point

What is the role of the `--wrap=X` option of `ld`?

A. The function `X` is replaced by a dummy function.

B. A library named `libX.so` is used to resolve references to the function `X`.

C. This creates a wrapper to the function `X` with debug information.

D. References to `X` are resolved as `__wrap_X`, and those to `__real_X` as the original `X`.

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 20

1 / 1 point

What is the role of the environment variable `LD_PRELOAD`, when a program is started?

A. If set, its value should be a text file listing the order in which libraries should be loaded.

B. If set, its value should be a library that will be loaded before any other library.

C. If set, the libraries required by the program are preloaded, rather than loaded when needed (*on demand*).

D. If set, all the libraries on the system will be loaded (debug only!).

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

View Feedback

Question 21

1 / 1 point

Arrange these exceptional control flow mechanisms by level of abstraction:

- Lowest level: closest to hardware,
- Mid-low level: kernel's job, not accessible to user processes
- Mid-high level: trigerring exceptional control flow from one process to another
- High level: exceptional control flow within a single process.

A. Context switch, exceptions, nonlocal jumps, signals.
B. Exceptions, context switch, signals, nonlocal jumps.
C. Signals, context switch, exceptions, nonlocal jumps.
D. Exceptions, signals, nonlocal jumps, context switch.
E. Signals, exceptions, nonlocal jumps, context switch.

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

View Feedback

Question 22

1 / 1 point

Why is exceptional control flow needed?

A. To react to external events (e.g., I/O, timer).

B. To escape the natural, single process control flow of CPU.

C. To react to internal events (e.g., assembly instruction failing).

D. To allow interprocess communication.

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

F. More than one of A, B, C, D, but not all.

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

View Feedback

Question 23

1 / 1 point

What is an exception (or interrupt)?

A. A purely physical event caused by setting one pin of the CPU to 1.

B. The CPU giving control to a portion of memory-resident code of the kernel in response to some event.

C. A way processes in an OS communicate between each other.

D. A nonrecoverable error in the CPU, that makes the computer crash.

E. More than one of the above.

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

View Feedback

Question 24

1 / 1 point

In a normally functioning computer, how often can you expect the CPU to receive exceptions?

A. Not at all if the computer is running smoothly.

B. Every tenth of a second (about 1 to 20 per second) .

C. Every few milliseconds (about 100 to 2000 per second).

D. Every few microseconds (about 100000 to 2000000 per second).

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

View Feedback

Question 25

1 / 1 point

Can a user process trigger an exception *explicitly*?

A. Yes, there is a specific assembly instruction that does this.

B. No, processes live in userspace, and have no way to access such low level details.

C. No, exceptions are only caused by the hardware.

D. Not directly: a process can only communicate with the kernel, and the kernel can then cause an exception.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 26 **1 / 1 point**

When an exception is triggered, how long does it take for the handler to be called?

A. It's immediate.

B. A few milliseconds.

C. Up to a second.

D. Exceptions can be ignored by the CPU.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 27

1 / 1 point

What is a system call?

A. Any function that is implemented by the `libc`.

B. A function implemented by the kernel, that a process can call by using an exception-based mechanism.

C. Any function that is *not* implemented by the `libc`.

D. A function implemented by low level libraries that are automatically linked with any program.

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

View Feedback

Question 28

1 / 1 point

Assume that a process tries to access some memory location, but for some reason, the kernel has flushed the data to disk temporarily (in a process known as *swapping*). The process executes the instruction:

```
movl $0xd, 0x80384d1
```

What happens next?

A. While processing the instruction, the CPU sees that the address is invalid, and executes a piece of code in the BIOS (Basic I/O System), stored on a very fast ROM on the motherboard. The CPU then reexecutes the instruction.

B. This triggers an exception, and the kernel takes control. It then loads the data from disk to memory, and returns control to the process, reexecuting the `movl` instruction.

C. The process crashes, since the kernel should make sure that this situation cannot occur by preemptively loading data from the disk before the instruction is executed.

D. A signal `SIGSEGV` is sent to the process, and the handler for that signal should make a system call to force the kernel to load the data from the disk.

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

View Feedback

Question 29

1 / 1 point

What is a process?

A. An *instance* of a running program.
B. The same thing as a processor.
C. An executable file.
D. The same thing as a program.
E. More than one of the above.

✔ Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

View Feedback

Question 30 **1 / 1 point**

On a modern OS, how are multiple processes running in parallel on a single-CPU system with one core?

A. Modern CPUs have a hardware way to create multiple *threads*, and each process is given one thread.
B. There is a chip that lives next to the CPU that is in charge of providing an *abstraction* that gives the illusion of parallelism.
C. The kernel frantically switches between all processes all the time, saving and restoring their states (register values, PC, ...), letting them run just for split seconds at a time.
D. Processes must call the `yield` system call to put themselves in the background and allow other processes to run.

Answer A.

- Answer B.
- ✔ Answer C.
- Answer D.

View Feedback

Question 31

1 / 1 point

What is a context switch?

A. The increment of an internal counter of the CPU; it indicates which process is currently running, and the increment triggers a switch between them.

B. A 3 step process: the kernel saves the state of the "current" process, then chooses another process, and loads its state back on the CPU, so that the other process is now running (*scheduled*).

C. When a process decides to put itself in the background so that one of its children is now in the foreground.

D. An interruption that is caught by a dedicated chip on the motherboard, whose job is to load in memory another process. The implementation of the chip is hardware-dependent.

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

View Feedback

Question 32

1 / 1 point

How is a context switch normally triggered?

A. When a process decides to put itself in the background.
B. With a process signal.
C. With a page fault exception.
D. With a timer/clock interrupt.

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

View Feedback

Question 33

1 / 1 point

Consider the directed graph whose set of nodes is the set of processes running at a given moment, and there's an edge (A, B) if B is a child of A. What *best* describes that graph?

A. A DAG (directed acyclic graph).
B. A forest (i.e., a set of disconnected trees).
C. A tree.
D. A binary tree.

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

View Feedback

Question 34

1 / 1 point

What happens when a child of a process terminates, but the parent does not read its return value immediately (*reaping*, using the `wait`... functions)?

A. The parent process will not be allowed to terminate.

B. The child becomes a *zombie process*, that occupies space in the process table of the kernel.

C. The parent process will receive periodical reminders that the child has terminated, via a signal.

D. After a few seconds, the child process times out and is removed from the process table of the kernel.

E. More than one of the above.

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

View Feedback

Question 35

1 / 1 point

Assume a program calls the system call `fork()`. This creates a child of the current process. The child is an exact duplicate of the parent process, with some subtleties; which ones?

A. The child PID is unique and different from the parent.

B. The return value of the `fork()` function is not the same in the two processes.

C. The child and parent processes run in *separate* memory spaces.

D. The function `getppid()` returns different values in the two processes.

E. More than one of A, B, C, D, but not all.

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

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

View Feedback

Attempt Score: 35 / 35

Overall Grade (highest attempt): 35 / 35

Done

Quiz Submissions - Week 3 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 4

Written: Sep 27, 2020 6:47 PM - Sep 27, 2020 6:58 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Exceptional Control Flow

Question 1 **1 / 1 point**

What is a signal?

A. The process equivalent of exceptions.
B. A way for processes to communicate between each other.
C. One of less than 32 possible messages.
D. A message sent by the kernel to a process.
E. All of the above.
F. Some but not all of the above.

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

View Feedback

Question 2

1 / 1 point

What is a signal handler?

A. A system call.

B. A kernel function that is called when a process sends a signal to another.

C. Another name for exception handler.

D. A user-level function that is called when the process receives a signal.

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

View Feedback

Question 3

1 / 1 point

Signals are sent if either requested by a process, or if the kernel decides to. What are examples of the kernel sending a signal to a process?

A. The kernel received an *exception* that needs to be directly forwarded to the current process itself (e.g., division-by-zero).

B. The kernel responded to a page fault *exception* triggered by a process, and the memory access was really invalid.

C. The kernel did a context switch on the process, and needs to let it known.

D. The kernel could not open a file requested by a `open()` system call.

E. A and B, but not the others.

F. A and C, but not the others.

G. B and C, but not the others.

H. B and D, but not the others.

I. Three or more of A, B, C, D.

- Answer A.
- Answer B.
- Answer C.
- Answer D.
- ✓ Answer E.
- Answer F.
- Answer G.
- Answer H.
- Answer I.

View Feedback

Question 4

1 / 1 point

A kernel will maintain, for each process, the list of pending signals that the process should receive. How is that list implemented, and what does this implementation entail?

A. It is a LIFO structure (last-in-first-out), so that each signal received is stored and delivered in the right order. Repeated signals of the same type are just added to the list.

B. It is an integer vector, counting how many times each type of signal needs to be delivered. The order in which the signals are received is lost.

C. It is a bit vector, where the bit at position i is set if the signal of type i is pending. Both the order in which the signals are received and the number of times each signal of a given type is received are lost.

D. It is not a list, but a single variable indicating which signal is pending, if any. There is no way two signals, even of different types, can be queued.

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

View Feedback

Question 5 **1 / 1 point**

Suppose that processes A, B, C are scheduled in this order, and that both A and B send a signal `SIGINT` to C. How many times will C's signal handler for `SIGINT` be called?

A. Twice.

B. Once.

C. Once or twice, there is no way to know.

D. Depending on whether B's signal cancels A's, this could be zero.

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

View Feedback

Question 6

1 / 1 point

After a call to `fork`, how is the process group ID (PGID) of the child process determined? (After `fork`, the PGID of the child can be changed using `setpgid`.)

A. It is set to the child's PID.

B. It is set to the parent's PID.

C. It is set to 1 (the PGID of `init`).

D. It is set to the parent's PGID.

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

View Feedback

Question 7

1 / 1 point

What is the purpose of the `SIGSTOP` signal, which cannot be caught, blocked, or ignored?

