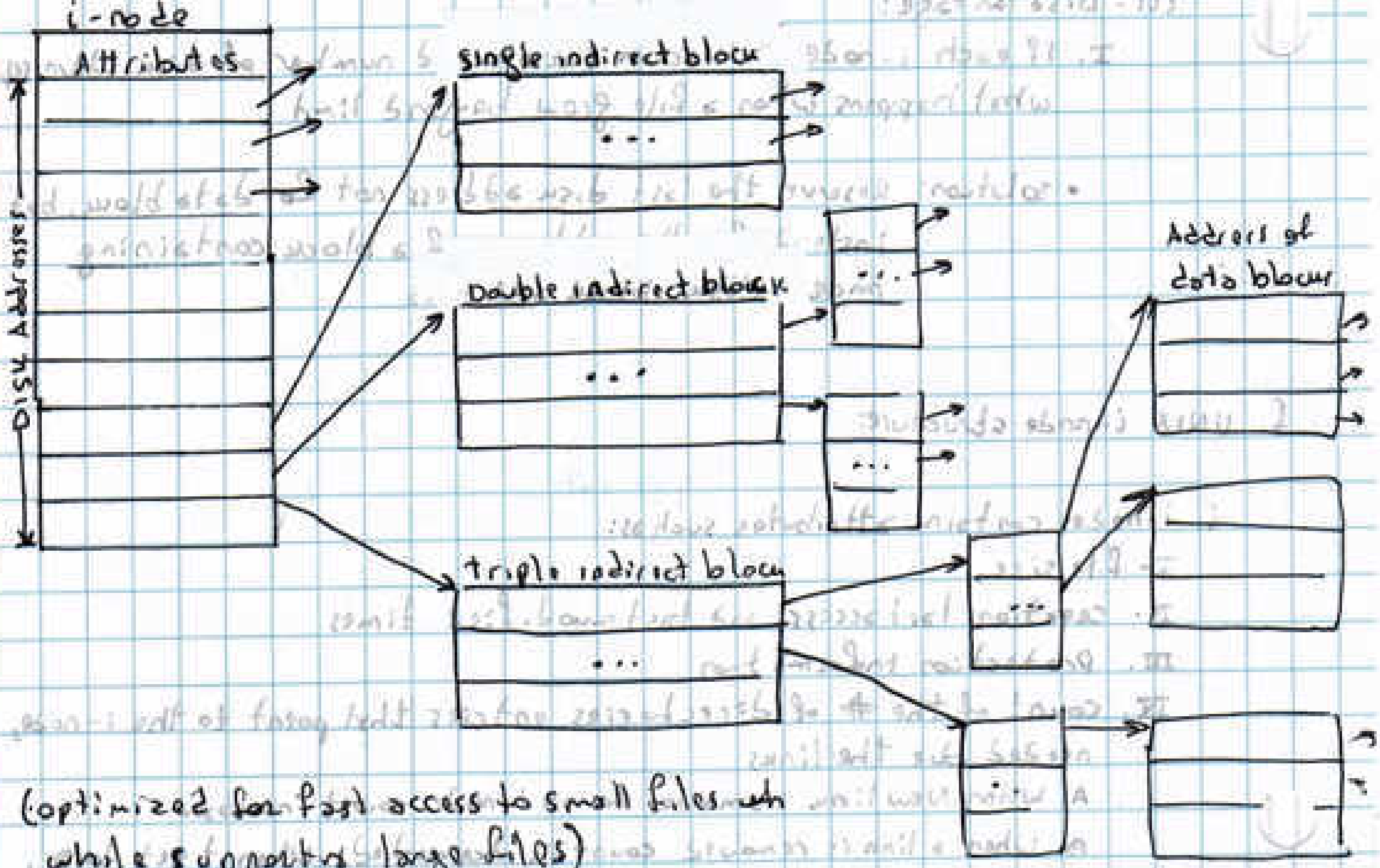
154 · iii.

[redacted]



(optimized for fast access to small files ~~wh~~ while supportin large files)

B. How path look up occurs access inodes and datablocks?

i. When a file opened, the file system must take the file name supplied and locate its disk block ie

I. path name /usr/ast/mbox is looked up.

A. File system locates the root directory

B. Reads the root directory and looks first component of the path, ~~usr~~ usr, in the root directory to find i-node number of the file /usr

