Questions from Test 1

One question on concurrency from Test 1 will be repeated verbatim in Test 2.

Memory management

(1) Show the typical memory hierarchy of a computer system. Explain the tradeoffs between latency (access times), capacity, and persistence, across different levels of memory hierarchy.

(2) What is a page table?

(3) How many page tables are maintained by an operating system?

(4) If there were no TLB, how would memory accesses be affected?

(5) What are the following? What do they do? Where are they located?

A. Memory Management Unit (MMU)
B. Translation Lookaside Buffer (TLB)
C. Page tables
D. Swap device

(6) Where are MMU, Page Table and TLB located?

(7) What is a page fault and a TLB miss? Which system component resolves each of them?

(8) Which statements are true? If a statement is false, then correct it.

A. A page table is an array of physical memory pages
B. Page table entries dictate whether a process can read, write, or execute the contents of a physical page.
C. A page table maps physical page numbers to virtual page numbers
D. Page table entries can be used to track which memory pages are infrequently accessed.

(9) How is a virtual address converted to a physical address in a virtual memory system? Explain the roles of MMU, TLB, and Page Tables.

(10) If you increase or decrease the page size in a system, how (and why) will it affect ***(a)*** the size of the page tables, and ***(b)*** the TLB miss ratio?

(11) What's TLB Coverage? Why is TLB coverage important? How can one increase the TLB coverage?

(12) In memory management, what is meant by relocation and protection? Why are they needed?

(13) Consider a machine with B-bit architecture (i.e. virtual address and physical address are B bits long). Size of a page is P bytes.

A. What is the size (in bytes) of the virtual address space of a process?
B. How many bits in an address represent the byte offset into a page?
C. How many bits in an address are needed to determine the page number ?
D. How many page-table entries does a process' page-table contain?

(14) A machine has a 32-bit address space and an 4KB page. The page table is entirely in

hardware. Each page-table entry is 4 bytes in size. When a process starts, the page table is copied to the hardware from memory, at the rate of one byte every 25 nano-second. If each process has a CPU burst of 200 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables? (Assume that each process uses its entire virtual address space during execution.)

(15) Consider a machine having a 32-bit virtual address and 8KB page size.

A. What is the size (in bytes) of the virtual address space of a process?
B. How many bits in the 32-bit virtual address represent the byte offset into a page?
C. How many bits in the 32-bit address are needed to determine the page number ?
D. How many page-table entries does a process' page-table contain?

2. **[10 pts]** Consider a machine having a 64-bit virtual address and 16KB page size.

A. What is the size (in bytes) of the virtual address space of a process?
B. How many bits in the virtual address represent the byte offset into a page?
C. How many bits in the virtual address are needed to determine the page number ?
D. How many page-table entries does a process' page-table contain?

(16) For each of the following decimal virtual addresses, compute the virtual page number and offset for a 4-KB page, an 8 KB page, and a 16KB page: 20000, 32768, 60000.

(17) A computer with a 32-bit address uses a two-level page table. Virtual addresses are split into a 9-bit top-level page table field, an 10-bit second-level page table field, and an offset. How large are the pages and how many pages are there in the address space?

(18) Which system components handle TLB misses and page-faults, and how, in a machine with

(a) architected page-table?
(b) architected TLB?

(19) What is the purpose of "Referenced" and "Modified" ("Dirty") bits in the page table entry? How are they manipulated by the (a) hardware, and (b) operating system?

(20) Describe Optimal Page Replacement (OPR) algorithm. Why is it called "Optimal"? Why is it not practical to implement OPR?

(21) Explain the Least Recently Used (LRU) page replacement algorithm.

(22) Why is LRU a good approximation of Optimal Page Replacement (OPR)?

(23) OPR (Optimal Page Replacement) and LRU (Least Recently Used)

(a) Why is it impossible to implement OPR?
(b) Why is it hard to implement LRU?

(24) What is meant by "Internal" fragmentation?