A. Terminate the process, and immediately remove it from the kernel's process table, not allowing its parent to reap it.

B. Suspend the process, so that the kernel never schedules the process for execution.

C. Zombify the process: terminate it, but make sure that it always stays in the kernel's process table.

D. Terminate the process, but allow its parent process to reap the child, reading its return value.

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

View Feedback

Question 8

1 / 1 point

Using the `/bin/kill` program, the user can send arbitrary signals to their processes. Recalling that signal 9 is `SIGKILL`, what is the difference between:

`kill -9 2388` and `kill -9 -2388`?

A. The first terminates *one* process, the second every process with PGID 2388.

B. The first terminates *one* process, the second recursively kills all the children of process 2388, from the lowest level to the highest level in the tree of processes.

C. The first terminates every process with a PID starting with 2388, the second terminates just the process with PID 2388.

D. The first terminates only process 2388, the second kills all the processes in the same process group as 2388.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 9

1 / 1 point

What is the difference between calling the command `/bin/kill -9 2388` in a shell, and using the system call `kill (2388, 9)` in C? (Signal 9 is `SIGKILL`.)

A. The shell command will check for permission before sending the signal, but not the system call.

B. No difference.

C. The shell command will kill the *process group* 2388, while the C function will kill just the process.

D. The shell command will try to kill the process with `SIGINT` to allow it to save some data, before killing it with `SIGKILL`.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 10

1 / 1 point

Suppose a process A is running, and sends a signal `SIGINT` to a process B. When will the signal handler of process B be called?

A. After process B has been executed, and just before it is to be unscheduled (i.e., before a context switch stopping B's execution).

B. Immediately: Process A is suspended, and control is passed to process B.

C. When B is scheduled to be executed, i.e., at the beginning of some context switch for B.

D. When the kernel unschedules A, and switches to another process, even if it is not B.

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

View Feedback

Question 11 **1 / 1 point**

What happens when a process A calls the system call `kill(pid, sig)`?

A. The call triggers the execution of `/bin/kill -sig pid`.

B. The CPU calls an exception handler in the kernel code (since the `syscall` instruction causes an interruption). The kernel then calls the signal handler of `pid` for signal `sig`. When the signal handler returns, A's execution resumes.

C. The CPU calls an exception handler in the kernel code (since the `syscall` instruction causes an interruption). This handler stores the signal `sig` in the list of pending signals for `pid`. When the kernel returns, A's execution resumes.

D. The `kill` function directly adds the signal `sig` to the pending signal list of `pid`, since pending signal lists are in shared memory.

- Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 12

1 / 1 point

Process A sends a signal `SIGINT` to B, but B is blocking (masking) `SIGINT`. What happens?

A. Process A's call to `kill` returns `-1`, and `errno` is set to `EINVAL`.

B. Process A is stuck as long as process B does not unblock `SIGINT`.

C. The signal is deleted, it will never be delivered.

D. The signal is not delivered, but it will be as soon as `SIGINT` is unblocked by B.

E. More than one of the above.

Answer A.

Answer B.

Answer C.

✔ Answer D.

Answer E.

View Feedback

Question 13

1 / 1 point

Assume process A has multiple nonblocked pending signals, and the kernel just treated one of them by calling the appropriate signal handler of A. What happens next?

A. The kernel clears all the signals, and resumes the normal execution of A.

B. The kernel resumes the normal execution of A, and will treat the other signals later.

C. The kernel treats another pending signal.

D. The behavior depends on the return value of the signal handler just called.

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

View Feedback

Question 14 **1 / 1 point**

In a given process A, can a signal handler be interrupted by a signal?

A. Yes, but it has to be a signal different than the one(s) currently handled.

B. Yes, and it could be the same signal as one that is currently handled.

C. No, during A's signal handling all signals are blocked.

D. No, during A's signal handling, calls to `kill` $(A$, `...)` return an error.

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

View Feedback

Question 15

1 / 1 point

Which function(s) change the list of signals blocked within a process?
A. `sigaddset`. B. `sigdelset`. C. `sigemptyset`. D. `sigprocmask`. E. `sigfillset`. F. All of those.

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

View Feedback

Question 16

1 / 1 point

Which of these are problems that arise with signal handling?

A. The process can be stuck in a signal handler and never return to its normal execution.

B. Local copies of global variables, made by the compiler, can become out of date if not declared as `volatile`.

C. Shared (global) data structures can become corrupted if the code that modifies them is interrupted.

D. If not saved and restored properly, the global `int errno` can indicate the result of another system call than the one just made in the code.

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

F. More than one of A, B, C, D, but not all.

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

View Feedback

Question 17

1 / 1 point

Which of these are *async-signal-safe* functions?

A. `malloc`
B. `exit`
C. `printf`
D. `write`
E. All of A, B, C, D.
F. More than one of A, B, C, D, but not all.

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

View Feedback

Question 18

1 / 1 point

Upon receiving a signal `SIGCHLD`, what is the proper way to count, using a variable `cnt`, how many children have terminated?

A. `cnt++`: each time a child terminates, the handler is triggered.

B. `if (wait(NULL)) > 0) cnt++;`: one can only be sure a child has terminated if it is reaped (i.e., has been waited for, and its return value read).

C. `while (wait(NULL) > 0) cnt++;`: one can only be sure a child has terminated if it is reaped, and `wait` would immediately return `-1` when no child is waiting to be reaped.

D. `while (waitpid(-1, NULL, WNOHANG) > 0) cnt++;`: one can only be sure a child has terminated if it is reaped, and `waitpid` is required to use the option `WNOHANG`, so that the function immediately returns `-1` when no child is waiting to be reaped.

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

View Feedback

Question 19

1 / 1 point

How can a process suspend itself until it receives a specific signal (say `SIGINT`)?

A. This is not possible.

B. By calling `kill(getpid(), SIGSTOP)` then `kill(getpid(), SIGINT)`.

C. By calling `pause()`: this suspends the process until *any* signal is received, the handler for `SIGINT` can then toggle a flag, so that the code can check whether `SIGINT` was received. If not, `pause()` is called again.

D. By first masking `SIGINT`, then calling `sigsuspend(mask)`, with `mask` unmasking `SIGINT`. The call to `sigsuspend` suspends the process until *any* signal set in `mask` is received, the handler for `SIGINT` then toggles a flag, so that the code can check whether `SIGINT` was received. If not, `sigsuspend` is called again.

E. This is possible, but not with the above options.

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

View Feedback

Question 20 1 / 1 point

What is a nonlocal jump?

A. A way to jump to the execution of another process of *any* program.

B. A user-space mechanism to save the state of the CPU and restore it later.

C. A `goto` statement that goes *accross* functions (i.e., the label can be defined in another function).

D. A way to jump to the execution of another process of the *same* program.

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

View Feedback

Question 21

1 / 1 point

What are nonlocal jumps used for?

A. They are a modern replacement for signals.
B. To simplify control flow in complex multithreaded applications.
C. In error recovery from deeply nested calls.
D. They are a cleaner version of `goto`s.

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

View Feedback

Question 22

1 / 1 point

What is the difference between these two programs:

```
int main () {
  int x = 0;
l:
  printf ("%d\n", x++);
  goto l;
}
```

```
jmp_buf b;

int main () {
  volatile int x = 0;
  setjmp (b);
  printf ("%d\n", x++);
  longjmp (b, 0);
}
```

A. The second code does not compile.

B. They run in the same way, but the second code creates more overhead (i.e., runs slightly slower, and uses more memory).

C. They run correctly, but produce different outputs.

D. The second code crashes.

E. They run in the same way, and end up executing the same assembly instructions. (No difference.)

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

View Feedback

Question 23

1 / 1 point

What is the main limitation of nonlocal jumps?

A. They can only be used to jump up in the call stack.

B. They can only be used to jump in functions in the same C file.

C. They can only be used in processes with admin credentials (`root`).

D. They can only be used if all signals are blocked.

E. More than one of the above.

✔ Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

View Feedback

Question 24 **1 / 1 point**

Consider the following code:

```
jmp_buf b;

int main () {
  int x = 0;
  setjmp (b);
  printf ("%d\n", x++);
  longjmp (b, 0);
}
```

When compiled with `gcc -O0` (no optimizations), this prints a sequence of numbers, but with `gcc -O1`, this only prints 0s. Why?

A. Without optimizations, `x` is in a register, and is unaffected by the `longjmp`. With optimizations, `x` is on the stack, and it would be saved on `setjmp` and restored by `longjmp`, having always a value of 0.

B. This is a known bug in `gcc` when dealing with nonlocal jumps.

C. Without optimizations, `x` is on the stack, and is unaffected by the `longjmp`. With optimizations, `x` is in a register, and it would be saved on `setjmp` and restored by `longjmp`, having always a value of 0.

D. This is because `longjmp` is called with 0 as second argument, which gets optimized out. This would work as expected with `longjmp(b, x)`.

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

View Feedback

Question 25

1 / 1 point

Is the `setjmp/longjmp` technique a system call, or is it entirely implemented by the `libc`?

A. It is only register manipulation, so this is implemented in the libc, in assembly.

B. This is a low-level control flow disruption, so it is implemented by a hardware mechanism.

C. This is an exceptional control flow, so it has to be implemented by the kernel.

D. This depends on the *distance* (in terms of memory addresses) between the `setjmp` and the `longjmp`.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Virtual Memory

Question 26

1 / 1 point

Virtual memory pages can be in three states; what are they?

A. Unallocated, cached, uncached.

B. Allocated, unallocated, uncached.

C. Allocated, unallocated, cached.

D. Allocated, cached, uncached.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 27

1 / 1 point

Why do we use virtual memory (VM)?

A. Efficiency: the DRAM is used as cache.
B. Consistency: memory looks the same to all processes.
C. Safety: different processes' address spaces are isolated.
D. Locality: VM improves spatial locality.
E. All but A.
F. All but B.
G. All but C.
H. All but D.
I. All of A, B, C, and D.

Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

Answer F.

Answer G.

✔ Answer H.

Answer I.

