CS-736: Midterm Exam (Fall 2000)

	Points	Total Possible
Writing your first and last name on all **7** pages	1	1
Part I: Short Answers	51	$(17 \times 3) \rightarrow 51$
Part II: Long Answers	48	$(3 \times 16) \rightarrow 48$
Total	100	100

Name: Smarty Pants

Part I: Short Questions

The following questions are multiple choice. There are two styles of questions. In the first, there is one and only one "right" answer. In the second, there are between one and four right answers - you should circle all that apply. Ones of the second style are marked as appropriate.

Each of the seventeen (17) questions is worth three (3) points for a total of fifty-one (51) points in this section.

1. In Pilot, a program can give advice to the operating system in order to tailor the OS's behavior to the program's liking. The Space.Activate() call on a region of a mapped file results in

   (a) the OS immediately fetching those pages

   (b) the OS knowing that those pages should be fetched soon

   (c) in-kernel data structures related to the space getting marked as "hot"

   (d) the pages of the space are all moved back on the global replacement list

   (b): Advice is just a hint, and the activate command just lets the system know that a particular "space" (a contiguous set of pages) will be needed by the application soon.

2. Assume there are three threads in a process, labeled A, B, and C. The process wishes to allocate 50% of its time to thread A, and let B and C share the other 50%. Assuming that we are using resource containers to account for CPU time of the jobs, what is the minimal number of resource containers that are required for this?

   (a) 1

   (b) 2

   (c) 3

   (d) 1.5

   (b): One resource container for thread A, and one shared between B and C. The scheduler can thus distinguish between the two and allocate 50% of time to A and the other 50% to B and C.

3. In Elephant, three policies were used to differentiate between historical accesses. "Keep all" keeps all versions of a file ever created; "Keep Safe" just keeps enough data around such that undo is possible; "Keep Landmarks" keeps a pruned history with landmark versions intact. If "Keep Safe" and "Keep Landmarks" were removed from the system, in which of the following ways could the system be simplified? **(Circle all that apply)**

   (a) No need for inode logs

   (b) No need for copy-on-write

   (c) No need for versioned directories

   (d) No need for the cleaner

   (d): By getting rid of "keep landmarks" and "keep safe", we never have to reclaim old data, and therefore the cleaner is obviated. All other mechanisms are needed by the "keep all" policy.

4. Which of the following are true for the VAX/VMS system? **(Circle all that apply)**

   (a) simple FIFOs are used to track process resident sets

   (b) a memory hog will not get more memory than other processes

   (c) the $system$ base register contains a virtual address

   (d) the P_0 and P_1 base registers contain virtual addresses

(a) and (d): FIFOs are indeed used to track resident sets, and because the user page tables are pageable, the P_0 and P_1 registers contain virtual addresses. The $system$ base register contains a physical address, though, and a memory hog can indeed get more memeory through the globally shared free or modified lists.

5. In the original Unix system, the i-number (or "index number") of a file can be used to locate the file's inode

   (a) by scanning the free list
   (b) by checking the bit map
   (c) by performing a simple "base + offset" calculation
   (d) not at all (the i-number can't be used to directly find the inode)

   (c): Remember that i-nodes are in a fixed place on disk in the original Unix file system (and FFS too). Thus, a simple base + offset calculation is all you need given a particular i-number to find the i-node.

6. In the exokernel, an "abort" protocol is sometimes used when the exokernel wants to take back some resources from a libOS. In which of the following situations could an abort be required? **(Circle all that apply)**

   (a) The exokernel wants the libOS to give back a page in a certain amount of time, and the libOS is tardy in doing so
   (b) The exokernel wants the libOS to give back a disk block in a certain amount of time, and the libOS is tardy in doing so
   (c) The exokernel wants the libOS to give up the CPU, and the libOS is tardy in doing so
   (d) The libOS is multiplexing between two application-level threads, and one thread is not yielding the processor

   (a) and (b): An abort protocol may be necessary when the exokernel wants to give the libOS some choice as to which resource is taken away – for example, when taking away a page of memory, let the libOS pick. However, what if the libOS takes too long? Then we abort them, and remove the resource forcefully. This does not apply to the CPU, which there is only one of, and therefore which the exokernel just takes away without telling anybody. Also, if the libOS is running threads, that is its business, and has nothing to do with the abort protocol.

