ATTENTION: The goal of these sample questions is to simply direct you when you review relevant topics for the exam. By no means, this set of questions is comprehensive (you still have to review all lectures!). **This also does NOT represent the actual format of the exam, style or number of questions, their difficulty, etc.**

Correct answers **are not** provided deliberately so that you can explore and look for the answer while you are preparing for the exam.

NOTE: I will mostly avoid theoretical questions about concurrent objects (e.g., histories, response/request notations, etc). But you still need to understand what sequential consistency and linearizability is. You also need to know what hardware actually implements. You need to know both blocking and non-blocking progress guarantees/conditions and be able to understand lock-free code (e.g., lock-free stack or queue). You need to know what TAS and CAS are.

Q1: Which of the following is true

A. Linearizability is easier to implement in hardware, while sequential consistency is harder
B. Sequential consistency is easier to implement in hardware, while linearizability is harder
C. Neither sequential consistency nor linearizability are typically implemented by hardware directly, but sequential consistency can typically be obtained by using memory barriers
D. Neither sequential consistency nor linearizability are typically implemented by hardware directly, but linearizability can typically be obtained by using memory fences
E. Neither sequential consistency nor linearizability are achievable in software or hardware

ANSWER: B, C

Q2: Which of the following are examples of the concept of “virtual memory”?
A. Paging through page tables
B. Segmentation
C. Disk emulation through qemu
D. /dev file system in Linux
E. Virtual file system

ANSWER: A, B

Q3: What is the monolithic OS? Please specify alternatives to this OS design.

ANSWER: Please review our lecture/lecture slides regarding OS architectures (Lectures 1-2, see the part where we talk about OS architectures). More importantly, see what the main characteristics of monolithic, microkernel and library OSs are. What are the components that they run in kernel and user space? What are the advantages of each model (e.g., performance vs. better isolation)? Reliability? Security?

Q4 [True or False]: UEFI has its own set of basic device drivers, which is independent from that of an operating system

ANSWER: True

Q5 [True or False]: UEFI and BIOS are both obsolete systems, neither are used by modern general purpose computer systems

ANSWER: False

Q6: Recall that TAS (Test-And-Set) is a special instruction which atomically sets a new value and returns and old value that was stored in a shared variable, e.g., can be considered as an **atomic** function

```
int TAS(int *variable, int new) {
   int old = *variable;
   *variable = new;
    return old;
}
```

Please show how you can implement a lock using this operation

ANSWER: Check lecture 22. There is a very similar function called “getAndSet” and the implementation of the lock there. (It is the same, just integer vs. bool value). Also presented in lectures 29-30 (1 – Spin on test&set).

Q7: Recall that TSS was historically used to manage tasks (processes, threads) to encapsulate task states when using ***hardware*** context switching. However, it is typically not used in the same manner anymore since ***software*** context switches are preferred. However, you still need to create one TSS segment per each processor (or core). What is the primary purpose of TSS (Task State Segment) when using it with modern x86-64 operating systems?

A. To store user-space stack pointer(s) since the default kernel stack pointers can be unreliable when an interrupt happens while CPU is in user mode
B. To store kernel-space stack pointer(s) since the default user stack pointers can be unreliable when an interrupt happens while CPU is in user mode
C. To store user-space stack pointer(s) since the default kernel stack pointers can be unreliable when an interrupt happens while CPU is in kernel mode
D. To store both user-space and kernel-space stack pointer(s) since all default pointers can be unreliable when an interrupt happens
E. Neither

ANSWER: B

ANSWER EXPLANATION: In the moment an interrupt happens, you can be running in user space your program. The user program uses an unreliable (potentially dangerous/malicious) stack pointer in the stack pointer register. Therefore, you cannot use it, and you need to have a known good kernel stack pointer which the CPU can immediately use when an interrupt happens! TSS needs to be initialized with a **known good kernel stack pointer** per processor (core), so that you have independent stacks for each processor (core). Recall that for interrupts, the CPU has to know and set up a new stack pointer EVEN BEFORE jumping to the interrupt handler (since it cannot use an old stack even for just a moment, i.e. to call an interrupt handler). The CPU will do that automatically using the kernel pointer supplied in its TSS segment.

Q8: What is the primary difference between NUMA (Non-Uniform Memory Access) and UMA (Uniform Memory Access)?

ANSWER: In NUMA, each CPU has its own local memory which it can access faster. The CPU can still access all memory, but other memory is slower. (See Lecture 7.)

Q9: What is the primary difference between SMP (Symmetric Multiprocessing) and AMP (Asymmetric Multiprocessing)?

ANSWER: In SMP, all CPUs/cores are treated equally. With AMP, not all CPUs/cores are treated the same. [We did not go into too much detail about SMP vs. AMP, so it is OK to just know this distinction for the exam.]

Q10 [True or False]: LibC is used to wrap system calls in a platform-independent manner

ANSWER: True

Q11 [True or False]: LibC does not provide anything beyond ordinary system call wrappers

ANSWER: False

Q12: Find bug(s) in the code which implements the ***pop*** operation for Treiber's stack. (top is a global variable which points to the top of the stack.) Note that the stack can be empty. Please ignore (i.e., do not consider) memory reclamation and/or the ABA problem. You also do not need to consider how values/objects are stored in the data structure.

```
1:      stack_node * pop () {
2:              stack_node *t;
3:              do {
4:                      t = atomic_load(&top);
5:              } while (!atomic_compare_exchange_strong(&top, &t->next, t));
6:              return t->next;
7:      }
```

ANSWER:
After Line 4, we need to add a check **if** (t == NULL) return NULL; // empty queue
Line 5 should be atomic_compare_exchange_strong(&top, &t, t→next));
Line 6 should be **return** t;

Q13 [True or False]: Memory reclamation does not typically solve the ABA problem, so you still need to use ABA tags for queues and stacks to be prevent false mismatches

ANSWER: False (but see below for more clarity)

DETAILED ANSWER:

If the memory objects are **not being recycled** directly (e.g., moved from one queue to another and then back), we simply retire memory objects and then allocate them again, then there is **no ABA** problem since memory objects cannot reappear until they are fully reclaimed. (e.g., Q3-Q4 of the homework)

If we do not use memory reclamation at all, the only option we have is to recycle elements (Q5-Q6) where the problem can arise.

