Student ID: _____

CS-736 Midterm: It's Really Quite Simple (Spring 2005)

An Arpaci-Dusseau Exam

Please Read All Questions Carefully!

There are eight (8) total numbered pages

Please put your NAME ONLY on this page, and your STUDENT ID on this and all other pages

Name: _____

Student ID: _____

This is the grading page.

	Points	Total Possible
1		5
2		5
3		5
4		5
5		5
6		5
7		5
8		5
9		5
10		5
11		5
12		5
13		5
14		5
15		5
16		5
17		5
18		5
19		5
20		5
Total		100

We all strive to build simpler systems. In this exam, we will! Below are descriptions of simplified versions of most of the systems we have read about in class.

Your task: For each question, describe the impact of the simplification on the system. What is the implication of the change? Is it good? Is it bad? Consider issues such as changes in the *performance*, *correctness*, or other important characteristics of the system (e.g., reliability if that's relevant). Also, use bullet points with short, supporting sentences; no need to write long and flowery paragraphs.

Just to make sure you understand what to do, here is an example. How nice!

Example Question:
Scheduler Activations (SA) informs you when a scheduler activation blocks on a page fault. SimpleSA doesn't.

Example Answer:
Implication: SimpleSA doesn't inform the user-level thread scheduler of an important event – a thread blocking on a page fault – and hence a processor may go idle while the page fault is being serviced.
Bad: This is definitely a performance problem; opportunities for the user-level library to schedule threads will be missed.
Good: Not much good about this, other than perhaps some reduction in complexity of the implementation. If you wanted this, you'd probably just be multiplexing user-level threads on top of kernel threads; no need to use a system like scheduler activations.
Good (alternate acceptable answer): Nothing much good about this.

You, dear student, unfortunately have to answer the remaining questions.

Student ID: _____

1. Hydra has a “walk” right for each object in the system. SimpleHydra doesn’t.

2. Disco has a second-level software TLB. SimpleDisco doesn’t.

3. Nooks installs and manages a different page table per device driver. SimpleNooks has *one* such page table for *all* drivers.

4. FFS spreads “large” files across multiple cylinder groups. SimpleFFS doesn’t do anything special for large files.

5. LFS caches the inode map (imap) in memory. SimpleLFS doesn't.

6. IO-Lite has an mmap() facility. SimpleIO-Lite doesn't.

7. Anticipatory Scheduling (AS) uses a cost-benefit formula to determine if and how long to wait after an I/O has completed. SimpleAS simply waits for about 10 milliseconds after a process that is reading a file sequentially has completed a read request.

8. RAID Level 5 rotates parity across disks. SimpleRAID Level 5 keeps all parity on a single disk.

9. D-GRAID replicates certain “popular” directories, containing program binaries and such, across many drives. SimpleD-GRAID doesn’t.

10. Mesa adds a little bit of extra state internal to condition variables that are used by devices in order to help with the “naked” notify. SimpleMesa just uses plain old condition variables when communicating with devices.

11. Lottery Scheduling uses randomness to determine who runs next. SimpleLottery simply uses a weighted round-robin approach without randomness.

12. Resource Containers (RCs) form a hierarchy, with parents and children. SimpleRCs have no hierarchy.

13. Vax/VMS has two page tables per user process. SimpleVax/VMS has just one.

14. Coda uses a prioritized hoarding algorithm to decide what to cache. SimpleCoda uses straightforward LRU.

15. Multics does a path lookup the first time an object is referenced, translating the "abstract" pathname into a machine-efficient form. SimpleMultics does all of these lookups when the program is loaded into memory (eagerly).

16. ESX Server uses content hashes to determine if two pages in memory are similar, and then compares potential matching pages byte-by-byte before coalescing them into a single copy-on-write copy. SimpleESX does not do the byte-by-byte comparison, using solely the hash match to decide when to coalesce.

17. An AFS client removes all callbacks from its cached files (but keeps the files around) when recovering from a crash. A SimpleAFS client simply keeps all of those callbacks and files.

18. NFS keeps file metadata (obtained from a `getattr` protocol message) in a cache, which expires every **3 seconds** or so. In SimpleNFS, there is no metadata cache.

19. In the Baker paper on "Measuring Distributed File Systems", Baker records open(), lseek(), and close() events (and the location of the file pointer at those points) in their trace. Baker's sister, SimpleBaker, only records open() and close() events in her trace.

20. Exokernel is so simple already, or is it? In what way could you simplify exokernel to make it a "better" system?