FS

- each file has a single owner-ID and group ID and others

  e.g. -rw-r--r-- (owner: read, write) (group: only read)
  (other: only read)

- Pathname

  e.g. /var/mail/root

  "/" "/var" "/var/mail" → dir root → file

- Name Resolution
  - Processes use pathname to get access to file system resources
    files and dirs.

- Name Server
  - performs name resolution to convert pathname to file system resources.

- Symbolic link

  -ln -s <target-name> <dest._name>

- Case sensitivity — Unix
  Case preserving — Windows, MAC OS

- block is the smallest unit of allocation in fs.

- FS (P3)

  File-system (dfilesys_t) Directory_entry (dDentry_t)
  file (fcb_t) blocks (dblock_t) → free list
  Directory (ddir_t) bitmap_size

  - 0th block for dfilesys_t. { refer to "/" dir
    the first free block in fs.

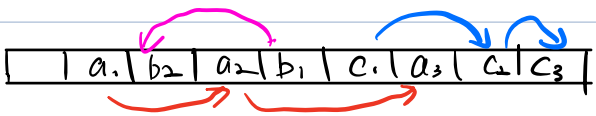
  - "/" dir is ddir → contains entries (ddentry_t)

  - ddentry_t: a data block for dir contains a list of its entries. { Block (ddir or fcb_t)
    Type → dir or file
    name

- file: fcb_t { uID
  Type: regular or sym link
  size
  Attrblk

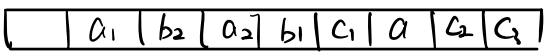
- Contiguous Allocation : |A|A|A|B|B|C|C|C|C|D|
  - advantage: ① good performance in reading successive block
    ② only need the starting location.
  - disadvantage: ① need to know file size at first
    ② external fragmention.

- Linked list Allocation:

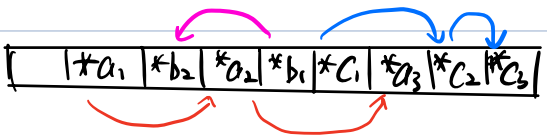


  - advantage: no external fragmention.
  - disadvantage: random access is slow since we only know the first pointer of a file.

- Linked list Allocation using Index (DOS) — FAT

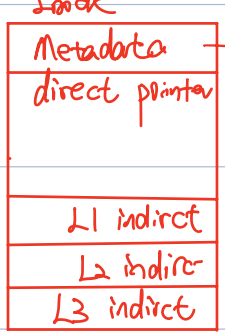


disk



FAT → originally in disk, put in mem when use.

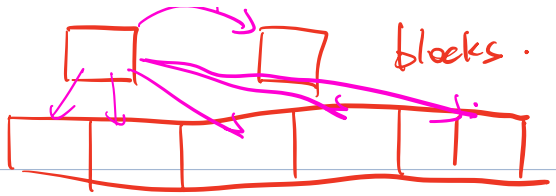
- Indexed Allocation (UNIX) — INODE
  - each file directly have pointers to all its blocks. the number of pointers for a file can be large.



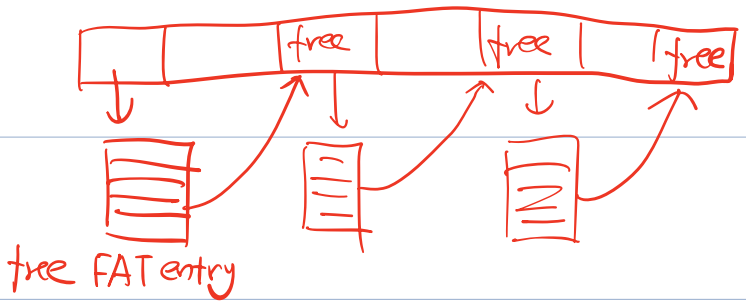
= data of data"
a set of data that describe or give information of other data.

