Section 14: Final Review

December 3, 2015

Contents

1 Operating System Concepts

1. True/False Questions

   (a) A Zombie process is one that has come back from the dead, i.e. has be restarted after being killed once.

   False. A Zombie process is a process that has completed execution but still has an entry in the process table. It is waiting for someone (typically a parent) to execute a "wait" system call to get their return value. The extended presence of a Zombie often indicates a bug in the parent.

   (b) A *Memory Management Unit* (MMU), is a piece of hardware that translates virtual addresses into physical addresses.

   True. This is true because it is true.

   (c) In a virtual memory system, a virtual page and a physical page frame must be the same size.

   True. Because virtual page is mapped to physical page.

   (d) For disks with constant areal bit density (bits stored/unit area of disk media), the disk head reads bits at a different rate on the outer tracks than on the inner tracks.

   True. The outer tracks are longer than the inner track and contain more bits than the inner tracks. Disk spinning at constant speed $\rightarrow$ the bits are passing under the read head faster on the outside than inside.

   (e) TLB lookup (i.e. address translation) must occur prior to checking for data in the first-level cache.

   False. With physical-mapped caches (where tags are physical addresses), we can perform TLB lookup in parallel with cache lookup by sending page offset portion of the address (doesnt get translated) to the cache while translation is occuring. Also, we mentioned virtual-mapped caches (where tags are virtual addresses); in this case, the cache lookup happens before TLB lookup which only occurs on cache misses.

   (f) Small time slices always improve the average turnaround time of all the processes in a system.

   False. Small time slices will sometimes improve the average response of the system. If the slice is too small, the context switching time will start to dominate the useful computation time and everything (including response time) will suffer.

   (g) Shortest Job First (SJF) or Shortest Completion Time First (SCTF) scheduling is difficult to build on a real operating system.

True. SJF/SCTF scheduling requires knowledge of how much time a process is going to take. This requires future knowledge. You might require a user to specify the maximum amount of time that a process could run (and kill it if it exceeds this amount), then use a variant on SJF/SCTF.

(h) Memory mapped I/O devices cannot be accessed by user-level threads.

False. Memory-mapped I/O is accomplished using load/store instructions to a special region of memory; a user-level thread can access this region if the users page table has a mapping for it.

(i) A direct mapped cache can sometimes have a higher hit rate than a fully associative cache with an LRU replacement policy (on the same reference pattern).

True. A pattern that repeatedly accesses (X+1) sequential memory addresses in a cache with X memory lines will exhibit a higher hit rate in a direct mapped cache than a fully associative cache with an LRU replacement policy.

(j) Because of the overhead of context switching, programs that use threads will always take longer to execute than programs that do not use threads.

False. Threading allows I/O to be overlapped with computation which can provide substantial performance improvements.

(k) Paging leads to external fragmentation.

False. Paging may lead to internal fragmentation.

(l) Each physical page belongs to only one process.

False. pages can be shared between processes.

(m) Every interrupt changes the CPU from user mode to kernel mode.

False, interrupts can happen when the processor is in kernel mode, in which case no switch to kernel mode is made.

2. Which of the following statements are true about file systems? Select **all** the choices that apply.

(a) Reading a random location in big files is usually faster in inode-based file systems than the FAT file systems.

(b) For better sequential read/write performance, the file system should try to allocate consecutive blocks to a file.

(c) If the journaling file system does not immediately flush the committed transaction record of each completed FS operation to disk, the FS will become inconsistent upon crash and recovery.

(d) If the journaling file system does not immediately flush the committed transaction record of each completed FS operation to disk, the FS might lose the last few FS operations but will remain consistent upon crash and recovery.

(a), (b) and (d).
(b) is correct. When performing sequential read/writes, the achievable throughput is limited by disk through-put, e.g. 40MB/s. This is much better than performing random read/writes whose achievable throughput is limited by disk seek time. E.g. if each read is 100 bytes, then random read throughput is $\frac{100}{10\times10^{-3}} = 10KB/sec$ for a disk with 10ms seek time.

2 Page Tables and TLBs

Orange Inc hires you to design the virtual memory system for a new cell phone with 32-bit virtual and physical addresses, in which memory is allocated in 2 KB pages. Suppose that you decide to use a single-level page table, in which you also store three metadata bits for each page: Writable, Executable and Valid.

1. Answer the following questions, briefly explaining your solution

   (a) How long, in bits, is a virtual page number?

   21 bits: a page is 2 KB = 2^{11} bytes, so an offset is 11 bits, so a page nr. is 32-11=21 bits.

   (b) How long, in bits, is a physical page number?

   21 bits: same as a virtual page number.

   (c) How long, in bits, is an offset within a page?

   11 bits, since a page is 2 KB = 2^{11} bytes.

   (d) How much memory is needed to store the page table of each process?

   Each page table entry contains a 21-bit physical page number plus 3 metadata bits, for a total size of 24 bits = 3 bytes. There are 2^{21} pages in each virtual address space. Therefore, the total size of the page table is $2^{21} \times 3$ bytes = 6 MB.

