Operating Systems

Exam 2 Review: Spring 2014

Paul Krzyzanowski
Rutgers University
Spring 2014

Exam 2 – Spring 2014

Question 1

A memory management unit (MMU) is responsible for:

(a) Translating a process' memory addresses to physical addresses.
(b) Allocating kernel memory.
(c) Allocating system memory to processes.
(d) All of the above.

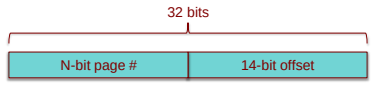
The MMU is hardware that is part of the microprocessor.

It translates from logical to physical memory locations.

Question 2

A system uses 32-bit logical addresses, a 16K byte (2^{14}) page size, and 36-bit physical addresses (64 GB memory). What is the size of the page table?

(a) 2^{18} entries (2^{32-14}).
(b) 2^{22} entries (2^{36-14}).
(c) 2^{4} entries (2^{36-32}).
(d) 2^{14} entries.



Page number = (32 – 14) = 18 bits
Page table size = 2^{18} page table entries

The physical memory address size does not matter.

Question 3

During a context switch, the operating system

(a) Modifies some page table entries for the new process to reflect its memory map.
(b) Switches a page table register to select a different page table
(c) Modifies the access permissions within the page table for the new process.
(d) Leaves the page table untouched since it is a system-wide resource.

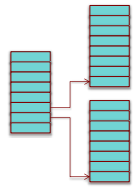
Each process has its own page table.

During a context switch, the operating system sets the **page table base register** to the page table for that process.

Question 4

Your process exhibits a 75% TLB hit ratio and uses a 2-level page table. The average memory access time is:

(a) Approximately 1.25 times slower.
(b) Approximately 1.5 times slower.
(c) Approximately 1.75 times slower.
(d) Approximately 2 times slower.



TLB hit = no page table lookup is needed
 # main memory accesses = 1

TLB miss = need to look up address via page table
We have a 2-level table
(1) look up in top-level table to find base of 2nd level table
(2) look up in 2nd-level table to find the PTE, giving us the physical frame #
(3) look up main memory ⇒ **3 main memory accesses for a miss**
Average memory access time = (0.75×1) + (0.25×3) = 1.5

Question 5

An address space identifier (ASID):

(a) Allows multiple processes to share the same page table.

(b) Enables shared memory among processes.

(c) Associates a process ID with TLB (translation lookaside buffer) contents.

(d) Performs logical (virtual) to physical memory translation for a process.

ASID is a field in the TLB
- associated with a process
- TLB lookups must match ASID & page #

Avoids the need to flush the TLB with each context switch

It ensures that the cached lookup of a virtual address is really for that process.

Question 6

Page replacement algorithms try to approximate this behavior:

(a) First-in, first-out.

(b) Not frequently used.

(c) Least recently used.

(d) Last in, first out.

(a) No – the earliest memory pages may contain key variables or code and be frequently used.
(b) No – we might have a high use count for old pages even though we're not using them anymore
(d) No – we'll get rid of pages we might still be using!

(c) Most reasonable guess but we can't implement LRU because MMUs do not give us a timestamp per memory access – we try to get close

Question 7

Large page sizes increase:

(a) Internal fragmentation.

(b) External fragmentation.

(c) The page table size.

(d) The working set size.

Internal fragmentation: unused memory within an allocation unit
External fragmentation: unused memory outside allocation units

Large page size: increases chance that we get more memory than we need

(b) No external fragmentation in page-based VM – all pages same size
(c) No. Page table becomes smaller – fewer pages to keep track of
(d) No. Working set comprises of fewer pages

Question 8

8.Thrashing occurs when:

(a) The sum of the working sets of all processes exceeds available memory.

(b) The scheduler flip-flops between two processes, leading to the starvation of others.

(c) Two or more processes compete for the same region of shared memory and wait on mutex locks.

(d) Multiple processes execute in the same address space.

The OS tries to keep each process' working set in memory.
The Working Set Size (WSS) of a process is the # of pages that make up the working set.
If Σ *WSS*$_i$ > *physical memory* then we end up moving frequently-used pages between main memory and a paging file ⇒ this leads to ***thrashing***

(b) That would be starvation, not thrashing
(c) That's just process synchronization
(d) That does not make sense ⇒ each process has its own address space

Question 9

The slab allocator allocates kernel objects by:

(a) Using a first-fit algorithm to find the first available hole in a region of memory that can hold the object.

