Operating Systems Practice

Project #3 – RAID4

Jeeyun Kim

(kimgyun@dgist.ac.kr)

Before we start…

- Clean-up your Git repository for this project
  - `git fetch && git reset --hard && git clean -fdx && git checkout raid-2024`
    - Your modifications will be deleted with this command!

- **For this project, make sure to perform `make clean` everytime!**

- Install ccache for faster compilation
  - apt install ccache

Project #3

- In the lecture, we learned about RAID…
  - RAID uses multiple disks in conjunction to build a faster, bigger, and more reliable disk system

- Objectives of this project
  - Understand how **RAID 1** system is performed in xv6 code (already implemented, provided!)
  - Implement **RAID 4** system to xv6 code

- Where to look and write code:
  - mkfs_2.c, bio.c, ide.c, param.h, (+etc)

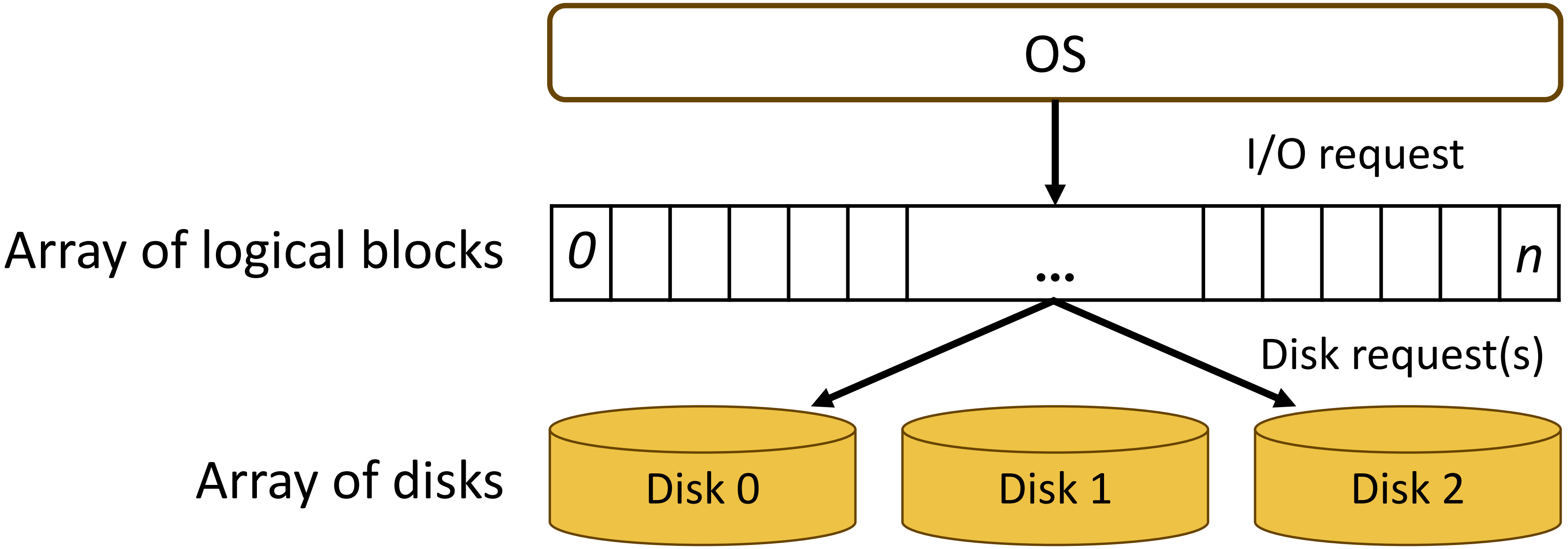
Block I/O

IDE 방식 HDD

SATA 방식 HDD

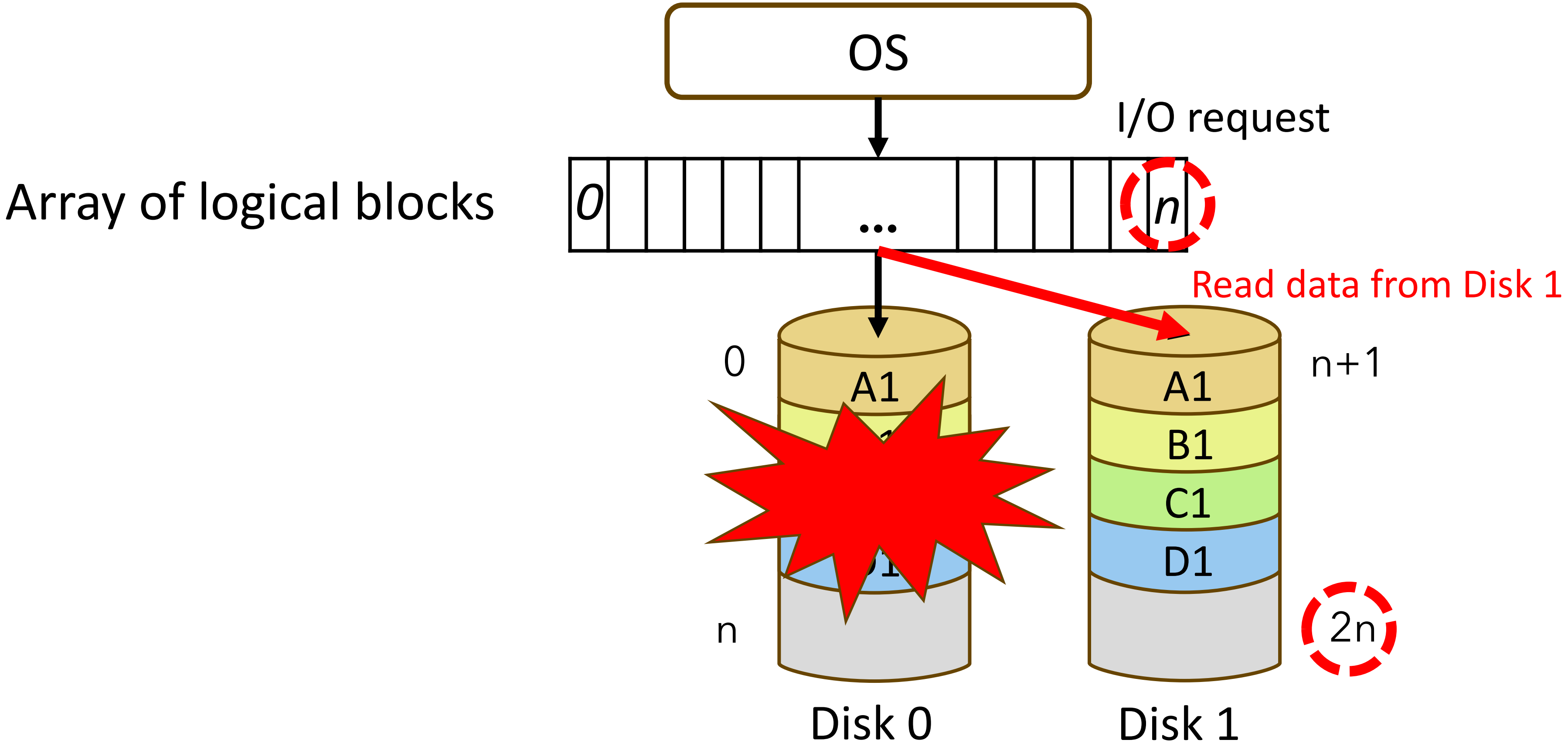
Overview – RAID

- RAID Abstraction & Interface
  - An array of disks are abstracted as a linear array of logical blocks
  - When an I/O request arrives, RAID calculates which disk to access and issues one or more physical I/Os
  - Advantage: Performance & Capacity, Reliability



