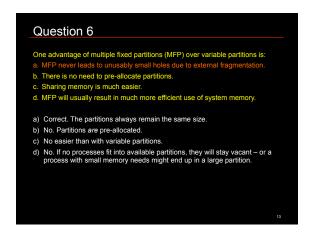
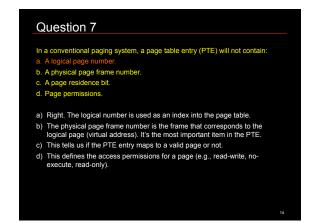
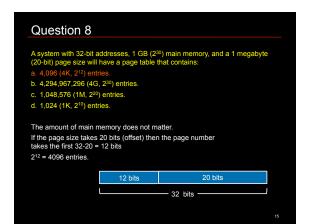


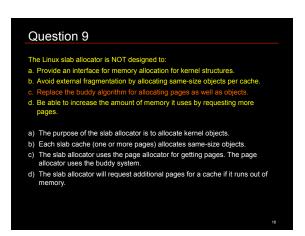
Aunning multiple processes in a single partition monoprogramming environment requires: a. Swapping the memory image of a process to disk at each context switch. b. Ensuring that each process uses the same amount of memory. c. Having each program compiled to start at the address of one of several partitions. d. That multiple processes can coexist within a single partition. 4) 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.

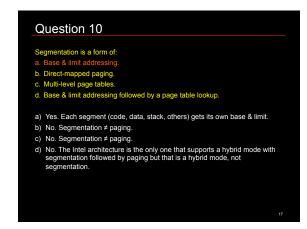
By adding base and limit addressing to multiple partitions, we can now use: a. Absolute code: b. Position independent code. c. Statically linked code. d. Dynamically relocatable code. a) Absolute code = absolute memory addresses are used. Base+limit addressing provides run-time translation of these addresses 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.

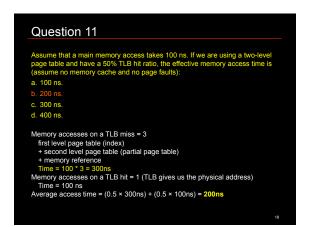


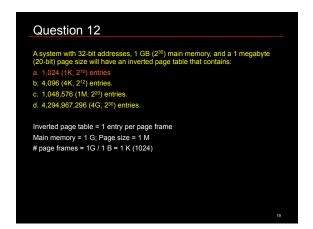


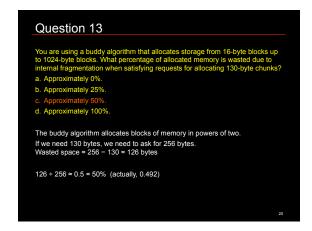


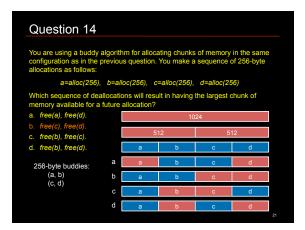


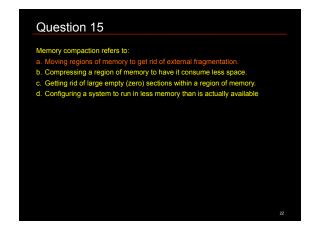


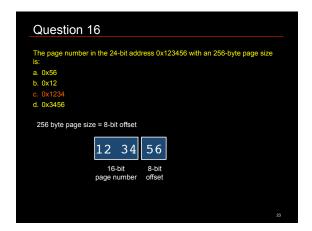


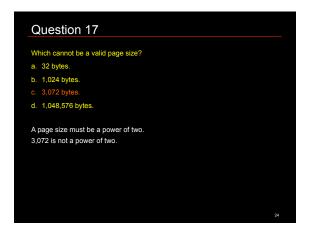


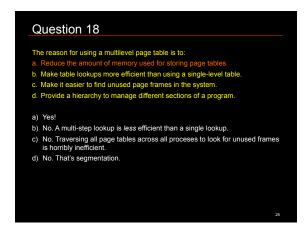


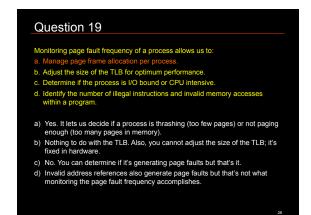












A buffer cache is useful only for:

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.

Cuestion 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

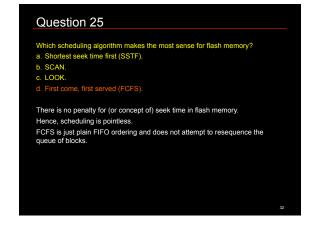
Minor number: passed as a parameter to the device driver to identify an instance of a specific device.

Question 23

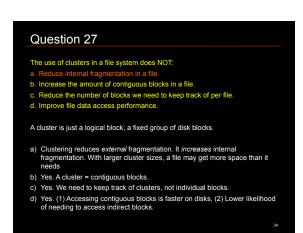
The top half of a device driver runs in:
a. Driver context.
b. User context.
c. Kernel context.
d. Interrupt context.

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.

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, 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 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.



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.