- Tracking free blocks.
  - each block contains ptrs to free blocks, and last ptr points to another blocks of ptrs. (UNIX)

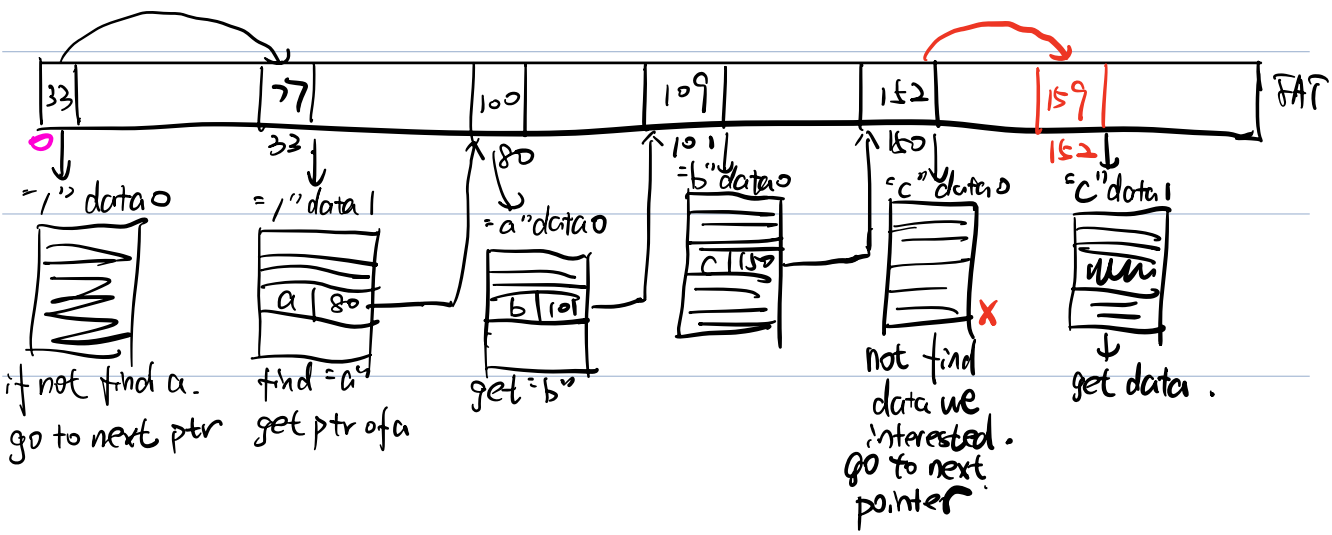
last ptr



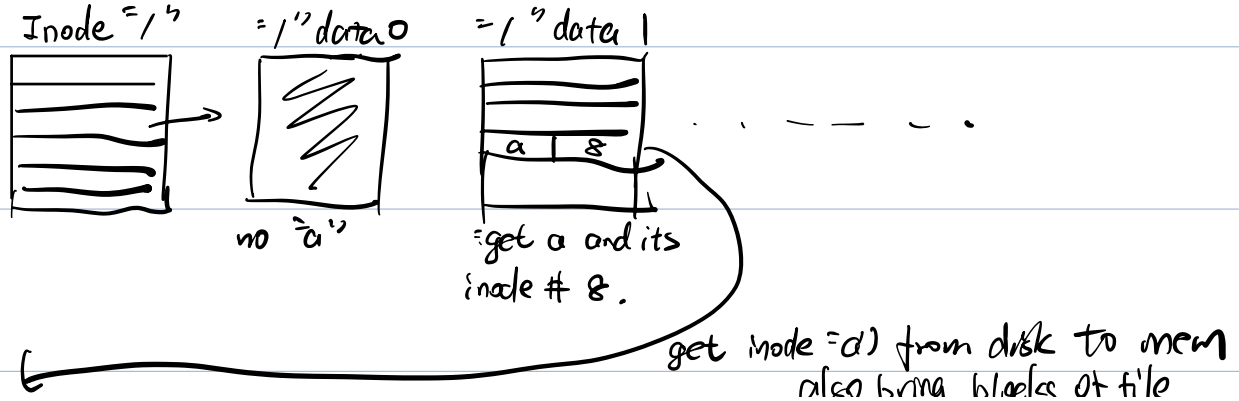
- pointer to a free FAT entry, which in turn points to another free entry (DOS)

