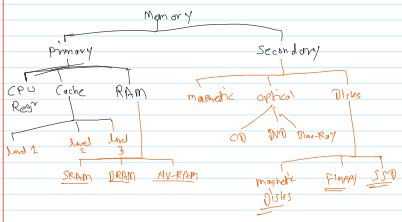
[[[

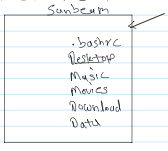
22 November 2023 08:38

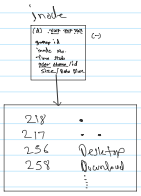
Memory and Storage

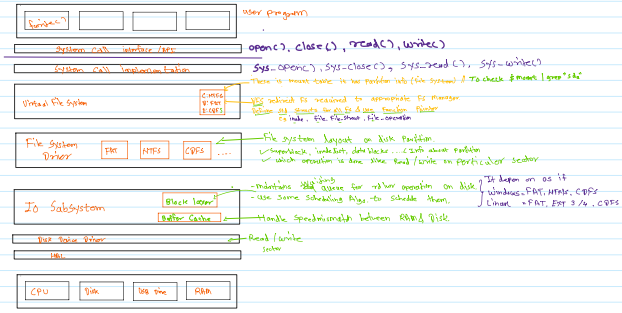


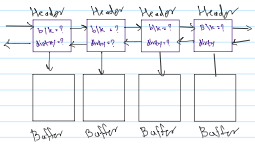
* Directory Entry Concept

* Name \ Sunbeam
Sunbeam









Buffer Cache

- Buffer cache kept in RAM
- Internally it is linked list (Doubly linked list)
- Every buffer has header (size 4kB)
- Header kept info like buffer block where in Disk
  - This buffer dirty buffer or not

Dirty Buffer : If we read data from buffer and after we can modify disk content of that particular buffer... means file has
- disk data on RAM and disk is known as Dirty Buffer
- If copy is both place same it not be Dirty Buffer

$ free -mh

* File - Sys call

① File Sys Sys calls (meta data)

① link( ) .... (Hard link)
② unlink( ) ... (remove link
③ symlink( ) .... symbolic link / shortcut
④ chdir( ) ..... change directory

② File io Sys-Call (rw_ data)

- ① open( )
② read( )
③ write( )
④ close( )
⑤ lseek( )

* File sys calls

① int fd = open ("file path", flags, mode)

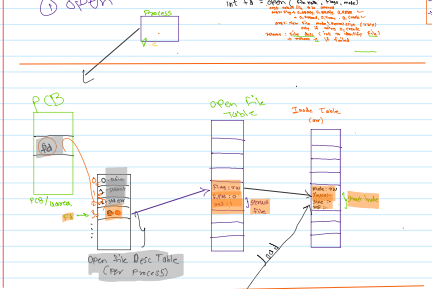
② close (fd)
arg 1 → file desc of the file to be closed

③ ret = write (fd, addr, nbytes)
= arg1 : File desc of file
arg2 : base address of data to write
arg3 : num of bytes to write
return : return num of bytes Sucessfully written

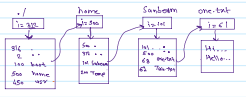
④ ret = read (fd, addr, nbytes)
= arg1 : File desc of file
arg2 : base address of var where to read data
arg3 : num of bytes to read
return : return num of bytes Sucessfully read

* Unix File Sys Call

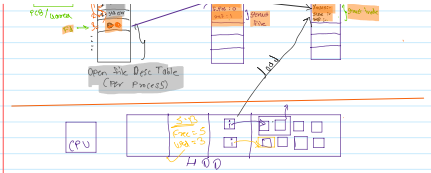
① open



O_TRUNC flag is often used with the open() system call to truncate an existing file to zero length or create an empty file if it does not exist.
- O_WRONLY → write only
- O_RDONLY → Read only
- O_RDWR → Read as well write
- O_CREAT → Create file

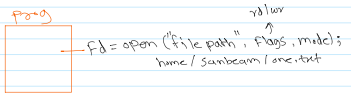


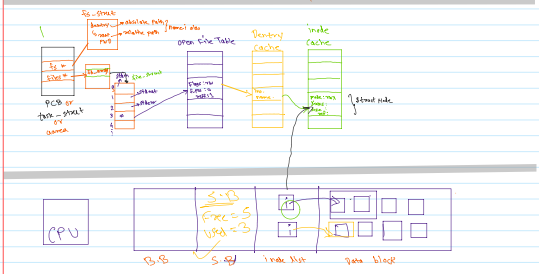
* open() Syscall
① File path → inode number (name's)
② load inode into in-memory inode table / cache
③ make an entry into open file table
④ add a pointer into open file descriptor table
⑤ return index to open file Table entry i.e. file descriptor



② load inode into in-memory inode table / cache
③ make an entry into open file table
④ add a pointer into open file descriptor table
⑤ return index to Open File Table entry i.e. fds descriptor

② Linux_open Syscall Internals

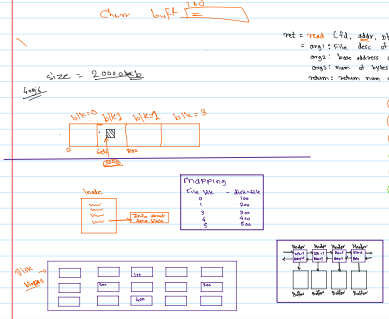




① open() Sys call api
② Sys_open Sys call
└ ③ Convert path into inode using name() algo.
④ Load inode into in-memory
inode table
→ make / create entry in dentry cache
⑤ make entry of file into open file Table
⑥ make entry into open file Descriptor Table
⑦ return index of OFT entry i.e file desc

