Final Exam for CSE 153 (Winter 2015)

17th March 2015 **Name:**

Student ID:

Instructions:

* This exam is out of a total of 20 points, with an additional 3 points for extra credit.

* 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.

For this problem, if you get all 3 right, you get 1 point. If you get 2/3 you get 0.5 point. 0 otherwise.

I. For each of the 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. How many page tables are present in a system with virtual memory?

____ One page table

__X_ One per process [Every process has its own page table. Threads in a proacess share an address space and so share the same page table]

____ One per thread

2. Which of the following is true about virtual memory pages?

__X__ They have to be power of two size

__X__ They have a fixed size [Some systems support superpages so have 2 sizes, but still usually one, sometimes 2 fixed sizes]

__X__ They have the same size as physical memory pages

3. FFS improves on the original unix file system in the following ways

____ It introduced the concept of a directory

__X__ It allocates files to a cylinder group to improve read and write performance

____ It batches writes into a log and stores them sequentially on the disk

4. Which of the following statements about hypervisors are true?

____ Hypervisors split the available physical memory statically between its guest operating systems

____ All operating systems running on a hypervisor must be identical

__X__ An OS can run without modification on a hypervisor

II. Give an advantage and a disadvantage for **any 3** of the following alternatives. (3 points + 0.5 bonus)

(a) Paging vs. segmentation:

Here are possible points you could make: Paging is simpler because its fixed size, which makes the hardware to manage it easier (bit masking, as opposed to adder and comparator for segmentation). It can have internal fragmentation but not external fragmentation. Paging is more friendly to relocation since all pages are the same size. It makes it easier to implement virtual memory. Paging can match page size to block size on disk, further improving efficiency. A segment could represent a very large memory area concisely whereas we would need many pages for that increasing overhead.

(b) Linked allocation vs. Indexed allocation:

Here are possible points you could make: Random access is inefficient in linked allocation as you need to traverse the link, reading many blocks. For small file sizes linked allocation can be more space efficient (one pointer for each block, rather than an index block) and faster to read (if the file is a single block, you can read it directly in linked, but you would need to read an index block first for indexed). Linked allocation can continue to grow whereas indexed is limited by the size of the index unless we allow linking indices.

(c) Lazy vs. aggressive policies:

Some points you could make: Lazy policies delay actions until they are needed, whereas aggressive tries to do things as early as possible. Lazy often gains because it does the minimum needed work. Aggressive may do some work eagerly that ends up being useless (e.g., pre-fetching). Aggressive when the work is needed, can reduce latency since it can do thinks like prefetching. Aggressive can increase the pressure on the bottlenecks in the system because it does extra work. For example, a prefetcher (aggressive) can increase demands on the memory bandwidth, and can replace blocks that are still needed in a cache.

(d) Xen style vs. VMWare style hypervisor:

I did not specify VMWare hosted or server model, so the question is flawed. Free credit!

IV. Answer **any 3** of the following questions in a couple of sentences. (3 points + 0.5 bonus)

a) Two processes share a page. Would there be one TLB entry or two TLB entries for this page? Explain.

TLB has entries representing one entry each from one page table. A shared page will have two separate entries in each page table and would be cached in the TLB independently.

b) Consider a disk where changing the direction of the seek incurs an additional delay of 3 msecs. Give a disk scheduling policy that this delay harms the most.

SSTF would be harmed more than the SCANs since it takes the shortest request next, regardless of direction.

c) Explain the intuition behind the page fault frequency (PFF) page replacement policy. Does it make sense to use global page replacement (i.e., pick a victim page from any process in the system when a page fault occurs) or local page replacement (pick a victim from the same process) with PFF?

It tried to approximate the working set of a process through feedback. If we are experiencing a lot of page faults, that means we need to expand the working set since some of it must be on swap causing the page faults. If we are experiencing too few, that probably means we can safely reduce the number of pages. PFF only makes sense with local replacement since we are keeping track of each process' share independently.

d) Explain how locality of reference helps the performance of Virtual Memory. What would happen if a program has bad locality?

VM caches the working set of a process. If there is no locality, this caching would not be effective and we would be constantly page faulting. To take a case where there is zero locality, every page is equally likely to be accessed meaning that we cannot have the working set in memory unless we bring in all the pages of a process into memory (losing a major advantage of VM).