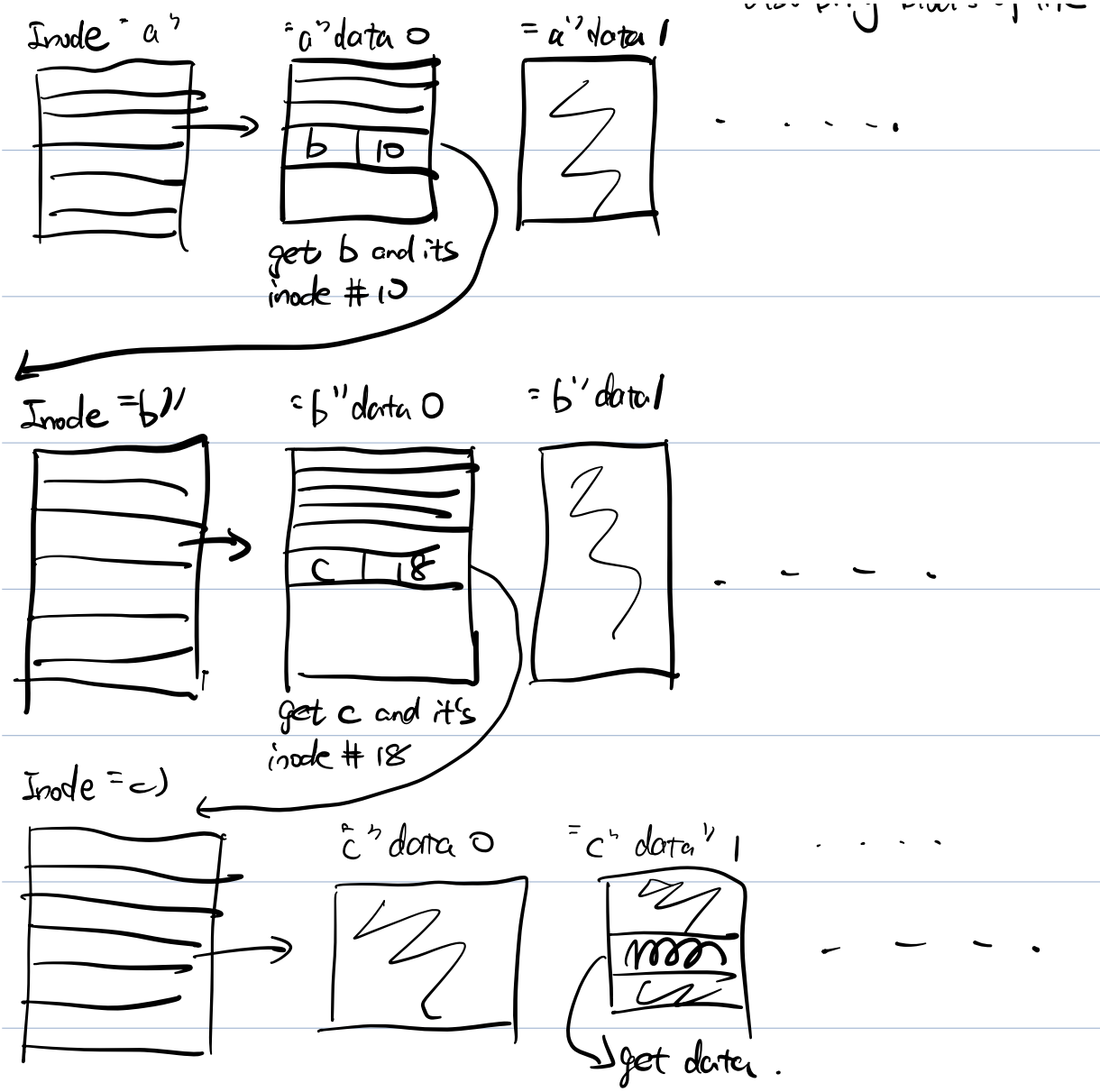


- Accessing a file blocks in DOS /a/b/c.



- Accessing a file blocks in UNIX /a/b/c





- File catching / buffering
  - read
  - write { write back : lose data when crash — UNIX
    write through : lose performance — DOS
  - replacement policy
    - LRU : not use on demand
    - High and low water marks

when the free block # is less than low water marks implement LRU to raise free block # to high water marks.