View Feedback

Question 28

1 / 1 point

What happens if the CPU tries to access an uncached page?

A. Some hardware on the DRAM chip is in charge of loading the data from the disk into the DRAM, then reexecuting the offending instruction.

B. This causes a page fault exception, and the kernel's handler kills the process that caused the access.

C. Some hardware on the CPU chip is in charge of loading the data from the disk into the DRAM, then reexecuting the offending instruction.

D. This causes a page fault exception, and the kernel's handler brings the page from the disk to the DRAM, then reexecutes the offending instruction.

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

View Feedback

Question 29

1 / 1 point

What does it mean for a page to be in the uncached state?

A. It is not in the caches between L1 and the DRAM (that is, neither L2 nor L3).

B. It is not initialized.

C. It is not in the DRAM.

D. It is not in the L1 cache.

- Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 30

1 / 1 point

VM is a nontrivial mechanism, but it *simplifies* memory management. How?

A. Processes can have a uniform, linear view of memory.

B. Multiple physical pages can be mapped to a single virtual page, allowing for multiple accesses with one query.

C. Multiple virtual pages can be mapped to a single physical page, allowing sharing code and data.

D. The location of a physical page in the DRAM can be changed without impacting the view of the process.

E. All but A.

F. All but B.

G. All but C.

H. All but D.

I. All of A, B, C, and D.

Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

✔ Answer F.

- Answer G.
- Answer H.
- Answer I.

View Feedback

Question 31

1 / 1 point

What is thrashing?

A. When processes on a computer are repeatedly accessing more memory than available on the DRAM, and pages are swapped in and out constantly.

B. A system call that asks the kernel to put all the memory to disk, so that future VM accesses will have to load data from the disk, forcing only the recently used data to be on the DRAM.

C. When the main memory is so saturated that only one process can run.

D. A system call that asks the kernel to put all the memory to disk, so that the computer can be suspended.

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

View Feedback

Question 32

1 / 1 point

What is the "global variable" that indicates the upper limit of the heap?

A. `%brk`, a register stored on the CPU, but not accessible to the user.

B. There is no such thing; if an access is made outside of the heap, the heap is automatically extended.

C. `brk`, a variable stored in the process-specific kernel virtual memory.

D. `%rsp`, the stack pointer: the heap stops where the stack begins.

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

View Feedback

Question 33

1 / 1 point

When are the first few instructions of a program (from the `.text` section) loaded into DRAM?

A. When the process is first scheduled (i.e., the first context switch to the program).

B. When the CPU causes a page fault trying to read the first instruction, and the kernel loads the page from disk.

C. When the process receives an initial `SIGCONT` message.

D. During the execution of `execve`.

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

View Feedback

Question 34

1 / 1 point

How is VM enabling protection features?

A. It distinguishes addresses for the kernel's VM and addresses for the current process's VM.

B. It gives different virtual memory addresses to different processes.

C. It equips pages with read/write/exec flags.

D. It saves which process can access which physical page.

E. Only A and B.

F. Only A and C.

G. Only B and D.

H. Only C and D.

I. Three or more of A, B, C, D.

- Answer A.
- Answer B.
- Answer C.
- Answer D.
- Answer E.
- ✔ Answer F.
- Answer G.
- Answer H.
- Answer I.

View Feedback

Question 35

1 / 1 point

Where are the read/write/exec permission bits of a page stored?

A. In the kernel, in the per-process memory management structures.

B. In the physical DRAM, as a header to each physical page.

C. In a specific cache that is only accessible by the MMU.

D. In the page table entry, since they pertain to *virtual* memory pages.

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

View Feedback

Question 36

1 / 1 point

What is the job of the MMU?

A. Translate a virtual address to a physical address, if possible (and generate an exception if this is not possible, e.g., if the virtual page is uncached), then request that physical address to the memory.

B. Retrieve the page table entry corresponding to a virtual address, and return it to the CPU, so that it can fetch data from memory.

C. Fetch data from memory, hiding the details of caching and loading from disk to the CPU. Except if there is a permission violation, the MMU will always return some data.

D. Implement caching of the mapping between virtual addresses and physical addresses. It can then return the cached physical address to the CPU, or raise an exception if the virtual address is not cached.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 37

1 / 1 point

Is it possible that two processes have access to the same *physical* page, but only one can write in it?

A. Yes, the virtual pages of each process can map to the same physical page, and the permission bits are per-process.

B. No, processes can share a physical page, but the permission bits are per physical page.

C. No, processes cannot share a physical page, they need to be duplicated.

D. Yes, but if both processes try to write to the page, the page is automatically duplicated in physical memory.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 38

1 / 1 point

What is a page fault?

A. An exception that is raised by the MMU when the valid bit of the PTE corresponding to the VPN is 0 (meaning unallocated or uncached virtual page).

B. An exception raised by the DRAM when the requested physical address is invalid (meaning the page is unallocated).

C. A signal sent to a process when it tries to access an unallocated or uncached portion of memory.

D. An unrecoverable error in the CPU memory management, that causes the CPU to halt. This is usually triggered by a hardware fault.

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

View Feedback

Question 39 **1 / 1 point**

Assume that the MMU successfully translated a virtual address (VA) to a physical address (PA). How did it do it?

A. The VA is split in two bit vectors, VPN and VPO. The VPN is taken modulo a preset value, giving a value PPN. The PA is then this PPN concatenated with VPO.

B. The VA as a whole is used a pointer in a cached table that contains the corresponding PA.

C. The MMU calls a bit of code from the kernel, which is in charge of doing the translation.

D. The VA is split in two bit vectors, VPN and VPO. The VPN is basically used as an index in a table of PPNs. The PA is then the corresponding PPN concatenated with VPO.

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

View Feedback

Question 40

1 / 1 point

The PTE for a given VPN has a *valid* flag indicating whether the virtual page is allocated and cached in DRAM. Who updates that flag?

A. The MMU, after a page fault exception has been handled.

B. The kernel, upon loading the page in memory after a page fault, or evicting a page from memory.

C. The process whose virtual memory is currently looked at.

D. The CPU, using a mechanism known as *write-back*.

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

View Feedback

Question 41

1 / 1 point

What happens when the MMU requests a PA to the L1 cache, but the L1 cache misses?

A. The MMU generates a page fault.

B. The L1 cache asks for that PA to the L2 cache, and this can go all the way to the DRAM in case of successive misses.

C. The L1 cache asks for that PA to the L2 cache, and this can go all the way to the disk in case of successive misses.

D. The L1 cache generates a page fault.

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

View Feedback

Question 42 **1 / 1 point**

How is the virtual memory context updated when there is a process context switch?

A. The MMU receives a process-switch interruption together with a value, and updates a private register with that value. This register indicates where the current page table resides in physical memory.

B. The kernel, restoring the context of a process, also updates a special, hidden CPU register that points at the page table in physical memory.

C. There is no need to change anything; the process ID is embedded in the virtual address.

D. The kernel maintains a table mapping process ids to page tables; when resolving an address, the MMU fetches the correct page table using the current process id.

- Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 43

1 / 1 point

Which component updates entries in the page table of a process?

A. The MMU, after it sends a page fault exception.
B. The CPU, when it is restarted on a given instruction.
C. The process, when it receives a `SIGCONT` signal.
D. The kernel, in response to a page fault.

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 44

1 / 1 point

What is the role of the TLB (Translation Lookaside Buffer)?

A. Translate virtual addresses to physical addresses.
B. Store kernel information to better handle page faults.
C. Cache frequently used pages.
D. Cache page table entries.

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

View Feedback

Question 45

1 / 1 point

Recall that a query to a TLB is a virtual page number (VPN). How is a TLB implemented?

A. The VPN is taken modulo N, with N the size of the TLB, and the TLB returns the element at that position.

B. The VPN is split into two bit-vectors, a tag and a set index, and the TLB then behaves like a normal set-associative cache.

C. The VPN is used as a physical address that is queried in the DRAM (or its caches).

D. The VPN is searched within a balanced tree implemented in DRAM.

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

View Feedback

Question 46

1 / 1 point

Recall that a query to a TLB is just a virtual page number (VPN), without any extra information about the current process. What happens to the TLB when there's a context switch (scheduling another process)?

A. The info stored in the TLB is now invalid, since it refers to virtual addresses of the old process (the TLB is then usually just flushed).

B. All the caches of the processor package are cleared on context switch, including the TLB, so there is no outdated information in these.

C. The process ID of the current process is part of the VPN, so there's no conflict, and the TLB is left as is.

✔ Answer A.

Answer B.

Answer C.

View Feedback

Question 47 **1 / 1 point**

Why do we use multilevel page tables?

A. To speed up access using parallelism.

B. To deal with multiple processes.

C. For the same reason as we use caches: using a small amount of expensive memory as if it were a large amount of memory.

D. Because a single-level page table would be too big to store in memory.

E. More than one of the above.

Answer A.

Answer B.

Answer C.

✔ Answer D.

Answer E.

View Feedback

Question 48

1 / 1 point

Page tables are indexed by virtual page numbers (VPN). How is a k-level page table indexed? Assume the level-1 page table is given by a pointer P (the value of the PTBR), and that there is a PTE stored for that VPN.

A. The VPN is searched in the table P; if it isn't found, the search continues on the level-2 table, which is stored at *double* the address of the previous page table. This goes on until the VPN is found.

B. The VPN is divided in k bit-vectors, with values $i_1, i_2, \ldots, i_k$. The PTE for the VPN is $P[i_1][i_2]\cdots[i_k]$.

C. The VPN is hashed using a hashing function (e.g., MD5, SHA1), and that value is searched in the level-1 table P. If it isn't found, the value is hashed again, and searched in the level-2 table, which is just after the level-1 table. This goes on until the VPN is found.

Answer A.

✔ Answer B.

Answer C.

View Feedback

Attempt Score: 48 / 48

Overall Grade (highest attempt): 48 / 48

Done

Quiz Submissions - Week 4 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 2

Written: Oct 6, 2020 3:57 PM - Oct 6, 2020 4:03 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1 — 1 / 1 point

Is it possible that a page table entry (PTE) would indicate a nonzero PPN, but a zero valid flag?

