Operating Systems Design
Exam 2 Review: Spring 2012

Paul Krzyzanowski
pxk@cs.rutgers.edu

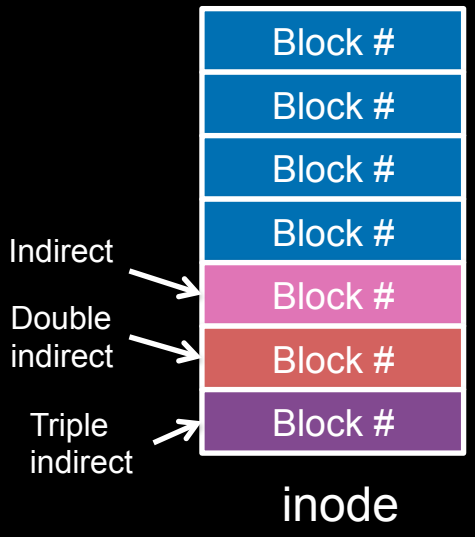
Question 1

Under what conditions will you reach a point of diminishing returns where adding more memory may improve performance only a little?

When there is enough memory in the computer to hold the working set of each running process.

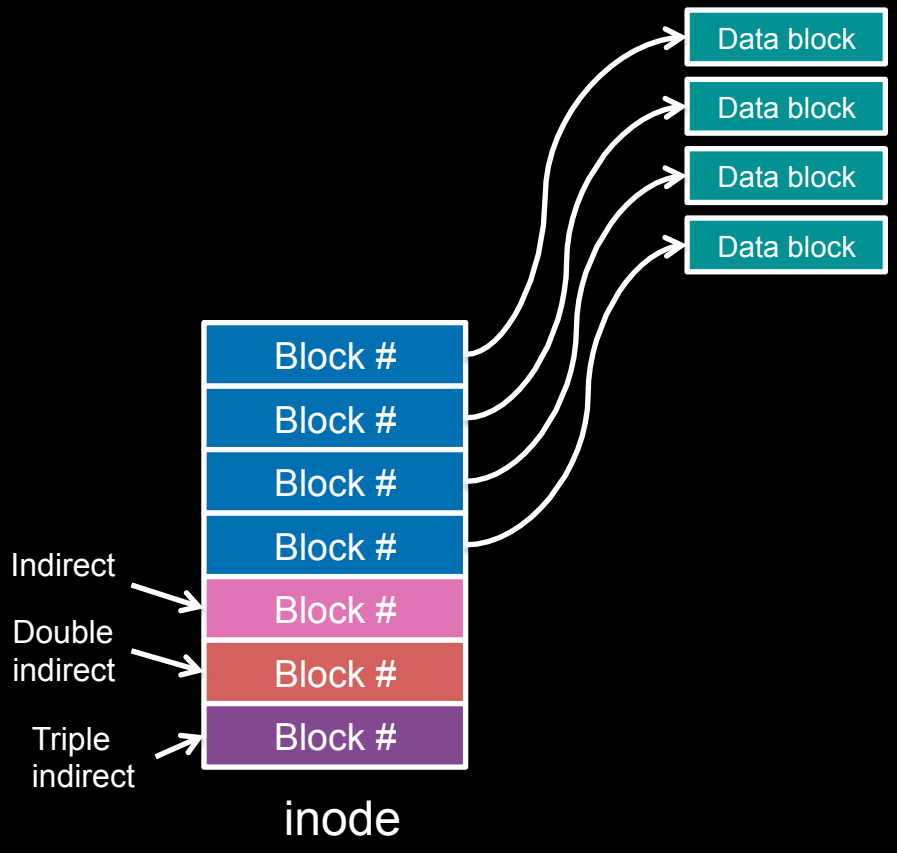
Question 2a

One of the ways that Berkeley improved performance in their design of the Fast File System was to use larger clusters. Assume that an inode has a small number of direct blocks, 1 indirect block, 1 double indirect block, and 1 triple indirect block. Further, assume that each disk cluster holds b block (cluster) pointers. Approximately how much bigger a file can the file system support if the cluster size is doubled? Show your work and approximate your answer as a single number.



