736 Midterm Exam
(Old School Style)

The Times are a Changin'

Name: _____

This exam contains 9 old-school pages.

What difference does it make? Sometimes, a small difference makes no difference, as in "I just won a million dollars!" as compared to "I just won a million and one dollars!"; sometimes a small difference makes all the difference, as in "I just DIDN'T win a million dollars!"

In this exam, we'll examine some small differences in systems, and determine whether a small difference makes a small or large difference overall.

Either way, you won't be winning a million dollars, but you might get a good grade on this exam, which of course is priceless.

1. The designers of VMS decide that two user segments (P0 and P1) are too many, and reduce it to just one. What is the main problem that arises due to this change?

ANSWER: Recall that VMS has paging as well as segmentation. Thus, when there are unused pages between the stack and the heap, that is OK, in that no space is allocated for the address space directly. However, space is allocated in the now large, contiguous page table, for a bunch of slots that say that the pages between the heap and stack are not valid. Thus, the biggest problem that arises: space wastage due to the size of the huge (now single) page table.

2. FFS no longer spreads large files across cylinder groups, but rather places all of such a file within one cylinder group. On a modern 1TB disk, does this make much difference? (explain)

ANSWER: I accepted arguments either way, as long as the argument was reasonable. Those who said "yes, this makes a big difference" assumed that cylinder group (CG) size didn't change, while perhaps files have gotten larger, and thus a single file still could fill up a CG and cause other (related) files to be spread across the disk, thus lowering performance.

Those who said "no, it doesn't, this is fine" argued that with larger disks, you should have bigger CGs, and thus very few files would fill up a CG. Further, caching of small files would make it such that read performance didn't matter as much, which is the primary target of FFS optimizations.

3. Nooks removes all parameter/result checking for data coming in and out of drivers. Does this small change make much of a difference? (describe)

ANSWER: The answer here is yes, it does. The address space boundary between drivers and the main OS is one that must be carefully maintained in Nooks. If the system doesn't check results (for example) on the reply from a driver, the (buggy) driver may return bad data, which in turn may crash the main kernel. While some problems will still be handled in an acceptable manner (such as an immediate driver crash upon hitting a bug), some class of problems will manifest in this manner and thus weakening the boundary is a bad idea.

4. Disco is changed to turn off page migration. When is page migration useful? What will the overall cost be of such a change?

ANSWER: Page migration is useful when a thread is accessing a page that is "remote" and thus paying a higher cost of access in a NUMA machine such as Flash. Migration moves such pages nearer to their accessing thread (assuming there is only one, in this example) and thus reduces main memory access time (in Flash, by a factor of 3). However, such a change would simplify the system, as no longer would any page location and access information need to be tracked.

5. On a system with a software-managed TLB, Exokernel usually provides some kind of interface to modify the contents of the TLB. What harm would come from removing this ability? How would the resulting system work?

ANSWER: The whole point of exokernel is making hardware exposed to clients (to applications, libOSes, etc.). By removing this interface, exokernel is essentially saying that it will now manage the TLB, and thus all the problems of doing so arise: what policy to implement? what if the policy is not what the application wants? etc.

6. The IRON file system paper presents an analysis of file-system failure handling that shows the failure handling is often implemented quite poorly. To remedy this, a small change in the lower layers of the I/O code is made to panic (crash) any time a disk failure occurs. When is this good? When is it bad? What do you think of this change?

ANSWER: This is good if passing the failure up to the file system results in something "worse" than a crash, such as corruption of the entire file system volume. This is bad if the file system would have handled it gracefully by say retrying or some other nice recovery approach. Overall, given what we saw in the IRON study, maybe this is a good thing, as it approximates the reiserfs policy of "do no harm", but perhaps it is an overreaction on reads which could at least be retried first.

7. ZFS keeps a checksum with every pointer, to detect data corruption. As a simplification, ZFS moves this checksum to reside next to the data itself, instead of with the pointer that points to the data. What is the impact of this change?

ANSWER: The key to this change is that it makes it difficult to detect certain types of problems that occur in disks. For example, think of lost/phantom writes. If you write out a checksum right next to the data, a single lost write will leave the old (but seemingly valid) contents on disk. By separating the checksum from the data, a single such lost write will be detected (and presumably recovered) by ZFS. Misdirected writes have the same problem.

8. LFS is changed to write the entire inode map to the checkpoint region, immediately after each segment is written. How is this different from what LFS usually does? What performance impact does it have?

ANSWER: This is different in that LFS usually only writes pointers to chunks of the inode map to the checkpoint region, and does so quite lazily, only every 30 seconds or so.

Writing out the entire inode map to the checkpoint region after each segment write introduces a seek and large write into the equation, and thus greatly reduces performance.

9. Journaling file systems are careful to order updates to ensure correct recovery in the case of a system crash. Describe how ext3, in ordered (metadata) journaling mode, carefully orders updates to disk. Can any of these ordering points be removed, thus changing the system to make it simpler?

ANSWER: Ordered journaling only writes metadata to the journal. The ordering of this protocol is:

1. write the data in place (and wait for completion)
2. write the tx begin and metadata to the journal (and wait)
3. write the tx end (and wait)
4. write the metadata in place (and wait)
5. update the journal super block to mark the space as free in the journal (wait)

You could certainly remove the wait after 1, because in that case the transaction is not yet committed. This would allow some parallelism if the journal is located on a separate device.

You could also remove the wait after 2 if you add a checksum over the entire transaction, as we did in the IRON paper.

10. Assume that you have an AutoRAID, and that you configure the system to use only two disks. How could you simplify AutoRAID in order to take advantage of the fact that it is running on a two-disk system?

ANSWER: RAID-5 and RAID-1 become virtually identical on a 2-disk system. Thus, AutoRAID has no point when running in this configuration, and thus you should just use mirroring.

11. As we all know, Multics solves every problem with indirection. Each code or data segment is referred to by a name, similar to a pathname in a Unix directory tree, basically something like "/a/b/foo" or whatever. To simplify this, Multics changes to a flat namespace with numbers for each segment. Discuss what this improves; discuss what it makes worse.

ANSWER: This is better in that it will improve lookup time, when segments are first made known to a process; instead of a long path traversal, a simple lookup would take place.

This is worse in that numbers are not a very good way to identify segments, especially given that humans must try to use these numbers in a meaningful way.

12. The designers of the VMware ESX memory manager decide it will be simpler to avoid all the trouble of maintaining a balloon driver, and thus remove it. What will the result of this removal be? Will the system still work?

ANSWER: The short answer is that the system would still work, but perhaps not as well, because the balloon driver was better at getting a good set of pages from a VM to give to another. Without such a mechanism, the VMM has to guess which page to reclaim, and thus has the potential of introducing the "double paging" problem.

13. THE and Nucleus are old systems. Remzi thus tries to change the 736 class by skipping all old papers like these. What is the most important lesson that would be lost by skipping these papers?

ANSWER: Almost anything that made sense was accepted here. A good answer probably said something about how it is important to understand the lessons of history to give some perspective on why we build systems the way we do. Some fun answers focused on Dijkstra and how awesome he was. More specific lessons focused on the importance of OS architecture and the beginnings of the debate between microkernels or monolothic kernels. Some pointed to Nucleus and the importance it placed on memory protection as an early understanding of the imperfect nature of software. And the list goes on...