c. since each inode has a fixed location on the disk, it straightforward

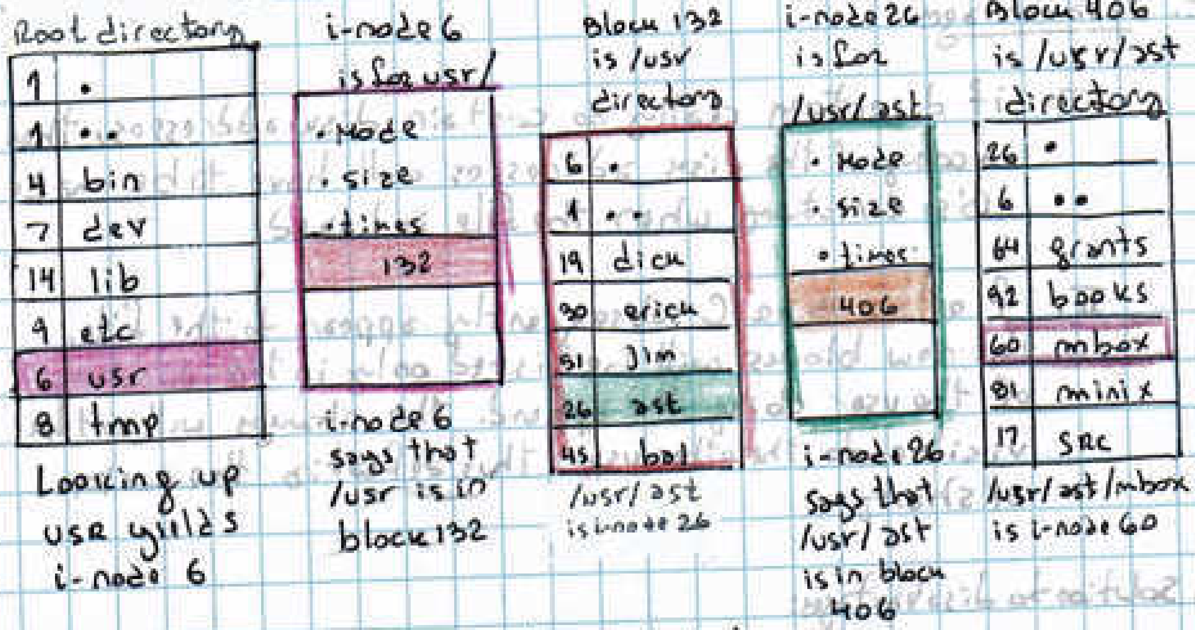
d. From this i-node, the system locates the directory for /usr and looks up for the next component, ast, in it

e. keep repeating until reaching mbox

f. i-node for this file file is read into memory and kept there until the file closed.

[redacted] final review

ii.



Root directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Looking up usr yields i-node 6

i-node 6 is for usr/

• Mode • size • times
132

i-node 6 says that /usr is in block 132

Block 132 is /usr directory

6	.
1	..
19	dick
30	erick
51	Jim
26	ast
45	bal

/usr/ast is i-node 26

i-node 26 is for /usr/ast

• Mode • size • times
406

i-node 26 says that /usr/ast is in block 406

Block 406 is /usr/ast directory

26	.
6	..
64	grants
92	books
60	mbox
81	minix
17	src

/usr/ast/mbox is i-node 60

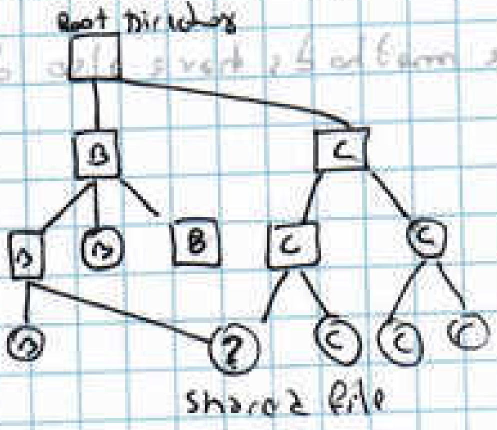
(steps ~~into~~ in looking up /usr/ast/mbox

note: Relative path names are looked up the same way as absolute ones, only starting from the working directory instead of starting from the root directory

9. <u>How shared files work?</u>

a. When a file is shared so it can appear simultaneously in different directories, the file system became a Directed Acyclic Graph (DAG), rather than a tree.

for example: lets assume one file belongs to user C is shared with user B by making a connection, link:



[redacted] final review

iii- Disavantage:

I. If each i-node has room for a fixed number of disk addresse, what happens when a file grow beyond limt

- solution: Reserve the last disk address not for data block, but instead for the addresr of a block containing more disk block addresses

f. UNIX i-node structure:

i. i-nodes contain attributes such as:

I- file size

II. creation, last access, and last modified times

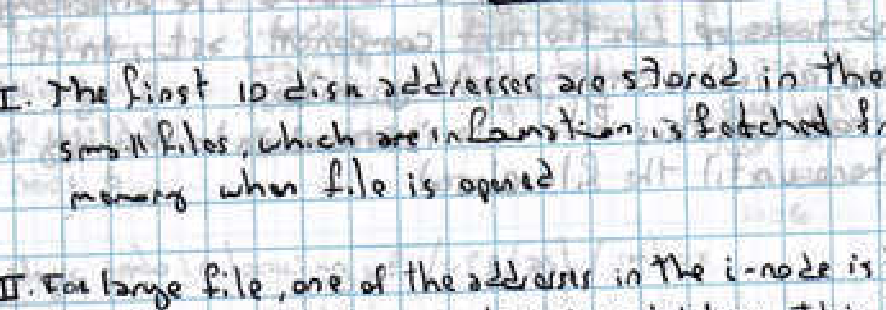
III. protection information

IV. count of the # of directories entries that point to the i-node, needed due the links

A. When New link is made to an i-node, count increased

B. When a link is removed count is decremented. When it gets to 0, the i-node is reclaimed and the disk blocks put back in the free list

ii. Keeping tracks of disk blocks is done using generalization



File Attributes

Adress of disk 0

...

