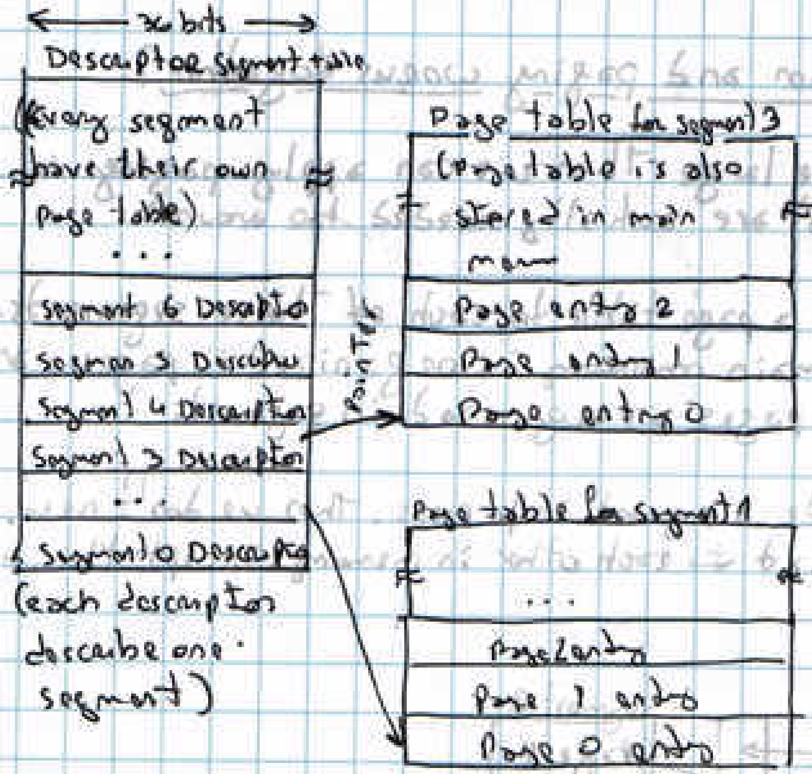
EX:



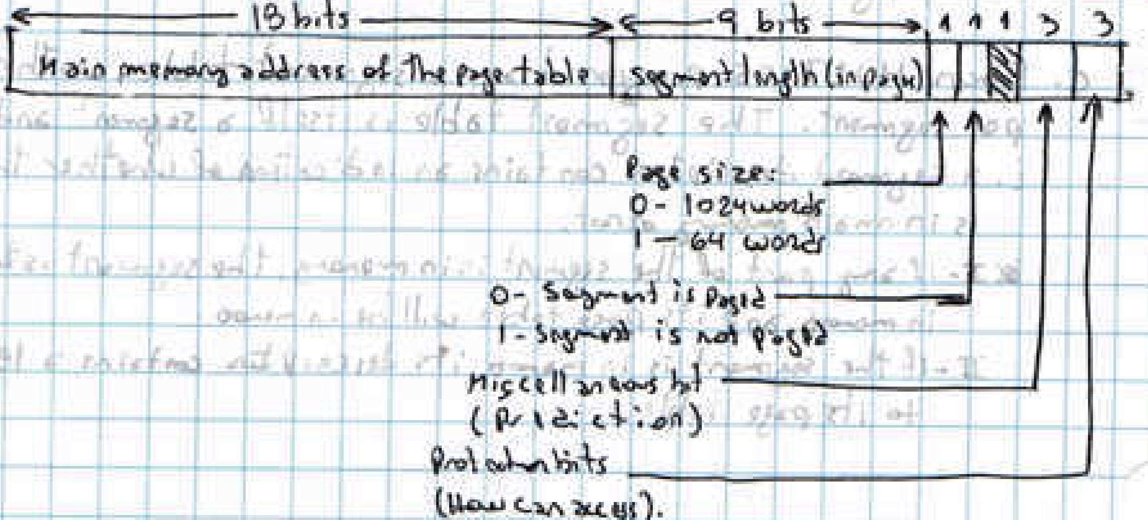
• you can either keep pages in memory or keep segment in it.

• Descriptor segment points to page tables

d. Since physical memory address are 24 bits and pages are aligned on 64-byte boundaries (6 lower bits of page address 000000), only 18 bits are needed in the descriptor to store a page address

• Descriptor also contains: segment size, protection bits, etc

i.



I. Segment descriptor numbers are field length.

II. The address of the segment in secondary memory is not in the segment descriptor but in another table used by the segment fault handler.

e. An address in MULTICS consist of two parts
  i. the segment
  ii. The address with in the segment.

• The Address with in the segment is futher divided into a page number and a word with in the page

	Address within the segment	
Segment number	Page number	Offset within the page
18 bits	6 bits	10 bits

34 bits
Virtual Address

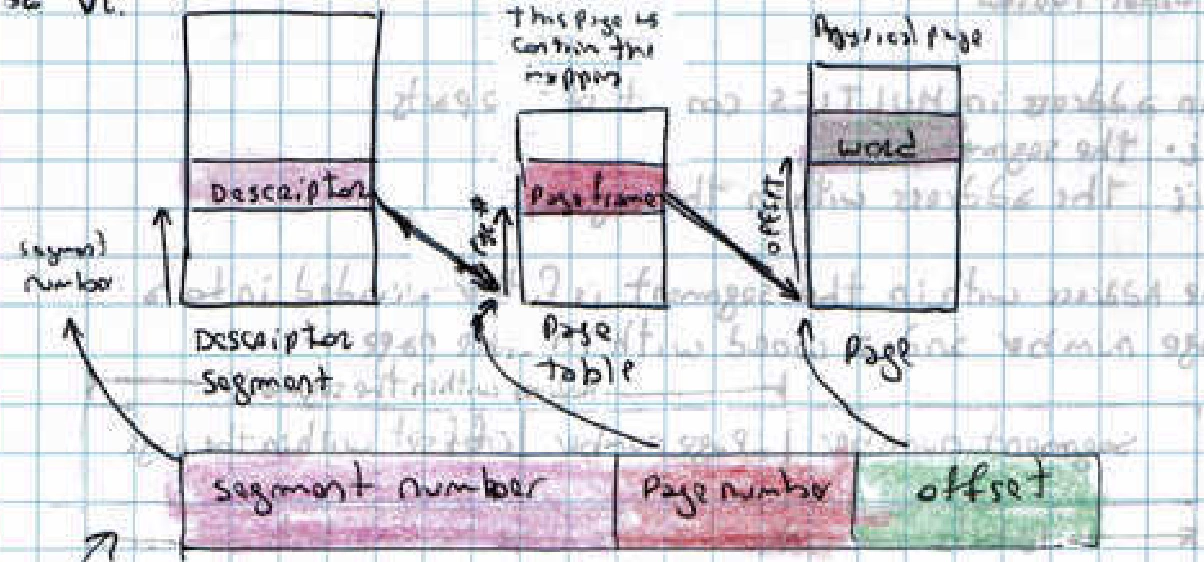
• Address within the segment is loaded in on ther egisters