Question 2a

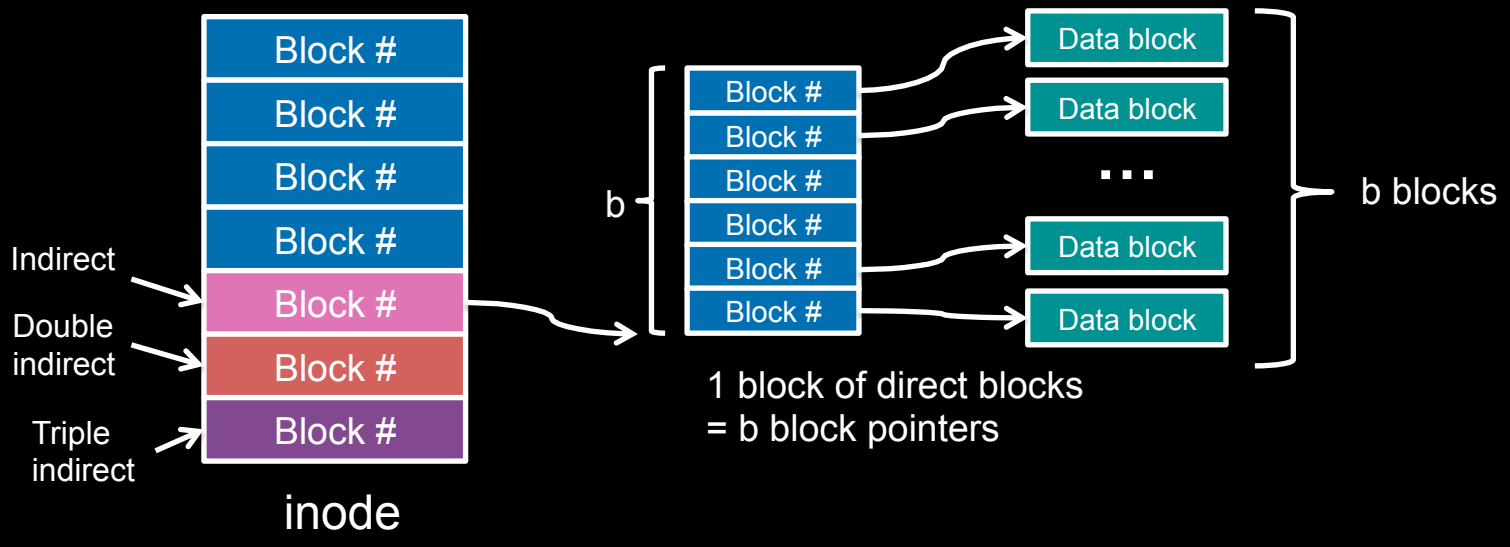
Max file size:
N direct blocks * M bytes per block = NM bytes



Question 2a

Adding an indirect block

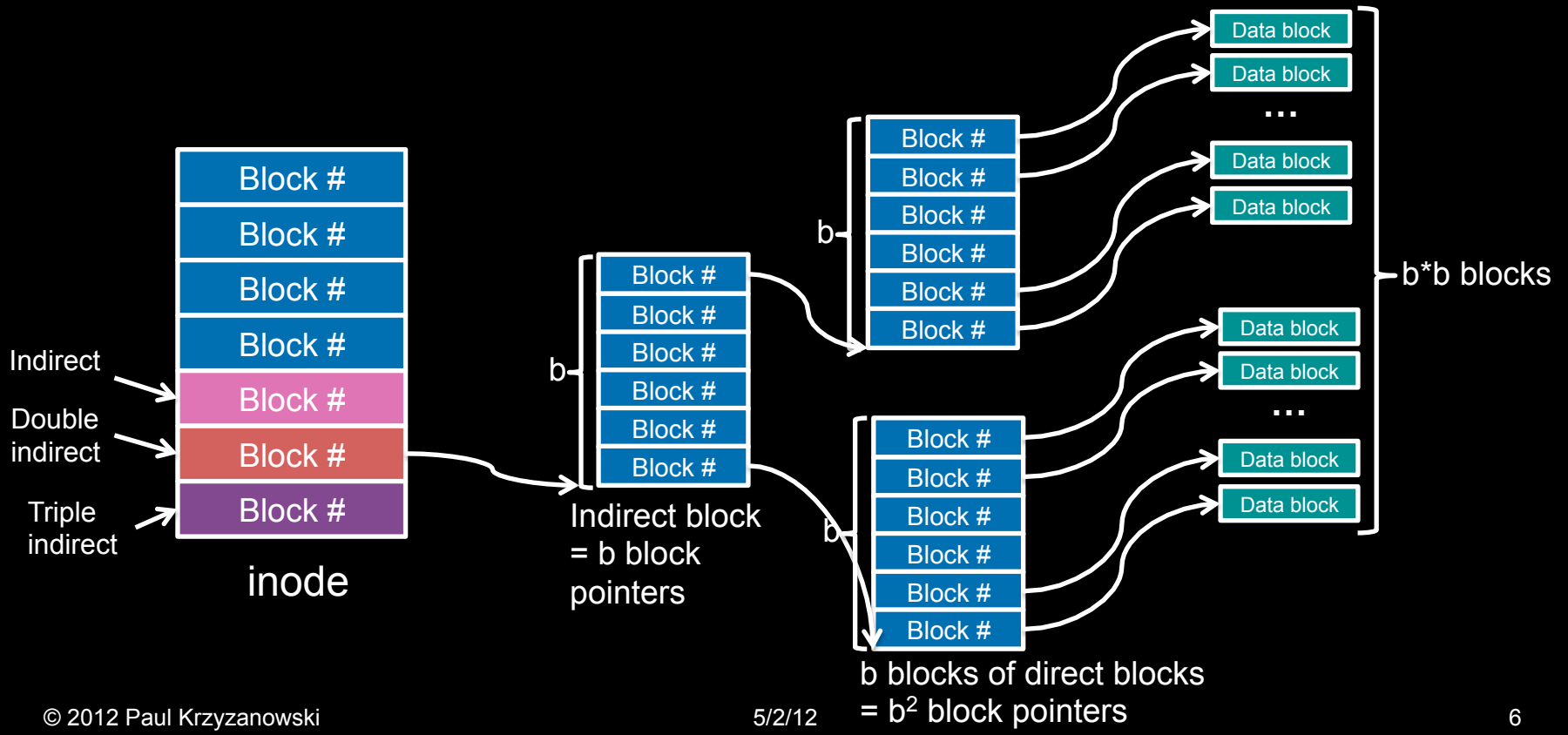
Max file size:
N direct blocks * M bytes per block = NM bytes
+ 1 indirect block * b pointers/block * M bytes = bM bytes