7. Multics procedure segments are said to be "pure". What does that mean? **(Circle all that apply)**

   (a) They are generated by a compiler
   (b) When executed, they will run to completion
   (c) When executed, they will not write into other segments' data sections
   (d) When executed, they will not change their own contents

   (d): Pure and simple - the execution of a pure procedure "must not cause a single word of its content to be changed".

8. Unix FFS used which of the following techniques to reduce the ill effects of *disk seeks* that were prevalent in the old Unix file system? **(Circle all that apply)**

   (a) Bit maps
   (b) Sub-blocks
   (c) Cylinder groups
   (d) Rotational layout tables

   (a) and (c): Bit maps remove the need for the free list, which led to bad allocations for files and lots of seeks. Cylinder groups put related data and inodes close to one another, and therefore also reduce seeks. Rotational layout tables are used to reduce rotational costs, not seeks! And sub-blocks improve bandwidth, and are not a seek-reducing thing.

9. When calling read() from an application to read a single data block from an open file in LFS, how many disk reads could occur? **(Circle all that apply)** *Smarty Pants*

   (a) Zero
   (b) One
   (c) Two
   (d) More than two

   (a), (b), (c), and (d): Zero if the data is in the file cache, one if the inode is cached but the data block isn't, two if we have to read an inode block and a data block, or have to read an indirect block before reading the block, and more than two with many levels of indirect pointers.

10. In Nucleus, assume a server process implements a user-level file system. What prevents a malicious process from sending a "fake" response from the file system to a client waiting for a response?

    (a) Mutual trust among processes
    (b) The client waits for a particular buffer from the server
    (c) Only the server can send send a message to waiting clients
    (d) The Nucleus kernel monitors all communication carefully

    (b): A particular buffer identifies a complete communication in Nucleus, which is allocated during send_message() and then used in the subsequent wait_message(), send_answer(), and wait_answer() calls.

11. Which of the following is true for RAID-5 as compared to RAID-4? **(Circle all that apply)**

    (a) RAID-5 has potentially higher read bandwidth
    (b) RAID-5 has potentially higher write bandwidth
    (c) RAID-5 has potentially lower read bandwidth under a single failure
    (d) RAID-5 potentially uses more capacity

    (a) and (b): RAID-5 has higher read bandwidth because it uses all N disks for reading, whereas RAID-4 uses N-1. RAID-5 has higher write bandwidth under many concurrent small writes because it does not have a single parity disk as a bottleneck. They both use the same capacity.

12. In Hydra, if a process has a reference to an object, it can for certain

    (a) modify the object's entire representation
    (b) modify only the object's data part
    (c) destroy the object
    (d) perform operations in accordance with the access rights of the reference

    (d): A reference in Hydra is a pointer and a set of kernel-maintained capabilities. Those capabilities dictate exactly what you can do with the object.

13. In which of the following aspects would you say a deterministic proportional-share scheduler has *clear benefits* over randomized, lottery-based scheduling? **(Circle all that apply)**

    (a) predictability
    (b) amount of state per job in system
    (c) precision in allocation of proportions
    (d) amount of time to pick who runs for the next quantum

    (a) and (c): A non-deterministic system is more predictable (as it does not rely on randomness), and is quite precise in its allocations. Randomized lotteries maintain less state (just the number of tickets), and the time to pick who runs next is probably comparable in each.

14. Mach and Nucleus are both quite similar in basic structure. Which of the following are enhancements that Mach provided that were not available in Nucleus? **(Circle all that apply)** *Smarty Pants*

   (a) A communication system that is tightly integrated with virtual memory
   (b) Communication between processes that is kernel protected
   (c) Some typical "system" services can be provided by user-level processes
   (d) Multiple threads of control per address space

   (a) and (d): Both (b) and (c) exist in both systems. Mach went beyond Nucleus with its strong integration with virtual memory (recall that Nucleus did not even support paging), and with true threads (Nucleus just had processes).

15. The THE system is organized in a strict hierarchy, virtualizing aspects of the machine one at a time, with higher-level abstractions built on top of the earlier layers. The lowest level of virtualization ("level 0") is of:

   (a) the processor
   (b) core memory
   (c) the drum
   (d) the keyboard console

   (a): The processor is the first thing virtualized via the clock interrupt – at higher levels, the system is a virtual multiprocessor.