A. Yes, this would correspond to an *uncached* page.

B. Yes, this would correspond to an *invalid* page.

C. Yes, this would correspond to an *unallocated* page.

D. No, if the valid flag is zero, the PPN has to be reset.

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

View Feedback

Question 2 — 1 / 1 point

What is the TLB doing with the VPO?

A. It uses it as the tag in its cache structure.
B. Nothing, it does not even receive it.
C. It uses it as the base offset in its cache structure.
D. It transfers it to the PPO.

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

View Feedback

Question 3 **1 / 1 point**

What is the dirty bit in a PTE?

A. A bit indicating that the page has been written to.
B. A bit indicating that the page on the disk that corresponds to the DRAM page has been discarded.
C. A bit indicating that any further write to the page should trigger a page fault.
D. A bit indicating that any further read to the page should trigger a page fault.

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

View Feedback

Question 4

1 / 1 point

What are memory features one can find in a modern CPU (such as the Intel Core i7)?

A. Multi-level page tables.

B. A cache hierarchy for the TLB.

C. A cache hierarchy for the DRAM.

D. A shared cache between cores.

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

F. Two or more of A, B, C, D, but not all.

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

View Feedback

Question 5

1 / 1 point

How does a kernel decide to terminate a process with a `SIGSEGV` signal?

A. If the kernel receives a page fault exception, then the memory access wasn't legal, either because it accessed an unallocated page, or because it tried to bypass protection bits. In both cases, the kernel immediately terminates the process.

B. Upon receiving a page fault exception, the kernel checks whether the process's access was authorized using the process's `vm_areas`. If it was, this means that the page wasn't in memory, and it needs to be loaded. Otherwise, the process is terminated.

C. The only acceptable virtual addresses for each program are between `0x400000` and `0xFFFFFFFFFFFF`. When a page fault interrupt is received by the kernel, it checks that the requested virtual address is between these values, and only if it is not, terminates the process. Otherwise, it loads a page in memory for that address.

D. It does not, the CPU directly terminates the user process.

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

View Feedback

Question 6

1 / 1 point

What are the main pieces of info pertaining to the virtual memory that the Linux kernel keeps in memory?

A. A pointer to the level-1 page table.

B. A virtual memory map of the process, describing the location and size of each section.

C. The location and size of the heap.

D. Read/write permissions for memory areas.

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

F. Two or more of A, B, C, D, but not all.

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

View Feedback

Question 7 **1 / 1 point**

What is memory mapping?

A. The general concept of mapping virtual memory to physical memory.

B. The process of assigning a virtual memory segment a direct byte-for-byte correlation with some disk object.

C. The mathematical map (function) between the physical locations of a cache, and the logical locations in the DRAM.

D. A kernel algorithm that is part of context switching. It computes a *map* of the virtual memory, and stores it in the process's VM, so that the process can load shared libraries.

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

View Feedback

Question 8 **1 / 1 point**

When a process forks, this creates an identical copy of the process but any modification to one process's memory does not impact the other's. How?

A. The virtual memory map of the child is the same as the parent, but every write access to physical memory triggers an exception that forces duplication of the data.

B. All the physical pages used by the parent process are duplicated during a fork, so that the processes hold the same values, but they do not correspond to the same locations in physical memory.

C. The processes are given separate virtual memory address spaces, with the parent using lower addresses than the child.

- ✓ Answer A.

- Answer B.
- Answer C.

View Feedback

Question 9

1 / 1 point

Why is memory mapping critical to modern library management?

A. It provides a user-space direct access to the page table of a process.

B. It allows to share code from a library between processes, without the processes' knowledge. They would share the physical memory pages pertaining to the library.

C. It allows code from libraries to be freely relocated in a process's virtual memory.

D. Processes can synchronize to share a part of the memory, and manually map the shared library there.

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

View Feedback

Question 10

1 / 1 point

Which component is in charge of copying memory segments flagged as Copy-On-Write?

A. The MMU, through a flag it stores in the PTE of each page. The kernel sets the flag.

B. The process: upon writing in such a memory segments, it receives a signal indicating that it has to first copy the memory.

C. One of the cache levels between the CPU and the DRAM (L1, L2, or L3).

D. The TLB of the MMU: only if a page's physical page number is TLB-cached can it be marked as Copy-On-Write.

E. The kernel: the flag is stored in the `vm_flags` of the corresponding area. The MMU doesn't have a concept of Copy-On-Write flag.

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

View Feedback

Question 11

1 / 1 point

How is the code of a program `a.out` loaded during `execve`, conceptually?

A. The file `a.out` is opened, then its `.text` section is memory mapped to VA `0x400000`; the `.text` section is not actually read.

B. The file `a.out` is opened, then its `.text` section is entirely copied to physical memory.

C. The kernel creates a fake memory map that just contains the name of the executable `a.out`. When the process is next scheduled, its `.text` section is entirely copied to physical memory, and the memory map is updated.

✔ Answer A.

Answer B.

Answer C.

View Feedback

Question 12

1 / 1 point

What are typical uses of the system call `mmap`?

A. Sharing the code of libraries so that they are loaded only once in physical memory.

B. Deferring duplicating physical memory pages that *shouldn't* be shared by waiting for the first write to them.

C. Loading a process but deferring reading portions of the program file until they are needed (*demand paging*).

D. Sharing memory between multiple processes, by `mmap`ing a common file on disk.

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

F. Two or more of A, B, C, D, but not all.

Answer A.

Answer B.

Answer C.

Answer D.

✔ Answer E.

Answer F.

View Feedback

Question 13

1 / 1 point

Apart from stack memory, how is the kernel dealing with the changing memory needs of a process during its execution?

A. The kernel creates a per-process *swap file* on the disk that can contain an unbounded amount of data, and use a memory mapping to make it accessible to virtual memory.

B. The kernel implements two system calls: `malloc` and `free`, that allocate and free memory on the heap, respectively.

C. The kernel uses a virtual memory section called the *heap*, whose size, given by a pointer `brk`, is controlled by the syscalls `brk` and `sbrk`.

D. The MMU is in charge of this, hiding details of memory allocation to the kernel.

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

View Feedback

Question 14

1 / 1 point

Recall that `sbrk(0)` returns the current value of `brk`. When will the following program crash, if it does?

```
1  #include <unistd.h>
2  int main () {
3    char *c;
4    c = sbrk (0) - 1;
5    *c = 'a';
6    c = sbrk (0) + 1;
7    *c = 'a';
8  }
```

A. Line 4. B. Line 5. C. Line 6. D. Line 7. E. It won't crash.

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

View Feedback

Question 15

1 / 1 point

What are strategies that implementations of memory allocators are allowed to pursue?

A. They can move allocated blocks to minimize fragmentation.

B. They can buffer requests to return a bigger block.

C. They can use the free memory for their own housekeeping.

D. They can reallocate blocks once they have been freed by the user.

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

F. Two of A, B, C, D, but not more.

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

View Feedback

Question 16

1 / 1 point

A memory allocator can be explicit, in which case the application both allocates and frees space. It can also be implicit; what does it mean?

A. Memory is allocated contiguously on the heap, and the kernel does memory management implicitly.

B. The application can allocate memory, but won't have to free it.

C. The application cannot allocate memory, and has no memory to free.

D. The application can allocate memory but only on the stack; upon popping the stack frame of the current call, the memory is implicitly freed.

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

View Feedback

Question 17

1 / 1 point

The two performance goals of memory allocators are to maximize throughput and peak memory utilization. After some time, and a few `malloc` and `free` requests, how is peak memory utilization at current time defined?

A. This is the difference between the current size of the heap and the maximum total memory used by the application at any previous time.

B. This is the maximum ratio between total memory used by the application and heap size, at any previous time.

C. This is the current total memory used by the application divided by the current size of the heap.

D. This is the maximum total memory used by the application at any previous time, divided by the current size of the heap.

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

View Feedback

Question 18

1 / 1 point

The two performance goals of memory allocators are to maximize throughput and peak memory utilization. Given a sequence of `malloc` and `free` requests, how is throughput defined?

A. This is the number of completed requests per unit time.

B. This is the number of DRAM hits per unit time.

C. This is the number of L1, L2, L3 misses per unit time.

D. This is the number of page faults per unit time.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 19

1 / 1 point

The two performance goals of memory allocators are to maximize throughput and peak memory utilization. How are these two goals conflicting?

A. The fastest memory allocator would always allocate at the end of the heap, but utilization would be terrible.

B. The best utilization would be provided by a *slow* algorithm searching for the best free block for each `malloc` request.

C. They are not conflicting, as exemplified by the libc implementation.

D. The kernel tends to maximize throughput, but the memory allocator wants to maximize peak memory utilization.

E. Only A and B.

F. Only A, B, and D.

Answer A.

Answer B.

Answer C.

Answer D.

✔ Answer E.

Answer F.

View Feedback

Question 20

1 / 1 point

What is internal fragmentation?

A. A measure of the number of contiguous free blocks in memory.

B. The concept of splitting a block when allocating only a portion of it.

C. A measure of the difference between the payload of a block and its actual size.

D. The average free space between allocated blocks.

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

View Feedback

Question 21

1 / 1 point

What is external fragmentation?

A. A pattern in a program execution where `frees` and `mallocs` are perfectly alternating.

B. The idea that virtual memory can appear contiguous without the physical memory backing it being so.

C. The average free space between allocated blocks.

D. The presence of lots of small free blocks that can't be combined to make a large one.

- Answer A.
- Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 22

1 / 1 point

Memory allocators keep track of the size of each block. Where is this information stored?

A. In a linked list stored on the heap.

B. In a table allocated at the beginning of the heap.

C. Within the block, in the word before the payload.

D. In a table stored in a specific segment `.malloc` of virtual memory.

Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 23

1 / 1 point

How is the free/allocated flag of each block stored?

