OS Overview, ISA

(1) What is an Operating System? List its primary responsibilities.

(2) What are the three different ways in which OS code can be invoked? Explain.

(3) Explain the following interfaces in a computer system
   (a) Instruction Set Architecture (ISA)
   (b) User Instruction Set Architecture (User ISA),
   (c) System ISA,
   (d) Application Binary Interface (ABI).
   (e) Application Programmers' Interface (API)

(4) Why doesn't a program (executable binary) that is compiled on the linux machine execute on a Windows machine, even if the underlying CPU hardware is the same (say x86)?

(5) Describe the process lifecycle illustrating the states and transitions.

(6) Which state transitions (if any) occur in a process lifecycle when a process
   A. Makes a blocking read() system call
   B. Runs too long on the CPU?
   C. Is interrupted by a hardware interrupt
   D. Dereferences a NULL pointer.
   E. Attempts to acquire a blocking lock that is taken by another process?
   F. Is pre-empted
   G. Voluntarily yields the CPU
   H. Tries to read keyboard input, but no input is available?
   I. Receives a signal?
   J. Attempts to execute a System ISA instruction in user space?
   K. Attempts to perform down() operation on a semaphore whose value is zero?

(7) During a process lifecycle, what events can cause the following transitions?
   (a) Ready to Running state
   (b) Running to Ready state
   (c) Ready to Blocked state
   (d) Blocked to Ready state

(8) Why are frequent context switches expensive in terms of system performance?

(9) What is cold-start penalty? What are some ways to reduce it?

(10) What are some key factors that affect application performance after a context switch?

Segmentation

(11) In memory management, what is meant by relocation and protection?

(12) How segmentation is used to achieve relocation and protection in Pentium architecture?

(13) How is segmentation different from paging? Why was each technique invented?

(14) What problem does segmentation solve that paging doesn't solve? What problem does paging solve that segmentation doesn't solve?

(15) How is a virtual address converted to a physical address, considering both segmentation and paging in (a) Multics and (b) Pentium architectures?

(16) What is meant by between internal and external fragmentation?

(17) How many page tables are there per segment in Multics and Pentium. Which one (Multics/Pentium) is better? Why?

(18) What kind of fragmentation occurs (and why) when using

A. Pure paging, without segmentation
B. Pure segmentation, without paging
C. Paging with Segmentation together
D. Using superpages of the same size
E. Using a mix of superpages of different sizes

(19) Consider segment-level protection and CPU privilege levels in x86. Explain how the operating system ensures that user-level processes don't access kernel-level memory?

File Systems - inode

(20) What is a File system?

(21) What's an i-node? Where is it stored?

(22) What's the simplest data structure possible for an i-node? Then why is UNIX i-node so complicated?

(23) Explain the structure of a UNIX i-node. Why is it better than having just a single array that maps logical block addresses in a file to physical block addresses on disk?

(24) If you increase or decrease the disk block size in a file system, how (and why) will it affect ***(a)*** the size of the inode, and ***(b)*** the maximum size of a file accessible only through direct block addresses?

(25) How does the inode structure in UNIX-based file-systems (such as Unix V7) support fast access to small files and at the same time support large file sizes?

(26) What's wrong with storing file metadata as content within each directory "file"? In other words, why do we need a separate i-node to store metadata for each file?

(27) Assume that the

- Size of each disk block is B.
- Address of each disk block is A bytes long.
- The top level of a UNIX i-node contains D direct block addresses, one single-indirect block address, one double-indirect block address, and one triple-indirect block address.

(a) What is the size of the ***largest "small"*** file that can be addressed through direct block addresses?

(b) What is the size of the ***largest*** file that can be supported by a UNIX inode?