16. A TLB miss occurs in an application running on Irix running on top of Disco. Which of the following represents a legal control flow? **(Circle all that apply)**

   (a) Irix TLB miss handler → resume application
   (b) Disco → resume application
   (c) Disco → Irix TLB miss handler → Disco → resume application
   (d) Disco → Irix TLB miss handler → Irix page fault handler → Disco → resume application

   (b), (c), and (d): (b) is possible when the software TLB inside of Disco has the proper translation, so it can thus do the update directly and resume the application. (c) just dictates the normal course of action when Disco doesn't have the translation but Irix does, and (d) is possible on a page fault (not just a TLB fault). (a) is not possible because the application does not trap directly into Irix.

17. In Scheduler Activations, assume that we have a process running on four processors (and hence 4 scheduler activations). The kernel then decides to preempt two of those processors to give to another process. How many processor preemptions are required to remove two processors from the process?

   (a) 1
   (b) 2
   (c) 3
   (d) 4

   (c): 3 are needed, 2 to take away two processors, and the third to take away another processor and tell the application what happened. The system does not need 4 to do this simply because more than one piece of information can be combined into a single upcall.

Part II: Longer Questions

Smarty Pants

The second half of the exam consists of three longer questions. Try to sketch out the basic idea of your answers before getting bogged down in details.

Each of the three (3) questions is worth sixteen (16) points for a total of forty-eight (48) points in this section.

1. **A Question Of Structure.**

   In this question, we explore using the mechanisms in Hydra as building blocks for operating system kernels. Hydra provides a generalized protected control transfer known as a *call* and an early form of object oriented programming to enable programmers to assemble whatever type of operating environment they would like.

   a): Describe the call mechanism. How is it similar to and different from a trap into a modern-day operating system? How does one increase privilege across a call boundary?

   A call in Hydra is a protected transfer of control from one domain to another. It is the basic primitive that Hydra provides on which to build all other services. A standard OS trap is quite limited: an application calls into the OS for service, and the OS is at a full privilege, whereas applications have very limited privileges. Hydra provides much more flexibility, and much finer-grained control over levels of privilege among arbitrary pieces of code. Finally, the mechanism for increasing privilege is the *amplifation template*, which allows the creator of the module to specify the needed rights to call into a routine, and then a new set of rights with which the caller will execute during the lifetime of the call. Those new privileges may exceed the normal rights of the caller.

   b 1): Assume we wish to build a communication infrastructure similar to that of the Nucleus system with the Hydra primitives. First, describe the nucleus communication system.

   The Nucleus communication system is quite simple. Four primitives are provided by the kernel: send_message, wait_message, send_answer, and wait_answer. A buffer is allocated during send message and can be used by a process to identify a particular communication exchange. Processes can send messages to one another (to other "internal processes"), or to the kernel to perform I/O (to "external processes").

   b 2): Now, describe how to build the Nucleus communication system on top of the Hydra primitives. What objects would you provide? Where would you use the call mechanism?

   There are three keys here. First, we must remember that the Hydra kernel does not provide any real services, just the mechanisms with which to build them. Second, we must have a way for processes to find one another (when process A wants to send to process B, how does it name process B?). Finally, the exact Nucleus communication primitives and their semantics must be supported.

   Thus, we provide two basic objects. First, there is a communication manager object, which each process must register with. In registering, each process is assigned a small number which other processes can use as a communication address. The communication manager also provides the basic four Nucleus calls, and therefore does all of the necessary buffer allocation and management therein, perhaps keeping a queue of messages for each process that registers. The basic protection mechanisms of Hydra can be used here to prevent arbitrary insertion of messages or answers and the like. What's actually quite hard is that the manager needs to know when a process exits, in order to free up its resources and send dummy messages as Nucleus semantics dictate – functionality the process sub-system would need to provide.

   Second, and less importantly, there is a buffer object, which is what gets passed among processes. This could be quite simple, and does not require much in terms of the protected aspects of Hydra, because only the designated receiver of a message will have access to the buffer.

2. **Be Very Af-RAID.** *Smarty Pants*

Assume you have a RAID-5 system with D + 1 disks. Each disk has B blocks of capacity, and all disks are of identical make, model, and performance. Unfortunately, one of your data disks fails, and you replace it with a brand new disk. The system must then carry out the task of updating the new disk so that the RAID-5 returns to its normal "working" status (*i.e.*, all the data blocks have all their data and the parity blocks have all their necessary parity information), a process known as *reconstruction*.

a): Describe how the process of reconstruction would work for RAID-5.

