Operating Systems Exam

Instructions:
Answer any six of the following seven questions.

Question 1. Security:

One of the factors that contributed to the explosive growth of the World Wide Web (WWW) was its simplicity of implementation. A *browser* program sends a string called a URL to a server, and the server sends back a document identified by the URL. The document is usually an ASCII file containing HTML (HyperText Markup Language), which consists of text and simple formatting commands (such as begin a paragraph or switch to Italics).

As the popularity of the Web and HTML grew, people wanted more and more features, such colors, images, sound, animation, tables, etc. The trouble was that anyone using a new feature in a document has to worry that the document will be unintelligible to any browser that doesn't implement the feature. Since there are millions of browsers in use, so this becomes a gigantic software maintenance and distribution problem.

Then someone got the bright idea of including with each document executable code capable of displaying it. This idea, while quite powerful, introduces a giant security problem. The user of a browser is being asked to run code from an untrusted source on his machine.

(1a) List three distinct nasty kinds of things that can happen. Assume the operating system is Unix.

(1b) What sorts of counter-measures could you employ to prevent some of the security breaches you listed in part (1a)?

To get around these problems, someone came up with the idea of sending code for a virtual machine implemented in software, rather than machine code for the actual hardware. Each browser would have to implement an interpreter for the virtual machine, but this would be the last new feature that would be required for all browsers.

(1c) How does the this idea ease the securing concerns listed above? What are some of the security violations that could be prevented by an interpreter that would be difficult or impossible to prevent if the document contained actual machine code?

Question 2. Page Tables:

A "standard" page table is very simple: It contains one entry for each page in a process' virtual address space. Each entry contains a bit to indicate whether the page is resident. If is, the entry points to (contains the physical address of) a *page frame* in physical memory. Otherwise, it may have the disk address of a copy of the page.

(2a) Processor architectures are also moving towards longer and longer virtual addresses, leading to larger and larger virtual address spaces and hence larger and larger page tables. On the other hand, RAM is getting cheaper, so physical memories are getting bigger. Nonetheless, it is now impractical to insist that the entire page table reside in physical memory. Why?

(2b) One solution to the problem is to put the page table itself in virtual memory. Outline how this solution works. In particular, there must be a page table listing the pages that contain the page table. How do you avoid getting into a vicious circle?

(2c) Another solution is to use an *inverted* page table, which contains one entry for each page frame, together with a translation lookaside buffer (TLB). Outline how this solution works, including all the data structures needed. Assume the TLB is implemented in hardware, but a TLB miss traps to the operating system (software), which must do everything else.

Question 3. Synchronization:

Lots of primitives have been proposed for synchronizing processes in shared memory, but monitors have emerged as the clear favorite.

(3a) Two variants of monitors have been proposed: the original Hoare (wait/signal) variety and the Mesa (wait/notify) variety. What is the difference between these two proposals? What are the arguments in favor of each one?

(3b) The programming language Java has monitors built in. They follow the Mesa (wait/notify) model, but there are no explicit condition variables. Instead, each monitor has exactly one (implicit) condition variable. A monitor procedure can call wait() to block the current thread, notify() to wake up a waiting thread (if any) or notifyAll() to wake up all waiting threads. Show how to solve the bounded buffer problem with this kind of monitor. Do you think the lack of explicit condition variables is a major shortcoming of the language? Why or why not?

Question 4. Transactions:

File systems that support transactions need to be able to atomically commit or abort a collection of file operations. This means that, before a commit or abort, the file system must keep both old and new versions of the parts of the files that have been modified. Two approaches are typically considered: (1) logging and (2) intention lists (or shadow pages).

(4a) Discuss the advantages and disadvantages of each of these approaches for keeping track of file data.

(4b) With each of the above two approaches, how would you handle meta-data operations such as file deleting and file name changing?

Depth 5. Interprocess Communication:

Consider two processes communicating with messages across a wide area net (WAN).

(5a) Identify each step that a message much go through from the time that the sending process does a **send** or **write** system call until the time that the receiving process completes it **recv** or **read** system call.

(5b) Give approximate costs for each of these steps (state the technology assumptions on which you're basing your estimates).

In both of the above questions, make sure that you describe the differences (if any) between a (1) sequence of small (less than 100 byte) messages, (2) a sequence of large (more than 64K byte) messages, and (3) and a sequence of RPC-like (request/response) operations.

Question 6. Distributed File Systems: Network file systems such as the Andrew File System (AFS) and the World Wide Web (WWW) both provide users access to remote files, but the two systems have very different user interfaces. In network file systems, the user only needs to mount the remote file system onto the local machine. Then he or she can access remote files just as if they were local. In WWW, a "browser" sends an "address" (a URL) of the file to a Web Server and displays the result to the user. (See also Question 1).

(6a) List the major differences between the two systems' services. In particular, point out their differences with respect to

- granularity of file accesses;
- semantics of the caching of remote files on local disks; and
- handling concurrent reads and writes of a file.

(6b) Since the two systems provide different services to the user, naturally, their implementations are different.

(i) Briefly outline the main features of the AFS file system implementation.

(ii) Suggest an algorithm for implementing the WWW. You do *not* have to describe how a browser implements the display of the Web documents, but you should suggest a framework for maintaining a cache of recently-accessed documents to decrease network traffic and for making sure the documents in the cache are up-to-date. If you happen to know the WWW actually does it, you can describe the algorithm. Otherwise, you can make up a plausible algorithm of your own.

(6c) If AFS were universally available, how would that simplify the task of implementing the WWW?

Question 7. Datagram Fragmentation

When datagrams are transmitted through a series of networks connected by gateways, fragmentation is sometimes required.

(7a) Why might it be necessary to fragment datagrams at gateways?

(7b) Compare the relative merits of reassembling the datagrams at gateways versus doing the reassembly at the destination host.