A. Within the size word, as it's known to be an even number.

B. In a table stored in a specific segment `.malloc` of virtual memory.

C. In a linked list stored on the heap.

D. In a table allocated at the beginning of the heap.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 24

1 / 1 point

In implicit lists of free blocks, how can we find the next free block, starting at some block?

A. The address of the next free block is stored in a table at the beginning of the heap.

B. Read the current block size K, reach the next block by jumping K words, check if it is free, if not repeat.

C. The address of the next free block is stored within the block.

D. Read the current block size K, reach the next free block by jumping K words.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 25

1 / 1 point

In dynamic memory allocation, what are *first-fit*, *next-fit*, and *best-fit*?

A. These are strategies to evict pages from the DRAM.

B. These are policies to choose whether to coalesce or split free blocks.

C. These are policies to choose a free block to answer a `malloc` request.

D. These are different ways to store the address of the next free block.

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

View Feedback

Question 26 **1 / 1 point**

Coalescing is the task of fusing neighboring free blocks. Assume an allocator implements coalescing, but not splitting, would that be bad?

A. Yes, this would drastically increase both internal and external fragmentation.

B. Yes, this would drastically increase external fragmentation.

C. Yes, this would drastically increase internal fragmentation.

D. No, this would provide good performances.

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

View Feedback

Question 27

1 / 1 point

Assume a block of size K has been selected for a `malloc` request asking for T words (including header/footer/padding). The block can undergo *splitting*; what is that?

A. That's when the block is split into two blocks, of size T and $K - T$, the first one being returned, and the second being set as free. This decreases internal fragmentation.

B. That's the concept that $K - T$ words are lost, increasing internal fragmentation.

C. That's when the block is shifted before being returned, so that it may span two blocks.

D. That's when the block is fused with surrounding allocated blocks, in an attempt to split memory at multiples of the page size.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 28

1 / 1 point

Why do free blocks need a boundary tag (i.e., a footer replicating the block header)?

A. For coalescing a newly freed block with the previous block.

B. For implementing the *best-fit* strategy.

C. For coalescing a newly freed block with the next block.

D. For splitting a newly allocated block.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 29

1 / 1 point

Why aren't implicit lists of free blocks used in the libc?

A. Because `free` can take time linear in the current number of blocks.

B. Because it does not implement splitting.

C. Because it does not implement coalescing.

D. Because the search for a free block that can fit a `malloc` request can take time linear in the current number of blocks.

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 30

1 / 1 point

Office hours are to be booked on BlueStar; what are the office hours of the instructor for this course?

Monday & Tuesday, 3am to 9am

✔ Wednesday & Thursday, 3pm to 6:30pm.

Saturday & Sunday, 10pm to 11:42pm

St Patrick's Day & Towel Day, 17am to 25pm.

Question 31 **1 / 1 point**

This class is supported by a Supplemental Instructor (SI leader). When are the SI sessions?

Monday 5pm to 6pm.

Thursday 6pm to 7pm.

✔ All of the above.

Attempt Score:31 / 31

Overall Grade (highest attempt)**:**31 / 31

Done

Quiz Submissions - Week 5 Quiz

Baris Dingil (username: BDINGIL)

Attempt 2

Written: Oct 13, 2020 1:48 PM - Oct 13, 2020 1:53 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1

1 / 1 point

How is tracking of free blocks using an explicit list implemented?

A. A table stored at the beginning of the heap gives a list of all free blocks.

B. Each block contains a pointer to the previous and next blocks.

C. Each *free* block contains a pointer to the previous and next *free* blocks, while *allocated* blocks just contain the size information.

D. Each block contains a pointer to the previous and next *free* blocks.

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

View Feedback

Question 2

1 / 1 point

In normal usage, and compared to implicit lists, how do explicit lists impact internal fragmentation?

A. Internal fragmentation is greater with implicit lists.

B. Internal fragmentation is greater with explicit lists.

C. Internal fragmentation stays the same.

D. This heavily depends on the definition of "normal usage."

- Answer A.

- Answer B.
- ✔ Answer C.
- Answer D.

View Feedback

Question 3 **1 / 1 point**

What is the strategy used by the explicit list implementation of `malloc` to choose a free block?

A. Same as implicit list: there are multiple choices (e.g., *first-fit*, *next-fit*, *best-fit*).
B. By coalescing the first few free blocks at the beginning of the list.
C. By taking the free block that is at the beginning of the list.
D. By doing a binary search within the explicit list.

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

View Feedback

Question 4 **1 / 1 point**

In explicit lists of free blocks, once a block is freed, how is it inserted into the list of free blocks?

A. It is always inserted at some position that depends on the address of the block, and any other choice would be incorrect.
B. It is always inserted at the beginning of the list, and any other choice would be incorrect.
C. This is implementation dependent; it can for instance be inserted at the beginning of the list, or at some position that depends on the address of the block.
D. It is inserted in its own new list, to allow for coalescing.

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

View Feedback

Question 5 **1 / 1 point**

Which statement holds?

A. The smaller the number of lists in segregated lists, the more *first-fit* search in segregated lists approximates *best-fit* in explicit lists.

B. The greater the number of lists in segregated lists, the more *best-fit* search in segregated lists approximates *first-fit* in explicit lists.

C. The greater the number of lists in segregated lists, the more *first-fit* search in segregated lists approximates *best-fit* in explicit lists.

- Answer A.
- Answer B.
- ✔ Answer C.

View Feedback

Question 6 **1 / 1 point**

In segregated lists of free blocks, several external lists are used. What are they?

A. They are a few lists (about 128), each of them keeping track of free blocks in a fixed size range (about 64 of them are ranges of size 1).

B. They are an unbounded number of lists, each of them keeping track of free blocks in a fixed size range (about half of them are ranges of size 1).

C. They are a few lists (about 128), each of them keeping track of free blocks in a size range, and each range can evolve during execution.

D. They are an unbounded number of lists, each of them keeping track of free blocks in a size range, and each range can evolve during execution.

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

View Feedback

Question 7 **1 / 1 point**

In the context of memory management, what is garbage?

A. Heap-allocated storage that is no longer referenced.
B. Another term for Java.
C. A heap-location that is referenced by the user, but is within a free block.
D. Another term for free memory (free blocks).

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

View Feedback

Question 8 **1 / 1 point**

When a memory snapshot is seen as a graph, the nodes are memory locations. We distinguish the *heap* nodes and the *root* nodes. What are the root nodes?

A. Any variable accessible in the current function of the original C program (scope-wise).
B. Variables stored in the registers.
C. Variables stored in the `.data` and `.bss` segments.
D. Variables stored on the stack.
E. All of A, B, C, D.
F. Two or more of A, B, C, D, but not all.

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

View Feedback

Question 9

1 / 1 point

What are assumptions required to implement mark-and-sweep perfectly?

A. We can find the beginning of a block given an address *within* the block.

B. We can list all the roots in the memory graph.

C. We can decide whether a memory location contains a pointer.

D. We can go from one allocated block to the next.

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

F. Two or more of A, B, C, D, but not all.

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

View Feedback

Question 10

1 / 1 point

What is the strategy of mark-and-sweep garbage collection?

A. Traverse the memory graph from the roots, then delete nodes that have not been visited.

B. Increment a reference counter (marker) each time a new variable points to a zone in memory, and decrement when that variable is removed (changes value or is popped from stack). If the marker is 0, free the memory zone.

C. Start at the leaves of the memory graph, and perform a random traversal for no more than `brk` steps, then delete nodes that have not been visited.

D. Mark all the memory zones returned by `malloc`, then on garbage collection, explore the current call stack to find addresses that are not marked, and free them.

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

View Feedback

Question 11 **1 / 1 point**

Which English description corresponds to the following type:

```
int *e[2]
```

A. e is an array[2] of pointers to ints.

B. e is a function returning an array[2] of ints.

C. e is a pointer to an array[2] of ints.

D. e is an array[2] of pointers to functions returning ints.

E. This is not a correct type.

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

View Feedback

Question 12 **1 / 1 point**

Which English description corresponds to the following type:

```
int (*(*t())())[2]
```

A. t is a function returning an array[2] of pointers to functions returning pointers to ints.

B. t is an array[2] of pointers to functions returning pointers to functions returning ints.

C. t is a function returning a pointer to a function returning a pointer to an array[2] of ints.

D. t is a pointer to a function returning a function returning a pointer to an array[2] of ints.

E. This is not a correct type.

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

View Feedback

Question 13 **1 / 1 point**

Which English description corresponds to the following type:

```
int (*(*k[2])())[3]
```

A. k is a pointer to a function returning an array[2] of pointers to array[3] of ints.

B. k is an array[2] of pointers to pointers to functions returning array[3] of ints.

C. k is a pointer to an array[2] of pointers to functions returning array[3] of ints.

D. k is an array[2] of pointers to functions returning pointers to array[3] of ints.

E. This is not a correct type.

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

View Feedback

Question 14 **1 / 1 point**

Which type corresponds to the following English description:

r is a function returning a pointer to an array[2] of pointers to functions returning pointers to pointers to ints.

A. `int **(**(r()))()[2]`

B. `int **(*(*r())[2])()`

C. `int *(**(*r))[2]()`

D. `int **(*(r()))[2]()`

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

View Feedback

Question 15

1 / 1 point

Let's start with a bit of C fun. In C, if we have two statements `stat1` and `stat2`, then the *compound statement* `(stat1, stat2)` executes `stat1`, then `stat2`, and has the value of `stat2`.

For instance, `(v = 51, 42)` will set `v` to 51, and the compound statement itself has value 42. As another example, `x = (3, 14)` will evaluate 3 (with no effect), and set `x` to 14, as this is the value of the compound statement.

Now assume I have a standard linked list type `list_t`, and a list denoted by its head, like this:

```
typedef struct list {
  int val;
  struct list *next;
} list_t;
list_t *head;
```

What is a proper way to free this list?