2. Your manager asks you to consider using a multi-level page table in your design. Explain one advantage and one disadvantage of multi-level page tables over single-level page tables. (Use no more than four sentences in total.)

   Advantage: When the address space is sparsely filled, a multi-level page table consumes less memory than a single-level one, because index pages at levels beyond the first level dont need to be present for parts of the address space that arent used. This is the primary motivation for multi-level page tables.

   Disadvantage: Multiple memory accesses are needed for a lookup in a multi-level page table,

decreasing performance on TLB misses.

3. In this question, you are asked to predict the results of a sequence of memory accesses on a machine with a single-level page table and a Translation Lookaside Buffer (TLB). Suppose that the machine has 20-bit virtual and physical addresses and 256-byte pages, and that the TLB is fully associative and holds 3 entries. The initial contents of the TLB and page table are shown as follows. (For the page table, we show only a subset of the entries; assume that the ones not shown are not valid.)

Initial TLB:

Virtual Page #	Physical Page #	Writable?	Valid?
0x100	0x200	0	1
0x101	0x300	0	0
0x200	0x320	1	1

Page Table:

Virtual Page #	Physical Page #	Writable?	Valid?
0x100	0x200	0	1
0x101	0x100	0	1
0x200	0x320	1	1
0x201	0x321	1	0
0xFFF	0x100	1	1

For each memory access in the sequence on the next page, write whether the TLB is hit, whether the access succeeds, and, if so, which physical address is accessed. Also show the state of the TLB after each access, assuming that TLB entries are replaced using a Least Recently Used (LRU) policy. Assume that the TLB is not updated if an invalid page is accessed.

Page Table:

Instruction	TLB Hit?	Success?	Physical Address	New TLB State			
				Virtual Page #	Physical Page #	Writable?	Valid?
LOAD 0x20012							
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0x10001							
				Virtual Page #	Physical Page #	Writable?	Valid?
LOAD 0x10101							
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0xFFFFF							
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0x20009							
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0x32000							

Instruction	TLB Hit?	Success?	Physical Address	New TLB State			
				Virtual Page #	Physical Page #	Writable?	Valid?
LOAD 0x20012	Y	Y	0x32012	0x100	0x200	0	1
				0x101	0x300	0	0
				0x200	0x320	1	1
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0x10001	Y	N	0x20001 (ok to omit)	0x100	0x200	0	1
				0x101	0x300	0	0
				0x200	0x320	1	1
				Virtual Page #	Physical Page #	Writable?	Valid?
LOAD 0x10101	N	Y	0x10001	0x100	0x200	0	1
				0x101	**0x100**	**0**	**1**
				0x200	0x320	1	1
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0xFFFFF	N	Y	0x100FF	0x100	0x200	0	1
				0x101	0x100	0	1
				0xFFF	**0x100**	**1**	**1**
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0x20009	N	Y	0x32009	**0x200**	**0x320**	**1**	**1**
				0x101	0x100	0	1
				0xFFF	0x100	1	1
				Virtual Page #	Physical Page #	Writable?	Valid?
STORE 0x32000	N	N	N/A (page fault reported to kernel)	0x200	0x320	1	1
				0x101	0x100	0	1
				0xFFF	0x100	1	1

3 File Systems and Disks

1. Consider a file system with 2048 byte blocks and 32-bit disk and file block pointers. Each file has 12 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer.

   (a) How large of a disk can this file system support?

   2^{32} blocks x 2^{11} bytes/block = 2^{43} = 8 Terabytes.

   (b) What is the maximum file size?

There are 512 pointers per block (i.e. 512 4-byte pointers in 2048 byte block), so: $blockSize \times (numDirect + numIndirect + numDoubly - indirecty + numTriply\ indirect)$

$$\begin{aligned} 2048 \times (12 + 512 + 512^2 + 512^3) &= 2^{11} \times (2^2 \times 3 + 2^9 + 2^{9\times 2} + 2^{9\times 3}) \\ &= 2^{13} \times 3 + 2^{20} + 2^{29} + 2^{38} \\ &= 24K + 513M + 256G \end{aligned}$$

2. Briefly (2-3 sentences) state the differences between a hard link and a soft link.

Hard links point to the same inode, while soft links simply list a directory entry. Hard links use reference counting. Soft links do not and may have problems with dangling references if the referenced file is moved or deleted. Soft links can span file systems, while hard links are limited to the same file system.

3. Rather than writing updated files to disk immediately when they are closed, many UNIX systems use a delayed *write-behind policy* in which dirty disk blocks are flushed to disk once every 30 seconds. List two advantages and one disadvantage of such a scheme.

Advantage 1: The disk scheduling algorithm (i.e. SCAN) has more dirty blocks to work with at any one time and can thus do a better job of scheduling the disk arm.
Advantage 2: Temporary files may be written and deleted before data is written to disk.
Disadvantage: File data may be lost if the computer crashes before data is written to disk.

4. List the set of disk blocks that must be read into memory in order to read the file /home-/cs162/test.txt in its entirety from a UNIX BSD 4.2 file system (10 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer). Assume the file is 15,234 bytes long and that disk blocks are 1024 bytes long. Assume that the directories in question all fit into a single disk block each. Note that this is not always true in reality.