(28) In a UNIX-like i-node, suppose you need to store a file of size 32 Terabytes (32 * 240 bytes). Approximately how large is the i-node (in bytes)? Assume 8096 bytes (8KB) block size, 8 bytes for each block address, and that i-node can have more than three levels of indirection. For simplicity, you can ignore any space occupied by file attributes (owner, permissions etc) and also focus on the dominant contributors to the i-node size.

1. In a UNIX-based filesystems, approximately how big (in bytes) will be an inode for a 200 Terabyte (200 * 240 bytes) file? Assume 4096 bytes block size and 8 bytes for each entry in the inode that references one data block. For simplicity, you can ignore intermediate levels of indirections in the inode data structure and any space occupied by other file attributes (permissions etc).
2. In a UNIX-based filesystems, approximately how big (in bytes) will be an inode for a ***400 Terabyte (400 * 2^{40} bytes) file***? Assume 4096 bytes (4KB) block size and 8 bytes for each entry in the inode that references one data block. For simplicity, you can ignore intermediate levels of indirections in the inode data structure and any space occupied by other file attributes (owner, permissions etc).
3. Assume that the size of each disk block is 4KB. Address of each block is 4 bytes long. What is the size of the ***largest*** file that can be supported by a UNIX inode? What is the size of the ***largest "small"*** file that can be addressed through direct block addresses? Explain how you derived your answer.
4. Assume all disk blocks are of size 8KB. Top level of a UNIX inode is also stored in a disk block of size 8KB. All file attributes, except data block locations, take up 256 bytes of the top-level of inode. Each direct block address takes up 8 bytes of space and gives the address

of a disk block of size 8KB. Last three entries of the first level of the inode point to single, double, and triple indirect blocks respectively. Calculate **(a)** the largest size of a file that can be accessed through the direct block entries of the inode. **(b)** The largest size of a file that can be accessed using the entire inode.

Answer:

Size of first level of the inode = 8KB
Size of attributes = 256 bytes
Space taken up by last three entries of the first level = 8 bytes * 3 = 24 bytes
Space remaining to for direct block entries = (8KB - 256 bytes - 24 bytes)

Largest file that can be accessed through direct block entries = (8KB - 280 bytes)*8KB/8bytes

Largest size of a file that can be accessed using the entire inode =
Size accessible from direct blocks +
Size accessible from single indirect blocks +
Size accessible from double indirect blocks +
Size accessible from triple indirect blocks
=
$8192*(8192 - 280)/8 + 8192*8192/8 + 8192*(8192/8)^2 + 8192*(8192/8)^3$ bytes

5. Consider a UNIX i-node for a file of size F bytes. What is the size of the i-node in bytes? Assume that disk block size is B bytes, each block address size is A bytes. The top level of the i-node contains D direct block addresses, one single-indirect block address, one double-indirect block address, and one triple-indirect block address.

Answer: Depends on how big is F.

Max number of block addresses stored in each inode block, say, $\mathbf{N_a} = B/A$
Number of disk blocks needed for a file of size F, say, $\mathbf{N_f} = \text{ceil}[F/B]$

Inode has at least one initial block to store attributes plus up to D direct block addresses, plus more single, double, and triple indirect blocks as needed.

Hence, inode size = $(1 + N_{single} + N_{double} + N_{triple})$xB bytes

Where N_{single}, N_{double}, and N_{triple} refer to the number of single-, double-, and triple-indirect blocks needed for a file of size F. These are calculated as follows.

If $N_f > D$, then $\mathbf{N_{single}} = \text{ceil}[(N_f - D)/N_a]$, else $\mathbf{N_{single}} = \mathbf{0}$

Single indirect block allows access up to a max of $D+N_a$ blocks of file data.
If $N_f > (D + N_a)$, then $\mathbf{N_{double}} = \text{ceil}[(N_{single}-1)/N_a]$, else $\mathbf{N_{double}} = 0$

Double indirect block allows access up to another N_axN_a addresses of file blocks.
If $N_f > (D + N_a + N_a x N_a)$ then we need space for **at most one more triple-indirect block,** i.e., $\mathbf{N_{triple}} = 1$, else $\mathbf{N_{triple}} = 0$

