CSE 451 Winter 2012: Final Exam Sample Study Questions

Note: for some questions, there may be multiple answers.

(1. Suppose there is a user level application that lets you run Linux binaries in Windows, called lin.exe. lin.exe is simply a user application that runs on Windows, and is not part of the Windows kernel, nor does it modify the kernel in any way.

- Suppose there is a binary called "linuxApp" that runs on Linux. Why can't this run directly on Windows? (Assume you compiled for the correct processor.)
- Briefly explain how lin.exe could work. Why can a program designed for Linux run in lin.exe but not Windows directly?

Binaries compiled for Linux rely on Linux bytecode layouts to interpret the binary: segments, symbols tables, memory mappings, system APIs, etc. are all different from Windows. To resolve this, lin.exe exists as a translation layer between the Windows system and the Linux system. All features and APIs, as well as how bytecode is interpreted, are translated from Linux to Windows.

(2. Assuming we are using a disk (not a solid state drive), name one advantage on each side for each of the following comparisons:

- Log-based FS vs. Journaling FS
- Unix FS vs. Log-based FS

Log-based FS writes faster than Journaling FS in the worst case: all writes are appended to the disk and requires very little seek time. Journaling FS flushes writes to disk, but in the event of a very large set of continuous writes, the disk will still require multiple seeks to write data from the log on disk to the data section on disk.

Both Journaling FS and Unix FS are more space efficient than the Log-based FS. Because Log-based FS appends to the log with every change, the file system contains multiple copies of the same file across multiple revisions, which creates a rather large space overhead.

Log-based FS (and Journaling FS) have faster crash recovery than the Unix FS. Because data is stored as a write-ahead log, atomicity of file updates is ensured. In the event of a crash, Log-based FS and Journaling FS only need to recover from the previous snapshot, whereas Unix FS requires an entire disk check to verify non-dangling file and block pointers.

(3. When would using separate processes be advantageous over using separate threads?

Separate processes are ideal for large tasks that share very little or no data with each other. Even when multiple instances of the same code are executed, processes may be ideal if each task is “heavyweight,” such as tasks that require large process spaces. Example: Chrome browser uses separate processes for each tab to implement sandboxing.

(4. Which scheduling algorithms (at the user level, assuming one processor) are best used for the following:

- web server (FIFO, Round Robin)
- compiling a kernel (FIFO, Priority)

- web browser fetching a webpage (Priority)
- cracking multiple encrypted messages of different lengths (Any)
- multi-threaded file copy (Any)

(5. What advantage does RAID 5 have over RAID 1? Over RAID 0? Disadvantages?

RAID 5 has better fault tolerance over RAID 0, since it can handle one failure, whereas RAID 0 can handle none. RAID 5 has better storage utilization than RAID 1, since RAID 5 doesn't require full replicas of each disk.

RAID 0 has better data utilization than RAID 5, since it doesn't require recovery information (parity, checksums, etc.). RAID 1 has better read performance than RAID 5 in some but very frequent cases; under RAID 1, a disk may be closer to you and serve you the file faster than RAID 5 can piece together multiple shards of one file across all disks.

(6. Suppose we doubled the size of the TLB. How does this affect performance of virtual memory?

Virtual memory lookups from memory would be up to twice as fast, since twice as many recent entries are stored inside the TLB. However, each individual TLB lookup will be up to twice as slow, since the lookup is still a linear-time operation.

(7.
- Assume we have a system that writes large files often but seldom reads them. What is the best FS to use?
- Assume we have a system that writes and reads small files very often. What is the best FS to use?

Writes large files often but seldom reads: Log-based FS, since writes are very fast on Log-based FS. If space utilization is an issue, then Unix FS may be used, since large file writes are still sequential, at the cost of seek time between writes.

Writes and reads small files very often: Journaling FS, since each small write on a single file only writes to that one file, and all changes are gathered in the log, rather than appending changes to the disk (as in Log-based FS).

(8. When would it make sense to use a Mesa monitor over a Hoare monitor, and vice versa? (We don't care about programmer convenience in this case.)

Mesa monitor is better used for situations where overall speed is more important, because a context switch isn't required. Hoare monitor is better for situations where a thread absolutely needs to execute immediately after it finishes waiting, e.g. if the thread is running a time-critical task.