Address of disk 10

Address of block of pointers

Disk block containg additional addresses

I. The first 10 disk addresses are stored in the i-node, for small files, which are information is fetched from disk to main memory when file is opened

II. For large file, one of the addresses in the i-node is the address of a disk block called a single indirect block. This block contain additional additional disk address

A- If not enough double and triple indirect block can be used as needed.

g. How shared files work?

158 b. Advantage: shared file to appear simultaneously in different directories (even if these directories belong to different users) [redacted]

c. <u>disavantages</u>:

I. if directory really do contain disk addresses, then a copy of the disk addresses will have to be made in B's directory when the file is linked

II. if either B or C subsequently appears to the file, the new blocks will be listed only in the directory of the user doing the append. the changes will not be visible to the other user, thus defeating the purpose of sharing

d. Solution to disavantages:

I. Disk blocks are not listed in directories, but in a little data structure associated with the file itself. The directories would then point to the little data structure (i-node).

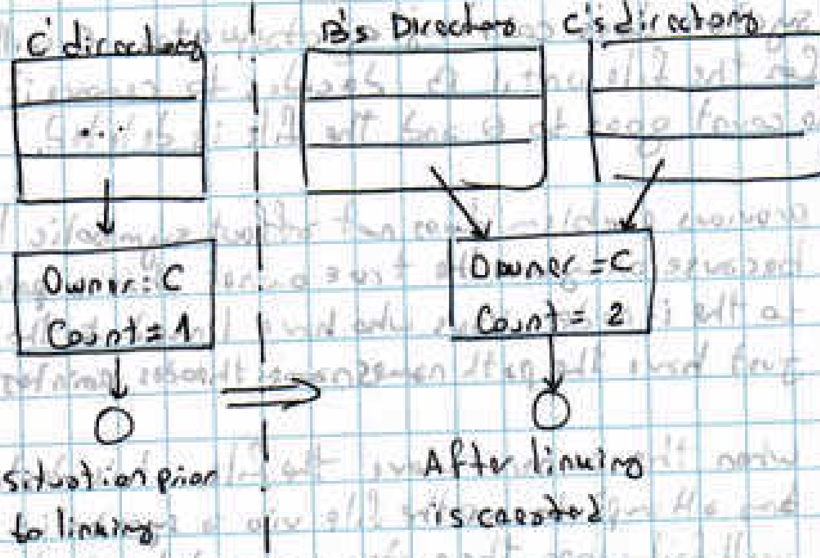
II. B links to one of C's files by having the system create a new file, of type LINK, and entering that file in B's directory The new file contains just the path name of the file to which it is linked. When B reads from the linked file, the OS sees that the file being read from is of type LINK, looks up the name of the file, and reads that file (this approach is called symbolic linking, to constrast it with traditional hard linking).

III. these methods have also drawbacks

[redacted] final review

e. disovantages of the solutions

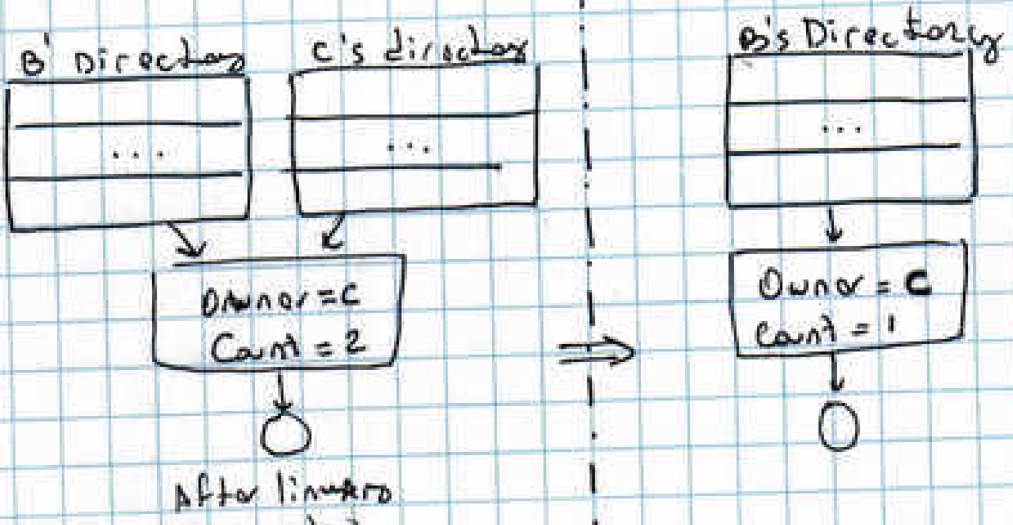
I. for the first solution: the moment that B lines to the shared file, the i-node records the file's owner as C. Creating a link doesn't change the owner ship, but does increase the link count in the inode, so the system knows how many directory entries currently point to the file



situation prior to linking

After linking is created

A If C subsequently tries to remove the file, the system is faced with a problem
 • If it removes the file and clean the i-node, B will have a directory entry point to an invalid i-node
B • If the i-node is later reassigned to another file, B's link will point to the wrong file



After linking is created

100 C. System can see from the count in the i-node that the [redacted] is still in use, but there is no easy way for it to find all the directory entries for the file, in order to erase them.