RAID

1. Distinction between logical and physical I/O address spaces.
1. What was the original & current motivation for RAID?
2. Why is a multiple-disk system less reliable than a single disk?
3. How does Mean Time to Failure (MTTF) change as number of components in a system increases?
4. What are the different levels of RAID and how do each of them work?
5. What are the relative benefits/drawbacks of each RAID level?
6. How is data distributed in each RAID level?
7. How is parity calculated and stored in each RAID level?
8. What is the extent of read and write parallelism in each level?
9. How is the parity calculation bottleneck in RAID 4 solved?
10. In RAID-5, explain how can you perform a single logical write operation in no more than one physical read and two physical writes?
11. Consider RAID levels 0, 1, 3, 4, and 5: Which RAID level provides the best (a) reliability (b) I/O Parallelism. Explain why.
12. In order to save power, disks are usually spun down (placed in sleep or low-power mode). This works well if there is only one disk in the system, if all data resides on the single disk, and if performance is not a major concern. Consider a RAID-5 system consisting of N+1 disks. Explain how you can redesign RAID-5 so that all the following requirements are satisfied: (1) fault-tolerance of original RAID-5 is maintained under all conditions, (2) energy consumption is minimized by spinning down one or more disks whenever possible, and (3) performance (read/write throughput) of the system is maximized to the extent possible. Again, while there is no single correct answer, you must explain all salient aspects of your design, justify any assumptions you make, and examine any design tradeoffs (e.g. energy savings to performance).

13. In RAID 5, describe how you can complete a write I/O operation using just 2 disk reads and 2 disk writes.
14. (a) Explain (with formula), how does parity computation differ between RAID 3 and RAID 4? (b) How does parity placement on the disk (not parity computation) differ between RAID 4 and RAID 5? Explain with example.
15. How should parity be computed in RAID 5 to increase parallelism of write operations? Explain with parity computation formula.
16. What is the write parallelism problem in RAID and how is it solved?

17. Describe the design of a parity-based RAID system that can survive two-disk failures (as opposed to single-disk failure discussed in class). In your design, be sure to explain the following: (a) How your system would compute the parity required for recovery from a two-disk failure? (b) How your system would recover from two-disk and single-disk failures, (c)

How much additional space would parity information occupy, compared to data, and (d) How many parallel read and write I/O operations can your system support?

18. For a RAID system **with N disks, including data and parity,** compare the level of parallelism provided by RAID 1, RAID 3, RAID 4, and RAID 5 for multiple simultaneous (i) read I/O operations, (ii) write I/O operations, and (iii) combination of read and write I/O operations? Explain your answers.

19. Consider RAID levels 1, 3, 4, and 5 (ignore RAID 0 and 2). Which RAID level provides the best (a) reliability (b) I/O Parallelism. Explain why. Assume data worth N disks of same size and additional space for any parity disks.