Question 2a

Adding a double indirect block

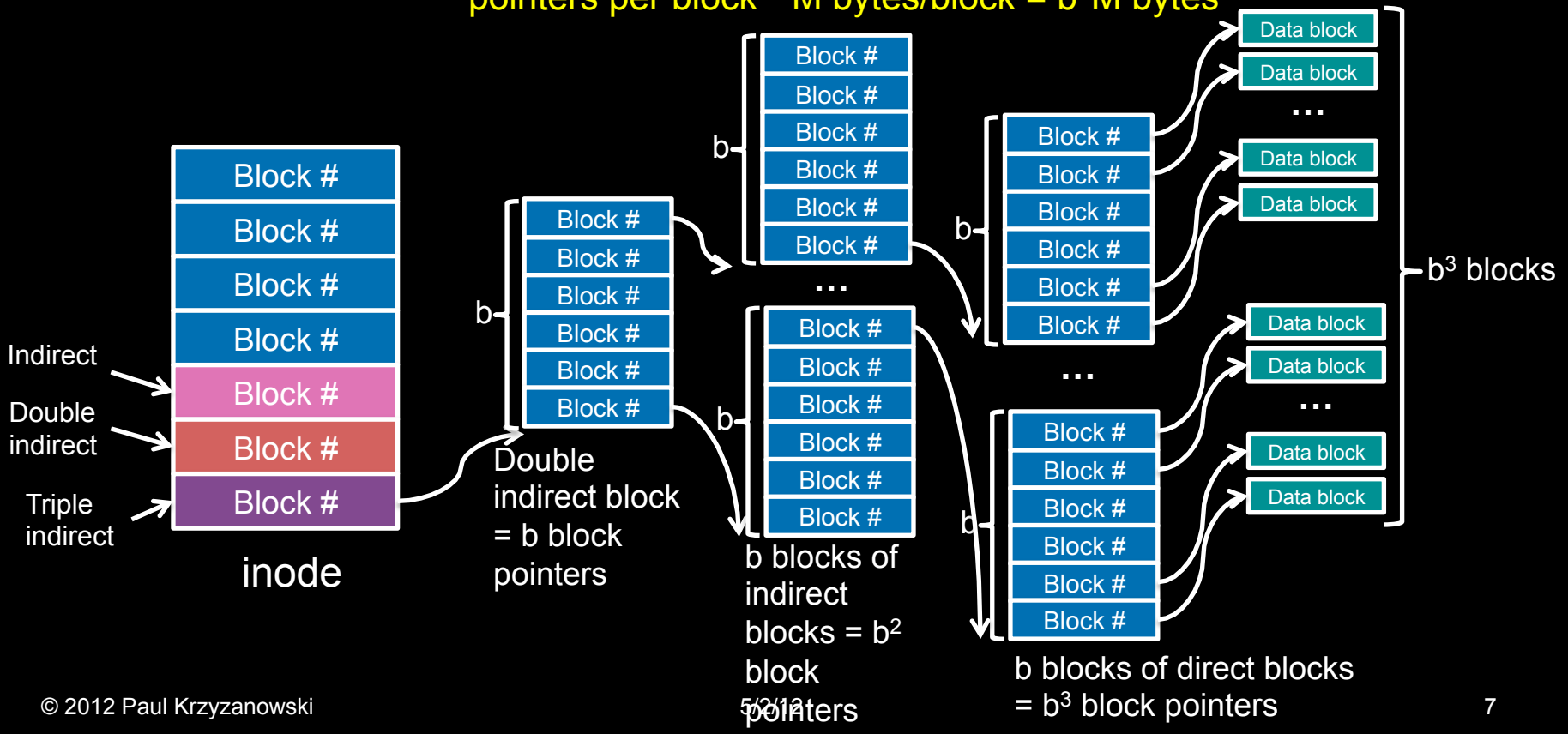
Max file size:
N direct blocks * M bytes per block = NM bytes
+ 1 indirect block * b pointers/block * M bytes/block = bM bytes
+ 1 double indirect block * b pointers/block * b pointers per block * M bytes/block = b^2M bytes



