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.

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

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

   (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.

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

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

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

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

(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).

(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.

(k) Paging leads to external fragmentation.

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

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

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.

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?

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

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

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

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.)

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
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				Virtual Page # / Physical Page # / Writable? / Valid?

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?

   (b) What is the maximum file size?

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

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.

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.

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:

(b) SSTF: Shortest Seek Time First:

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

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.

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?

(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?

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?

(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?