f. When a memory referrence occurs:

  i. The segment number is used to find the segment desceiptor
  ii. A checu is made to see if the segment's page table is in memory
    I- If page table is in memory, it is located.
    II- If it is not, a segment fault occurs
    III- If there is a protection violation, a fault (trap) occurs
  iii. The page table entry for the regiested virtual page is examined.
    I- If the page itself is not in memory, a page fault is triggered.
    II- If it is in memory, the main memory address of the start of the page is extracted from the page table entry
  iv. The offset is added to the page origin to give the main memory address where the word is located.

  v. The read or store finally takes place.

vi.



(the fact that the ~~page~~ descriptor segment is itself paged has been omitted for simplicity)

I. A register (descriptor base register) is used to locate the descriptor segment's page table

II. the descriptor segment's page table points to the pages of the descriptor segment

III. Once the descriptor for the needed segment is found the addressing process begins

9. <u>Segmentation with Paging: the Intel pentium</u>

a. Pentium has 16Kb independent segments, each holding one billion 32-bits words.

b. Pentium virtual memory consist of two tables:
   i. Local Descriptor table (LDT): Each program have its own LDT
   ii. Global Descriptor table (GDT): Shared by all program on

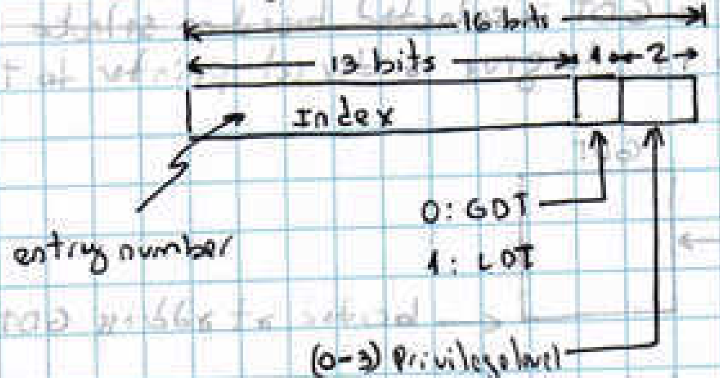
LDT: Describe segments local to each progr
GDT: Describe system segments including the OS itself

[redacted] final exam review

c. To access a segment, a pentium program first load a selector for that segment into one of the six segment registers:

i. CS register holds the selector for code segment

ii. DS register holds the selector for the data segment

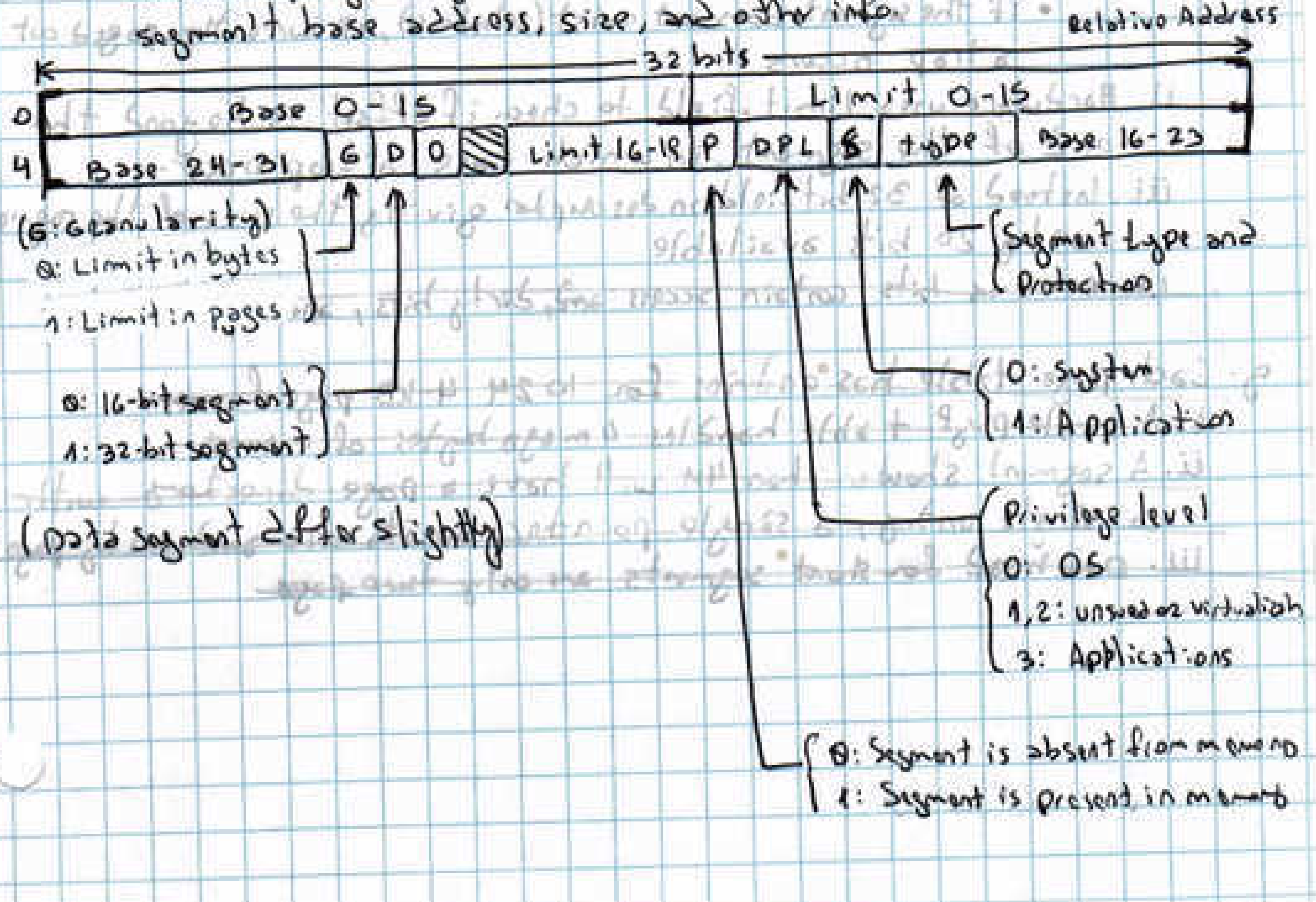


(selector: index to segment descriptor table, equivalent to segment Number on MULTICS)

d. Tables are restricted to holding 8K segment descriptors

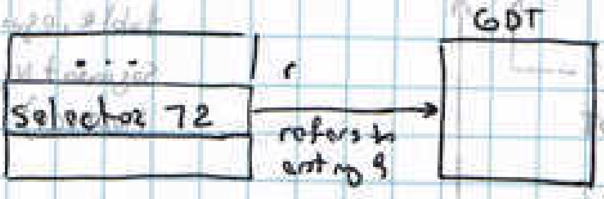
<u>note</u>: Descriptor 0 is forbidden, it causes a trap if used

e. When a selector is loaded into a segment register ~~to indicate that the segment~~ descriptor from LDT OR GDT is fetched and stored in microprogram registers. This descriptor is 8 bytes and include segment's base address, size, and other info.



(Data segment differ slightly)

i. Selector format chosen to make locating descriptor easy
~~ii. Corresponding descriptor LDT or GDT is selected based of~~
ii. From LDT or GDT, the corresponding descriptor is fetched and stored in microprogram registers (fast access)
iii. Selector is copied to an internal scratch register, and the 3 low-order bits set to 0
iv. Address of either LDT or GDT ~~is selected based on selector bits~~ 2 table is added to it, to give a direct pointer to the descriptor



← located at address GDT+72

f. Set steps by which a (selector, offset) is converted to physical address

i. ~~Assum~~ As soon microprogram knows which segment register is being used:
I - Find complete descriptor corresponding to that selector in internal registers
• If the segment does not exist (selector 0), or currently paged out a trap occurs
ii. Hardware uses Limit field to check if offset is beyond the end of the segment, in which case a trap occur.
iii. Instead of 32 bit field in descriptor giving the size of the segment, only 20 bits available
~~iv. remaining bits contain access and, dirty bits, and utility bits~~

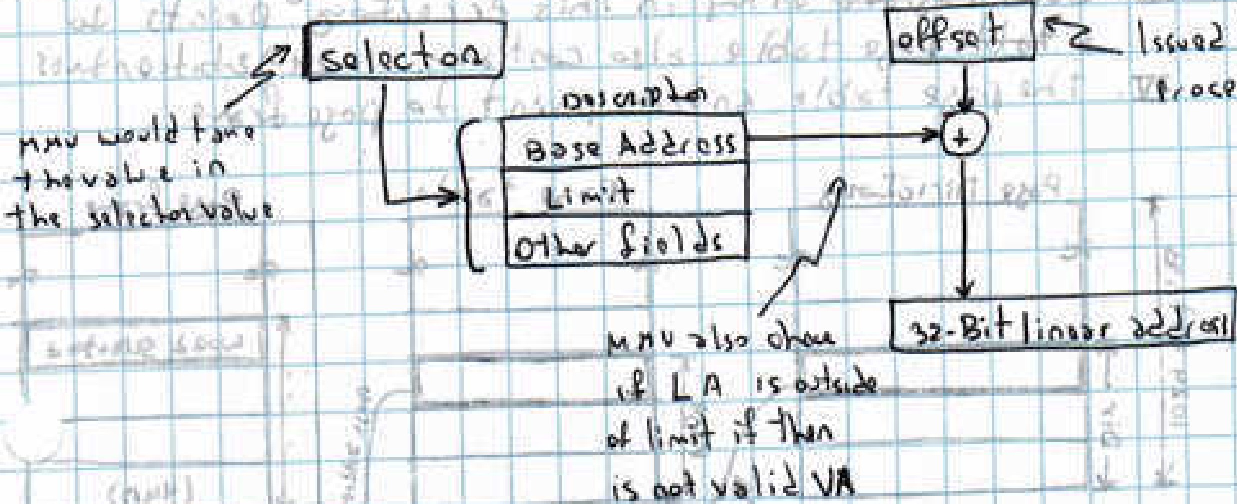
~~g. Each page table has entries for 1024 4-KB page frames~~
~~i. A single page table handles 4 megabytes of memory~~
~~ii. A segment shorter than 4M will have a page directory with a single entry, a single pointer to its one and only page table~~
~~iii. overhead for short segments are only two pages~~

[redacted] final exam review

iv. • If G (Granularity) is 0, the Limit field is the ~~exact~~ exact segment size, up to 1MB.
• If G is 1, Limit field gives the segment size in pages instead of bytes.

v. Pentium pages size is fixed at 4KB, so 20 bits are enough for segments up to 2^{32} bytes.

vi. Assuming segment is in memory and offset is in range, pentium adds 32-bit Base field in the descaiption to the offset to form called linear address



(Conversion of a (selector, offset) pair to a linear address (LA))

vii. • Base field is broken up into three pieces and spread all over the descriptor (for compatibility with 286 which base is 24 bits)
• Base field allows each segment to start at an arbitrary place within 32-bit linear address space.