Question 2a

Adding a triple indirect block

Max file size:
N direct blocks * M bytes per block = NM bytes
+ 1 indirect block * b pointers/block * M bytes/block = bM bytes
+ 1 double indirect block * b pointers/block * b pointers per block * M bytes/block = b^2M bytes
+ 1 triple indirect block * b pointers/block * b pointers per block * b pointers per block * M bytes/block = b^3M bytes



Question 2a

Maximum file size

Max file size:
N direct blocks * M bytes per block = NM bytes
+ 1 indirect block * b pointers/block * M bytes/block = bM bytes
+ 1 double indirect block * b pointers/block * b pointers per block * M bytes/block = b^2M bytes
+ 1 triple indirect block * b pointers/block * b pointers per block * b pointers per block * M bytes/block = b^3M bytes

= $NM + bM + b^2M + b^3M$ bytes = $M(N + b + b^2 + b^3)$ bytes
≈ Mb^3 bytes

Double the cluster size:
- Each block containing block pointers now contains 2b block pointers
- Each block of data now contains 2M bytes.

Max file size = $2M[N + (2b) + (2b)^2 + (2b)^3]$ bytes = $2M(N + 2b + 4b^2 + 8b^3)$ bytes
≈ $2M * 8b^3$ bytes = $16Mb^3$ bytes

Compare the two – how much bigger is the maximum file size?

$$\frac{16Mb^3 \text{ bytes}}{Mb^3 \text{ bytes}} = 16 \text{ times bigger}$$

Question 2b

You have the option of doubling the cluster size or adding two additional triple indirect blocks to the inode. Completely ignoring the performance implications of using triple indirect blocks, which file system will support a larger file size? Show your work and explain why.

Original max = $M(N + b + b^2 + b^3)$ bytes ≈ Mb^3 bytes

Max with 2x cluster size = $2M(N + 2b + 4b^2 + 8b^3)$ bytes ≈ $16Mb^3$ bytes

Max with 2 extra triple indirect blocks =
$M(N + b + b^2 + 3b^3)$ bytes ≈ $3Mb^3$ bytes

$16\ Mb^3 > 3\ Mb^3$

Answer: Doubling the cluster size allows for a file size > 5x bigger than adding two extra triple indirect blocks.

Question 3

Dynamic linking:

a. Brings in libraries as needed to create the final executable file that can then be loaded and run.

b. Links libraries to create the final executable file without having the user identify them.

c. Loads libraries when the executable file first references those libraries.

d. Is a technique for re-linking an executable file with newer versions of libraries.

a) No. That's what a static linker does: create an executable file.

b) No. A static linker knows some default libraries but this is otherwise not done by any component.

d) No. A dynamically-linked file may pick up newer libraries but there's no re-linking done.

Question 4

Running multiple processes in a single partition monoprogramming environment requires:

a. Swapping the memory image of a process to disk at each context switch.
b. Ensuring that each process uses the same amount of memory.
c. Having each program compiled to start at the address of one of several partitions.
d. That multiple processes can coexist within a single partition.

a) Yes. Monoprogramming = one memory partition; one process at a time. This approach is not practical due to the time involved but is the only valid answer.

b) No. Monoprogramming = one process in memory at a time

c) No. There are not multiple partitions.

d) This makes no sense.