(b) Keeping multiple lists of same-size chunks of memory, each list for different object sizes.

(c) Allocating large chunks of memory that are then divided into pages, one or more of which hold an object.

(d) Using the buddy system to allocate memory in a way that makes recovering free memory easy.

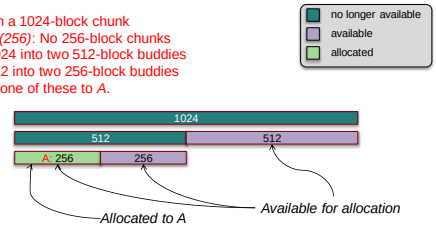
Slab allocator – designed to make it quick to allocate kernel data structures and avoid fragmentation: create pools of equal-sized objects (1 or more pages per pool)
(a) No.
(c) The vast majority of kernel structures are smaller than a page
(d) The **page allocator** does this; the slab allocator uses the page allocator to get memory when it needs it.

Question 10

You are using a Buddy System memory allocator to allocate memory. Your initial buffer is 1024 blocks. The following memory chunks are allocated and released:

A=alloc(256), B=alloc(128), C=alloc(256), D=alloc(256), free(B), free(C).

1. Start with a 1024-block chunk
2. *A = alloc(256)*: No 256-block chunks
   - Split 1024 into two 512-block buddies
   - Split 512 into two 256-block buddies
   - Return one of these to *A*.

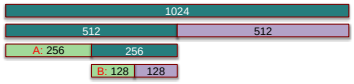


Question 10

You are using a Buddy System memory allocator to allocate memory. Your initial buffer is 1024 blocks. The following memory chunks are allocated and released:

A=alloc(256), B=alloc(128), C=alloc(256), D=alloc(256), free(B), free(C).

3. *B=alloc(128)*: No 128-block chunks
   - Split 256 into two 128-block buddies
   - Return one of these to *B*.

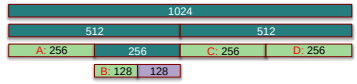


Question 10

You are using a Buddy System memory allocator to allocate memory. Your initial buffer is 1024 blocks. The following memory chunks are allocated and released:

A=alloc(256), B=alloc(128), C=alloc(256), D=alloc(256), free(B), free(C).

4. *C=alloc(256)*: No free 256-block chunks
   - Split 512 into two 256-block buddies
   - Return one of these to *C*.
5. *D=alloc(256)*: Return the 2nd free 256-block to *D*

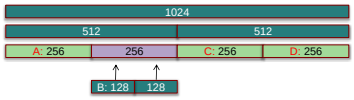


Question 10

You are using a Buddy System memory allocator to allocate memory. Your initial buffer is 1024 blocks. The following memory chunks are allocated and released:

A=alloc(256), B=alloc(128), C=alloc(256), D=alloc(256), free(B), free(C).

6. *free(B)*: Release B – its buddy is also free
   ⇒ coalesce into 1 256-block chunk.

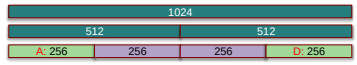


Question 10

You are using a Buddy System memory allocator to allocate memory. Your initial buffer is 1024 blocks. The following memory chunks are allocated and released:

A=alloc(256), B=alloc(128), C=alloc(256), D=alloc(256), free(B), free(C).

7. *free(C)*: Release C.



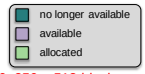
Question 10

You are using a Buddy System memory allocator to allocate memory. Your initial buffer is 1024 blocks. The following memory chunks are allocated and released:

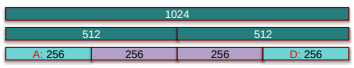
A=alloc(256), B=alloc(128), C=alloc(256), D=alloc(256), free(B), free(C).

What is the largest chunk that can be allocated?

(a) 128.
(b) 256.
(c) 384.
(d) 512.



Even though total free memory is 256+256 = 512 blocks, we do not have two 256-block buddies to coalesce.