- Block size.
  - Larger block size
    - higher internal fragmentation
    - unnecessary data
    + higher disk transfer rate

- FS reliability:
  - Availability of data and integrity of data
  - 2 scenarios: { disk can go bad → RAID; Machine can crash; user can make mistake }
  - disk can go bad
    - RAID
    - Once bad blocks/sectors are detected, you mark them, and do not allocate on them
  - Machine crash
    - Fsck
      ① Blocks:
      - for every block, keep 2 counters:
        a) # of occurrences in files
        b) # of occurrences in free list
      - Ideally, (a)+(b) = 1 for every block
      - However
        If (a) = (b) = 0,
        miss block, add to free list
        If (a) = (b) = 1.
        remove from free list

If (b) > 1
remove duplicate from free list

If (a) > 1
make copies of this block, and insert each of the other files

② files

(a) counter for each file, increment the counter for the inode

(b) link number in inode.

- Ideally (a) = (b)
- If (a) > (b)
  (b) = (a)
  If (a) < (b)
  (b) = (a)

IO

- Disk
  - A disk has platters, and each platters have 2 surfaces.
  - Each surface has circular tracks.
  - Each track has sectors
  - The same track across all surfaces of all

platters, together constitute a cylinder

- There is a head for each surface of each platter.

- perform
  - (a) seek a specific cylinder
  - (b) wait for the given sectors to come under the head
  - (c) transfer the data.
  - → (a) is typically most dominant

- Disk scheduling

waiting request for cylinders: head 11

1, 36, 16, 34, 9, 12.

① FCFS

- service order: 1, 36, 16, 34, 9, 12
- cost: (11-1) + (36-1) + (36-16) + (34-16) + (34-9) + (12-9)
  = 10 + 35 + 20 + 18 + 25 + 3 = 111

② Shortest Seek time First (SSTF) — not fair

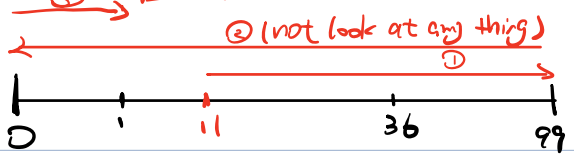
- service order: 12, 9, 16, 1, 34, 36
- cost: 1 + 3 + 7 + 15 + 33 + 2 = 61

③ SCAN Algorithm

not fair (center number have short search time)

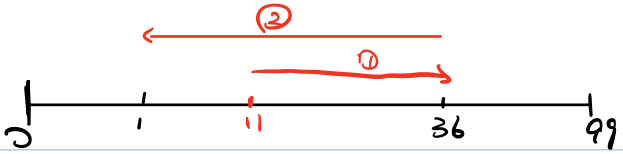
- service order: 12, 16, 34, 36, 99, 9, 1
- cost: 1 + 4 + 18 + 2 + 63 + 90 + 8 = 186

④ C-SCAN



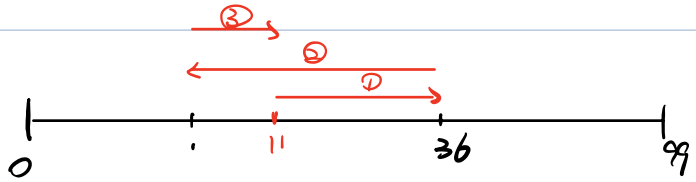
- service order : 12, 16, 34, 36, 1, 9.
- cost : 1+4+18+2+63+99+1+8 = 197

⑤ Look



- service order : 12, 16, 34, 36, 9, 1
- cost : 1+4+18+2+27+8 = 60

⑥ C-Look



- service order : 12, 16, 34, 36, 1, 9.
- cost : 1+4+18+2+35+8 = 68

- I/O software
  - layer :
    - lowest layer : Device driver
    - Middle layer : Device independent OS software
    - High level : User-level software / libraries

(part of kernal)

- Device independent OS software
  - Each device given a major #, minor #
    - major #: identify the driver
    - minor #: passed on to the driver (to handle sub-driver)
  - buffering / catching
  - handle error
- DMA controller
  - why RAM in card?
    - to address the speed mismatch between the bit stream coming from disk and the transfer to main memory.

- Physical not Virtual mem.
  - since the address generated by the DMA do not go through the MMU.
  - do not want to give this to user program
  - address need to be pinned in mem.