(25) What is "External" fragmentation of memory? How can it be resolved?

Answer: External fragmentation occurs when all free memory regions are small and distributed at different non-contiguous locations in the main memory. As a result, an allocation request for a large contiguous memory region cannot be satisfied, even though enough free memory exists. External fragmentation is resolved by **compaction**, i.e. moving all allocated memory regions to one end of physical memory so that all free memory is contiguous at the other end.

(26) What is a "working set"? Why is it so important?

(27) Explain how the following page replacement algorithms work: (a) Clock, (b) WSClock (c) Second Chance.

(28) How does the Second Chance page replacement algorithm improve upon the FIFO page replacement algorithm?

(29) How does Clock page replacement algorithm improve upon the Second Chance page replacement algorithm?

(30) Briefly explain how the following page replacement algorithms work:
(a) FIFO (First In First Out)
(b) Second Chance
(c) Clock.

(31) Suppose that "page table pages are paged", meaning that (some or all of) the memory allocated to hold page tables can be paged-in and out of the main memory by the operating system. Suppose further that you have two-level page-tables, i.e. a first-level page-directory which tracks the second-level page table blocks.
(a) Which parts of the page-table can be paged (moved in and out of main memory)?
(b) Where are the memory address translations (i.e. page table entries) for the "paged page-table"?
(c) Can the memory used for your answer in (b) be paged? Why? Or Why not?

(32)

(33) Consider two processes that set up one page of shared memory for inter-process communication with each other. Given what you know about virtual memory management, explain how the OS would set up this shared memory page at the level of page tables?

(34) Suppose that the Operating System wanted to track (or intercept) every write performed to a specific memory page by a user-level process. Explain how the OS would achieve this goal?

(35) How does TLB Coverage and TLB miss ratio vary with the size of a page?

(36) Consider a virtual memory system running on an architected page-table hardware supporting two-level page tables. Page tables are not locked in memory and may be swapped to disk. An lw (load word) instruction reads one data word from memory; the address is the sum of the value in a register and an immediate constant stored in the instruction itself. Neither machine instructions nor page-table entries nor data words can cross a page boundary. In the worst case, how many page faults could be generated as a result of the fetch, decode, and execution of an lw instruction? Explain why?

(37) What is hysteresis? Explain how the page-out/page-in mechanism in the OS uses hysteresis and why?

OR

How does the swap daemon (paging mechanism) avoid rapid oscillations in paging activity when memory pressure increases?

(38) Under what condition does thrashing occur in memory management? How can the OS resolve thrashing?

(39) Considering memory protection, explain how the operating system ensures that user-level processes don't access kernel-level memory?

(40) How can the operating system track

A. Dirty (or updated) memory pages for the purpose of eviction?
B. Every memory write performed by a process to specific memory pages?

(41) What are superpages? Why are they useful? What are the constraints on their sizes, placement, and page attributes (protection, reference, and dirty bits)?

(42) Superpages

A. What are the advantages and disadvantages of superpages?
B. How many TLB entries and page table entries are there for each superpage?
C. What are the restrictions on superpage size, allocation, and placement?

(43) If a superpage has a size equal to 4 base pages, how many TLB entries are occupied by the superpage? How many page table entries are occupied by the same superpage? Explain why.

Superpages

(44) How do superpages affect (increase/decrease/don't change) internal and external memory fragmentation? Why?

(45) In the paper, superpage allocations are performed in two steps: preemptible reservations and incremental promotions. Why do we need this two-step mechanism? If we already know how much to reserve, why not create the entire superpage in one shot the first time any of its base pages is accessed?

(46) Why is it that using a mix of multiple superpage sizes tends to yield better application speedups than using superpages of only one size?

(47) In the "superpages paper" (Navaro et. al.), when and how are (a) reservation, (b) promotion, and (c) demotions carried out? (d) Why are reservations called "pre-emptible"?

