Final Exam for CSE 153 (Spring 2016)

6th June 2016 **Name:**

Student ID:

Instructions:

* This exam is out of a total of 20 points.

* Be brief in your answers. You will be graded for correctness, not on the length of your answers.

* Make sure to write legibly. Incomprehensible writing will be assumed to be incorrect.

* Read all questions carefully. Every word is in there for a reason.

I. For each of the 4 questions below, select the correct options by clearly making an X mark next to the options that you think are correct. Every question has at least one correct option, and may have multiple correct options. **(4 x 1 = 4 points)**

1. Which of the following are true about contiguous allocation for files

__X__ It leads to better random access performance than linked allocation

____ It minimizes external fragmentation

____ The index size increases with the size of the file

2. Which of the following are true about mechanisms and policies

____ A good design customizes (rather than separates) the mechanisms to the policies

__X__ page faults are mechanisms

____ Clock LRU is a mechanism

3. Given a system where the page size is 4Kbytes and given two variables x and y who are placed in memory at addresses 0x1230 and 0x2340 respectively, which statement(s) are true?

______ The physical address of y is larger than the physical address of x

___X____We cannot tell which physical address is larger until we translate the addresses

_______It depends on whether we have a TLB hit or not

4. Which of the following are true about RAID file systems

____ The hard drives used in a RAID system must be special drives

__X__ RAID can provide higher performance than normal drives

__X___RAID can offer higher reliability than normal drives

II. (**4 points +1 bonus**) We discussed several cross cutting principles in operating system design. **Pick any two** and briefly explain the principle **using one or two sentences** then provide an example from operating systems that illustrates them. If you answer all three correctly, you get one bonus point.

a. Lazy vs. Aggressive policies

Explanation: Lazy tries to delay work until it is absolutely necessary. Aggressive policies try to do work as soon as possible. Lazy policies could save the system some work if the delayed work is never needed or if it can combine with other work that arrives later.

Example: write back (lazy) vs. write through (aggressive) in caches. Prepaging (aggressive) vs. demand paging (lazy)

b. Make the common case fast

Explanation: Focus your optimizations and investments on the part of the system that is needed often and/or takes a lot time.

Example: memory accesses occur very frequently, so translation has to be very efficient.

c. The hardware/software tradeoff (should something be implemented in software or hardware?)

Explanation: This is similar to (b) but from the perspective of the implementation. Any functionality can be generally implemented in HW or SW. The decision depends on whether the case is common enough and critical enough in term of time available to do it to warrant hardware investment. Usually, we invest in hardware for the most common cases.

Example: The same example above applies: translation from virtual to physical memory has to be in hardware because it occurs very often. We typically focus on mechanisms rather than policies in HW. On the other hand, page replacement policies occur very infrequently (when there is a page fault) and are not critical in time since the disk access will take millions of cycles anyway -- so they are implemented in software. In the middle, things like TLB misses are in the gray area and some systems implement them in software and others in hardware.

III. (**3 points**) This problem is about the Translation Lookaside Buffer.

(a) (0.5 point) What is the primary purpose of the TLB?

Speed up the translation by caching page table entries.

(b) (1.5 points) What hit ratio on the TLB is needed to bring the translation time to10% of its value without a TLB, assuming TLB accesses are free.

Average access time $= h * 0 + (1-h) * m$

Where m is the memory access time. We want average access time to be 0.1 m, so

$0.1\ m = (1-h)\ m$

and $h = 0.9$

(c) (1 point) What happens if the size of the TLB is smaller than the size of a process working set?

We keep getting TLB misses as no matter what PTE entries we keep in the TLB, we will be accessing others that are not there since our working set is larger than the TLB size.

IV (**4 points**) Suppose that a disk drive has 5000 tracks, numbered 0 to 4999. The drive is currently serving a request at track 143, and the previous request was at track 125. The queue of pending requests, in FIFO order, is

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

(a) Starting from the current head position, what is the total distance (in tracks) that the disk arm moves to satisfy all the pending requests using the following disk-scheduling algorithms?

FCFS (0.5 point): 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

SSTF (1 point): 130, 86, 913, 948, 1022, 1509, 1750, 1774

SCAN (1 point): 913, 948, 1022, 1509, 1750, 1774, 130, 86

(b) (1.5 points) Suppose that we know that the i-nodes are stored on a few tracks near the beginning of the disk. We know that for our file system, about half the requests go to the i-nodes. What suggestions can you make to improve the performance of the disk?

One option is to change the file system layout so that i-nodes are collocated with the data blocks such as the solution with FFS.

Another option is to use disk scheduling. Something like SCAN or C-SCAN should work ok, but perhaps we can be more intentional and schedule a period for i-nodes followed by a period for data.

Other ideas may also work. This is a design question.

V (5 points) Consider a system where virtual memory address size is 38 bits and physical address size is 30 bits. The memory is paged with a total of 2^16 frames in physical memory.

(a) (1 point) What is the page size?

We know that the physical memory is 2^30 and that it is split into 2^16 frames. So, the frame size (which is also the page size) can be computed as:

2^30/2^16 = 2^14

(b) (1 point) What is the size of a flat page table assuming that the Page table entry is 8 bytes?

Number of pages in the virtual address space = 2^38/2^14 = 2^24. We need a PTE for each one, which requires 8 bytes. So, total is 2^27 bytes.

(c) (1 point) If we use a two level page table, what is the size of the outer (or first level) page table?

We page the page table. The number of pages in the page table is 2^27/2^14 or 2^13. Each of these needs an 8 byte PTE so, the total size of the outer page table is 2^16 bytes. This is 4 pages (2^16/2^14).

(d) (2 points) The system has a TLB and as was mentioned in part (c) a two level page table. Explain in detail how a memory access is performed, outlining all the possible scenarios.

Assuming that this is a valid address for an allocated page:

The VPN can be determined from the address. Check the TLB to see if the translation is there. If so, we are done with the translation and we can proceed to access the data.

If the translation is not in the TLB, we must go to the page table.

We first check the directory/first level of the page table to get the page which holds the second level of the page table with respect to this address using the most significant bits (13 bits in our system above to get the offset in the directory for the PTE).

This second level page could be in swap, causing a page fault; if so, we bring it from swap. At this point we access the PTE in the second level of the page table (using the next 11 bits in the example above to get the offset in the page).

At this point we have the PTE. If the valid bit is 0, that means the data page is on swap and we have another page fault. After swapping the page in, we can do the translation and access the data.