D. Pointers to the directories cannot be stored in the inode because there can be an unlimited number of directories

E. Only thing to do: Remove C's directory entry, but leave the inode intact, with count set to 1

*Now B is only user having a directory entry for a file owned by C.

- If the system does accounting or quotas, C will be continue billed for the file until B decides to remove it, at which time the count goes to 0 and the file is deleted.

f. Symbolic lines: previous problem does not affect symbolic links because only ~~one~~ the true owner has a pointer to the i-node. Users who have linked to the file just have the path names; not i-nodes pointers

When the owner remove the file, it is destroyed. Any attempt to use the file via a symbolic link will fail when the system is unable to locate the file. Removing a sybolic link doesnot affect the file.

I. Problem: extra overhead required

[redacted] final review

10. <u>DISK Free space (block) managment</u>

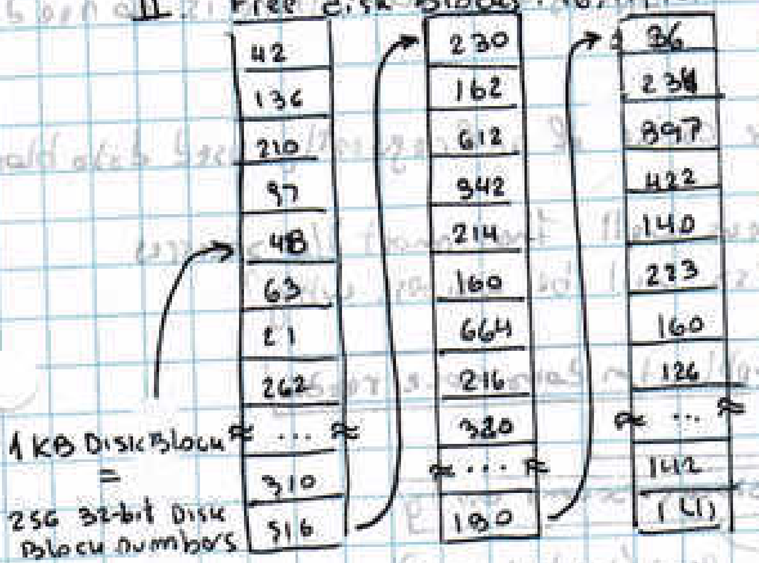
a. Using a linked list of disk blocks, with each block holding as many free disk block numbers as will fit.

I. With 1KB block, 32-bit disk block number, each block on the free list holds the number of 255 free Blocks

ie: 500 GB HD = $488 \cdot 10^6$ disk blocks

to store all there address at 255 per block = $1.9 \cdot 10^6$ blocks

II. Free disk blocks: 16, 17, 18



b. Using a bitmap, a disk with n blocks requires a bitmap with n bits. Free blocks are represented by 1s in the map, allocated blocks by 0s.

ie: 500 GB = 488000000 bits for map

= 60,000 1KB Blocks to store

I. Therefore bitmaps require less space than linked list

II.

1	0	0	1	1	0	1	1	0	1	1	0	1	1	0	0
0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
1	0	1	0	1	0	1	0	1	0	1	1	0	1	1	1
						...									
1	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1

162

[redacted]

11. File System Cache

a. block cache or buffer cache is a technique used to reduce disk accesses.

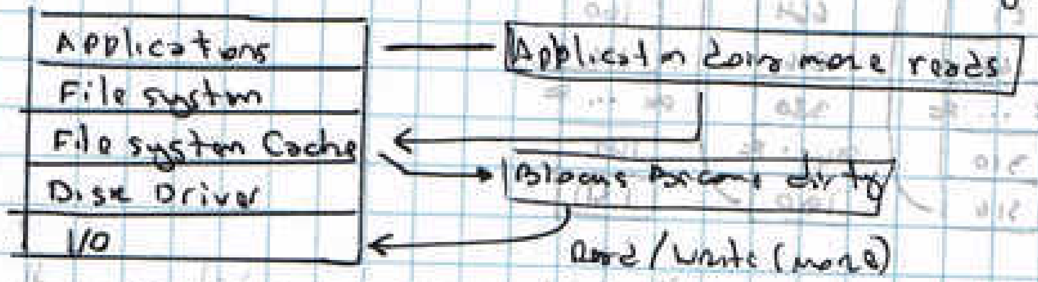
b. A cache is a collection of blocks that logically belong on the disk but are being kept in memory for performance reasons

c. Small area in main memory that store frequently accessed data blocks in the file system

d. Before accessing the disk, look in the filesystem cache
   I. if you find it in the file system cache, there is no need to go to the disk

e. periodically, purge the cache of infrequently used data blocks

? claim: If the cache works well, then most I/O accesses to the pysical disk will be writes. why?