Virtualization

1. What is meant by virtualization? Give examples of many(virtual)-to-one(physical, one-to-many, and many-to-many resource virtualization.
2. For system virtual machines, explain how virtual memory addresses are translated to physical addresses when (a) hardware supports EPT/NPT (extended/nested page tables) and (b) hardware only supports traditional (non-nested) page tables.
3. How does Intel VTx extending the traditional CPU execution privilege levels to support system virtual machines?
4. Compare different approaches for virtualizing I/O devices for virtual machines.
5. Explain the key hardware-level virtualization support provided by Intel for
   (a) Memory translation for VMs
   (b) CPU privilege levels for guest OS execution
   (c) Direct I/O device access by VMs
6. Explain briefly with examples (1) Process virtual machine, (2) System virtual machine, (3) Emulator, (4) Binary optimizer, (5) High-level Language Virtual Machine.
7. Which interface does a Process VM virtualize? Which interface does a System VM virtualize?
8. (a) How do Interpreters differ from Dynamic Binary Translators? (b) How do Binary Optimizers differ from Emulators?
9. What are the advantages and disadvantages of Classical System VMs compared to Para-virtualized VMs?
10. What is a co-designed virtual machine? Briefly describe and give an example.
11. What type of virtual machine (VM) is each of the following and why? Be as specific as possible. (a) Java Virtual Machine (JVM) (b) VMWare (c) Xen (d) Digital FX!32 (e) VirtualPC (f) (e) Transmeta Crusoe (Code Morphing)
12. Explain the difference between the concepts of full-virtualization and para-virtualization, giving at least one example of both virtualization techniques.
13. When you have design a system that does emulation, under what circumstances would you opt for Interpretation and when would you opt for Binary Translation? Justify your answer.
14. Let's say that you are asked to modify the Linux OS so that programs and libraries compiled on Windows OS could run natively on Linux, meaning they should be executed as normal programs (without using any emulator or virtual machine). What would be your high-level approach?
15. What is the difference between a Type-1 hypervisor and a Type-2 hypervisor? Give examples.

Security

1. What is the difference between security and privacy? Are they entirely the same? Or entirely different?
2. Explain the three key principles of computer security?

3. What is a threat model? What factors should you consider when defining threat model?
4. What hardware mechanism does x86 ISA provide to ensure that Operating System's code and data are protected from user-level processes?
5. What is the role of privilege levels (defined by the ISA) in a computer system? How many privilege levels are defined in the x86 ISA? In which privilege level does the OS execute?
6. Explain the basic security mechanisms supported by (a) the CPU execution hardware, (b) Memory management hardware and software, (c) File system. Assume that the machine uses x86 ISA.

7. In x86, how does the MMU figure out whether a code currently executing on CPU has permissions to read/write to/execute a given address in memory?

8. What is authentication?
9. Describe different techniques to authenticate users.
10. What are some ways in which by which authentication mechanisms can be subverted?
11. What's a computer virus? What's a computer worm?
12. Explain a buffer overflow attack.
13. What is sandboxing? List two sandboxing mechanisms.
14. Explain Discretionary, Mandatory, and Role-based access control mechanisms.
15. What is meant by "trust" in computer security?

16. Explain (a) trusted computing base (TCB) including why is it called "Trusted", (b) Reference Monitor, and (c) relationship between TCB and reference monitor.
17. Explain the two key data access principles of multi-level security (MLS) systems (also called Mandatory Access Control).
18. Why is Mandatory access control called "mandatory"? What's the alternative?
19. What type of systems require mandatory access control?
20. Give an example of a scenario where the software doesn't trust the OS, hypervisor, and/or the hardware platform on which it runs? What can the software possibly do to "secure" itself in this situation?
21. Considering memory protection, explain how the operating system ensures that user-level processes don't access kernel-level memory?

CPU Scheduling

(1) What is a CPU scheduler? When does it execute?

(2) What is the difference between a CPU Scheduler and a Dispatcher?

(3) How does the operating system (or hypervisor) maintain control of the CPU? In other words, how does the OS prevent a process, such as a *while(1);* loop, from indefinitely running on the CPU without returning control back to the OS?

(4) Give at least three mechanism(s) by which the highest privileged software, such as an operating system or a hypervisor, retains control over the CPU? Which mechanism is absolutely essential for the OS/hypervisor to retain control over CPUs? Why?

Input/Output I/O

1. What is an interrupt? What is an interrupt handler?
2. What are character, block, and network devices? Give examples.
3. What are the factors affecting read/write latency in traditional magnetic hard disks?
4. Why is IOPS (I/O operations per second) considered a better measure of hard disk performance than raw bandwidth (bytes per second)?
5. Explain the role of *on-board disk cache* in hard disks during (a) read I/O operations and (b) write I/O operations.