Operating Systems Depth Exam

Instructions:
Answer any six of the following seven questions.

Question 1. Naming Services

(1a) Grapevine provides location information for (among other things) services and mailboxes. Describe the steps used by Grapevine to translate the name of a user or service into location information.

(1b) Grapevine is used in a corporate network. Discuss limitations in Grapevine that prevent it from being used effectively as the naming service in a global internet. How you would modify the design of Grapevine to remove these limitations?

(1c) How would you use Grapevine to additionally support reverse name mapping? For example, while Grapevine can map from host name to host address, you would also like it to be able to map from host address to host name. Answer in the context of part (b), i.e. for an internet name service.

Question 2. Distributed Time Stamps

Lamport describes an algorithm to logically order events in a distributed system. In his algorithm, events *a* and *b*, in processes *i* and *j*, have logical time stamps $C_i(a)$ and $C_j(b)$.

If $C_i(a) < C_j(b)$, we do not know if $a \rightarrow b$ or if *a* and *b* are un-ordered.

How would you extend the logical time information sent with each message to included enough information that you could disambiguate the case described above?

Question 3. Transport Protocols

A *connectionless service* (at any layer) accepts data packets from its user with the understanding that it will use its best efforts to deliver these packets to the user's peer entity. No guarantees are made with respect to reliability (packets may be lost, delayed, etc.) or with respect to the order of delivery of packets sent.

A *connection-oriented service* accepts data packets from its user with the understanding that they will all be delivered, without errors, to the user's peer and in the order sent. Moreover, the user will be informed if this is not possible.

Assume that a transport protocol is to provide a connection-oriented service to its users. Furthermore, assume that the network used by the transport protocol provides a connectionless service.

(3a) Why is such a transport protocol more complicated than a link protocol that provides a connection-oriented link service?

(3b) Describe two mechanisms employed by such transport protocols for dealing with the problem of old duplicates.

Question 4. Interprocess Communication

Consider two processes that need to communicate by sending messages.

(4a) If the processes are running on a *uniprocessor* and use operating system kernel calls (*send()* and *recv()* routines) to communicate, what would you expect the main costs (i.e., the most time-consuming activities) to be in communicating from process to process? How does the size of a message effect your answer?

(4b) Suppose that the two processes can share a part of their address spaces, and use this shared space to store their message queues. How does this effect your answer to part (a)?

(4c) Now suppose that the processes are running on separate processes in a symmetric shared-memory *multiprocessor*. Answer part (a) for this new scenario.

Question 5. Typed Objects

The Unix file system only distinguishes three "types" of objects: "plain" files, directories, and symbolic links. (It also supports block and character devices and—in some variants—sockets, FIFO's etc., but for the purposes of this question we will only consider objects that represent chunks of space on disk). It strictly enforces the distinction between these types but does not make any finer distinction among plain files, for example distinguishing binary and ASCII files, source files from executables, or C from Pascal.

(5a) How does the Unix kernel enforce these type distinctions? What operations are allowed or prohibited on each type of object?

(5b) Why does the Unix kernel enforce these restrictions? What sorts of things could go wrong it if relied on application programs, conventions, or libraries to enforce the distinction? Consider not only authorization issues (such as confidentiality and prevention of unauthorized access), but also questions about the integrity of critical system structures.

(5c) Suppose the system were modified to support a more extensive and detailed type structure. How could this facility help support modern distributed environments in which files are accessed by multiple hosts with varying architectures?

Question 6. Synchronization Give a solution, using monitors, to the ''Restroom Scheduling Problem'':

> There are two kinds of users, designated M and F, and one sharable resource (the restroom). Multiple users of the same type may use the resource simultaneously, but it is forbidden for users of different types to be accessing the resource at the same time. The following fairness constraint is imposed to prevent one kind of user from monopolizing the resource. The resource alternates among three *phases*: M phases, in which one or more M-type users are using it, F phases, in which one or more F-type users are using it, and I phases, in which the resource is idle. If there is a waiting M user and the number of F users who have accessed the resource since the beginning of the current phase is N or more, then all subsequently arriving F users must wait until the next M phase completes. The symmetric constraint (with M and F reversed) also holds.

List all the shared variables and condition variables you need, and indicate the code for the *request* and *release* entry procedures. (You need only show the code for one kind of user, since the code for the other kind of user is likely to be nearly identical.) You may assume either Hoare-style (wait/signal) or Mesa-style (wait/notify) conditions, but indicate which kind you are using.

Question 7. File System Performance

(7a) The original Unix file system allocated pages to files essentially randomly, so that the pages of a file would be scattered all over the disk. The result was that sequentially scanning (*reading*) a single large file was quite slow (only a few percent of the raw disk bandwidth). However, performance in a multi-user timesharing environment was acceptable, as was performance for sequentially *writing* (or creating) a file. Explain why.

(7b) The Berkeley "fast filesystem" fixed this problem, making the cost of sequentially reading or writing a file roughly the same. More recently, however, Rosenblum and Ousterhout argue in *The design and implementation of a log-structured file system* that rapidly dropping cost of RAM, and the resulting large disk-block caches, mean that file systems should work harder to optimize the performance of writing, even if it is at the cost of poorer performance for reading. Justify or refute this claim.

(7c) The solution proposed by Rosenblum and Ousterhout is to write modified pages sequentially, in a free area of the disk, rather than modifying them in-place (that is, overwriting them at their original locations). They propose a background "segment cleaner" process that reclaims obsolete copies of pages from "segments" (large (multi-megabyte), fixed-size chunks of disk). At first, they designed the segment cleaner to clean segments in order of the amount of free space they contained (emptiest first), but that turned out to be a bad policy. Simulations showed that the efficiency of the cleaning process was poor and did not benefit from locality in the workload (in which some files are modified more often than others). Explain why this "greedy" policy did not perform well.