A.
```
while (head) {
  list_t *p = head->next;
  free (head);
  head = p;
}
```

B.
```
void f (list_t *l) {
  if (l) f (l->next);
  free (l);
}
f (head);
```

C.
```
list_t *mf (list_t *a) {
  return a + (free (head), 0);
}
while (head && (head = mf (head->next)));
```

D.
```
for (list_t *p; head && (p = head->next, 1); head = p)
  free (head);
```

```
free (head);
```

E. All of the above.

F. Some, but not all of the above.

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

View Feedback

Attempt Score:15 / 15

Overall Grade (highest attempt)**:**15 / 15

Done

Quiz Submissions - Week 6 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 4

Written: Oct 16, 2020 12:13 PM - Oct 16, 2020 12:31 PM

Submission View

Released: Sep 7, 2020 10:15 PM

System I/O

Question 1 **1 / 1 point**

What can be found in `/proc`?

A. "Virtual" files that can be written to in order to change some kernel parameters.

B. "Virtual" files that the kernel fills with system information, on demand, upon reading.

C. One directory per process, containing a file-based view of the info that the kernel keeps about it.

D. Only A and B.

E. Only A and C.

F. Only B and C.

G. All of A, B, and C.

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

Answer E.

Answer F.

✔ Answer G.

View Feedback

Question 2

1 / 1 point

What are example of files, on a Unix system?

A. A directory.
B. A PNG image.
C. A disk partition.
D. A network device.
E. Two or three of the above.
F. All of the above.

Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

✔ Answer F.

View Feedback

Question 3

1 / 1 point

How does the kernel distinguishes between a text file and a binary file?

A. It sees that the characters are only ASCII or Unicode characters.

B. It sees that the file is made up of *lines*, terminated by `\n`.

C. It uses an approach similar to the `file(1)` command, which reads the first few characters of a file (the *magic*) to determine its type.

D. It does not.

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

View Feedback

Question 4

1 / 1 point

Assuming we are deep enough in the directory hierarchy, where is the following command leading us?

```
cd ./../././../.
```

A. The current directory.

B. The parent directory.

C. The grandparent directory.

D. The great-grandparent directory.

E. This is not a correct command; this is some Morse code spelling EIEEIE.

- Answer A.
- Answer B.
- ✔ Answer C.

Answer D.

Answer E.

View Feedback

Question 5

1 / 1 point

After creating an empty directory using the command `mkdir`, how many entries does the directory contain?

A. 0. B. 1. C. 2. D. 3.

Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 6

1 / 1 point

What are the basic *system calls* used for I/O?

A. `open`, `close`, `read`, `write`, `stat`.

B. `fopen`, `fclose`, `fread`, `fwrite`, `fstat`.

C. `openf`, `closef`, `readf`, `writef`, `statf`.

D. There are no system calls, just library calls.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 7

1 / 1 point

What is the *proper* way to write abcd in a file `foo.txt`, but in two steps, first ab then cd?

A.
```
int fd = open ("foo.txt", O_WRLONLY, 0);
write (fd, "ab", 2);
write (fd, "cd", 2);
```

B.
```
int fd = open ("foo.txt", O_WRLONLY, 0);
write (fd, "ab", 2);
// file offset set to current location + 2
lseek (fd, 2, SEEK_CUR);
write (fd, "cd", 2);
```

C.
```
int fd = open ("foo.txt", O_WRONLY, 0);
write (fd, "ab", 2);
// reopen file in append mode.
fd = open ("foo.txt", O_WRONLY | O_APPEND, 0);
write (fd, "cd", 2);
```

D.
```
int fd = open ("foo.txt", O_WRONLY, 0);
fd = write (fd, "ab", 2);
write (fd, "cd", 2);
```

✔ Answer A.

- Answer B.
- Answer C.
- Answer D.

View Feedback

Question 8

1 / 1 point

How does one access the list of files in a directory?

A. Directories are text files that contain the name of the files within, so it's just a matter of reading that file.

B. Using functions provided by the libc.

C. Using system calls.

D. Using a specific directory library, `libdir`.

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

View Feedback

Question 9

1 / 1 point

How does one access metadata info about a file (mode, size, modification time, etc.)?

A. By allocating a `struct stat`, and passing it, along with the file name, to the `stat` syscall.

B. By calling the `stat` syscall, with the file name and the property wanted (mode, size, etc.) as argument; this returns the info queried.

C. By calling `open` on the file, as this returns a file descriptor that contains the metadata info.

D. By calling the `stat` syscall, with the file name as argument; this returns a `struct stat` object with the info.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 10 **1 / 1 point**

Entries in the file descriptor table points to entries in the open file table. What is stored in an open file table entry?

A. The number of the file descriptor, the PID of the process that opened the file.

B. A pointer to a v-node, the current position in the file, and the number of times this entry is referenced.

C. The file access mode, the file size, the file type.

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 11

1 / 1 point

The kernel maintains descriptor tables, open file tables, and v-node tables. Which tables are per-process, and which ones are shared by all processes?

A. All of these tables are shared by all processes.

B. Only the descriptor table is per-process.

C. Both the descriptor table and the open file table are per-process.

D. All of these tables are per-process.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 12

1 / 1 point

When a process forks, what happens to the file descriptors, in the child?

A. The corresponding files are duplicated, and new open file table entries are created.

B. The files are closed in the child, and attempts to use the old file descriptors would result in an error.

C. The descriptor table is duplicated, and the corresponding open file table entries have their reference counters incremented.

D. The descriptor table and the corresponding open file table entries are duplicated.

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

View Feedback

Question 13 **1 / 1 point**

If a file is opened twice by a process, then:

A. One open file table entry is created, implying that the file can't be read at two distinct positions at the same time.

B. Two open file table entries are created, allowing to read a file at two distinct positions at the same time.

C. The second call to `open()` returns an error.

D. The file is temporarily duplicated on the disk, and the last FD to be closed determines which version stays on the disk.

- Answer A.
- ✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 14

1 / 1 point

A short count occurs when a `read` or `write` call doesn't read or write the number of bytes that was asked. When can this happen?

A. When reading or writing to network sockets.

B. When a long read is interrupted by a signal.

C. When the end of file is reached on a read.

D. When reading from a terminal, as the reads are magically stopped at newline characters if the opened file is a terminal.

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

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

Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

✔ Answer F.

View Feedback

Question 15

1 / 1 point

What is the `dup2(4, 2)` system call doing?

A. It duplicates the file table entry of FD number 4, and makes FD number 2 point to that new entry.

B. It duplicates the file that corresponds to FD number 4, and opens it as FD number 2.

C. It makes FD number 2 point to the same open file table entry as FD number 4.

D. It reopens the file that corresponds to FD number 4 as FD number 2.

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

View Feedback

Question 16 **1 / 1 point**

What is the main feature of the libc's I/O compared to the bare Unix syscall interface?

A. Being userspace.

B. Buffering.

C. Async-signal-safety.

D. Multi-threading.

- Answer A.
- ✔ Answer B.
- Answer C.

Answer D.

View Feedback

Question 17

1 / 1 point

What are advantages of buffered reads?

A. Less system calls.
B. Better memory utilization.
C. Efficiently check for newline characters.
D. Overall faster execution.
E. Two or three of A, B, C, D.
F. All of A, B, C, D.

Answer A.

Answer B.

Answer C.

Answer D.

✔ Answer E.

Answer F.

View Feedback

Question 18

1 / 1 point

A file descriptor is just a small integer, but the libc uses the type `FILE*`, pointing to a structure. What is in that structure?

A. A pointer to the current write position within an internal buffer.

B. A pointer to the current read position within an internal buffer.

C. A file descriptor.

D. A pointer to an internal buffer.

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

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

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

View Feedback

Question 19 **1 / 1 point**

Are there functions to read or write a *single* byte to a file, in the libc?

A. Yes, and the use of buffering make them inexpensive.

B. Yes, but they are very costly and should be avoided.

C. No, one should use `fread` or `fwrite`.

D. No, they would encourage costly micro operations on files.

- ✔ Answer A.
- Answer B.

- Answer C.
- Answer D.

View Feedback

Question 20

1 / 1 point

What is `strace`?

A. A performance monitoring library for C.
B. A system call that turns on verbose debugging features for the current process.
C. A program that prints out all the system calls made by another program.
D. An alternative to GDB.

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

View Feedback

Question 21

1 / 1 point

Considering our three sets of I/O functions (Unix I/O, Standard I/O, Robust I/O):

A. They are all independent and implement on top of low-level assembly instructions to access files.
B. Robust is implemented on top of Standard, itself implemented on top of Unix.
C. Standard is implemented on top of Robust, itself implemented on top of Unix.
D. Standard and Robust are independent, and implemented on top of Unix.

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

View Feedback

Question 22

1 / 1 point

In general, which I/O framework should you use to read files on the disk?

A. Memory maps (`mmap`).
B. Standard I/O.
C. Robust I/O.
D. Unix I/O.

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

View Feedback

Question 23

1 / 1 point

When should you prefer using Unix I/O over Standard I/O?

A. When dealing with sockets.
B. When short courts should be avoided.
C. When reading text lines.
D. When speed is important.
E. Two or three of A, B, C, D.
F. All of A, B, C, D.

✔ Answer A.

Answer B.

Answer C.

Answer D.

Answer E.

Answer F.

View Feedback

Network Programming

Question 24

1 / 1 point

Rank these types of networks from most local to most global:

A. WAN, LAN, SAN.
B. SAN, LAN, WAN.
C. LAN, SAN, WAN.
D. SAN, WAN, LAN.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 25

1 / 1 point

Which sentence is correct?

A. The internet is an internet.
B. The Internet is an Internet.
C. The Internet is an internet.
D. The internet is an Internet.

Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 26

1 / 1 point

What is the main difference between a hub and a switch?

A. A hub broadcasts the incoming data to all its hosts (pure hardware), a switch only sends the data to the target host (some software needed).

B. A switch broadcasts the incoming data to all its hosts (pure hardware), a hub only sends the data to the target host (some software needed).