* inode table entry
Struct inode
- i no, permission, uid, gid
- time stamps, size, type
- info of data blocks.
* Open file table Entry
- Struct file
- flags, file_pos, refcount

Read System Call



ret = read (fd, addr, nbytes)
- arg1: file desc of file
- arg2: base address of var where to read data
- arg3: num of bytes to read
- return: return num of bytes succesfully read

① read (fd, buf, nbytes): → sys_read()
② Get current file position (OFT) → (File offset) fpos = ...
③ Calculate file logical block & block offset → (File block = ... offset: ...
④ get the inode of the file (inode has info about disk data blocks)
⑤ Convert file logical block to corresponding disk block → file blk: → disk block ...
⑥ Check data is present in buffer Cache
⑦ If not found in cache, instructs disk driver to load it
↳ and block (waiting) the calling process.
⑧ read position of file block & copy it into user buffer.
Also modify curr file pos.
⑨ return num of bytes sucessfully read.

→ read (fd, buff, len) → User space

read (file, file_pos, buff, len)
file_pos = (current file block)

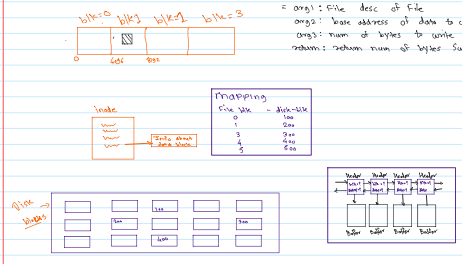
read ('inode, file_block_no) : physical fs
file block = disk block

read (dev_no, disk_block_no);
- check buff cache & return if found
- call disk driver if not found

disk_io (dev_no, disk_block_no); // Device Driver

(disk)
disk controller

* Write System Call



ret = write (fd, addr, nbytes)
= arg1: file desc of file
arg2: base address of data to write
arg3: num of bytes to write
return: return num of bytes Succesfully written

① write (fd, buf, nbytes);
② Get current file position (OFT)
③ Calculate file logical block & block offset
④ Get inode of file
⑤ Convert file logical block to corresponding disk block
⑥ Check if disk block is present in buffer Cache
⑦ If not found in cache, instruction disk driver to load it
and block the calling Process.
⑧ write portion of file\ disk block Copying from user buff.
⑨ mark buffer as dirty and instruct disk driver to write it
⑩ return num of bytes Successfully written.

* lseek() System Call

① To Set file position to Start of file
↳ rewind (fp); //c
↓
lseek (fd, 0, SEEK_SET); // Sys Call

② To set file pos to End of file
└ lseek (fd, 0, SEEK_END)
↳ f_pos = file size

③ To change File position wrt Current file
→ lseek (fd, -10, SEEK_CUR)
offset Origin

\

② To change file position until (current file

→ lseek (fd, -10, SEEK_CUR)
offset origin

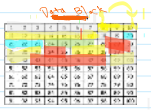
→ lseek (fd, +10, SEEK_CUR)
offset origin

→ f_pos = f_pos + offset

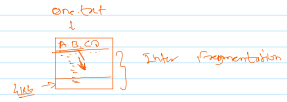
★ Disk Allocation

→ ① Contigeous allocation
② Linked allocation
③ Indexed allocation

① Contigeous allocation

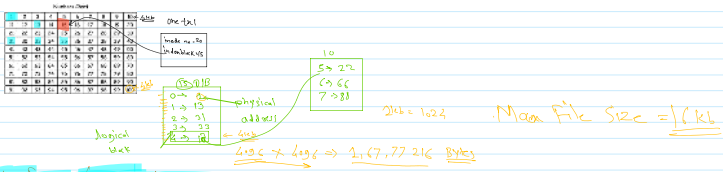


one.txt
inode
1 mdc_num - 20
starting block add - 8
Count - 16



② Linked list allocation

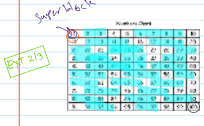
③ Indexed Allocation:



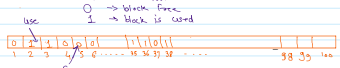
★ Disk Free Space Management

→ ① Bit Vector
② Linked list
③ Grouping
④ Counting

① Bit Vector



n blocks = n bits
nth bit = nth block
0 → block free
1 → block is used



- when free data block is needed, file system manager will find first 0 bit in the bit vector.
- If nth bit is 0 means nth block is free & is given I/O to file.

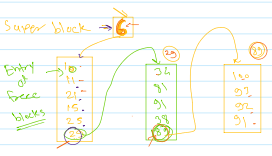
② Linked list

- Super block keep address of first free data block.
- Each free block keep address of next free block.

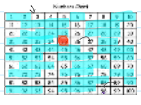
Super Block : 8 →

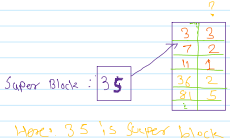
- when a block of memory for a file, it will take from it from free list (Delete first).
- when block is released (due to file delete or truncate) it will be added in the list.

③ Grouping



④ Counting





Here, 35 is super block store count how many free blocks available from free block.