(48) Why do superpages make both internal and external fragmentation worse?
OR
Which fragmentation (Internal/external) is made worse by superpages? Why?

(49) In the superpage paper, how does the system proposed avoid the need to page-out an entire superpage to the disk upon memory pressure? What is the source of performance overhead in this mechanism?

(50) Use of superpages (as they are currently designed) requires the use of contiguity restoration techniques in the operating systems. To avoid the need for contiguity restoration, one could get rid of the requirement that the base physical pages of a superpage be contiguous - in other words, allow the base pages in physical memory not to be next to each other. If one does so, describe how you would redesign (if at all) the TLB, page-tables, and the mechanism for translating virtual to physical addresses? You can assume either architected page-tables or architected TLB. Remember to list any assumption you make, justifying why they are reasonable.

Segmentation

(51) How are relocation and protection implemented in Pentium architecture? Consider the roles of both segmentation and paging.

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

(53) Using either Multics or Pentium architecture as an example, explain how segmentation is used in enforcing protection?

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

(55) In paged-segmentation implementations: Multics has one page table per segment, whereas Pentium has one page table for multiple segments. Why the difference? Which one is better? Why?

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

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

(57) Which of the following memory designs can cause internal fragmentation only, external fragmentation only, both, or neither? Briefly explain why.

A. Pure paging
B. Pure segmentation
C. Paging with Segmentation
D. Using superpages of the same size (no base pages or any other superpage size)
E. Using a mix of superpages of different sizes

(58)

File Systems

1. What is a File system

2. What's an i-node? Where is it stored?

3. What's the simplest data structure for an i-node? Then why is UNIX i-node so complicated?

4. In a file-system, (a) What is meta-data? (b) Where is meta-data stored? (c) Why is it important for a file system to maintain the meta-data information? (d) List some of the typical information that is part of the meta-data.

5. If you collect a trace of I/O operations below the file system cache (at device driver or physical disk level), what type of I/O operations do you expect to see more of -- write I/O requests or read I/O requests? Explain why.

6. (a) Suppose you collect a trace of I/O operations above the file system layer (in applications or in system calls). Do you expect to see more write I/O operations or read I/O operations? (b) Now suppose you collect a similar trace of I/O operations below the block device layer (in the disk or device driver). Do you expect to see more write I/O operations or read I/O operations? Explain why?

7. 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?
8. 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.
9. What does the file system cache do and how does it work? Explain with focus on the data structures used by the file system cache.

10. Explain the role of *file system cache* during (a) read I/O operations and (b) write I/O operations.

11. Describe two different data structures using which file system can track free space on the storage device. Explain relative advantages/disadvantages of each.
12. How does a log-structured file system work? Why is its performance (typically) better than conventional file systems?

13. In a file-system, explain how two different directories can contain a common (shared) file. In other words, how do hard links work?

14. 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***.
15. 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?
16. Explain the steps involved in converting a path-name */usr/bin/ls* to its i-node number for the file *ls*.
17. 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?

18. 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?

Explain your answers.

19. 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 pointer (entry in the inode)., 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.

20. 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).
21. 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).
22. 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.
23. 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.

24. In the "UNIX/Ritchie" paper, consider three major system components: files, I/O devices, and memory. UNIX treats I/O devices as special files in its file system. What other mappings are possible among the above three components? (In other words, which component can be treated as another component)? What would be the use for each possible new mapping?
25. Suppose your filesystem needs to store lots of uncompressed files that are very large (multiple terabytes) in size. (a) Describe any alternative design to the traditional UNIX inode

structure to reduce the size of inodes wherever possible (NOT reduce the file content, but reduce inode size)? (Hint: maybe you can exploit the nature of data stored in the file, but there may be other ways too). (b) What could be the advantage of your approach compared to just compressing the contents of each file?

26. Why doesn't the UNIX file-system allow hard links (a) to directories, and (b) across mounted file systems?

27. Why did the authors of the "UNIX" paper consider the hierarchical file-system to be their most important innovation?