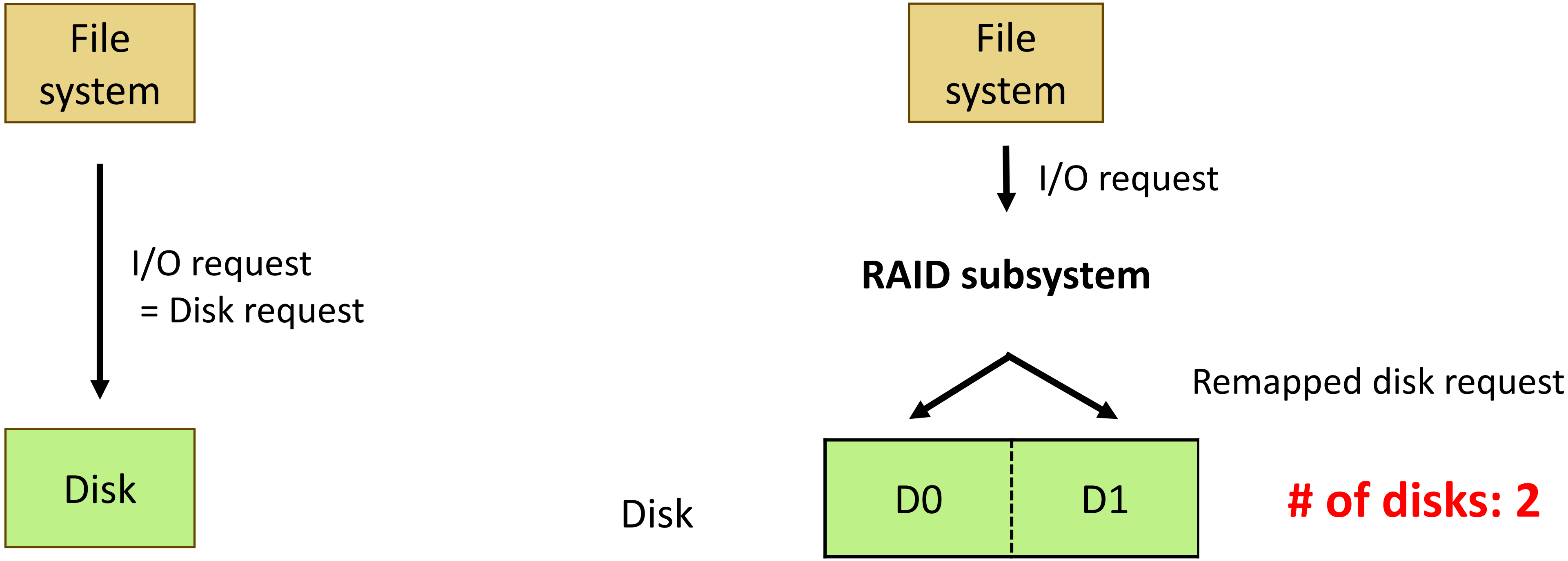
Overview – RAID 1

- RAID 1 consists of an exact copy (or mirror) of a set of data on two disks
- A classic RAID 1 mirrored pair contains two disks



Overview – RAID 1 in Xv6

- In our project, disks are not physically separated
  - We extend the size of disk to virtually implement the RAID 4
  - RAID 1: Disk size: 2×, usable file-system size: 1×
- **The finished RAID 1 implementation is provided**
  - We map the logical block number of each I/O request to the physical block number.



File system

I/O request
= Disk request

Disk

File system

I/O request

RAID subsystem

Remapped disk request

Disk

D0

D1

# of disks: 2

RAID in Practice

How does it “detect” failures?