~~viii. If paging is disable (by bit in global control register), the linear address is interpreted~~

viii. If paging is disable (by bit in global control register), the linear address is interpreted as the physical address and send to the memory for the read or write

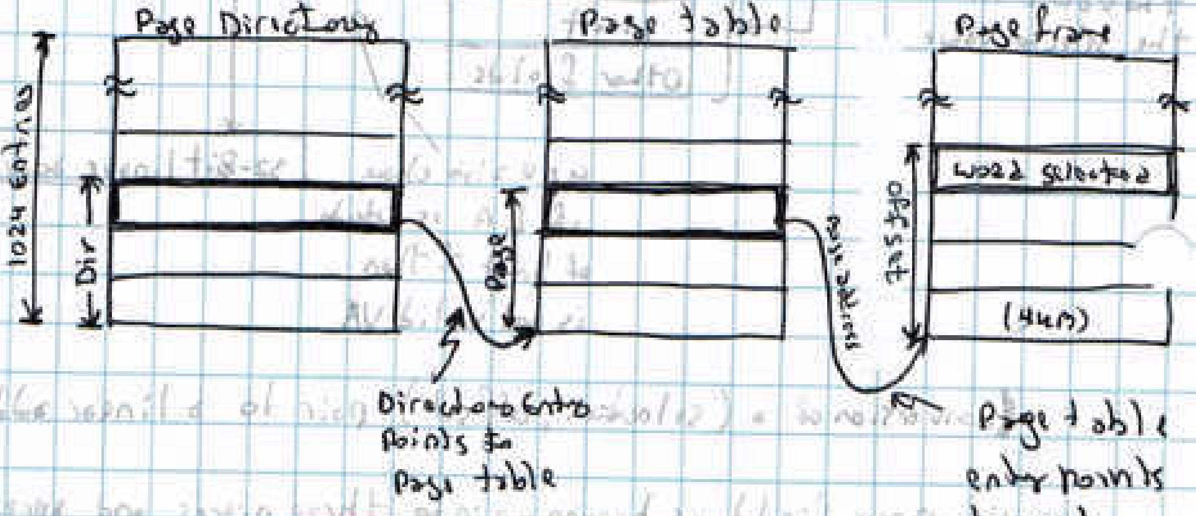
I - with paging disable, we have pure segmentation scheme, each segment's base address given in its descriptor

II - segments are not prevented from overlapping

- If paging is enabled, linear address is interpreted [redacted] a virtual address and mapped onto physical address using page tables.

ix. A segment may contain 1 millions pages, with 32 bit virtual address and 4-KB page, two level mapping is used to reduce the page table size for small segments.

x. Each running program has a page directory
   I. Consisting of 1024 32-bit entries.
   II. Located at an address pointed to by a global register
   III. Each ~~register~~ entry in this directory points to a ~~table~~ page table also containing 1024 32-bit entries
   IV. The page table entries point to page frame



(Mapping of linear address onto physical address)

xi. Linear address divided into three fields Dir, page, and Offset.

10 Bits	10 Bits	12 bits
DIR (Page Directory index)	Page Page table index	offset (used for access actual byte in page frame)

$$2^{12} = 2^{10} * 2^{2}$$
$$= 1KB * 4$$
$$= 4KB$$

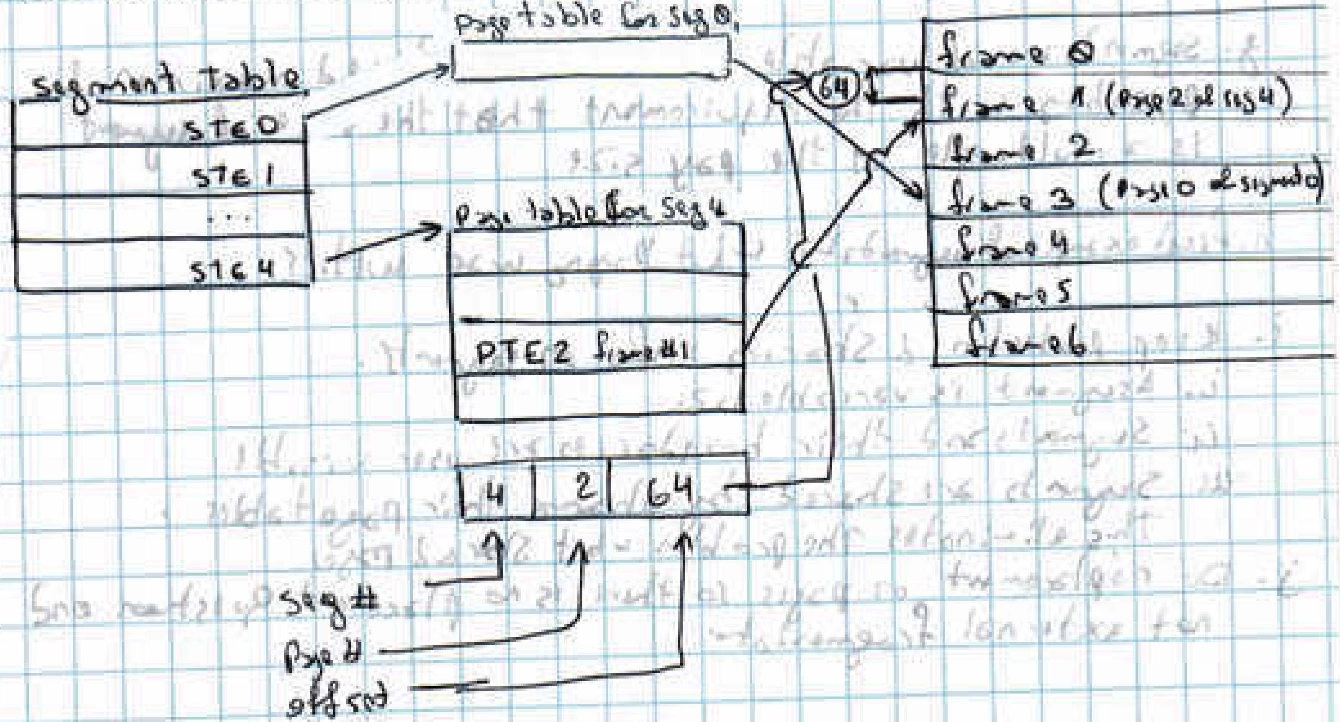
[redacted] final exam review

I. Dir field: used to index into the page directory to locate a pointer to the proper page table

II. Page field: used to index into the page table to find the physical address of the page frame

III. Offset: ~~used to the~~ added to the address of the page frame to get the physical address of the byte or word needed

10. General example of both segmentation and paging



a. A virtual address become a triple: (seg #, page #, offset)
b. each segment table entry (STE) points to the page table for that segment. Compare this with multi level pages
c. the size of each segment is a multiple of the page size (since the segment consists of pages).
   i. Perhaps not. Can keep the exactly size in STE (limit value) and shoot the process if it referenced beyond the limit. In this case the last page of each segment is partially valid.