Question 11

The advantage of a multilevel (hierarchical) page table over a single-level one is:

(a) Page number lookups are faster.
(b) The page table can consume much less space if there are large regions of unused memory.
(c) Each segment (code, data, stack) can be managed separately.
(d) Page permissions can be assigned at a finer granularity.

(a) No. Page number lookups are slower.
(c) No. That's segmentation, not paging.
(d) No. The granularity is still that of a page.

(b) Any regions of memory that are not mapped for a region covered by a top-level page number do not need to have a lower-level page table allocated. This avoids the need to allocate the lower-level page table for that region.

Question 12

A USB mouse is a:

(a) Block device.

(b) Character device.

(c) Network device.

(d) Bus device.

Block devices: devices with addressable, persistent block I/O – capable of holding a file system

Network devices: packet-based I/O driven by the network subsystem
Even a bluetooth mouse is presented as a device, not as a network interface. The bluetooth controller is a network device

Character device: accessed as a stream of bytes

Bus device: not a device category, although devic drivers usually interact with a bus driver (e.g., USB, PCIx, …)

Question 13

On POSIX systems, devices are presented as files. When you send an I/O request to such a file, it goes to the:

(a) Device driver whose name is stored in the device file's data and read when it is opened.

(b) Device driver, which is stored in the device file and loaded into the operating system when opened.

(c) File system driver (module), which forwards it to the device driver based on the name of the device file

(d) Device driver based on the major and minor numbers contained in the file's metadata.

(a) No. Driver names are not stored in device files.

(b) No. Device drivers are modules but are not stored in the device file.

(c) No. The file system driver is consulted to look up the device file's metadata but the name of the file does not matter. Also, the file system driver is out of the loop after the initial lookup.

(d) Yes.

Question 14

The *buffer cache*:

(a) Is memory that is used to copy parameters between the process (user space) and the kernel

(b) Is a pool of memory used by the kernel memory allocator.

(c) Is a pool of memory used to allocate memory to user processes.

(d) Holds frequently used blocks for block devices.

Question 15

Interrupt processing is divided into two halves to:

(a) Provide a generic device-handling layer in the top half with device-specific functions in the bottom half

(b) Partition work among user processes and the kernel.

(c) Provide separate paths for reading versus writing data from/to devices.

(d) Minimize the time spent in the interrupt handler.

Goal: tend to the interrupt : grab data from the controller & place it in a work queue.

(a) No: generic work is not done by drivers. There is usually generic code around both top-half and bottom-half components

(b) No. The kernel does both. The bottom half is usually handled by worker threads – kernel threads that are scheduled along with user processes.

(c) No.

Question 16

Which disk scheduling algorithm is most vulnerable to starvation?

(a) SCAN.

(b) Shortest Seek Time First (SSTF).

(c) LOOK.

(d) First Come, First Served (FCFS).

SCAN and LOOK algorithms guarantee to traverse the disk in its entirety so all queued blocks get a chance to be written.

FCFS guarantees that all blocks will be served in the order they are requested.

With SSTF, outlying blocks may be delayed indefinitely if there's a steady stream of blocks closer to the current head position.

Question 17

17. Which disk scheduling algorithm works best for flash memory storage?

(a) SCAN.

(b) Shortest Seek Time First (SSTF).

(c) LOOK.

(d) First Come, First Served (FCFS).

There is no need to optimize I/O based on the current disk head position. Access time for all blocks is equal – there is no moving read/write head.

Question 18

A superblock holds

(a) The contents of the root directory of the volume.

(b) A list of blocks used by a file.

(c) Metadata for a file.

(d) Key information about the structure of the file system.

(a) No. The root directory holds that. The superblock may identify where that root directory is located.

(b) No. The file's inode contains that information.

(c) No. The file's inode contains that information.

(d) Yes.

Question 19

In file systems, a *cluster* is:

(a) The minimum allocation unit size: a multiple of disk blocks.

(b) A collection of disks that holds a single file system.

(c) A variable-size set of contiguous blocks used by a file, indicated by a starting block number and length.

(d) A set of disks replicated for fault tolerance.

A cluster is a logical block size = N × physical block size
↑ cluster size:
fewer indirect blocks
more consecutive blocks per file
↓ file fragmentation
↑ internal fragmentation

Question 20

Creating a directory is different than creating a file because:

(a) The contents of a directory need to be initialized.

