COS318 Final Exam SOLUTIONS
Princeton University
Fall, 2011
Instructors: Profs. Margaret Martonosi & Vivek Pai

(Total Time = 180 minutes)

Question	Score
1-3	/15
4	/15
5	/15
6	/15
7	/10
8	/15
9	/15
Total	/100

- **This exam is closed-book, closed-notes. 1 double-sided 8.5x11" sheet of notes is permitted.**
- **Calculators are ok but unneeded. Laptop or palmtop computers are not allowed.**
- **Show your work clearly in the spaces provided in order to get full or partial credit.**
- **Excessively long and/or vague answers are subject to point deductions.**
- **If you are unclear on the wording/assumptions of a problem, please state your assumptions explicitly and work it through.**

Honor Code (Please write out these words and then sign): I pledge my honor that I have not violated the Honor Code during this examination.

MAX SCORE = 98.5

Mean = 76.5

Std Dev = 11.8

Print name clearly and sign:

Short Answer:

1. (3 points) What is a TLB and what does it do?

A TLB is a form of hardware cache that tries to store a small number of recently-used virtual-to-physical page translations, so that V-to-P mappings can be performed at hardware speeds, without invoking the OS on each memory reference.

2. (6 points) One program reads a large file via malloc/read and the other uses mmap. If the OS needs to evict dirty pages from these regions, where do they go?

The malloc/read program has its pages written to swap. Note that the read itself makes the pages dirty from the standpoint of the OS.

The mmap'd program would have its dirty pages written back to the file in question unless you told mmap to do otherwise. The question assumed that there were dirty pages, but if you said that reading via mmap doesn't cause pages to become dirty, so the OS can just discard the pages instead of writing them back to the file, that was also accepted.

3. (6 points) A multi-threaded database program uses a single file to store all customer balance information. Each thread performs a seek, followed by a read to obtain the old balance, and then a write to update the balance.

a) (3 points) Using pread and pwrite can avoid the race conditions inherent in the original design. Explain why

The original program can have a race if the two seeks occur before the reads. Both reads will occur at the location of the second seek. Other combinations are possible too. The pread/pwrite call specifies the location of the read/write, so there's no implied position that can be changed by other calls. Note that pread/pwrite don't lock the entire file, which is the whole point.

b) (3 points) If the system runs on a single disk, is it preferable to use pread/pwrite or just have each thread lock the file during a transaction?

With a single disk, it's actually better from a performance standpoint to lock the file so that the head doesn't move between the read and write. With pread/pwrite, you may service the next pread before seeing the pwrite, so you'll end up doing more seeks.

4. (15 points) Virtual Memory: Recall the x86 page table structure from project 5: given a 32-bit linear address, the high-order 10 bits index into the page directory, the next 10 bits index into the page table, and the low-order 12 bits index into the page.

Consider each the following changes **separately** (i.e., they are **NOT** cumulative):
a) (8 points) Given a linear address, the high-order 8 bits index into the page directory, the next 8 bits index into the page table, and the low-order 16 bits index into the page. What is the size of a page, in bytes, in the new system? What are the pros and cons of the change? Give an example usage pattern under which the change would be beneficial.

2 points:
2^16 = 65536

4 points:
Pros
- less overhead due to paging structures and swapping to/from disk
- more of virtual memory fits into the TLB at once
Cons
- more wasted space due to internal fragmentation
- less granularity for swapping
- swapping an individual page is more expensive

2 points:
Example workload: random access of a 64 KB array.

b) (7 points) Given a linear address, the low-order 10 bits index into the page directory, the next 10 bits index into the page table, and the high-order 12 bits index into the page. Will this work? If so, is it a good or bad idea? Justify your answer.

correct answers: 3 points
justification: 4 points
It will work, since there's still a valid mapping between virtual and physical addresses. It's a bad idea, though, because it eliminates the benefits of spatial locality. Neighboring addresses will translate to different pages, leading to high overhead in paging data structures and frequent swapping.