142 d. The page # field in the address gives the entry in the chosen page table and the offset gives the offset in the page.

e. From the limit field, one can easily compute the size of the segment in pages (which equals the size of the corresponding page table in PTEs). Implementations may require the size of a segment to be multiple of the page size in which case the STE would store the number of pages in the segment

f. A straightforward implementation of segmentation with paging would requires 3 memory references (STE, PTE, reference word) so a TLB is crucial

g. segments are of variable size with a fixed maximum & possibly with the requirement that the size of segment is a multiple of the page size

h. first example of segmentation w/ paging was Multics

i. keep protection & sharing info on segments.
 i. A segment is variable size
 ii. Segments and their boundaries are user visible
 iii. Segments are shared by sharing their page tables. this eliminates the problem w/ shared pages

j. Do replacement on pages so there is no placement question and not external fragmentation

[redacted] Final Exam Review

1. Function of File System
   a. long term information Storage
   b. Store large amounts of data
   c. Information stored must survive the termination of the process using it
   d. Multiple processes must be able to access the information concurrently

2. Type of files
   a. regular files: contain user information
      ie: text files and binary files
   b. Directory: system files for maintaining the structure of the file system
   c. Character special files: Related with input/output and used to model serial I/O devices such as terminals, printers, etc
      ie: If I have a file /dev/sda this is a raw interface to the device
   d. Block special files: used to model disks

3. Sequencial access versus Random access (File Access)
   a. Sequencial Access: A process could read all the bytes or records in a file in order. Most applications do sequencial access
      i. Read all bytes/records from the beginning
      ii. Cannot jump around, could rewind or back up
      iii. Convenient when medium was magnetic tape.

144 b. Random Access: Possible to read a file out of order, [redacted] access records by key rather than by position. (i.e. open, Jump to position of the file, etc)

i. Bytes/Records read in any order

ii. Essential for data base systems

iii. Read can be:

I. Move file marker (seek), then read

II. Read and then move file marker

4. <u>Common file attributes and Operations</u>

a. Every file has a name and its data. All OSs associate other information with each file such as date and time file was last modified and size of file. We call these extra information the file's attributes. or meta data

File Attributes

	Attribute	Meaning
File Protection	Protection	Who can access the file and in what way
	Password	Password needed to access the file
	Creator	ID of the person who created the file
	Owner	Current owner
	Read only flag	0 for read/write; 1 for read only
	Hidden flag	0 for normal files; 1 for do not display in listing
	System flag	0 for normal files; 1 for system file
	Archive flag	0 for has been backed up; 1 for needs to be backed up
	ASCII/binary flag	0 - ascii / 1 - binary file
	Random Access flag	0 - ~~Random~~ sequencial access only / 1 - random access
	Temporary flag	0 - Normal / 1 delete file on Process exit
only record that can be look at with key	Lock flag	0 - unlocked ; non zero locked
	Record length	# of bytes in record
	Key position	Offset of the key within each record
	Key length	# of bytes in the key field
	Creation time	Date & time the file was created
	Time of last access	Date & time file was last accessed
	Time of last change	Date & time file was last changed
	Current size	# of bytes in the file
old mainframes →	Maximum size	# of bytes the file may grow

[redacted] final review

b. File/Directory Operations: most common system calls

i. Create: File is created with no data. The purpose of the call is to announce that the file is comming and to set some of the attributes

ii Delete: when file not longer needed, it has to be deleted to free up disk space

iii Open: Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls

iv Close: When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up internal table space. A disk is written in blocks, and closing a file forces writing of the file's last block, even though that block may not be entirely full yet.

v. Read: Data are read from file. Usually the bytes come from the current position. The caller must specify how many data are needed and also provide a buffer to put them in.

vi. Write: Data are written to the file again, usually at the current position. If the current position is the end of the file, the file's size increase. If the current position is in the middle of the file, existing data are overwritten and lost forever

vii. Append: This call is a restricted form of write. It can only add data at the end of the file

viii Seek: For random access files, a method is needed to specify from where to take the data. Common approach is a system call, seek, that repositions the file pointer to a specific place in the file. After this call has completed, data can be read from or written to, that position

ix. Get attributes: Processes often need to read file attributes to do their work
i.e.: make

x. Set attributes: some of the attributes are user settable and can be changed after the file has been created i.e. protection mode flags

xi. Rename: this system call make possible to change the name of an existing file

xii. Link: Linking is a technique that allows a file to reappear in more than one directory. This system call specifies an existing file and path name, and create a link from the existing file to the name specified by the path

xiii. Unlink: A directory entry is removed. If the file being unlinked is only present in one directory (normal case), it is removed from file system.
If present in multiple directories, only the path name specified is removed

Note: • Hard link: A link of this kind increments the counter in the file's i-node (to keep track of the number of directory entries containing the file)

• Symbolic link: Instead of having two names point to the same internal data structure representing a file, a name can be created that points to a tiny file naming another file.
i.e: First time file used, opened, the file system begins process all over using the new name
Advantages: they can cross disk boundaries and even name files in remote systems
disadvantages: less efficient that hard link