[redacted]

12. Data Structure for File-System Cache

a. since many blocus in the cache, there is a need of some method to determine quickly if a given blocu is present.

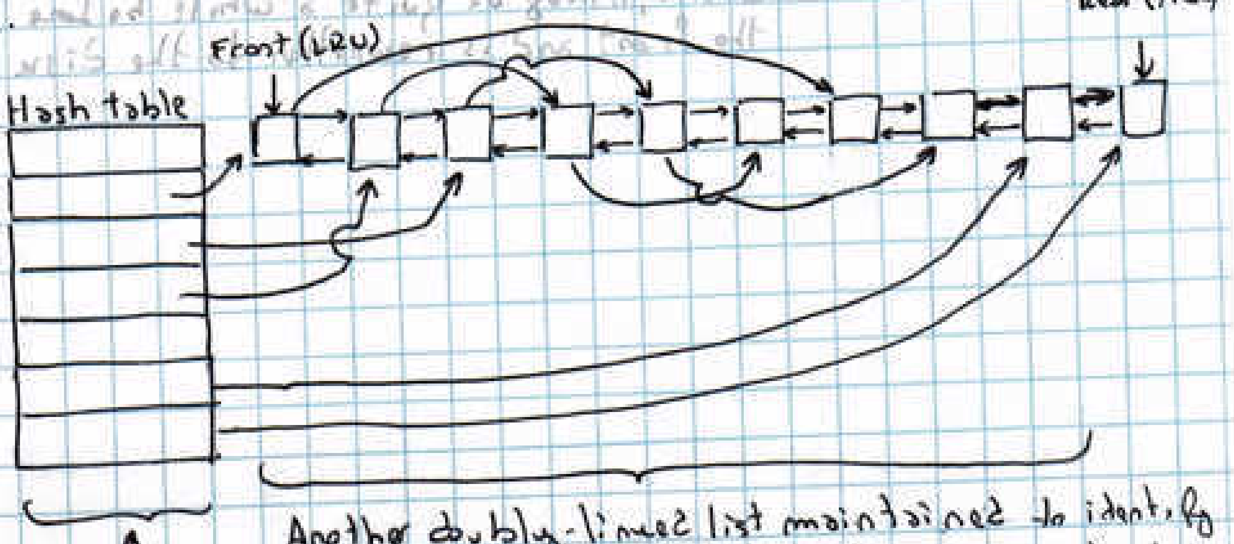
b. A method usually usedis to hash the device and disk address and look up the result in a hash table.

i. All the blocks with the same hash value are chained together on a linked list so that the collision chain can be followed

ii. when a block has to be loaded into a full cache, some blocu has to be removed (and rewritten to the disk if it has been modified since being brought in)

c. Similar to Page replacement algorithm. Normally the file system cache and profile system go together in linux systems since they have the same size as pages system even do they don't have to.

d.



Another doubly-linked list maintained to identify least recently used (LRU) pages that are periodically purged to disk

Pages in cache are normally looked up via a hash table for fast access. Best case O(1); worse case O(n)

i. Similar to paging and page replacement algorithm such as F.[redacted] second chance and LRU are appliable; however the difference is that cache references are relatively infrequent, so is feasible to keep all the blocks in exact LRU order with linked list

ii. In addition to collision chains starting at the hash table, there is a bidirectional list running through all the blocks in the order of usage, with the least recently used block at the end of the list

iii. When a block is referenced, it can be moved from its position on the bidirectional list and put at the end. In this way exact LRU order can be maintained.

iii Catch: situation in which exact LRU possible, it turn out LRU is undesired because crashes and filesystem consistency.

I. If a critical block such as an i-node block, is read into the cache and modified, but not rewritten to the disk, a crash will leave the filesystem in an inconsistent state.

II. If the i-node block is put at the end of the LRU chain, it may be quite a while before it reaches the front and is rewritten to the disk

[redacted] Final Review

13. Performance impact of i-node placement

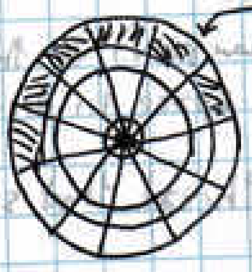
a. Apart of caching and read ahead, reducing the amount of disk arm motion by putting blocks accessed in sequence close to eachother (preferably in the same cylinder).