(b) Ordinary users are not allowed to create directories.

(c) The parent directory's link count needs to be incremented.

(d) A directory does not use up an inode.

Sorry – two correct answers
(a) File contents are empty upon creation
Directories get initialized with two files upon creation:
- Link to the parent (..)
- Link to itself (.)
(b) Ordinary users *can* create directories
(c) The link count is incremented because a link to the parent is created.
(d) A directory uses an inode … just like a file.

Question 21

Which of the following is not part of the metadata for a directory:

(a) A link to the parent directory.

(b) Modification time.

(c) Length of directory.

(d) Owner ID.

(a) A link to the parent directory is just another directory – no different from any subdirectory.
(b) Modification time: standard metadata for files and directories
(c) Length (of file or directory): standard metadata for files and directories
(d) Owner ID: standard metadata for files and directories
Other metadata includes permissions, creation time access time, group ID

Question 22

The File Allocation Table in a FAT file system can be thought of as:

(a) A table of all files with each entry in the table identifying the set of disk blocks used by that file.

(b) A bitmap that identifies which blocks are allocated to files and which are not.

(c) A set of linked lists identifying the set of disk blocks used by all files in the file system.

(d) A table, one per file, listing the disk blocks used by that file.

(a) Each entry does *not* represent a file – it represents one disk block
(b) No, the FAT is *not* a bitmap.
(c) Each entry represents a disk block. The contents of the entry contain the *next* disk block for the file.
The directory entry contains the starting block number.
Put together, the FAT is a series of linked lists representing file & free space allocation
(d) There is one FAT per file system, not one per file.

Question 23

23. An inode-based file system uses 4 Kbyte blocks and 4-byte block numbers. What is the largest file size that the file system can handle if an inode has 12 direct blocks, 1 indirect block, and 1 double indirect block?

(a) Approximately 64 MB.

(b) Approximately 1 GB.

(c) Approximately 4 GB.

(d) Approximately 16 GB.

Direct blocks: 12 block pointers × 4K bytes = 48 KB
Indirect block: 1 × 1K block pointers × 4K bytes = 4 MB
Double indirect block: 1 × 1K block pointers × 1K block pointers × 4K bytes = 4 GB

Total size = 4 GB + 4 MB + 48 KB ≈ 4 GB [0.1 % accuracy]

Question 24

When using metadata (ordered) journaling in a journaling file system:

(a) The file system is implemented as a series of transaction records.

(b) The overhead of journaling is reduced by writing file data blocks only to the journal.

(c) The overhead of journaling is reduced by writing file data blocks only to the file system.

(d) All file system updates are written into the journal in the order that they occur.

(a) No. The file system is implemented in a conventional manner.

(b) No. Data blocks have to get written to the file system with any type of journaling

(c) Yes – metadata journaling avoids double writes of data blocks

(d) This does not distinguish metadata journaling from full data journaling. Moreover, file system updates are written in the order that the buffer cache flushes them out to the disk.

Question 25

25. An advantage of using extents in a file system is that:

(a) They can allow a small amount of pointers in the inode to refer to a large amount of file data.

(b) No additional work is necessary to implement journaling.

(c) They make it easier to find the block that holds a particular byte offset in a file.

(d) They ensure that a file's data always remains contiguous on the disk.

An extent is a {starting block, # blocks} tuple.
If contiguous allocation is possible, an extent can represent a huge part of the file (or the entire file).

(b) Extents have nothing to do with journaling.

(c) No. Extents can make it more difficult.
You can no longer index to a particular block number in the inode or compute an indirect block number: you need to search.

(d) No. They can take advantage of contiguous allocation but do not enable it.

Question 26

Differing from its predecessor, ext3, the Linux ext4 file system

(a) Uses block groups.

(b) Uses extents instead of cluster pointers.

(c) Allows journaling to be enabled.

(d) No longer uses inodes.

(a) block groups were introduced in ext2 – a replacement for FFS's cylinder groups

(b) Yes. inodes can now use extents

(c) Journaling was supported in ext3, the successor to ext2

(d) Inodes are still used although indirect blocks are not used in their traditional form

The End