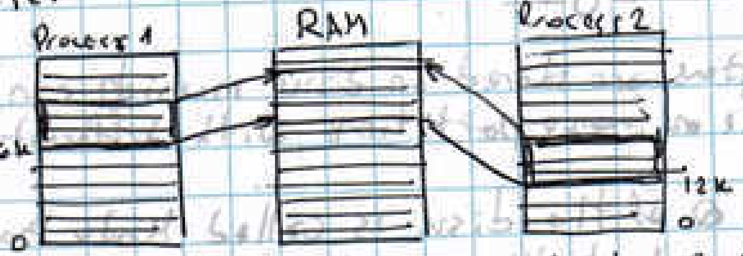
[redacted] Final review

5. Mapped files

a. A process can issue a system call to map a file onto a portion of its virtual address space

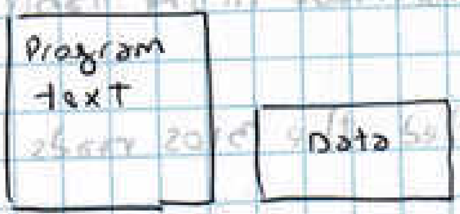
b. Mapped files provide an alternative model for I/O. Instead of doing reads and writes, the file can be accessed as a big character array in memory

c. If two ~~or~~ more processes map onto the same file at the same time, they can communicate over shared memory. Writes done by one process to the shared memory are immediately visible when the other one reads from the part of its virtual address spaced mapped onto the file.

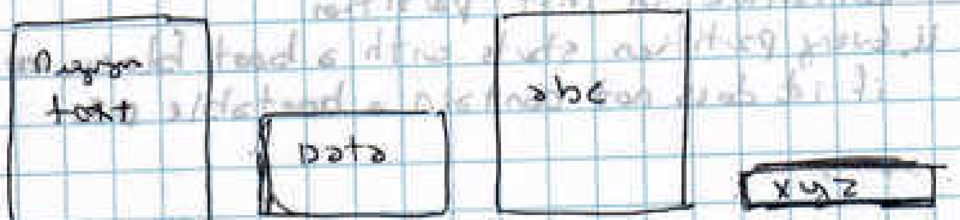


(shared library being used by two processes)

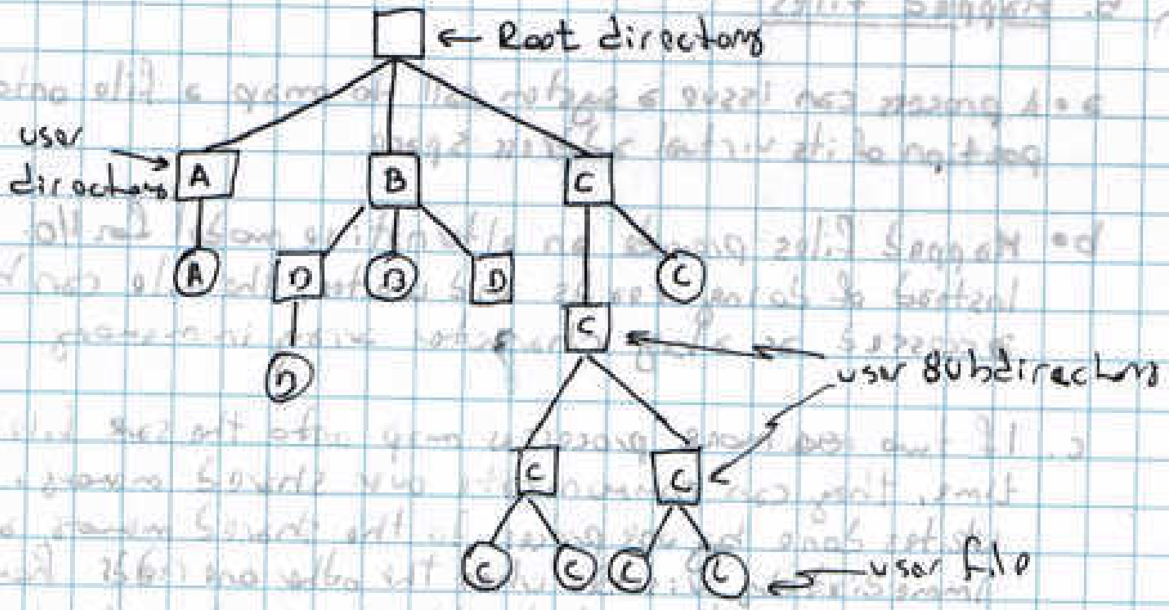
d. i Segmented process before mapping files into its address space



ii Process after mapping: Existing file abc into one segment creating new segment for xyz



6. Hierarchical Directory Systems



7. File system layout

a. File systems are stored on disks in which can be divided into one or more partitions, with different filesystems

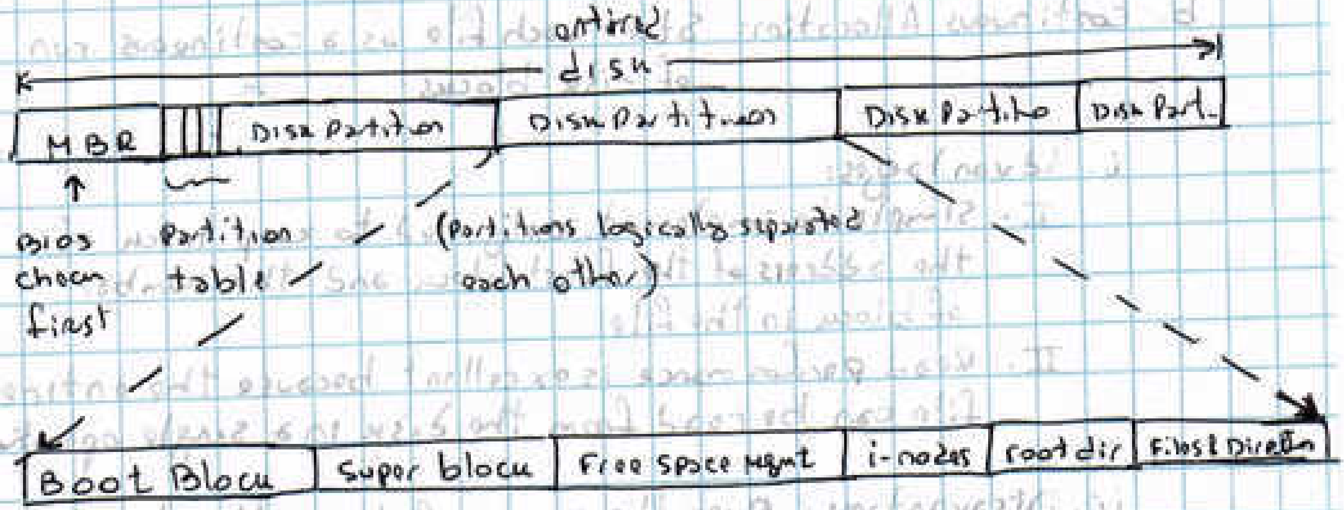
b. Sector 0 of the disk is called Master Boot Record (MBR)
   i. used to boot the computer
   ii. the end of the MBR contains the partition table.
      I. Table gives the starting and ending addresses of each partition
      II. One of the partitions in the table is marked as active.