Question 5

By adding base and limit addressing to multiple partitions, we can now use:

a. Absolute code.
b. Position independent code.
c. Statically linked code.
d. Dynamically relocatable code.

a) Absolute code = absolute memory addresses are used. Base+limit addressing provides run-time translation of these addresses relative to where the program is loaded
b) No. You can use PIC without base+limit addressing; all address references are relative.
c) This has nothing to do with the question. We don't know if the linked code uses absolute or relative addresses.
d) You don't need run-time address translation; memory references are relocated at load-time.

Question 6

One advantage of multiple fixed partitions (MFP) over variable partitions is:

a. MFP never leads to unusably small holes due to external fragmentation.
b. There is no need to pre-allocate partitions.
c. Sharing memory is much easier.
d. MFP will usually result in much more efficient use of system memory.

a) Correct. The partitions always remain the same size.
b) No. Partitions *are* pre-allocated.
c) No easier than with variable partitions.
d) No. If no processes fit into available partitions, they will stay vacant – or a process with small memory needs might end up in a large partition.

Question 7

In a conventional paging system, a page table entry (PTE) will not contain:

a. A logical page number.
b. A physical page frame number.
c. A page residence bit.
d. Page permissions.

a) Right. The logical number is used as an index into the page table.
b) The physical page frame number is the frame that corresponds to the logical page (virtual address). It's the most important item in the PTE.
c) This tells us if the PTE entry maps to a valid page or not.
d) This defines the access permissions for a page (e.g., read-write, no-execute, read-only).

Question 8

A system with 32-bit addresses, 1 GB (2^{30}) main memory, and a 1 megabyte (20-bit) page size will have a page table that contains:

a. 4,096 (4K, 2^{12}) entries.

b. 4,294,967,296 (4G, 2^{30}) entries.

c. 1,048,576 (1M, 2^{20}) entries.

d. 1,024 (1K, 2^{10}) entries.

The amount of main memory does not matter.

If the page size takes 20 bits (offset) then the page number takes the first 32-20 = 12 bits

2^{12} = 4096 entries.

Question 9

The Linux slab allocator is NOT designed to:

a. Provide an interface for memory allocation for kernel structures.

b. Avoid external fragmentation by allocating same-size objects per cache.

c. Replace the buddy algorithm for allocating pages as well as objects.

d. Be able to increase the amount of memory it uses by requesting more pages.

a) The purpose of the slab allocator is to allocate kernel objects.

b) Each slab cache (one or more pages) allocates same-size objects.

c) The slab allocator uses the page allocator for getting pages. The page allocator uses the buddy system.

d) The slab allocator will request additional pages for a cache if it runs out of memory.

Question 10

Segmentation is a form of:

a. Base & limit addressing.
b. Direct-mapped paging.
c. Multi-level page tables.
d. Base & limit addressing followed by a page table lookup.

a) Yes. Each segment (code, data, stack, others) gets its own base & limit.
b) No. Segmentation ≠ paging.
c) No. Segmentation ≠ paging.
d) No. The Intel architecture is the only one that supports a hybrid mode with segmentation followed by paging but that is a hybrid mode, not segmentation.

Question 11

Assume that a main memory access takes 100 ns. If we are using a two-level page table and have a 50% TLB hit ratio, the effective memory access time is (assume no memory cache and no page faults):

a. 100 ns.

b. 200 ns.

c. 300 ns.

d. 400 ns.

Memory accesses on a TLB miss = 3
first level page table (index)
+ second level page table (partial page table)
+ memory reference
Time = 100 * 3 = 300ns
Memory accesses on a TLB hit = 1 (TLB gives us the physical address)
Time = 100 ns
Average access time = (0.5 × 300ns) + (0.5 × 100ns) = **200ns**

Question 12

A system with 32-bit addresses, 1 GB (2^{30}) main memory, and a 1 megabyte (20-bit) page size will have an inverted page table that contains:

a. 1,024 (1K, 2^{10}) entries

b. 4,096 (4K, 2^{12}) entries.

c. 1,048,576 (1M, 2^{20}) entries.

d. 4,294,967,296 (4G, 2^{30}) entries.

Inverted page table = 1 entry per page frame

Main memory = 1 G; Page size = 1 M

page frames = 1G / 1 B = 1 K (1024)

Question 13

You are using a buddy algorithm that allocates storage from 16-byte blocks up to 1024-byte blocks. What percentage of allocated memory is wasted due to internal fragmentation when satisfying requests for allocating 130-byte chunks?

a. Approximately 0%.

b. Approximately 25%.

c. Approximately 50%.

d. Approximately 100%.