However, it is also possible to have something in between: if memory objects are **being recycled** directly sometimes instead of being retired and reallocated, then the ABA problem can happen with memory reclamation. (We did not have any question like this in the homework but you could imagine having Q5-Q6 enhanced so that the free queue would be periodically drained if it gets too big. In this case, we would still prefer to recycle nodes, would also wrap all operations in SMR, and drain the free queue when it gets too big.)

In the actual exam, if the question like this appears, the assumptions about recycling will be clearly specified.

Q14: What is the difference between RAID 0 and RAID 1?

ANSWER: RAID 0 (striping) splits data across multiple disks, which makes writes/reads faster but reliability **will become worse** due to multiple source of failures. RAID 1 (mirroring) duplicates data on multiple disks for better reliability. (See Lecture 28.)

Q15: Give an example of at least three different components (including separate bits) of page table entries?

ANSWER: Physical address, present bit, user/supervisor bit (i.e., user-only mode vs. user and kernel mode access), etc

Q16: Can interrupts happen while the program is in user space?
Yes / No

ANSWER: Yes

Q17: Which of the following is true about page tables?
A. User program page tables reside in user space
B. Kernel portion is not shared across different page tables
C. Kernel portion is shared across different page tables
D. User portion is shared across different page tables (processes)
E. User program page tables reside in kernel space

ANSWER: C, E

Q18: Which of the following is true about a journaling file system
A. It is also known as a “virtual file system”
B. The last operation (create file, delete file, etc) can either fully succeed or not be performed at all during power loss
C. The last operation (create file, delete file, etc) can only fully succeed during power loss
D. The last operation (create file, delete file, etc) will never be performed during power loss

E. Neither

ANSWER: B

Q19: What is the difference between system calls and interrupts?

ANSWER: The basic idea behind system calls is to trigger an artificial trap (exception), which would switch take the CPU from user mode to kernel mode. This is very similar to ordinary traps (exceptions), with the only difference being that it is triggered by the user rather than some CPU error. Exceptions and system calls are events that are internal to the CPU.

Interrupts are triggered by other devices and they are fundamentally external to the CPU.

Q20: Please provide an example when a deadlock is possible (e.g., use more than one lock to demonstrate the condition)

ANSWER:

```
T1:                 T2:
lock(A);            lock(B);
lock(B);            lock(A);
…                   ...
```

Q21: Consider the following code that is executed in user space

```
void func(const char *str) {
        printf(str);
}
```

What does the 'str' variable contain?

A. Physical address to a read-only string
B. Virtual address to a read-only string
C. Character of type ‘const char’
D. Constant physical address to a string
E. Neither

ANSWER: B

Q22: What is difference between hard links and soft (symbolic) links

ANSWER: Hard links are providing an alternative name for the same file (i-node). Thus, they do not have any separate i-node. Also recall that i-nodes do not store file names, thus you can have multiple file names/paths to the same file from different (or even the same) directories.
Symbolic (soft) links are separate files (i-nodes) which contain a path (shortcut) to another file. This path is traversed automatically (by default) when a symbolic link is opened, but it is still a separate file.

Q23: Please explain the difference between different thread models (1:1, M:N, N:1)

ANSWER: Please review Lecture 8. Specifically, see where we would always need to do mode switches (go from user mode to kerne mode) to have a context switch (switch from one thread to another).

Q24: Suppose you need to set up a periodic APIC timer. Please explain how IDT (Interrupt Descriptor Table) is related to that and what you need to do with IDT.

ANSWER: IDT stores interrupt handlers. APIC timer will use its own interrupt vector, thus the corresponding IDT entry needs to be initialized with the APIC timer handler.

Q25: Which of the following represent actual physical memory addresses when using hypervisors

A. Virtual addresses
B. Physical addresses
C. Machine addresses
D. Shadow page table
E. Nested page table

ANSWER: C

Q26: Please explain what is typically stored in file system i-nodes

ANSWER: File permissions (chmod bits), file modification time, file owners (user/groups), file size, etc. Note that i-nodes **do not** typically store file names!

Q27: Which of the following is true about virtualization with hypervisors

A. Popek and Goldberg were proven to be wrong about virtualization requirements
B. Binary translation is typically impossible, thus paravirtualization is the only option when using virtualization
C. Paravirtualization is preferable for x86-64 due to availability of protection rings (Ring 0, Ring 1, Ring 2, Ring 3)
D. Hardware virtualization was historically used but is now fully obsolete due to paravirtualization
E. Neither

ANSWER: E

Q28: [True or False] Containers provide better isolation than hypervisors

ANSWER: False

More importantly – please review lecture 14 for the description of hypervisors and containers

Q29: Please explain what file descriptors are, where they are stored. Give an example of three (default) file descriptors that are typically available for any program.

ANSWER: A file descriptor is an integer which identifies a file object inside the kernel for a given process. Each process has its own table (mappings) of file descriptors to file objects (i-nodes). The table itself is modified/accessed in kernel space. Default descriptors: stdin (0), stdout (1), stderr (2).

Q30: Remember that write() is used to write data to a file. How is it possible that LibC implements printf() through write() to output strings to the terminal.

ANSWER: printf is writing output to stdout (1). The terminal output uses regular file I/O operations.