Operating Systems   Final part II   06/13/18

Name ______________________________

Student ID # ______________________________

Seat Row ________   Seat Col ________   Exam # ______

This exam is comprised of six problems, each problem being a full page of related questions. You must choose THREE of these problems to answer. You cannot answer more than three problems. Review all of the questions and consider what answer you would give to EACH part before deciding which three to answer. All problems have the same value.

Read each question CAREFULLY, make sure you understand EXACTLY what question is being asked and what type of answer is expected, and make sure that your answer clearly and directly responds to the asked question. Many students lose many points for answering questions other than the one I asked. If you are unsure of what a question is asking for, raise your hand and ask.

I am looking for depth of understanding and the ability to solve real problems. I want to see specific answers. One-liners and vague generalities will receive little or no credit. Superficial answers that I may have accepted previously will not be accepted on this exam.

None of these questions requires long answers, but many of the questions may require you to do a lot of thinking and sketching before you come up with a reasonable answer. Feel free to use scratch paper to organize your thoughts. If the correct part of your answer is buried under a mountain of rambling, we may not find it.

If you need more space, you can overflow onto the back page. Note (on the problem page) that you are doing this.

If you answer more than three problems, we will only grade the first three.
If you change your mind, cross out the problems you do not want us to grade.

1: (a) List two important respects in which a virtual machine is fundamentally very similar to a (user-mode) process.

(b) List two important respects in which a virtual machine is fundamentally very different from a (user-mode) process.

(c) How will we enable the (guest) operating system in a virtual machine to execute privileged instructions (e.g. load processor status word or disable interrupts) ... while still retaining control of the computer and its critical resources?

(d) How will we enable the (guest) operating system in a virtual machine to perform device (e.g. disk or network) I/O operations?

(e) How will we enable the (guest) operating system in a virtual machine to get control when an application running in that virtual machine executes an instruction to request a system call?

(f) Ignoring time spent in the (guest) operating system, How fast should we expect an application in a virtual machine to run, compared to its speed on a comparable physical computer? Briefly justify your answer.

2: In a land-mark paper, Peter Denning asserted that the cost of any memory management strategy can be measured by the sum (over all processes) of the time-integral, for each process, of the number of page frames assigned to that process (or more simply the total number of page-seconds consumed by all of the processes in the system). If, for instance, a process has an image size of "N" pages and we leave it in memory for "T" seconds, the time-space cost of this decision is N*T page-seconds.

(a) Why is this time-space product a reasonable metric for the cost or performance of a memory management strategy? (e.g. show how this metric is well correlated with goodness)

(b) If we were to use pure demand-paging (w/o any pre-loading) rather than swapping, what (approximately) would be the time-space cost of allowing the same process to be in memory for the same "T" seconds (if we assume that the we started with no pages and that the process faulted for its N pages uniformly over the "T" second period)?

(c) Which I/O time likely to be longer:
Ts(N) ... the time to swap N contiguous pages into memory or back out to disk
Tp(N) ... the time to fault N individual pages in from, or flush them back to disk
Explain your answer.

(d) Is it likely that the demand paged process would require the same number of pages (n) to be read from disk into memory as the swapped process (N) during a single time-slice? Why or Why not?

(e) If the paged process did require the same number of in-memory pages (n=N), would the number of pages that had to be written back to disk (n') be equal for swapping and paging? Why, or why not?

(f) Write an equation (in terms of T, N, and Ts(N) for the total time-space cost of swapping a process in, running it, and swapping it back out to disk.

(g) Write an equation (in terms of T, n, n' and Tp()n) for the time-space cost of paging a process in, running it, and writing it back out to disk.

3: It is vital that the OS be trust-worthy. We can probably trust code on the vendor CD, but we will probably have to download some of the device drivers we need from the web. These "after-market" device drivers may not be as well designed and tested, or some may even contain Trojan Horses.

(a) Suggest a general approach to enable customers to know that a downloaded kernel module could be trusted.

(b) Suggest a responsible process that a trusted authority (e.g. the vendor) could use to ascertain that a submitted module was worthy of customer trust.

(c) How, IN DETAIL, would the authority (e.g. the vendor) label a module as trust-worthy?

(d) How, IN DETAIL, would a customer verify a module as being trust-worthy?

(e) Would it be a good idea for the OS to refuse to load un-certified modules? Justify your answer.

4: Consider three alternative cache invalidation approaches for a distributed file system that wants to provide open-after-close consistency:

1. clients re-validate cached files with the server before each open.
2. the server broadcasts notifications after each client-close/flush.
3. caching clients register with the server, and the server sends update notifications to them after each client close/flush.

(a) List and briefly describe the key load-characterization parameters (zero points for bottom-line event/message counts) we will need to estimate the number of messages associated with each approach.

(b) Approach 1: Write an equation for the number of re-validation messages and responses.

(c) Approach 2: Write an equation for the (total) number of change messages sent to/by the server.

(d) Approach 3: Write an equation for the (total) number of registration/unregistration and update messages sent to the server, and the number of update notifications from the server.

(e) estimate a plausible value for each parameter

(f) compute the number of messages for each alternative.

5: In discussing the increasing use of logging file systems, it was mentioned that they are often used to implement key-value stores (very simple and highly scalable non-relational databases whose only operations are get, put, and delete). (a) List two distinct key advantages of a logging file system, and briefly describe why they have these advantages.

(b) How can we make the look-up of a desired key efficient in a logging file system?

(c) What operation(s) might cause an old assignment (recorded in the log) to become stale (no longer valid)?

(d) Briefly describe the garbage collection process to be performed when the log wraps around?

(e) Briefly describe the process of recovering a log-based KVS after a crash, and how we manage the cost of this recovery.

(f) Is it possible to provide Read-After-Write consistency in such a system? Briefly explain why, or why not.

6a: A space probe runs an ever-changing mix of experiments, all of which require power, which is in very limited supply. The current draw for any given experiment is predictable, but varies over time. Aborting an experiment after it starts will waste irreplaceable resources, or perhaps even damage the instrument. If concurrent experiments are using most of the available power and all of them need more power to complete, we may find ourselves in a power-deadlock. How can we prevent this?

6b: A huge database contains merchandise, inventory, shipping, invoicing, order status and other such information. Most interesting business operations involve related changes to multiple records, and/or records of multiple types, but the records affected by an operation are known at the start of the operation. The database must be able to support thousands of concurrent operations at a time. How can we ensure that parallel requests for the same records do not deadlock, while preventing the locking from turning the database into a performance bottleneck.

6c: A critical section in a system event log is protected by a mutex. The event log is used throughout the OS, including by some interrupt handlers. How can we ensure the integrity of the log while preventing/avoiding deadlock.

6d: Some system calls (e.g. create(2), mkdir(2), link(2), unlink(2), rename(2)) require the locking of multiple I-nodes (e.g. the affected file and the containing directory/directories). How can we prevent deadlocks from coincidental concurrent operations involving the same files and directories?