Operating Systems Depth Exam

Instructions:
Answer any 6 of questions 1-7.

Depth 1. Network File Systems:

(1a) Network file systems often using caching to improve performance. Describe three places that caching may occur between a client's file access and the actual read from disk. Give examples from systems that use caching.

(1b) In what way can each of these caches improve performance?

(1c) Your computer and the file servers have just been placed on a new network with 1 gigabit/second throughput. How would this new facility change your caching strategies?

Depth 2. Authentication:

The Kerberos authentication service produces *tickets* that a client can use to authenticate itself to a service. But previous versions of Kerberos (including the one describe in the paper in the exam reading list) did not allow the service then to pass this ticket on to other services (on behalf of the original client).

(2a) What caused this restriction?

(2b) In what cases is passing the tickets to other services a desirable feature? Give an example.

(2c) In what cases is passing the tickets to other services a bad idea? Give an example.

Depth 3. Network Partitioning:

In reliable systems that keep replicated data on different hosts, a network partition can be a difficult fault to handle.

(3a) What is a network partition? What problems can it cause for systems that replicate data?

(3b) Describe a mechanism that allows a service to continue to provide service in the face of a network partition.

(3c) In the cases that you can proceed, what operations need to be done when the partition fault is repaired? Which of these operations can be done automatically and which must have human intervention?

Depth 4. Protection:

Compare file protection based on user groups (as in Unix) with access-control lists, in which individual users are designated as having specific access privileges to a file. Consider complexity of implementation, ease of use, flexibility, and security in your answer.

Depth 5. Synchronization:

The Unix kernel uses a very simple low-level facility for its internal synchronization. The primitive

```
void event_wait(void *event);
```

blocks the current process until another process calls

```
void event_signal(void *event);
```

with the same value of `event`. If multiple processes are waiting for the same `event`, an arbitrary one is started. The `event` value can an arbitrary. A common convention is to use the address of a data structure as the `event` value. For example, a process might wait for a queue to become non-empty by calling `event_wait` with the address of the queue header as an argument. Any process adding to the queue is expected to call `event_signal`.

[**Note:** For parts (5a)-(5c) of this question, assume a conventional uniprocessor system.]

(5a) The only "data structure" used to implement this facility in the Unix kernel is an `event_waited_for` field in each process descriptor. Thus the total space overhead is proportional to the number of processes and is independent of the number of distinct `events` waited for. Outline how `event_wait` and `event_signal` might be implemented with this structure. What is the time overhead of your scheme? What fairness guarantees do you provide?

(5b) This facility is awkward to use because signals can easily be "lost." Give an example of how this problem might arise. Why are semaphores "better" than `event_wait/event_signal` in this respect?

(5c) Sometimes a process may want to wait for a combination of events. Outline how you might modify your answer to (5a) to support the primitives

```
void or_wait(int n_events, void *events[]);
void and_wait(int n_events, void *events[]);
```

(`or_wait` blocks the process until any of listed events is signaled; `and_wait` blocks until they are all signaled).

(d) How would your answer to part (5a) change if you were working on multi-processor like the Sequent, which has a few processors sharing cache-coherent memory? What if there were a few hundred processors? What about a multicomputer without no shared memory?

Depth 6. Typed Messages:

Most message-based operating systems support messages that are untyped byte streams. User code often has many lines of message packing and unpacking (which is tiresome to write and visually clutters the code).

(6a) Support for typed messages can be placed in the language or the operating system (or in both). Discuss the trade-offs of where to put typed message features.

(6b) How do heterogeneous architectures affect the design of a typed message system?

Depth 7. Write-Once File Server:

Consider the problem of building a file server from a write-once optical disk. This disk allows each sector of the disk to be written at most once (unwritten sectors read as all 1's). How would you construct a Unix-compatible server that uses such a disk as its *only* non-volatile memory. Both client and server machines have 10 MIPS processors and 256 MB of RAM. Client machines have no non-volatile memory.

Describe the significant file system data structures, including those on disk and those kept in RAM. Which functions will be performed on the server and which will be done on the client? How is your design influenced by the characteristics of the expected workload and by estimated reliability (mean time between failures) of the workstations?