5. (15 points) A UNIX filesystem has 2-KB blocks and 4-byte disk addresses. Each i-node contains 10 direct entries, one singly-indirect entry and one doubly-indirect entry.

a) (5 points) What is the maximum file size?
10*2KB + (2048/4)*2KB + (2048/4) * (2048/4) *2KB = 20K + 1024K + 524288K = 525332KB (or 537939968B or 513.02MB etc.)
If you assume 1K equals to 1000 instead 1024, your answer is still considered to be correct.

Rubric: If the final result is correct, you will get all 5 points. Otherwise, size of singly- indirect worth 1.5 points; size of doubly-indirect worth 1.5 points; the formula of adding up everything together worth 1.5 points; final result worth 0.5 point.

b) (5 points) Suppose half of all files are exactly 1.5-KB and the other half of all files are exactly 2-KB, what fraction of disk space would be wasted? (Consider only blocks used to store data)

Both 1.5-KB and 2-KB file will use 2KB space. For each 2-KB file, 0KB is wasted; for each 1.5-KB file, 0.5KB is wasted. Therefore, the fraction wasted is (0/2)*50%+(0.5/2)*50% = 12.5%.

Rubric: If the final result is correct, you will get all 5 points. Otherwise, the correct formula worth 4 points final result worth 1 point.

c) (5 points) Based on the same condition as in b), does it help to reduce the fraction of wasted disk space if we change the block size to 1-KB? Justify your answer.

No. Nothing is changed. Both 1.5-KB and 2-KB file will still use 2KB space. For each 2-KB file, 0KB is wasted; for each 1.5-KB file, 0.5KB is wasted. Therefore, the fraction wasted is (0/2)*50%+(0.5/2)*50% = 12.5%, which is unchanged.

Rubric: No (2 points). Reasonable reason (3 points).

6. (15 points) Virtual Machines. Virtualized environments seek to provide isolation between different workloads by allowing multiple guest OS's to run distinct workloads simultaneously on one physical system. In theory, someone running a program within a perfectly-virtualized environment would find it indistinguishable from running on bare hardware. In practice, discuss some ways that a piece of running code might be written to be able to discern the difference.

The best strategies all revolve around timing the performance of functionality known to highlight differences between native and virtualized execution. The three main examples of this would be: specific instructions like (1) x86 popf, (2) I/O functionality, (3) memory management and allocation.

A reasonable but general discussion of timing strategies got 12 points. Detailed specifics on the categories above got 15. Attempts to get at the above sorts of information without mentioning/using timing usually earned 10 points, since they usually wanted to look at info unavailable to a virtualized client.

7. (10 points) Your boss decides that your first project will be to take a standard Unix filesystem, and make one change -- to place the full inode in the directory file instead of just placing the inode number.

a. (5 points) What benefits does this change provide? For each benefit, briefly explain why.

The biggest is that it reduces the number of seeks, since you don't have to do a separate seek for the inode. It also gives you more flexibility on the number of inodes, since they're only allocated as needed. This will either save space when you don't need as many inodes, or will give you greater flexibility when you need a lot more inodes. It may also save some space by using up more of the directory file for smaller directories. It may also help lessen the impact of damage to the disk, since scattered inodes reduce the chance of lots of inodes getting wiped out at the same time.

b. (5 points) What drawbacks/limitations does this change introduce. Briefly explain each one.

Hard linking becomes much harder, and you need some other mechanism for it. Without some other mechanism, hard links become impossible. Finding inodes also requires a new mechanism. Detecting corruption and recovering from it becomes more complicated since you have no idea where to look for inodes.

8. (15 points) Consider the following simplified implementation of the link() system call from an early Unix system.

```
algorithm link:
input: existing file name A, new file name B
output: none

{
1       get inode for existing file A; /* returns a locked, reference counted inode */

2       if (( too many links on file ) or (linking directories without superuser permission)) {
3         release inode A; /* removes reference count and unlocks the inode */
4         return (error);
     }
5       increment link count on inode A;
6       update disk copy of inode A;
7       unlock inode A;

8       get inode for parent directory to contain new file B; /* returns a locked, ref-counted
inode */
9       if ( ( new file name already exists ) or ( existing file, new file on different file systems ) ) {
10        undo update done above;
11        return (error);
        }
12      create new directory entry in parent directory of new file B:
13          include new file name B, inode number of existing file A

14      release parent directory inode for B; /* decrements ref-count and unlocks the inode */
15      release inode of existing file A; /* decrements ref-count */
}
```

Now answer the following questions, with no more than 3-4 sentences per part.

a. (3 points) You see mentions of "link counts" and "reference counts", both on inodes. What's the difference between them?