c. When computer is booted, the BIOS reads in and executes the MBR.

d. The MBR locate the active partition, read its first block, called boot block, and execute it.
   i. Program in the boot block loads the operative system contained in that partition
   ii. Every partition starts with a boot block, even if it does not contain a bootable OS.

[redacted] final review

e. layout



i. super block: Contains all key parameters about the file system and is read into memory when the computer is booted or the file system is first touched. Information holders:

I. magic # to identify the file system type
II. # of blocks in the file system
III. other key administrative information

ii. Free space Mgmt: free blocks in the file system
ie: bitmap or list of pointers

iii. i-nodes: An array of data structures, one per file, telling all about the file

iv root directory: top of the file system tree

v. Remainer of the disk contains all the other directories and files.

1507. <u>Implementing files</u> [redacted]

a. keeping track of which disk blocks go with which file.

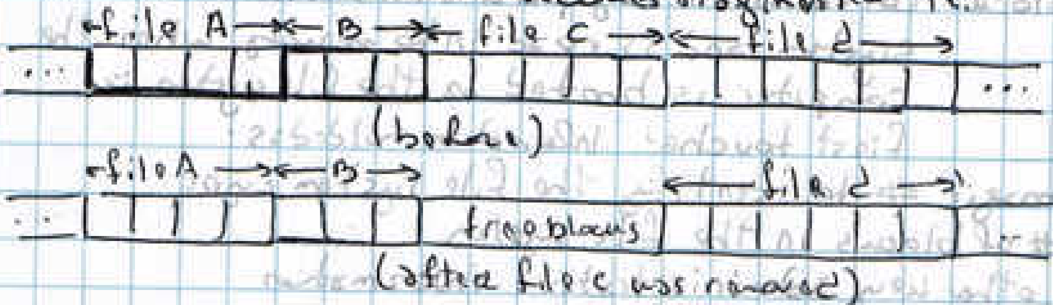
b. continuous Allocation: Store each file as a contineous run of disk blocks

i. Advantages:

I. Simple to implement. only need to keep track of the address of the first block and the number of block in the file

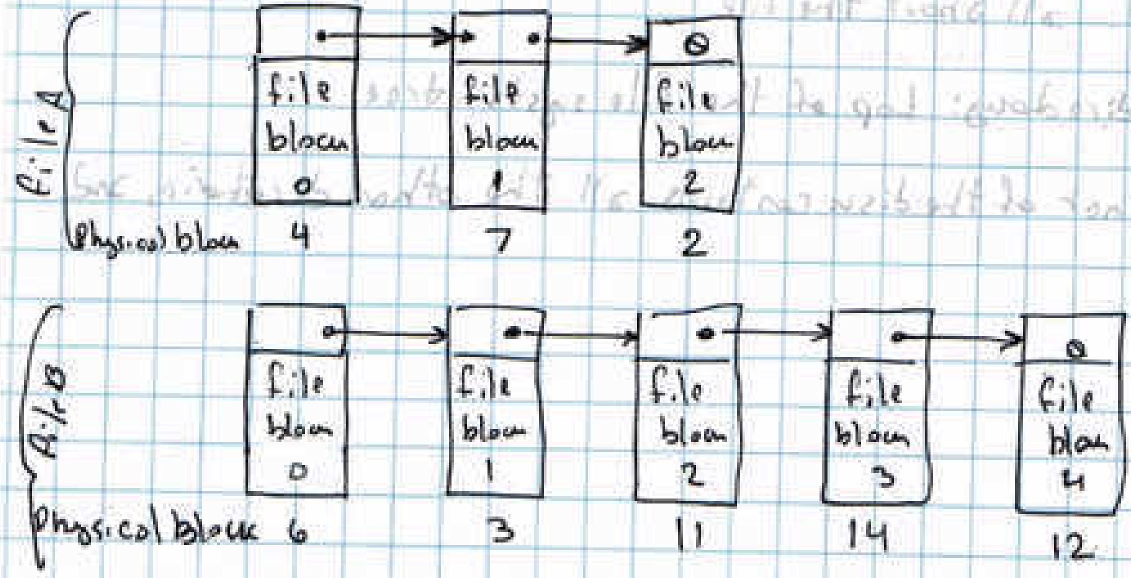
II. Read performance is excellent because the entire file can be read from the disk in a single operation

ii. Disavantages: Over the course of time, the disk becomes fragmented ie:



c. Linked List Allocation: Keep each one as a linked list of disk blocks.

i. The first word of each block is used as a pointer to the next block. the rest of the block is data.