To reconstruct the data, assuming a simple XOR parity scheme, we simply need to XOR the data from the other disks (read blocks from each disk) and then write the result of that to the new replacement disk. Turns out it doesn't matter if the failed blocks are data blocks or parity blocks.

Example: On disk A we have 0, disk B we have 1, and disk C (parity block) we have a 1 (assume even parity). Assume that disk A fails. To reconstruct A, we simply XOR B and C (1 XOR 1), and get a 0. If B fails, 0 XOR 1 gets us the needed 1. If C fails (the parity information is lost), we simply XOR A and B, and still get the right answer, 1.

b): During this reconstruction, how many blocks must be read from the RAID-5 storage system, and how many written?

We must read all the non-failed blocks from D disks ($D \cdot B$ blocks), and then write out the reconstructed blocks to one disk (B blocks).

c): Now assume that we had instead created a storage system with mirroring instead of RAID-5, with a total of 2D disks for the system. Describe how reconstruction would work in the mirrored system.

Mirrored reconstruction is much easier - just copy the data from the non-failed mirror to the new disk. B reads and B writes.

d): Which is faster - reconstruction in RAID-5 or reconstruction in the mirrored system? (justify)

RAID-5 seemingly has more work to do, as it has to read $D \cdot B$ blocks and compute parity, but the reads can be done in parallel, and parity computation is fast. Therefore, the answer is that they roughly proceed at the same rate (assuming a simple mirrored layout).

3. **It's Your Lucky Day.** *Smarty Pants*

In this question, we explore the application of lottery scheduling to disks. Assume we have machine with a single disk attached, and we want to provide applications with proportional share of the disks both in terms of capacity and performance.

a): In CPU lottery scheduling, a single list was used to track each job; each node of the list held the number of tickets allocated to that job. What kind of data structures would you use to organize the disk-based lottery? (describe and justify)

Because we wish to proportionally share performance (bandwidth) and capacity, we keep two lists instead of one. In basic structure, the bandwidth list just tracks the number of tickets per process, quite similar to the standard CPU list. For the capacity list, we will need to hold an inverse lottery, so the list will need to reflect that (described further in part d).

For all disk requests in the system, we will have a queue of requests per process, and pull requests off of the queues as determined by the lottery (described in part b).

b): With CPU lotteries, a lottery was held every quantum to determine who would run for the next quantum. When applying lottery scheduling to disk performance, how often would you hold a lottery? (justify)

The bandwidth lottery could be held every time we wish to schedule a disk request, because fundamentally, that is when we decide which application's requests to service. However, this would not allow us to perform any sort of fancy disk scheduling (*e.g.*, C-SCAN), so instead we might hold N lotteries after some number of requests greater than N have been made, and thus allow us to improve overall performance. That said, by holding lotteries on the granularity of requests, we may not be sharing bandwidth in terms of bytes/second proportionally, but instead are sharing requests/second.

c): Any time we are applying a proportional-share approach, the effect on performance comes into question. In the CPU-based system, how did lottery scheduling affect overall system performance? What kind of effect could a proportional-share system have on overall disk performance?

In any proportional-share system, performance is a secondary consideration. Consider that in CPU scheduling, shortest-job first is the best ideal policy. A standard multi-level feedback queue provides a real-world approximation of this, by favoring new jobs and decreasing the priority of jobs that have run for a long time. Thus, lottery scheduling forsakes true global performance for proportional-sharing.

The same problem occurs for disks. Because disk scheduling is so important (the order that requests are presented to the drives dramatically impacts performance), any sort of proportional-share scheme has the potential to negatively impact performance.

d): Finally, we also wish to use lotteries to proportionally share disk capacity. When would we hold a lottery in this case? What other issues arise in using lotteries for sharing capacity instead of performance?

Capacity is quite hard to share, but we could do it by holding a lottery when we run out of disk space. This lottery would be an *inverse* lottery; that is, the more tickets you have, the less of a chance of getting selected you should have. If selected, your process would have to give up a disk block.

The really hard issue here is how to reclaim a disk block that someone is already using! Unlike CPU time, disk space is finite, and thus we have this thorny issue to deal with. We could do something like the exokernel with the abort protocol and all of that, but basically we are stuck: if we have to reclaim a block, someone will be unhappy. Maybe quotas are a better way to go.