Operating Systems Depth Exam

Instructions:
Answer any six of the following seven questions.

1. Disk Allocation

Disk performance is greatly improved if disk pages that are often accessed at about the same time are stored near each other on disk. In Unix for example, two pages of the same file are likely to be accessed together, so they should be clustered together on disk.

A recent paper suggests separating disk page allocation from the rest of the file system by defining a "Logical Disk" (LD) interface. This interface manages disk space using a two-level hierarchy. Blocks are named by logical block numbers. Each block is allocated from a *list*, and the set of lists is also organized as a linear list. The interface calls are

L′ = new_list(L,hints)	create a new list following *L* in the list of lists
B′ = new_block(L,B)	allocate a new block in list *L* following block *B*
read(B,buffer)	read block *B* into *buffer*
write(B,buffer)	write block *B* from *buffer*
delete_block(B)	release block *B*
delete_list(L)	delete *L* and all of its blocks

The *hints* in the *new_list* call indicate whether the blocks in the list should be compressed and/or clustered, and whether the list itself should be clustered near its predecessor.

How would you use the LD interface to implement a Unix file system (inodes, directories, indirect blocks and data blocks)? In particular, explain how you would use the two-level hierarchy to get the kind of performance achieved by the Berkeley Fast File System (ffs) currently used by many flavors of Unix.

Depth 2. Distributed Barriers

Barriers are a synchronization mechanism that requires all processes in a program to reach a given point before proceeding. Each process calls `barrier(b)` and each process blocks until all processes have made the barrier call; when the last process makes the barrier call, all processes are unblocked.

Assume that you are writing distributed programs that run on a network of UNIX workstations. These workstations support standard stream (TCP/IP) and datagram (UDP) message communications.

(2a) How would you implement barrier synchronization in this environment? Assume that you want to make the operations as fast as possible (minimal delay).

(2b) Is load on the network (assume a typical Ethernet) likely to be a problem in your solution? If yes, describe in what cases this will occur. If no, explain why not.

Depth 3. Multiple Page Sizes

New processors are being designed to support multiple memory page sizes. Traditional paging systems assume that each page in a page table is the same size. These new processors allow *any* page to be *any* size that is a power of two within a certain range (typically from 512 bytes to 16K).

The page table can be thought of as a function that translates from virtual page numbers (VPN) of physical page numbers (PPN). In current systems, when the operating needs to translate a VPN, they simply treat the page table as an array and use the VPN as the subscript. Assume that there is no segmentation (only paging).

(3a) What kind of data structure could the operating system use to represent a page table with different page sizes?

(3b) What kind of data structure would you use to keep track of free space in physical memory?

Depth 4. User-level Threads

Consider the implementation of threads on top of regular processes (*user-level threads*). Assume that your operating system allows multiple processes to share an address space. For each of the following design choices, describe the alternatives from the points of view of implementation complexity and usefulness to the applications programmer.

(4a) Preemptive vs. non-preemptive.

(4b) Threads running in a single user process vs. running in multiple user processes.

Depth 5. Byzantine Generals Meet Reality

One solution to the Byzantine Generals problem makes the following assumptions about the underlying system:

(5a) Every message that is sent is delivered correctly.

(5b) The receiver of a message can reliably determine the identity of the process from which it was (most recently) sent.

(5c) The receiver of a signed message can reliably determine whether the message has been modified since it was signed.

(5d) Anyone can verify the authenticity of a signature.

Discuss whether each of these assumptions is reasonable in light of the way distributed systems really work. If an assumption is reasonable, describe how a system would support such a feature. Where applicable, provide examples from systems that you know. If an assumption is unreasonable, explain why the feature is difficult to provide.

Depth 6. Process Placement vs. Migration

Process *migration* allows a process to be moved any time during execution, while process *placement* only allows the location of a process to be chosen at the time it is created.

Describe the advantages and disadvantages of migration and placement. For each of these two mechanisms, consider

(6a) ease of implementation,

(6b) usefulness, and

(6c) cost of performing the operation.

Depth 7. File Servers

Choose three of the four systems NFS, Sprite, Locus, and Andrew, and compare them with respect to strategies for

(7a) Granularity of caching.

(7b) Prevention or recovery from conflicting write operations.

(7c) Flushing “dirty” data from the client cache to the server.

(7d) Preventing readers from getting “stale” data from the client cache.