CPS 110: Twenty Questions

The final exam will feature some synchronization problems. The final exam often includes questions about basic operating system mechanisms that you would learn from the Nachos labs if not in class. Be sure that you are familiar with your group's approach to the Nachos labs. Here are some questions to help guide your study of other material that may appear on the final exam this semester. These are a bit more open-ended and "thoughty" than the questions that will appear on the exam.

1. Briefly summarize the mechanisms that isolate the operating system kernel from interference by buggy or malicious user programs. How does this protection impact the ease of application development, and why? How does it impact application performance, and why? Is protection worth these costs?

2. Briefly summarize the mechanisms that isolate a process from interference by other processes running buggy or malicious programs. How does this protection impact the ease of application development, and why? How does it impact application performance, and why? (Consider the need for multiple applications to work together like the "puzzle pieces" in Microsoft Office.)

3. What machine features are needed to support protected operating system kernels? Briefly explain each feature and how it empowers the kernel to control user programs.

4. Many system features are typically implemented in the kernel, while others are typically implemented in user-mode libraries. Examples of features often provided at the library level include heap management (*new*, *malloc*, and *delete*), *printf*, and perhaps even support for threads. So-called "microkernel" systems aggressively move features out of the kernel and into user-mode code. What criteria should be used to decide where in the system to implement a given feature?

5. A key element of Unix philosophy is that the kernel interface for I/O to files and other I/O objects deals only with raw, unstructured byte streams. What are the advantages of this philosophy? What are the disadvantages? As a general rule, the internal structure of the data in Unix files is defined by user code and is not known to the kernel. I can think of three key exceptions to this rule: directories, symbolic links, and executables. Explain how these are exceptions to the rule, and why these exceptions exist.

6. How does a typical kernel (e.g., your Nachos kernel) decide how much physical memory to allocate to each user process at any given time? Is this policy "fair"? What program behaviors might cause a process to gain or lose memory "unfairly"? If the policy is unfair, what are its advantages? How could it be made more fair without sacrificing its advantages?

7. Why are threads useful for building server programs? It has been noted that construction of large server programs using threads is inefficient. What are the sources of inefficiency in threads? Can efficient server programs be built without using threads? What does this assume about the kernel I/O interfaces?

8. Briefly summarize how the Unix process creation primitive (*fork*) differs from the *Exec* primitive in Nachos. What are the advantages of *fork*? What are its disadvantages?

9. Briefly outline how to add a file cache to your Nachos kernel. When are entries added to the cache? When are they removed from the cache? The entries in most file caches (e.g., in Unix systems) are fixed-size pieces of files (blocks) rather than entire files. What are the advantages and disadvantages of caching blocks rather than complete files? Whether the entries are files or blocks, how and when are entries located in the file cache, i.e., what constitutes a cache "hit"? Under what conditions would a hit occur? Under what conditions are file caches effective for speeding up file reads, i.e., what do file caches assume about the file access patterns of user programs? How should the kernel decide how much memory to devote to the file cache vs. backing the virtual memory of user processes?

10. How does file (block) cache replacement/eviction differ from virtual memory replacement? Most block caches use a hybrid of exact LRU and MRU policies for victim selection in the file cache. Under what conditions is LRU a better policy than MRU? Under what conditions is MRU a better policy than LRU? When LRU is used, block caches typically use exact LRU rather than the approximate LRU replacement used in most virtual memory systems. Why is exact LRU feasible for file block caches but not for virtual memory page caches?

11. Summarize the differences between file block caches and VM page caches with respect to their handling of writes/updates to the cached blocks or pages. How and why are they similar? How and why are they different?

12. Disk block allocation is the problem of determining which physical disk block to assign to each file system block. This policy is the most important determinant of file system performance. Why? What kinds of disk access patterns do good allocation policies try to produce? What kinds of disk access patterns do they try to avoid?

13. How does a file system track which blocks are free on disk? Many schemes are possible: describe one that shows that you understand the basic challenges.

14. In Unix file systems, how do symbolic ("soft") links differ from "hard" links? Why have both? In a system with hard links (multiple names for the same file), how can the system determine when it is safe to free the blocks of a file, i.e., what constraints do hard links impose? Related question: does your favorite Unix system allow a file to be removed while some process has it open? How and why does the system handle this case in the way that it does?

15. What is a hierarchical page table? How does a hierarchical page table differ from the linear page table structure used in Nachos? What is the purpose of this difference? Draw a picture of a hierarchical page table, and outline the steps needed to handle a page fault with such a structure. How and why are hierarchical page tables similar to the block map (inode and indirect block) structure used in the classic Unix file system? How and why are these structures different?

16. What actions does your Nachos kernel take to select a page replacement candidate (victim) for a page fault? How might the problem be more difficult or the solution more complex in a "real" system?

17. What actions does your Nachos kernel take to evict a page? How might the problem be more difficult or the solution more complex in a "real" system?

18. What actions does your Nachos kernel take to initialize the contents of a page frame to handle a page fault? How might the problem be more difficult or the solution more complex in a "real" system?

19. What are the *mechanisms* needed to support the *policies* for demand paging and page replacement? Broadly speaking, what architectural (machine) features are necessary to support virtual memory, and how does the operating system use them?

20. Speculate on the actions taken by a disk interrupt handler in a typical operating system with a block buffer cache, i.e., what happens when an I/O operation completes? What state transitions are needed in the buffer cache structures, or in the threads and processes using the buffer cache?

20. What happens on process exit? Your answer should be specific and comprehensive.

20. Most VM operating systems support sharing of virtual memory between programs (e.g., for shared libraries or inter-process communication). Files may also be shared in various ways, e.g., two processes may open the same file by name, and children in Unix inherit open files from their parents. How does sharing complicate the schemes for virtual memory management (questions 16, 17, 18) and file block caching (if at all)?