Link counts keep track of the number of directory entries anywhere under the filesystem root which refer to the file in question. The inode reference count is used to count the number of threads/processes¬ actively modifying the inode.

Scoring: 1.5 points each for link count and reference count definition/illustration of difference. 1 or 2 points deducted if you mention any incorrect usages of these counts.

b. (4 points) Why did early Unix system implementations require superuser permission to link directories (line 2)?

To simplify system call implementations having to deal with loops in the file system hierarchy--e.g., if a user were to link an existing directory (file A) from a node below it (file B) in the hierarchy. (Superusers are trusted to know what they're doing...) So, now system call implementations can assume that anything other than "../" does not lead it *up* the file system hierarchy.

Scoring: 2.5 points for mention of loops or circular links; 1.5 points for explaining why loops are bad.

A common answer to this question was that this helps avoid users from getting access to protected parts of the filesystem (e.g., /etc) or to other users' home directories. But this is incorrect because the file owner, group and permission information resides on the inodes, which are checked when the file is actually being accessed.

c. (8 points) Why do lines 6-7 update the disk copy and unlock inode A *before* it is clear that the link call can be successfully completed? (i.e., before the checks in lines 8-11)

The essence of this question is: why unlock inode A before including the new inode entry for B, and not after? The answer is that you would end up with the classic deadlock situation of acquiring resource locks in reverse order. These bad orderings of lock acquire()s are hard to foresee, because link()s can in principle be called from any node on the file system.

A concrete example: Suppose thread A wants to link "/e/f/g" (new file) to "/a/b/c/d" (existing file) and thread B wants to link "/a/b/c/d/ee" (new file) to "e/f" (existing file). Consider what happens if A finds and locks the inode for "/a/b/c/d" at the same time that B finds and locks the inode for "/e/f". Now they are deadlocked waiting to obtain the locks for the second half of the system call to complete.
It's important to note that the need to unlock the inode drives the need to update the disk copy, and not the other way around.

Scoring: If you mention the possibility of a deadlock, you receive 5 points. A correct, plausible example sequence of operations where deadlock happens gets you 3 more points.

If you don't mention deadlocks (with or without example) but provide valid performance or fault tolerance reasons, you get 2 points. These reasons don't address why you need to unlock the inode (though they may address why you update the disk copy).

9. (15 points) RAID. Consider that many RAID devices now ship with the following options:
RAID 0 - data striped across all disks
RAID 1 - each disk mirrored
RAID 5 - striped parity

Assume a system with 8 disks

For each level, how much usable storage does the system receive?
RAID 0 – 8 disks
RAID 1 – 4 disks
RAID 5 – 7 disks

Assume a workload consisting only of small reads, evenly distributed. Assuming that no verification is performed on reads, what is the throughput of each level assuming one disk does 100 reads/sec?
RAID 0 – 800 reqs/sec
RAID 1 – 800 reqs/sec – reads can be satisfied from both disks in a pair
RAID 5 – 800 reqs/sec – no need to read the parity, so no loss of read performance, only space

Assume a workload consisting only of small writes, evenly distributed. Again, calculate the throughput assuming one disk does 100 writes/sec
RAID 0 – 800 reqs/sec
RAID 1 – 400 reqs/sec – need to write to both disks in a pair
RAID 5 – 200 reqs/sec if you do two reads + two writes to update the parity, or 100 reqs/sec if you read all of the disks to recalculate the parity

For each level, what is the minimum number of disks that may fail before data **may** be lost?
RAID 0 – 1, but data loss is guaranteed at the first lost disk
RAID 1 – 2, if you happen to lose both disks in a pair
RAID 5 – 2, but data loss is guaranteed on the second disk

For each level, what is the minimum number of disks that must fail to **guarantee** data loss?
RAID 0 - 1
RAID 1 – 5, if you happen to get really lucky and lose one from each pair before losing the 5th
RAID 5 - 2