C. They are two names for the same thing.

D. A hub is used only in local networks, a switch only in global networks.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 27 **1 / 1 point**

What is a MAC address?

A. Another term for IP addresses when considered within private networks.

B. A unique ID assigned to each network interface controller.

C. The IP address of an Apple computer on a local network.

D. An ID that combines the hardware specs of the current computer, to check when some hardware component is changed.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 28

1 / 1 point

What best describes the graph of nodes on an internet?

A. A mess.

B. A complete graph.

C. A DAG (directed acyclic graph).

D. A tree.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 29

1 / 1 point

What is the OSI (Open Systems Interconnection) model?

A. Another name for the TCP/IP model.

B. A conceptual model of layers in networking, from hardware to software.

C. The collection of all protocols used in a given network.

D. A hierarchy of specifications that network protocols should follow.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 30

1 / 1 point

We mostly focus on TCP (Transmission Control Protocol) in this course. What is UDP (User Datagram Protocol)?

A. A protocol built on IP, and on which TCP is constructed, that adds the notion of ports.

B. A peer to TCP, built on IP, offering unreliable port-based communication.

C. An extension of TCP, offering reliable connection over unreliable networks.

D. An alternative to IP.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 31

1 / 1 point

In terms of addresses, what is the main difference between IPv4 and IPv6?

A. IPv4 uses 16 bit addresses, IPv6 uses 128 bit addresses.

B. IPv4 uses 32 bit addresses, IPv6 uses 128 bit addresses.

C. IPv4 uses 16 bit addresses, IPv6 uses 32 bit addresses.

D. IPv4 uses 32 bit addresses, IPv6 uses 64 bit addresses.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 32

1 / 1 point

What is *network byte order*?

A. The order in which OSI levels interact.
B. The endianness used in the IP suite.
C. The order in which packets arrive.
D. The order in which headers are added to a payload.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 33

1 / 1 point

What is the meaning of double-colon (::) in IPv6 addresses notation?

A. It's not valid; it is used for discussion, but standard functions would fail on an IP containing this.

B. It's a joker, denoting a range of IPs.

C. It's a placeholder for routing purposes.

D. It's an ellipsis for zeroes.

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

View Feedback

Question 34 **1 / 1 point**

In IP notation (regardless of version), what does it mean to suffix an address with /11?

A. This denotes the group of IP addresses starting with the address before the slash, and for 11 addresses.

B. This denotes a group of IP addresses that share their first (highest) 11 bits.

C. This denotes a group of IP addresses that share their last (lowest) 11 bits.

- Answer A.
- ✔ Answer B.
- Answer C.

View Feedback

Question 35

1 / 1 point

What is the difference between broadcast and multicast?

A. They are the same.

B. Broadcast targets a group of devices, multicast targets a whole subnetwork.

C. Multicast targets a group of devices, broadcast targets a whole subnetwork.

- Answer A.
- Answer B.
- ✔ Answer C.

View Feedback

Question 36

1 / 1 point

What happens when multiple devices take on the same unicast IPv6 address?

A. The behavior is undefined.

B. The address becomes unavailable and unroutable.

C. The address becomes *anycast*, and a message sent to the address is received by a single device.

D. The address becomes *multicast*, and a message sent to the address is received by all the devices.

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

View Feedback

Attempt Score:36 / 36

Overall Grade (highest attempt)**:**36 / 36

Done

Quiz Submissions - Week 7 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 3

Written: Oct 26, 2020 2:29 PM - Oct 26, 2020 2:33 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Network Programming

Question 1 — 1 / 1 point

What is a first-level domain name?

A. These are the domain names of the most visited websites on the Internet.

B. The six original end of domain names: gov, edu, com, mil, org, and net.

C. Also called Top-Level Domains (TLD), they are domain names living as direct children of the DNS root.

D. A domain name that contains a dot (.).

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

View Feedback

Question 2 — 1 / 1 point

The main purpose of the Domain Naming System is to map domain names (text addresses) to IP addresses. Where is that database stored?

A. There's no single place, it is heavily distributed.

B. On an ISC (Internet Systems Consortium) server with "well-known" IPv4 address `192.5.5.241` (IPv6: `2001:500:2f::f`).

C. On the ISP (Internet Service Provider) DNS server.

D. On Google's DNS server (`8.8.8.8`).

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 3

1 / 1 point

In an iterative DNS query, how many DNS servers can be contacted to resolve `www.cdm.depaul.edu`?

A. 1 B. 2 C. 3 D. 4

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 4

1 / 1 point

To resolve an address, the libc will contact a DNS server. How does the libc, on a Unix-like system such as Linux, know the IP address of the DNS server?

A. The local router answers with the address of a DNS server when a connection is attempted to the "well-known" address `127.0.0.1`.

B. A list of DNS servers is provided by the file `/etc/resolv.conf`.

C. A default DNS server is hardcoded in the libc.

D. The libc does not know of a DNS server address: to resolve an address, it makes a system call and the kernel will use the DNS server it received when it was given an IP address.

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

View Feedback

Question 5 **1 / 1 point**

DNS can be seen as a function mapping domain names to sets of IPs (since a domain name can have multiple IPs). Calling f this function, is the following situation possible?

$$f(\texttt{domain1}) = \{\texttt{ip1}, \texttt{ip2}\}$$
$$f(\texttt{domain2}) = \{\texttt{ip1}\}$$
$$f(\texttt{domain3}) = \{\texttt{ip2}\}$$

A. No; since `ip1` serves `domain1`, it cannot serve another domain.

B. No; since both `ip1` and `ip2` serve `domain1`, the should also *both* serve `domain2` and `domain3`.

C. Yes.

D. Yes, but it is a temporary situation due to outdated DNS caches.

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

View Feedback

Question 6 **1 / 1 point**

HTTP relies on TCP, which relies on IP, which relies on Ethernet (or 802.11 Wi-Fi, for instance). All of these are *protocols*. Where is the notion of *port* defined? (E.g., HTTP works on port 80.)

A. At the application level, i.e., it is defined by HTTP.

B. At the transport level, i.e., it is defined by TCP.

C. At the network level, i.e., it is defined by IP.

D. At the data link level, i.e., it is defined by Ethernet.

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

View Feedback

Question 7

1 / 1 point

(TCP) connections are port-to-port. When a client connects to the HTTP port of a server (80), what is the port allocated on the client?

A. A random, so-called *ephemeral* port.

B. Port 80.

C. Port 80 + 1024, since ports below 1024 are reserved.

D. None; in a client-server connection, only the server needs a port.

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

View Feedback

Question 8

1 / 1 point

(TCP) connections are said to be process-to-process. When a connection is made to a server using a port number (say 80), how is the correct process notified?

A. All processes on the server-side are notified, and they should test that the target port is the one they expect.

B. The TCP packet should contain the process ID of the correct process on the server side.

C. A port can only be allocated by *one* process, so the server-side kernel knows which process is being targeted.

- Answer A.
- Answer B.
- ✔ Answer C.

View Feedback

Question 9 **1 / 1 point**

Applications read from and write to sockets. In C, what *is* a socket?

A. A pointer to `char`, that acts as a buffer to the connection.

B. A file descriptor.

C. An object of type `FILE*`.

D. An object of type `struct sockaddr`.

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

View Feedback

Question 10

1 / 1 point

What is a `struct sockaddr`?

A. A socket that can be read from or written to.

B. A structure with a large number of fields, that covers most common socket addressing uses.

C. A protocol-agnostic, generic type to address remote sockets.

D. An IPv4 addressing structure.

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

View Feedback

Question 11

1 / 1 point

What is the succession of calls necessary to open a client socket, and make a connection request?

A. `getaddrinfo, connect, socket.`

B. `getaddrinfo, socket, connect.`

C. `getaddrinfo, connect, bind, socket.`

D. `getaddrinfo, socket, bind, connect.`

- Answer A.
- ✔ Answer B.
- Answer C.

Answer D.

View Feedback

Question 12

1 / 1 point

What is the succession of calls necessary to open a listening socket, and wait for connections?

A. `getaddrinfo`, `socket`, `bind`, `listen`, `accept`.

B. `getaddrinfo`, `bind`, `socket`, `accept`, `listen`.

C. `getaddrinfo`, `socket`, `bind`, `accept`, `listen`.

D. `getaddrinfo`, `socket`, `listen`, `bind`, `accept`.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 13

1 / 1 point

Assuming that a server only *reads* from its clients, how is the server notified that a client has closed its end of the connection?

A. It will receive a EOF value (typically `-1`) on the next `read` on the socket.

B. It will receive a signal on the next `read` on the socket.

C. It will receive a signal as soon as the client closes.

✔ Answer A.

- Answer B.
- Answer C.

View Feedback

Question 14

1 / 1 point

Where is DNS resolving implemented on a UNIX-like system, such as Linux?

A. In a POSIX library known as `libdns`.

B. In the kernel.

C. In the network adapter.

D. In the libc.

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

View Feedback

Question 15

1 / 1 point

How is most of the information retrieved by `getaddrinfo` returned to the caller?

A. As an automatically allocated linked list of `struct addrinfo`.

B. Using a global variable `addrinfo` of type `struct addrinfo`.

C. It is written in a string buffer provided by the caller, of a given maximum size. This buffer will contain a string describing the info.

D. It does not return information explicitly, but the information is used implicitly in subsequent calls to socket interface functions.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 16

1 / 1 point

In a normal application relying on DNS, assume there is a call:

```
connect(fd, addr, addr_len);
```

How were the values of `addr` and `addr_len` typically found?

A. They are computed using various bit masks, and some macros are provided to simplify this.

B. They are constants found in the header file `sys/socket.h`.

C. They are the obvious values, for instance `addr` would be the string `"127.0.0.1"`, and in that case, `addr_len` would be 9.

D. They are returned by `getaddrinfo`, which should be called beforehand.

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 17

1 / 1 point

The function `getaddrinfo` can be seen as converting "human-readable" addressing information to a `struct sockaddr`. What function provides the converse direction?