Consistancy

- Exclusion: you don't want threads interleaving with each other
- Ordering:
- atomically: set of instruction to execute without interupting
- race condition: result of a sequence of instructions depends on its interleaving with other instructions.
- critical section: set of instructions or codes that only one thread can execute at a time
- mutual exclusion: only one thread can enter critical section at a time
- Implementing mutex:
  1. Disable Interrupts
     - Do not want to give user such option
  2. Busy-wait / spinlock Solution

- Pure software
- Integrated hardware support

③ Blocking Solution

- Correctness : mutex
  - progress : If one thread request access a C.S. it should be able to enter if no other thread is interested.
  - Bonded wait : all threads will go to C.S eventually

1. Strict Alternation

```
turn = 0;
T0 {
  while (turn != 0)
        ;
  ---------------
  CS
  ---------------
  turn = 1;
}

T1 {
  while (turn != 1) {
        ;
  }
  ---------------
  CS
  ---------------
  turn = 0;
```

no satisfy progress

```
bool flag[2];
T0 {
  flag[0] = T;
  while (flag[1] == T) {
      ;
  }
  ---------------
    CS
  ---------------
  flag[0] = F;
}
T1 {
  flag[1] = T;
  while (flag[0] == T) {
      ;
  }
  ---------------
      CS
  ---------------
  flag[1] = F;
}
```

no satisfy

Bounded wait

Peterson's Algo

```
int turn
bool flag[N] = false;

enter_cs(myid)
    otherid = 1 - myid;
    flag[myid] = T;
    turn = myid;
    while (turn = myid && flag[other] == T)
        ;
    }
```

CS

```
leave_cs
    flag[myid] = F;
```

Correctness:

- If both are in CS, then 1 condition at least must have been false for both condition.

- turn == my id can not be false for both

progress:

- If a thread is waiting in the loop, the other person have to be interested.
- One of two definitely get in

bounded wait:

- when only one thread, it will go through
- when two threads interested, the first did turn == my id will go through.
- when the current thread is done with CS, the next time it requests the CS, it will get it only after any other thread waiting at the loop.

busy wait (with HW

- Bool test & set (bool)

```
temp = bool;
bool = T;
return temp;
```

- Swap (a, b)

```
temp = a;
a = b;
b = temp
```

- using swap:

```
key = T;
while (swap (key, lock) {
    ;
}
```

- Using Test & Set

```
while (Test & Set (lock)) {
    ;
}
```

- Blocking solution
  - mutex lock, mutex unlock
  - signal wait

- Synchronization Problems

① Bounded-buffer Problem

```
int BB[N];
int count, head, tail = 0;
Semaphore  m = 0
S     notempty = 0
S     notfull = N
```

```
Append (element){
    P(notfull);
    P(m);
    CS
    V(m);
    V(notempty);
}
```

```
Remove(ele) {
    P(notempty)
    P(m);
    CS
    V(m);
    V(notfull);
```

② Reader–Writer Problem

```
int nreader = 0;
S wrt = 1;
S m = 1;
```

```
reader ( )
  P(m)
  nreader ++;
  if (nreader == 1)
     P(wrt)
  V(m)
  ---------
    dB
  ---------
  P(m)
  nreader --;
  if(nreader == 0)
     V (wrt)
  V(m);
```

```
Writer( );
 P(wrt)
 ---------
    dB
 ---------
 V(wrt)
```

② Dining Philosophers Problem

```
int state[N];
```

S S[N] = 0.

S m = 0;

```
#define Left (i-1)%N
#define Right (i+1)%N
philosopher(i){
    while( ){
        take_chopsticks(i)
        eat( );
        put_chopsticks(i)
        think( );
    }
}

take_chopsticks(i)
    P(m);
    state[i]=hungry;
    test(i)
    V(m);
    P(S[i])
```

```
Put_chopsticks(i)
  P(m);
  state[i]=think;
  test(left);
  test(Right);
  V(m)

test(i)
  if(state[i]==hungry && !state[left]==Eating
     && !state[right]==Eating){
    state[i] = eating;
    V(s[i]);
  }
```

Deadlock

-event: { wait for physical resource

| wait for condition to change
wait for critical section

- necessary conditions

① mutex

② hold and wait

③ No preemption

④ circular wait

- A knot: the reachability set of every node in the set is the set itself.

- Handling Deadlock

① Detect and recover

② Avoidance

③ Prevention: negating one of four necessary condition.