I. when output file is written, fs try to allocate the blocks one at a time, on demand.

A. If free blocks are recorded in bitmap & in main memory, easy to choose free block as close to previous block

B. In free list, some block clustering can be done by keeping track of the disk storage not in blocks, but in group of consecutive blocks

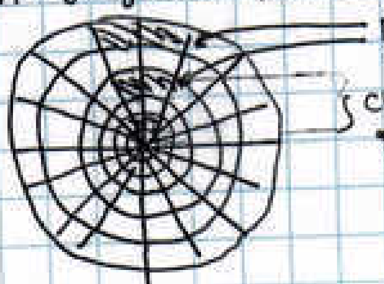
b. Another performance bottleneck in systems using inodes is that reading even a short file requires two disk accesses: one for the inode, and one for the block. e.g. of the usual i-node placement:



i. here all inodes are near the beginning of the disk, so the average distance between an i-node and its block will be half the number of cylinders, requiring long seeks.

c. Another performance is to put the inodes in the middle of the disk, rather that at the start, thus reducing average seek between the i-node and the first block by a factor of two.

d. Another idea is to divide the disk into cylinder groups, each with its own i-nodes, blocks, and free list (McKusick 1984). When creating a new file, any i-node can be chosen, but an attempt is made to find a block in the same cylinder group as the i-node. if none is available, then block in a nearby cylinder group is used.


Disk is divided into cylinder group, each with its own i-nodes
cylinder group

14. Log-Structured File Systems (LFS)

[redacted]

a. with CPUs faster & memory larger

   i. disk caches can also be larger

   ii. Increasing number of read request can be come from cache with no disk access needed

   iii. Most disk accesses will be writes

   iv. read ahead no longer needed to gain perform

   v. However, small writes are inneficient.

   vi. delayed writes expose fs to inconsistency problems therefore i-nodes writes are done immediatelly

b. LFS strategy structure entire disk as a log

   i. All writes initially buffered in memory are collected into a single segment and written to the disk as a single continous segment at the end of the log.

   ii. When file opened, locate i-node, then find blocks

   iii. If a file is overwritten, its inode will now point to the new blocks, but old ones will still be occupying space in previously written segments

      I. solution: LFS has a cleaner thread that scan the log to compact it

[redacted] final review

Input/Output Subsystem

1. I/o devices categories

a. block devices: store information in fixed-size blocks, each one with its own address.

i. All transfers are in units of one or more entire (consecutive) blocks

ii. It is possible to read or write each block independently of all the other ones

iii. ie: Hard Disk, CD Roms

Note: sometimes boundary between devices that are block addressable and those that are not is not well defined.

b. Character device: A character device delivers or accepts a stream of characters, without regards to any block structure

i. It is not addressable and does not have seek operation

ii. ie: Printer, mice,

c. Some devices are not block or characters such as Clocks.

i. Clocks are not accessable

ii. Do not generate or accept character streams

iii. They do cause interrupts at well-defined intervals

2. Device Controllers

a. I/o devices have components: mechanical and electronic components

b. The electronic component is the device controller which may be able to handle multiple devices. Electronic components are called device controller or adapter.

168 c. Controller tasks

[redacted]

i. Intermediary between I/O devices and CPU (commands from the CPU to the I/O device back and forwards)

ii. Convert serial bit stream to block of bytes and perform error correction as neccerary

iii. Block of bytes is first assembled, bit by bit, in a buffer inside the controller. After its checksum has been verified and the block has been declared to be free of errors, it can then be copied to main memory and move available

3. Memory - Mapped I/O

a. Each controller has few registers that are used for communication with the CPU.

i. By writing into these registers, the OS command the device to deliver data, switch itself on/off, or other actions

ii. By reading these registers, the OS can know what is the state of the device, whether is prepared to accept a new command, and so on.

b. In addition to control registers, many devices have a data buffer that the OS can write & read

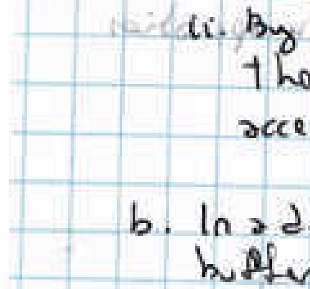
c. How the CPU communicates with the control registers and the device data buffer?

i. Each control register is assigned an I/O port number

I. set of all the I/O ports form the I/O port space

II. the I/O port space is protected so user programs cannot access, only the OS

III. Using an special I/O instruction the CPU read in control register port and store the result in the CPU register. Also the CPU can write the contex of the CPU registers to the control register



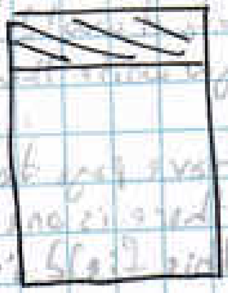
[redacted] final review

ii. another way is to map all the control registers into the memory space

I. Each control register is assigned a unique memory address to which no memory is assigned. This system is called memory-mapped I/O.