1. Read in file header for root (always at fixed spot on disk).
2. Read in first data block for root (/).
3. Read in file header for home.
4. Read in data block for home.
5. Read in file header for cs162.
6. Read in data block for cs162.
7. Read in file header for test.txt.
8. Read in data block for test.txt.
9. - 17. Read in second through 10th data blocks for test.txt.
18. Read in indirect block pointed to by 11th entry in test.txts file header.
19. - 23. Read in 11th 15th test.txt data blocks. The 15th data block is partially full.

5. Disk requests come in to the driver for cylinders 8, 24, 20, 5, 41, 8, in that order. A seek takes 6ms per cylinder. Calculate the total seek time for the above sequence of requests assuming the following disk scheduling policy (In all cases, the disk arm is initially at cylinder 20). Explain your answer.

   (a) FIFO: First-come, first-served:

Sequence is: 20[starting], 8, 24, 20, 5, 41, 8.
Time is: [(20-8)+(24-8)+(24-20)+(20-5)+(41-5)+(41-8)]*6ms=696ms

(b) SSTF: Shortest Seek Time First:

Sequence is: 20[starting], 20, 24, 8, 8, 5, 41.
Time is: [(20-20)+(24-20)+(24-8)+(8-8)+(8-5)+(41-5)]*6ms=354ms

(c) SCAN: Elevator algorithm (initially moving upward in cylinder value)

Sequence is: 20[starting], 20, 24, 41, 8, 8, 5.
Time is: [(20-20)+(24-20)+(41-24)+(41-8)+(8-8)+(8-5)]*6ms=342ms

6. Suppose that we have a disk with the following parameters:

- 1TB in size
- 7200 RPM, Data transfer rate of 40 Mbytes/s (40×10^6 bytes/sec)
- Average seek time of 6ms
- ATA Controller with 2ms controller initiation time
- A block size of 4Kbytes (4096 bytes)

What is the average time to read a random block from the disk (assuming no queueing at the controller). Show your work.

$$\begin{aligned} T_{read} &= Controller + Seek + Rotational + Xfer\ Time \\ &= 2ms + 6ms + \frac{1}{2}\left(\frac{60000\frac{ms}{min}}{7200\frac{revolutions}{min}}\right) + \frac{4096bytes}{(40 \times 10^6 \frac{bytes}{s}) \times (10^{-3}\frac{s}{ms})} \\ &= 12.269ms. \end{aligned}$$

4 Pintos Questions

1. Read the following pintos codes in project 2.

```
1  static bool setup_stack (void **esp)
2  {
3      uint8_t *kpage;
4      bool success = false;
5
6      kpage = palloc_get_page (PAL_USER | PAL_ZERO);
7      if (kpage != NULL)
8      {
9          success = install_page (((uint8_t *) PHYS_BASE) - PGSIZE, kpage, true);
10         if (success)
```

```
11              *esp = PHYS_BASE - 12;
12          else
13              palloc_free_page (kpage);
14      }
15      return success;
16  }
```

(a) Why do you need to subtract `12` from `PHYS_BASE` at line 11? What will happen if you don't do that?

> Because the new process might read argv and argc, which is located at the stack. You need to put the argc and argv on the stack and increase the stack pointer (i.e. decreasing the value of esp). If don't do that, the new process will be accessing a virtual address whose value is larger than PHYS_BASE, which is not a valid virtual address and has no physical address corresponding to it. So it will cause page fault in kernel.

(b) Assume the new process doesn't read argv and argc. And assume all syscalls' user space address are validated before being de-referenced. What could the new process do to also cause page fault?

> If the new process uses more than 4KB space on the stack, it will cause page fault in the kernel. Because we only allocated on page for the stack.

2. Read the following pintos codes in project 2.

```
1  /* Invokes syscall NUMBER, passing argument ARG0, and returns the
2     return value as an 'int'. */
3  #define syscall1(NUMBER, ARG0)                                           \
4          ({                                                               \
5            int retval;                                                    \
6            asm volatile                                                   \
7              ("pushl %[arg0]; pushl %[number]; int $0x30; addl $8, %%esp" \
8                 : "=a" (retval)                                           \
9                 : [number] "i" (NUMBER),                                  \
10                  [arg0] "g" (ARG0)                                       \
11                : "memory");                                              \
12           retval;                                                        \
13         })
```

(a) Briefly describe how the syscall interface works. How does the operating system obtain the syscall arguments passed from userspace?

> User syscall functions put arguments onto the stack and trigger an interrupt "int $0x30", which causes syscall handler to execute.
> The kernel obtains the syscall arguments from userspace by looking at the users stack using the stack pointer.

(b) As you learned in project 2, there are two ways for kernel to ensure a pointer passed by user program to syscall is valid. The first method is to verify the validity of a user-provided

pointer, then dereference it. The second method is to check only that a user pointer points below PHYS_BASE, then dereference it. An invalid user pointer will cause a "page fault" that you can handle by modifying the code for `page_fault()`. Which of this two methods are more efficient? Why?

The second method is more efficient, because it takes advantage of the processor's MMU.