A. `getinfoaddr`.

B. `getnameinfo`.

C. `getsockaddrinfo`.

D. `getaddrinfo_r`.

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

View Feedback

Question 18 **1 / 1 point**

Among the 5 fundamental socket system calls, only one is usually used by both the client and the server. Which one?

A. `socket`

B. `bind`

C. `listen`

D. `accept`

E. `connect`

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

Answer E.

View Feedback

Question 19

1 / 1 point

The call to `socket(domain, type, protocol)` returns a socket descriptor. How does one indicate that the socket is supposed to be a client type, rather than server type?

A. Using a specific value for `domain`.
B. Using a specific value for `type`.
C. Using a specific value for `protocol`.
D. The kernel does not differentiate between the two types of sockets.

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 20

1 / 1 point

What is the role of the syscall `bind`?

A. Mark a socket descriptor as *passive*, that is, mark it as a socket that will be used for incoming connections.
B. It creates a socket descriptor.
C. It extracts the first connection request on the queue of pending connections for the socket descriptor.
D. Assign a socket address to a socket descriptor.

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

View Feedback

Question 21

1 / 1 point

Calling `listen(sockfd, backlog)` lets the kernel know that we intend to use `sockfd` as a server-type socket. What is the role of `backlog`?

A. This is the maximum number of connections that `sockfd` can receive before automatically closing down (or 0 for no limit).

B. This is a value that indicates whether the kernel should log connections using the standard logging facilities (`/var/log` or `journald`).

C. This is the size of the buffer for the TCP stream; clients that try to write when that buffer is full will be blocked until the buffer has room.

D. This is the maximum length of the queue of pending connections to `sockfd`.

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

View Feedback

Question 22

1 / 1 point

In a server, what is the return value of a call to `accept(listenfd, ...)`, and how does one communicate with the new client?

A. It is a new file descriptor that directly connects to the client. The FD `listenfd` remains open and can accept more connections.

B. It is a new file descriptor that can be used to further accept more connections. The FD `listenfd` is now directly bound to the new client.

C. It is either `0` if `accept` succeeded, or `-1` otherwise. The FD `listenfd` is used to directly communicate with the client.

✔ Answer A.

Answer B.

Answer C.

View Feedback

Question 23 **1 / 1 point**

After a successful call to `connect`, a process-to-process connection is created. Usually, how is the TCP port on the *client* side chosen?

A. It is based on the MAC address of the client.

B. An *ephemeral* port is picked by the kernel.

C. The server returns the port that the client should use.

D. A call to `bind` should have been made before to link the client socket to a port.

Answer A.

✔ Answer B.

Answer C.

Answer D.

View Feedback

Question 24

1 / 1 point

In a client, what is the return value of a call to `connect(clientfd, ...)`, and how does one communicate with the server after the call?

A. It is a new file descriptor that directly connects to the server. The FD `clientfd` remains open and can be used to make more connections.

B. It is either `0` if `connect` succeeded, or `-1` otherwise. The FD `clientfd` is used to directly communicate with the server.

C. It is a new file descriptor that can be used to make further connections. The FD `clientfd` is now directly bound to the server.

- Answer A.
- ✔ Answer B.
- Answer C.

View Feedback

Question 25

1 / 1 point

We use `telnet` as an echo client (sends the user input raw to the server, and prints the server output as it comes). But this wasn't why `telnet` was created; what was it?

A. It was used to navigate DNS servers.

B. It was used as an elementary HTTP client.

C. It was used as a command-line interface to remote hosts.

D. It was used as an elementary FTP client.

- Answer A.

- Answer B.
- ✔ Answer C.
- Answer D.

View Feedback

Question 26

1 / 1 point

What are the most basic functions used to read and write to a socket?

A. `sockread` and `sockwrite`.
B. `read` and `write`.
C. `rio_readn` and `rio_writen`.
D. `fread` and `fwrite`.

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

View Feedback

Question 27

1 / 1 point

Can the HTTP protocol transfer any type of file?

A. Yes, but the files need to be transcoded to be using ASCII only (using, e.g., base64).

B. No, it can only transfer HTML files.

C. No, it can only transfer files that contain only ASCII characters, and transcoding to ASCII is not part of the protocol.

D. Yes, the files are sent raw over the connection.

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

View Feedback

Question 28 **1 / 1 point**

What is the distinction between *static* and *dynamic* content in the context of web servers?

A. Static is just a file on the server, dynamic is the output of an executable run on the server.

B. Static is a webpage whose location does not change over time, while dynamic content can be at any location.

C. Static is a webpage that does not change when it is refreshed, while dynamic pages always change.

D. Static is a webpage without Javascript, dynamic is a webpage with Javascript.

- ✔ Answer A.
- Answer B.

Answer C.

Answer D.

View Feedback

Question 29

1 / 1 point

Assume a web server serves websites `lorem.com` and `ipsum.net`. Recall that a GET request only provides the URL suffix, for instance:

```
GET /mypage.html HTTP/1.0
```

How does the server know whether to serve `lorem.com/mypage.html` or `ipsum.net/mypage.html`?

A. The GET request should indicate the website domain name as an extra header.

B. The webserver has one FD per website, so it knows which one the client was addressing.

C. The webserver knows what was the domain name the client used by inspecting the result of the `accept` function.

D. The server returns an error and the GET request is resent by the client with the full URL.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 30

1 / 1 point

What is CGI (Common Gateway Interface)?

A. A webpage that is dynamically generated.

B. An HTTP error, indicating that the connection gateway is busy.

C. A standard for transferring info between the browser and a script that is run on the web server.

D. A raw format for images, with no compression, used in the early years of the Internet.

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

View Feedback

Question 31 **1 / 1 point**

What is the standard way for passing arguments to a webpage with a GET request?

A. There is no way to do this with GET, one must use a POST request.

B. The GET request is followed by request headers of type `Argument:`, one for each argument.

C. The GET request takes an optional fourth argument which is a list of arguments, separated by ampersands, e.g.: `GET / HTTP/1.1 arg1&arg2`.

D. The URI in the GET request is suffixed with a question mark, then some arguments separated by ampersands.

- Answer A.
- Answer B.
- Answer C.

✔ Answer D.

View Feedback

Question 32

1 / 1 point

What is the difference between a POST request and a GET request?

A. GET requests cannot be used with HTML `<form>`s.
B. POST requests can send unlimited raw data to the server, GET cannot.
C. POST requests cannot provide arguments in the URL.
D. GET requests can request dynamic content, POST cannot.
E. More than one of the above.

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

View Feedback

Question 33

1 / 1 point

How does a CGI script retrieve the arguments provided in a GET request?

A. By the usual command line argument passing system (`argc` and `argv` in C).
B. By reading the environment variable `QUERY_STRING`.
C. By reading the file `/var/cgiargs`.
D. By reading a global variable `extern char *cgiargs`.

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

View Feedback

Question 34

1 / 1 point

How does a CGI script read the data provided by a POST request?

A. By reading the file whose filename is stored in the environment variable `CGI_DATA`.

B. By reading the file `/var/cgidata`.

C. By reading the FD stored in the global variable `extern int cgidata`.

D. By reading the standard input.

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

View Feedback

Concurrent Programming

Question 35

1 / 1 point

Why is it so hard to picture the interactions within n concurrent executions?

A. Because the number of possible interactions is exponential in n.

B. Because it's hard to remember which function calls are libc, and which are system calls.

C. Because it's hard to keep track of the tree of forks (children processes).

D. Because (single-core) processors only *emulate* concurrency.

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

View Feedback

Question 36 **1 / 1 point**

What are examples of problems that arise in concurrent programming?

A. Deadlocks: when a process waits for an event that will never happen.

B. Race conditions: the outcome depends on the order in which processes are scheduled.

C. Starvation: when a process is never scheduled.

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 37

1 / 1 point

Iterative servers can only serve one client at a time. If a second client tries to connect and make small interactions with the server while it is not in accept mode, when will the second client block?

A. When calling `socket`.
B. When calling `connect`.
C. When calling `write`.
D. When calling `read`.

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 38

1 / 1 point

In a process-based concurrent server, each client is assigned a child process (client process). Who is in charge of scheduling these processes? In other terms, what part of the system chooses which process is executed at any given time?

A. Client processes are normal processes and are scheduled by the kernel with no special consideration.

B. The CPU is in charge of scheduling client processes as soon as data from them is received.

C. Client processes are scheduled by the kernel in the order in which the server receives data from them.

D. The server chooses which client process is to be scheduled next, through system calls.

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 39 **1 / 1 point**

In a process-based concurrent server, how are processes for each client created?

A. Using `sockclient`.

B. Using threads.

C. Using `execve`.

D. Using `fork`.

Answer A.

Answer B.

Answer C.

✔ Answer D.

View Feedback

Question 40

1 / 1 point

In a process-based concurrent server, after creating a child process for a new client, the server should close the socket descriptor (file descriptor) associated with that client. Why?

A. This is optional, and not doing it would not lead to any problem.

B. If the server were not to do it, when the child process closes that socket the kernel will think that a process still needs that file descriptor, and won't actually close it.

C. If the server were not to do it, the FD would appear in both the client and the server, but only one of the two processes can read from it.

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:40 / 40

Overall Grade (highest attempt)**:**40 / 40

Done

Quiz Submissions - Week 8 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 2

Written: Oct 30, 2020 1:39 PM - Oct 30, 2020 1:41 PM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1 — 1 / 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 — 1 / 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

1 / 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

1 / 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

1 / 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

1 / 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

1 / 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

1 / 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

1 / 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

1 / 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 **1 / 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

1 / 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

1 / 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

1 / 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:22 / 22

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

Done

Quiz Submissions - Week 9 Quiz

Celine Villareal (username: CVILLAR5)

Attempt 8

Written: Nov 9, 2020 11:08 AM - Nov 9, 2020 11:29 AM

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