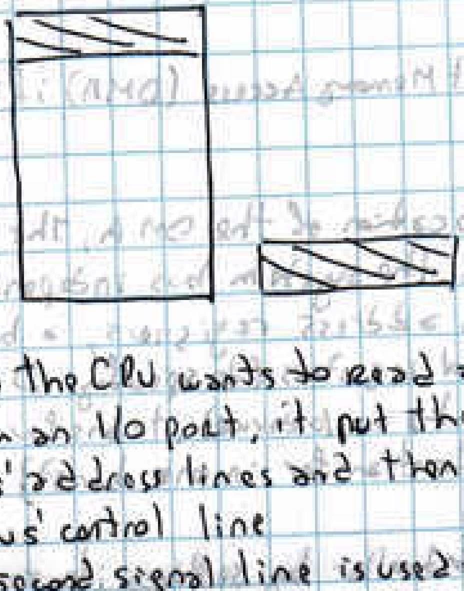
II. Assigned addresses are at the top of the address space.

one address space



iii. Hybrid scheme, with memory-mapped I/O data buffers ad separate I/O ports for the control registers

two addresses



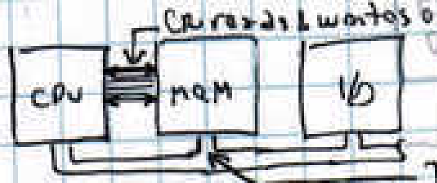
I. when the CPU wants to read a word, either from memory or from an I/O port, it put the address it needs on the bus' address lines and then asserts a READ signal on a bus' control line

II. A second signal line is used to tell whether I/O space or memory space needed.

A. If it is memory space, the memory responds to the request

B. If it is I/O space, the I/O device respond to the request

CPU reads & writes of memory go over this high-bandwidth bus



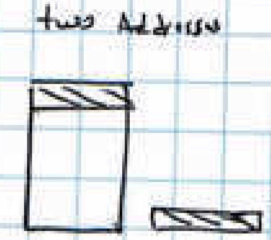
This mem port is to allow I/O devices access to memory

c. If there is only memory space, every memory module and every I/O device compares the address lines to the range of addresses that it services

- If the address falls in its range, it responds to the request

III. Since no address is ever assigned to both memory and an I/O device, there is no ambiguity and no conflict

iv. How if the same way you read and write to memory, you could do the same with devices?



I. The first part of memory it would be reserved for the I/O device, so if you write there. memory mapped

II. In page tables, we have page table entries.

A. In the page entries there is one field that take care of the I/O. this field is the disable cache (or don't do cache).

B. Every read and write will be performed directly to the device whent disable cache tells.

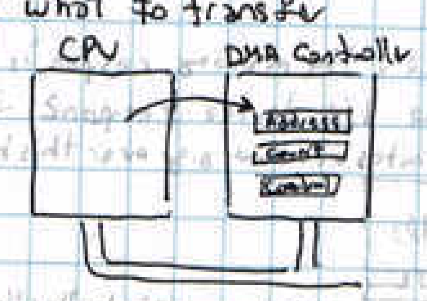
4- Direct Memory Access (DMA)

a. The OS can only use Direct Memory Access (DMA) if the Hardware has a DMA controller.

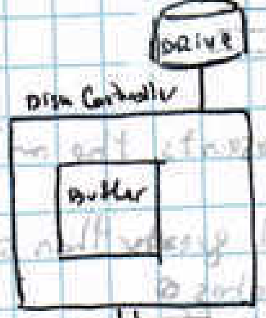
b. No matter the physical location of the DMA, the DMA controller must be able to acess the system bus independent of the CPU

I. These include memory address registers, a byte count Register, and one or more control registers. (to specify I/O port to use, direction of transfer (memory to I/O device), transfer unit (byte or word), and number of bytes to transfer in one burst]

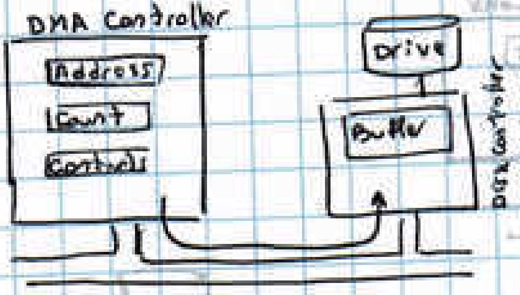
c. The CPU programs the DMA controller by setting its registers so it knows what to transfer



d. The CPU also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the check sum. When valid data are in the disk controller's buffer, DMA can begin

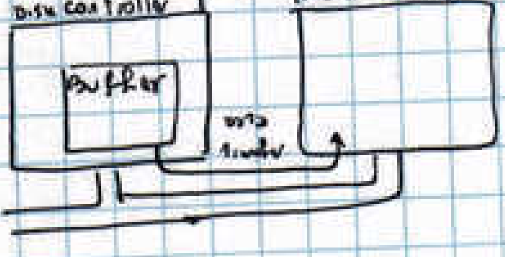


e. The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller

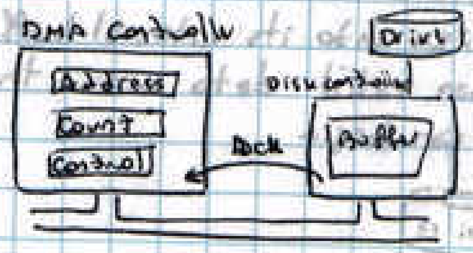


The disk controller read request As any other request since doesn't know or care if request came from the CPU or DMA controller

f. The memory address to write to is on the bus' address lines so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle.