V. (4 points) The size of the block on a disk drive is fixed. This makes drives similar to paged memory where the frame is of fixed size. An engineer in your company would like to build a disk organization where the block size is variable (like segments). Outline the challenges involved in building such a system. Discuss the implications of such a design on directory organization, file allocation and indexing, free space management, disk scheduling and disk caching. Be clear and precise.

Some points to make. You don't need to make all of them to get full credit. Make 3 or 4 points well and you got it.

Disk is not random access like memory. So, this idea about is bad because it requires us to change block sizes constantly on the disk, which is not possible in practice. Perhaps resizing but only in units of blocks would work making this scheme similar to contiguous allocation. However, this concern aside:

-We can possibly reduce internal fragmentation by allocating the right size of a file. We would also reduce access time because we are keeping the file all in one place on the disk.

-Although internal fragmentation is gone, we have an external fragmentation issue as is the case with any variable size partitioning scheme.

-Even though in theory a file can be stored in a single block, as a file is written to, there may not be space to grow a block and it may need to either be moved or be allowed to use multiple blocks (losing much of the advantage of this scheme).

-Directory is not affected beyond the complication of finding a new addressing scheme since block numbers are no longer indicative of location.

-Since the block sizes are variable, we cannot just use the block number to indicate location on the disk. The directories will have to be updated to give a starting location that allows finding the start of a block. So, we are back to needing either indexed or linked allocation. In this case, the pointers should not only point to the start of the block, but also to its size.

-Free space management cannot use a bit vector since the block sizes are variable. Need a more elaborate scheme.

-Disk scheduling is complicated because computing random access locations becomes very difficult if the file occupies multiple variable size blocks.

-Caching cannot be done at block level since blocks are variable size. Mapping from variable size blocks to a caching unit is difficult.

VI (1 bonus) Outline one of the common security exploits (attacks) on mobile phones discussed in the guest lecture by Prof. Qian.

I believe Zhuyin spoke at length about side-channel attacks (in particular, storage side-channel attacks). However, if you mention other relevant vulnerabilities such as buffer overflows, cross site scripting, etc.. I will work with you.

VII. (6 + 1 bonus) Consider a system with a page size of 2^{14}, and a two level page table with an outer page table size of 2^{12} entries. Assume that each page table entry is 8 bytes. Assume the size of a page is 1Kbyte. The system has a physical address space of 1 Gbytes (2^{30} bytes).

(a) (1 point) How many bits are there in the physical address?

30 bits

(b) (1 point) What is the maximum total size of the page table?

We have 2^12 PTEs in the top level page table. That means we have 2^12 pages of the lower level of the page table (one pointed to by each entry in the top level page table).

Each page is 2^14 bytes, so it can hold 2^14/8 PTEs since each PTE is 8 bytes. So, each page of the lower level of the page table has 2^11 PTEs. Since we have 2^12 lower level pages (we know this from the size of the top level page table), the lower level page table has 2^12*2^11 PTEs or 2^23 PTEs.

The total size of the page table = size of top level + size of bottom level

Top level size= Number of PTEs * size of PTE = 2^12*8 = 2^15 byts.

Bottom level size = number of pages * size of page = 2^12*2^14 = 2^26 bytes

Total size = 2^26 +2^15.

(c) (1 point) What is the minimum size of the page table that must be in memory for a (small) program to run?

Recall that a multi-level page table can be smaller if we are not using the whole address space since pages of the lower level of the page table do not have to be allocated if they are empty. The minimum size it can be is one page in the top level page table, and one page in the bottom level page table. This would allow us to address 2^14/8 pages (the PTEs held in the one page of the lower level page table). If you answered that the whole top level page table must be preallocated (2^12*8 or 2 pages) and one page of the lower level page table, that is fine too.

(d) (1.5 point) Given your answer in (b), how many bits are there in the virtual address for each process?

Each one of PTEs in the bottom level holds the translation for a data page. Since the total size of the lower level page table is 2^23 PTEs, the VPN is 23 bits. Another way to see this is to see that the first 12 bits indicate which entry in the top level we use to find the right page of the page table, and the next 11 bits to find the right entry inside that page (every page of the bottom level holds 2^11 PTEs). The VPO is 14 bits (since the size of the page is 2^14). So, the virtual address space is 2^37 in size.

(e) (1.5 + 1 bonus) Consider a situation where each process on average needs 1/10 of its pages to be resident (i.e., part of its working set). How many processes can you run before thrashing occurs (Hint: don't' forget the size of the page tables)?

1/10 of 2^37 is bigger than 2^33. Since the physical memory size is 2^30, that means we thrash even with one process. Not a very good system!