The buddy algorithm allocates blocks of memory in powers of two.

If we need 130 bytes, we need to ask for 256 bytes.
Wasted space = 256 − 130 = 126 bytes

126 ÷ 256 ≈ 0.5 = 50% (actually, 0.492)

Question 14

You are using a buddy algorithm for allocating chunks of memory in the same configuration as in the previous question. You make a sequence of 256-byte allocations as follows:

a=alloc(256), b=alloc(256), c=alloc(256), d=alloc(256)

Which sequence of deallocations will result in having the largest chunk of memory available for a future allocation?

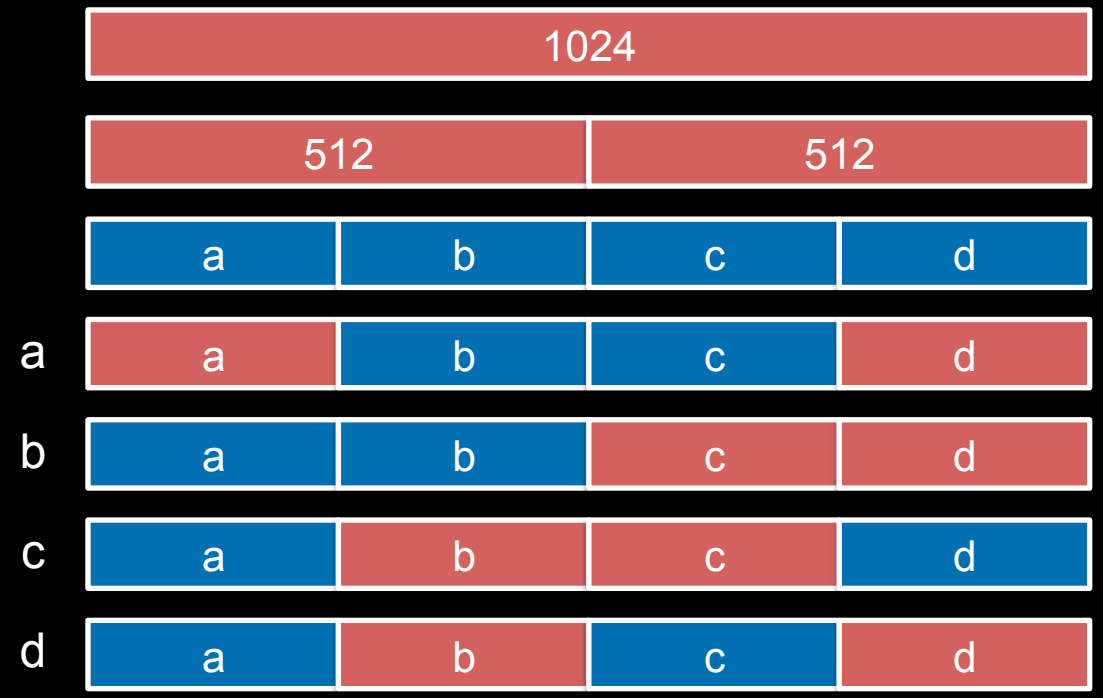
a. *free(a), free(d).*

b. *free(c), free(d).*

c. *free(b), free(c).*

d. *free(b), free(d).*

256-byte buddies:
(a, b)
(c, d)



Question 15

Memory compaction refers to:

a. Moving regions of memory to get rid of external fragmentation.
b. Compressing a region of memory to have it consume less space.
c. Getting rid of large empty (zero) sections within a region of memory.
d. Configuring a system to run in less memory than is actually available

Question 16

The page number in the 24-bit address 0x123456 with an 256-byte page size is:

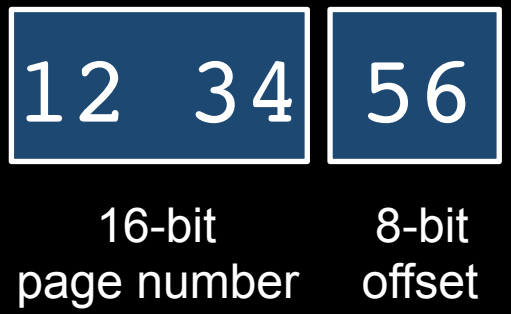
a. 0x56

b. 0x12

c. 0x1234

d. 0x3456

256 byte page size = 8-bit offset



Question 17

Which cannot be a valid page size?

a. 32 bytes.

b. 1,024 bytes.

c. 3,072 bytes.

d. 1,048,576 bytes.

A page size must be a power of two.

3,072 is not a power of two.