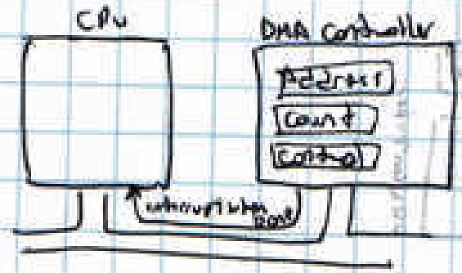
g. When the write is complete, the disk controller send an acknowledgment signal to the DMA Controller, also over the Bus



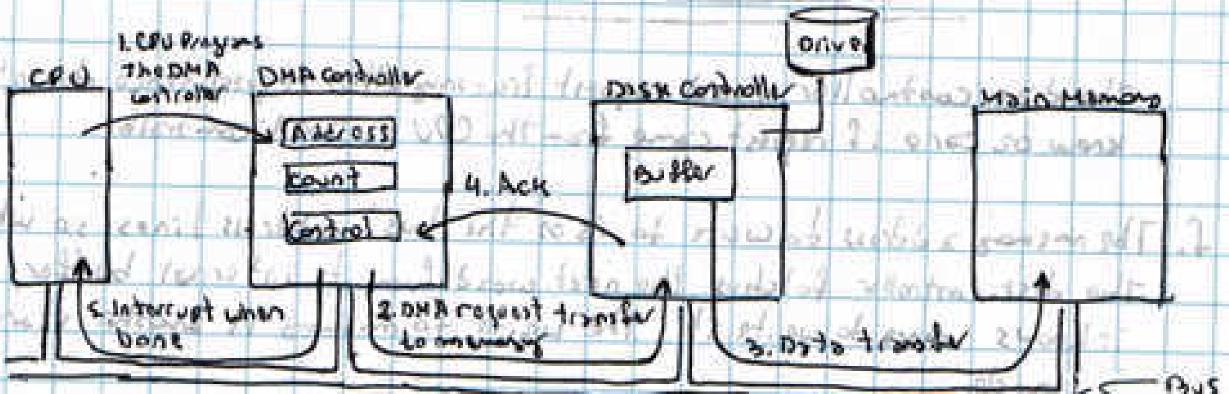
h. The DMA controller then increments the memory address to use and decrement the byte count

i. if the byte count is still greater than 0, repeat previous process of transfer until count reaches 0

ii. If byte count reaches 0, then the DMA controller interrupts the CPU to let it know that the transfer is now complete



i.

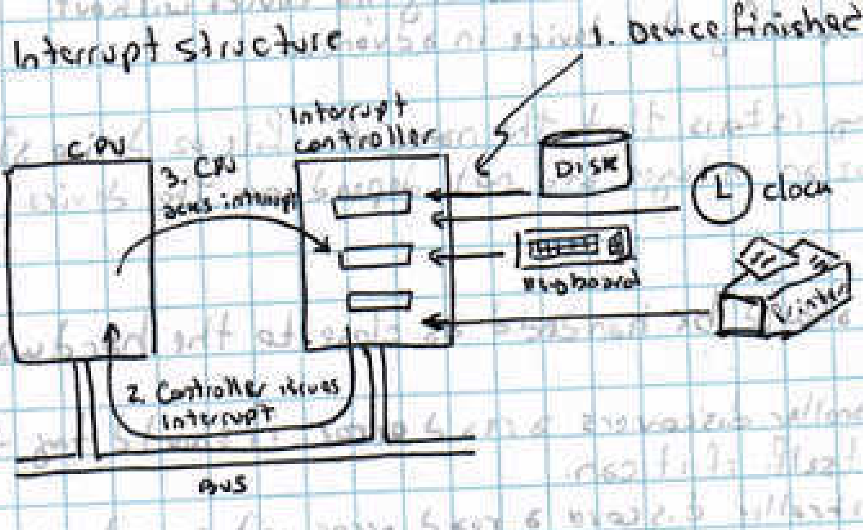


(operation of a DMA transfer)

[redacted] Final Review

5. Interrupts Revised

a. Interrupt structure



b. Interrupt controller will decide which devices can generate IRQ and what IRQ they can generate

I. If interrup tion controller do not recognize the IRQ, the IRQ will be lost

II. Interrupt controller designate IRQs to the devices

c. When an I/O device has finished the work given to it, it cause an interrupt (assuming that interrupts have been enabled by the OS). It does this by asserting a signal on a bus line that it has been assigned

d. this signal is detected by the interrupt controller chip on the motherboard, which then decides what to do.

e. If no other interrupt are pending, the interrupt controller processes the interrupt immediately.

f. If another one is in process, or another device has made a simultaneous request on a higher-priority interrupt request line on the bus, the device is just ignored for the moment. In this case it continues to assert an interrupt signal on the bus until it is serviced by the CPU