[redacted] final Review

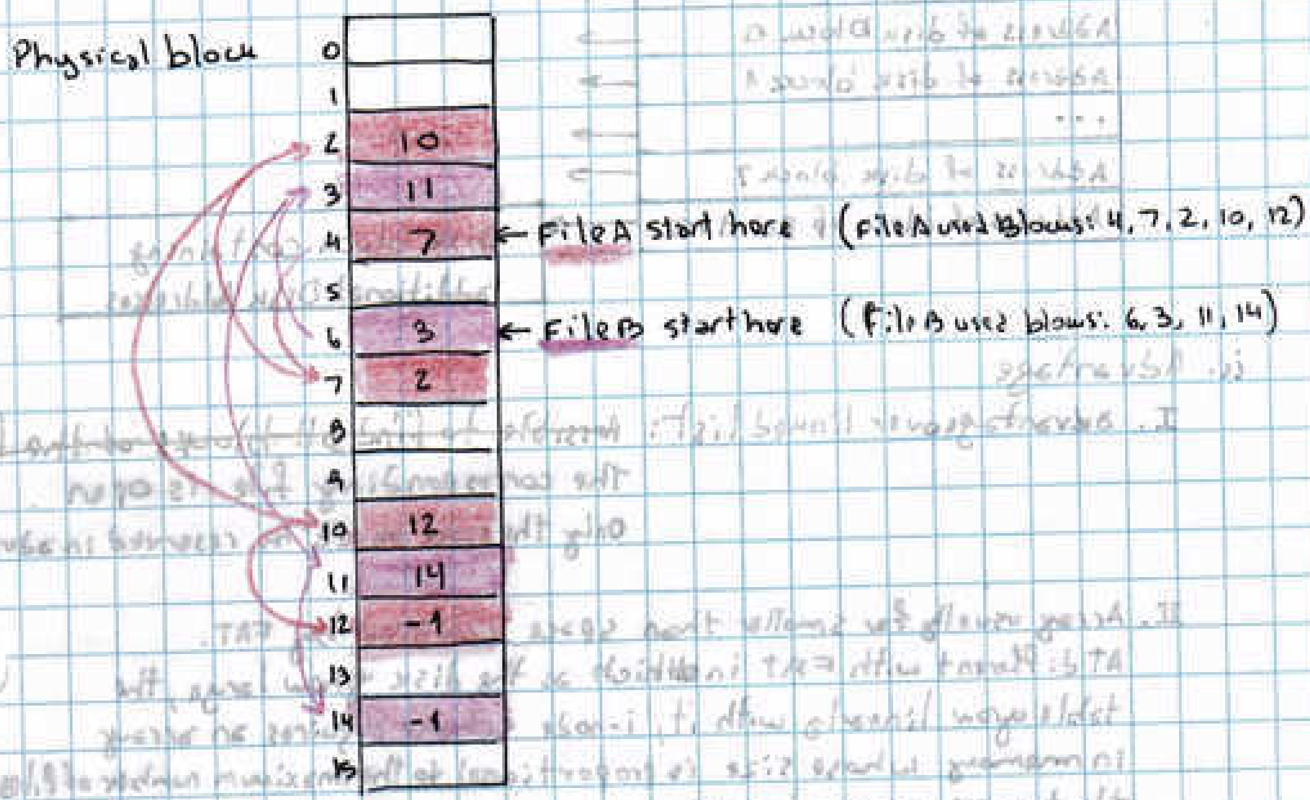
ii. Advantage: I. No space is lost to disk fragmentation
II. Sufficient for the directory entry to merely store the disk address of the first block

iii Disadvantages:
I. internal fragmentation in the last block
II. Although reading a file sequentially is straightforward, random access is extremely slow
III. Amount of data storage in a block is no longer a power of 2 because the pointer takes up a few bytes. Many programs read and write blocks of size power of two.

d. Linked List Allocation Using a table in Memory:
Both disadvantages of the linked list can be eliminated by taking the pointer word from each disk block and putting it in a table in memory.

i such table in memory is called File Allocation Table (FAT)

ii. Linked List allocation using a file allocation table in memory



iii. advantages:

152 I. Using this organization, the entire block is availabl[redacted] for data, making also random access much easier

II. Although chain must still be followed to find a given offset within the file, the chain is entirely in memory, so it can be followed without making any disk references

III. It is sufficient for the directory entry to keep a single integer (the starting block number) and still be able to allocate all the blocks, no matter how large the file is

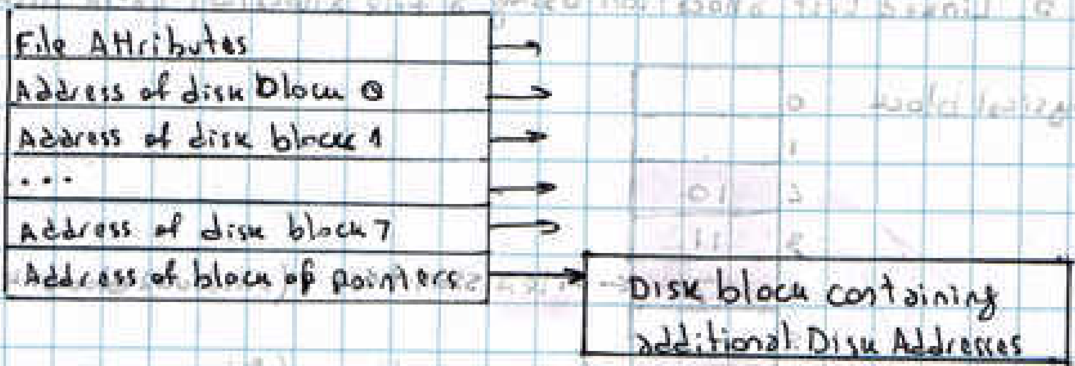
iv. disadvantages:

I. Entire table must be in memory all the time to make it work

II. The FAT idea does not scale well to large disks
ie: for 200 GB disk & 1KB Block size, each entry must have 3 bytes, therefore table will take up 600 MB to 800 MB of main memory

e. I-nodes: Keeping track of which blocks belong to which file is to associate each file data structure called an i-node (index-node), which list the attributes and disk addresses of the file's blocks

i.



ii. Advantage

I. advantage over linked list: i-node need only be in memory when the corresponding file is open
Only the space need be reserved in advance

II. Array usually far smaller than space occupied by FAT.
At different with FAT in which as the disk grow large, the table grow linearly with it, i-node scheme requires an array in memory whose size is proportional to the maximum number of files that may open at once.