04 11 4	TT	
Student	11):	

## CS-736 Midterm: Off By One (Fall 2007)

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. Why? So I can grade without knowing who you are. Particularly useful for students who think I don't like them.

Note: I like all of you, so this is not really a problem.

Name: \_\_\_\_\_

## 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
Total		70

Student	ID:	
Student	ш:	

One of the most common programming errors that occurs is known as "off by one." As defined:

off-by-one error /n./

Exceedingly common error induced in many ways, such as by starting at 0 when you should have started at 1 or vice-versa, or by writing < N instead of <= N or vice-versa. Also applied to giving something to the person next to the one who should have gotten it.

Unfortunately (for you), numerous "off by one" errors have recently been discovered in some of the systems have you have studying (shocking! but true). In this exam, your job is simple: discuss how the off-by-one error in each question will affect the system under examination.

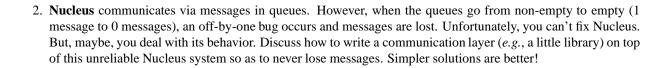
NOTE: Please read each question CAREFULLY, as the question will tell you exactly which ramification of the off-by-one bug to discuss.

Also, keep your answers short and to the point; there are no style points for long-winded answers (rather, quite the opposite!).

And, most of all, have fun!1

<sup>&</sup>lt;sup>1</sup>OK, I am not too serious about having fun. Mostly, just try to finish the exam in less than two hours and go home and take a nap.

1.	Scheduler Activations has an off-by-one bug that causes it to occasionally allocate an extra CPU to a running
	program (e.g., even though there are only three CPUs available, scheduler activations creates four activations
	and lets a program use those to run its code, time sharing the four processes across three processors). What are
	the implications of this bug on the performance and correctness of the system?



3. **Nooks** keeps per-driver page tables to build a wall of isolation between the drivers and the main kernel. When calling into driver D, Nooks is supposed to switch to page table D, but occasionally switches to page table D+1. Discuss what would happen when Nooks does this.

4. The random number generator a **Lottery scheduler** has an off-by-one bug that causes ticket 0 never to be picked. Given the implementation described in the Lottery scheduling paper, when (if ever) will this bug cause starvation of a process?

5. **Disco** has an off-by-one bug that causes it to occasionally think that a virtual CPU has gone idle, even though it hasn't. Discuss how Disco normally detects when a CPU has gone idle, why this is important, and what will happen when Disco fails to detect this event.

6. **LFS** uses an imap to find inodes. The in-memory version of the imap is always correct. However, an off-by-one bug in the write path causes the entry for inode I to be placed in the slot for inode I+1, for all inodes (each entry is thus shifted when written to disk by one inode slot). Assuming some files get written to disk, *when* will this problem become apparent? What will likely happen?

7. In a concurrent program built with monitors in the **Mesa** system, an off-by-one bug causes an extra signal to be generated occasionally, thus waking up a thread that we normally wouldn't expect to be woken. Discuss the ramifications of this spurious wake-up.

8. **Pilot** has an off-by-one bug that occasionally causes the file system index to point to the wrong block on disk. Oddly enough, Pilot detects this problem, rebuilds the index, and keeps chugging along until the next such problem occurs. How does Pilot do this?

9. **Anticipatory scheduling** has to decide how long to wait for a given process to issue a request, instead of scheduling the current best request the system has seen. This code, alas, has a bug where it waits 1 millisecond *more* than it should (on average). A different bug causes it to wait 1 millisecond *less* than it should (on average). Discuss the performance and correctness ramifications of these bugs. Which bug would you rather have in your system?

10. Someone purchasing **RAID** systems for the University has an off-by-one bug during purchase and buys RAID-4 instead of RAID-5. Discuss the performance impact of this human-level off-by-one bug.

11. The **Flash** web server paper has a performance evaluation that has an off-by-one bug. Specifically, the numbers reported in the paper were supposedly for a machine with 128 MB of memory. However, the actual amount of memory in the machine was 256 MB (which is off by one in binary). What would you say about the conclusions of the Flash paper if we knew about this bug?

12. **FFS** allocation occasionally has a bug where instead of placing a file in the cylinder group it is supposed to go into (e.g., cylinder group N), it places it in the adjacent one (e.g., N+1). Discuss the performance impact of this bug.

Student	ID:	
0 000000000		

13. The **IRON file system** is obviously awe-inspiring. It turns out, though, that the conclusions of the paper were off by one. Specifically, the ReiserFS "do no harm" policy was actually implemented by ext3; the ext3 "simple but reliable (but perhaps over-reactive)" policy was implemented by JFS; and the "kitchen sink" policy was implemented by ReiserFS. Describe these policies, and then tell me which you think is best and why.

14. **Exokernel** is from M.I.T. and therefore has no bugs. But, it may not be all perfect, alas. Discuss a scenario where exokernel performance will likely be *worse* than a typical general-purpose operating system.