Question 18

The reason for using a multilevel page table is to:

a. Reduce the amount of memory used for storing page tables.
b. Make table lookups more efficient than using a single-level table.
c. Make it easier to find unused page frames in the system.
d. Provide a hierarchy to manage different sections of a program.

a) Yes!
b) No. A multi-step lookup is *less* efficient than a single lookup.
c) No. Traversing all page tables across all proceses to look for unused frames is horribly inefficient.
d) No. That's segmentation.

Question 19

Monitoring page fault frequency of a process allows us to:

a. Manage page frame allocation per process.
b. Adjust the size of the TLB for optimum performance.
c. Determine if the process is I/O bound or CPU intensive.
d. Identify the number of illegal instructions and invalid memory accesses within a program.

a) Yes. It lets us decide if a process is thrashing (too few pages) or not paging enough (too many pages in memory).
b) Nothing to do with the TLB. Also, you cannot adjust the size of the TLB; it's fixed in hardware.
c) No. You can determine if it's generating page faults but that's it.
d) Invalid address references also generate page faults but that's not what monitoring the page fault frequency accomplishes.

Question 20

A buffer cache is useful only for:

a. Block devices.
b. Character devices.
c. Network devices.
d. Block and network devices.

A buffer cache is used for block addressable storage.

Data in character and network devices is not addressable.

Question 21

The following is not an example of a character device:

a. Mouse.
b. Sound card.
c. USB-connected printer.
d. Flash memory.

a-c are all character devices.

Flash memory is a block device and can hold a file system.

Question 22

The minor number of a device identifies:

a. The version number of a device driver.
b. Whether a device is a block, character, or network device.
c. The specific driver used by a block, character, or network device.
d. The instance of a specific device among devices sharing the same driver.

Major number: identifies device driver in device table

Minor number: passed as a parameter to the device driver to identify an instance of a specific device.

Question 23

The top half of a device driver runs in:

a. Driver context.
b. User context.
c. Kernel context.
d. Interrupt context.

Top half = interrupt-triggered event.

It's unknown what context is in place when the interrupt occurs so the top half is said to run in interrupt context. Unlike a user context (process) or kernel context (kernel thread), this is *not* preemptible.

There is no such thing as a *driver context*.

Question 24

A disk drive using the Circular LOOK (C-LOOK) algorithm just wrote block 1203 and then read block 1204. The following blocks are queued for I/O:
900, 1200, 1800, 2500.

In what order will they be scheduled?

a. 1200, 900, 1800, 2500
b. 1200, 900, 2500, 1800
c. 1800, 2500, 1200, 900
d. 1800, 2500, 900, 1200

C-LOOK schedules requests in sequence based on the current position and direction of the disk head. Requests are scheduled in one direction only (disk seeks back to the earliest block)

Current block: 1204.

Blocks to be scheduled next: 1800,2500. Then seek back to 900 (lowest block) and schedule I/O for 900,1200.

Question 25

Which scheduling algorithm makes the most sense for flash memory?

a. Shortest seek time first (SSTF).
b. SCAN.
c. LOOK.
d. First come, first served (FCFS).

There is no penalty for (or concept of) seek time in flash memory.

Hence, scheduling is pointless.

FCFS is just plain FIFO ordering and does not attempt to resequence the queue of blocks.

Question 26

The VFS inode interface does NOT allow you to:

a. Create a file.
b. Read file data.
c. Write a file's attributes.
d. Delete a directory.

VFS inode functions operate on file & directory names and other metadata.

VFS file functions operate on file data.

Creating a file, writing attributes, deleting a directory are not related to file data and are handled by inode_operations.

Other inode operations include: link/unlink files, create/read symbolic link, create/delete directory, create device file, rename a file, get/set attributes.

File operations include: seek, read data, write data, read directory, memory map a file, flush file data, lock a file.

Question 27

The use of clusters in a file system does NOT:

a. Reduce internal fragmentation in a file.
b. Increase the amount of contiguous blocks in a file.
c. Reduce the number of blocks we need to keep track of per file.
d. Improve file data access performance.

A cluster is just a logical block; a fixed group of disk blocks.

a) Clustering reduces *external* fragmentation. It *increases* internal fragmentation. With larger cluster sizes, a file may get more space than it needs
b) Yes. A cluster = contiguous blocks.
c) Yes. We need to keep track of clusters, not individual blocks.
d) Yes. (1) Accessing contiguous blocks is faster on disks, (2) Lower likelihood of needing to access indirect blocks.

Question 28

A File Allocation Table:

a. Stores a list of blocks for every single file in the file system.
b. Stores file names and the blocks of data that each file in the file system uses.
c. Is a table-driven way to store file data.
d. Is a bitmap identifying unused blocks that can be used for file data.

a) Yes. A FAT implements linked allocation. Each FAT entry represents a cluster. The table contains blocks for all files.
b) File names are stored as data in directory files.
c) This makes no sense.
d) No. That's a block bitmap.

Question 29

The Berkeley Fast File System did NOT improve the Unix File System by adding:

a. Cylinder groups.

b. Bitmap allocation for keeping track of free and used blocks.

c. Extent-based allocation.

d. Prefetching of blocks.

a) FFS added cylinder groups – improvement over big contiguous regions

b) FFS added bitmaps instead of lists on bitmaps

c) FFS used cluster addressing. Extents address <starting block, length> instead of clusters.

d) FFS prefetches blocks for improved performance.

Question 30

Unlike full data journaling, ordered journaling:

a. Improves performance by not writing file data into the journal.
b. Makes sure that all that all journal entries are written in a consistent sequence.
c. Provides improved consistency by writing the data before any metadata is written.
d. Imposes no order between writing data blocks and writing metadata journal entries.

a) Right. File data is written first, then metadata is journaled.
b) Full data journaling does this too.
c) No. It provides worse consistency.
d) No. It imposes a strict order. File data first, then journal.

Question 31

With NTFS:

a. File data may be present within the file record along with file attributes.
b. The main structure guiding the location of files is the File Allocation Table.
c. Journals, free bitmaps, file records, and file data are kept in distinct sections of the disk.
d. All data blocks are compressed to maximize available disk space.

a) Yes. File data is just another attribute of a file. If it doesn't take up much space, it may be stored in the file record as any other attribute.
b) No.
c) No. They are all treated as files and each may grow from the same pool of free blocks.
d) NTFS supports file compression optionally. Only data chunks that result in space saving get compressed.

Question 32

Which structure of ext3 is least useful for a flash-based storage device?

a. inode table.

b. Block groups.

c. Journal.

d. Free block bitmap.

None of these are optimal since they don't address wear leveling.
Conventional file systems are best on managed NAND flash.

a) Useful: stores info about files.

b) Least useful: Designed to reduce seek times: keep inodes & related blocks together

c) Useful for keeping file systems consistent

d) Useful for allocating storage (more efficient than a free list)

Question 33

The Linux ext4 file system differs from ext3 in that it uses:

a. Cluster-based allocation.
b. Extent-based allocation.
c. Block groups.
d. Journaling.

a) This was present in ext2 and ext3 (and FFS)
b) Yes. This is new to ext4. No more indirect blocks: Htree used instead.
c) This was introduced in ext2 as a variant of FFS's cylinder groups.
d) This was introduced in ext3.

Question 34

A log structured file system is the same as an inode-based file system with journaling added.

True False

A log structured file systems stores the entire file system as a set of transaction logs.

Journaling is just a scratch area for disk operations until they are committed to disk.

Question 35

YAFFS uses a separate area in the file system for storing directories and metadata.

True False

The entire file system is a log. There are no dedicated areas for specific functions.

Question 36

Because of page-based virtual memory, operating systems never need to worry about allocating contiguous pages.

True False

Unfortunately, they still do, which is why Linux uses a buddy allocator. Regions that may be used for DMA, for example, need to be contiguous.

Question 37

Each page table needs to have within it a reference to each page frame in the system.

True False

No. A page table will only reference the page frames that the process uses.

Question 38

A partial page table contains page table entries for all pages in the system but not all page frames.

True False

No. A partial page table contains a subset of page table entries (used in multilevel page tables).

Question 39

YAFFS2 implements dynamic wear leveling, not static.

True False

Yes. Static wear leveling means that you periodically move data that has not changed (“static data”) to ensure that all parts of the memory get worn out approximately evenly.

Question 40

A clock page replacement algorithm tries to approximate a Least Recently Used (LRU) algorithm.

True False

Yes, it tries. Pages that were referenced (reference bit set) are skipped over. Only if we don't find unused pages do we go back and grab one that was recently referenced.

Question 41

The Shortest Seek Time First (SSTF) algorithm has a danger of starving some requests.

True False

Yes. If there's a continuous stream of disk activity, outlying blocks may get deferred indefinitely if there are always blocks that are closer to the current block ready to be scheduled.

Question 42

The order that disk blocks are scheduled in flash memory is not critical.

True False

Correct. There is no concept of seeking.

Question 43

The loop device converts a regular file into a block device.

True False

The End