- 3 popular choices
  - Linux MD RAID
    - RAID of multiple block devices

Device alert

Data checksum

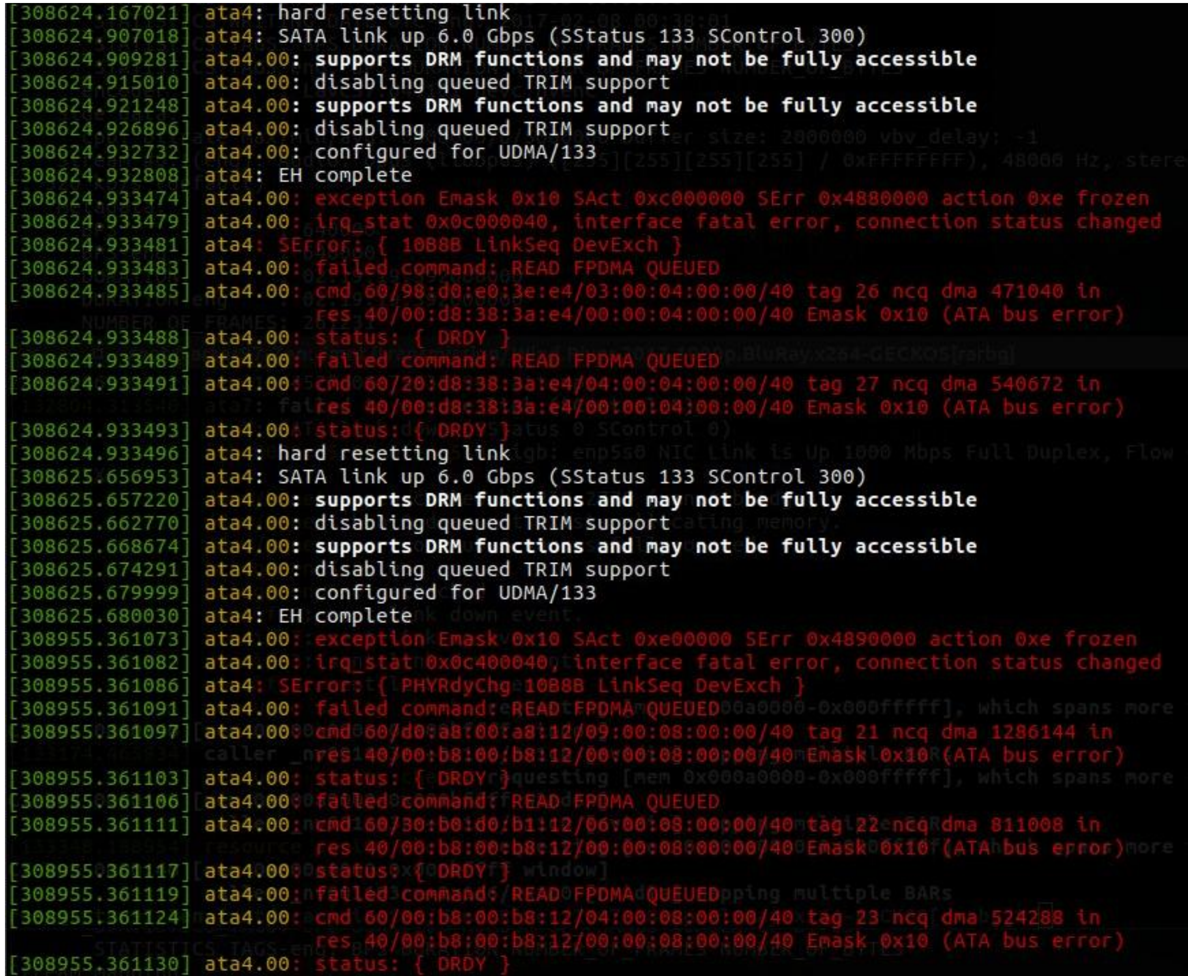
  - btrfs
    - Most feature-rich open-source CoW file-system in Linux
    - Infamous for its instability
    - Used in Synology NAS products
  - OpenZFS
    - Most feature-rich, open-source CoW file-system in Linux/BSD
    - Used in FreeNAS/TrueNAS
  - Bcachefs
    - “Next-generation” CoW file-system for Linux
    - Mainlined in Linux v6.7 (Jan 2024)

* Gross oversimplification

RAID in Practice - Device alert

- e.g., If too many errors on ata4 are accumulated, MD RAID disables ata4.
- Easy to implement

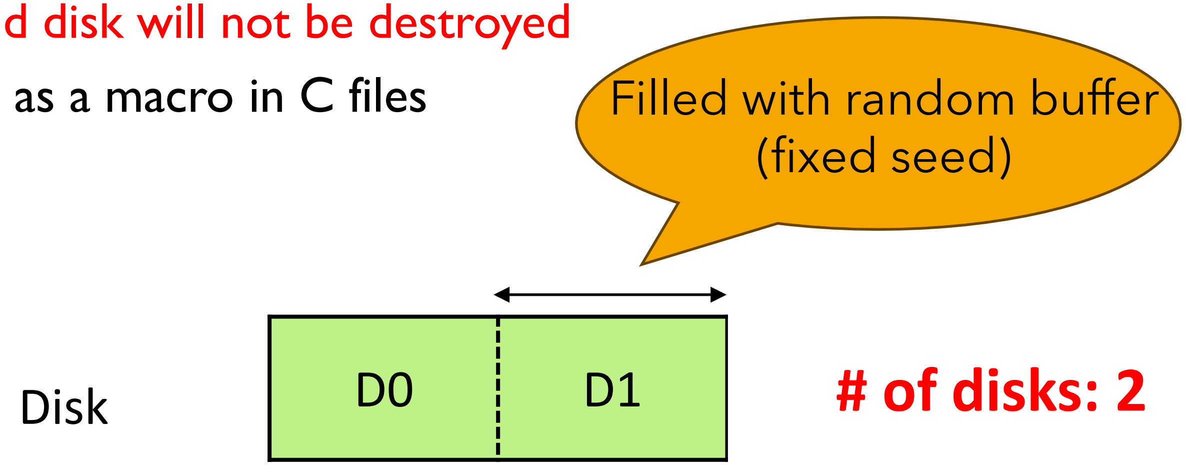
- We follow this approach!
  - `make qemu-nox BROKEN_DISK=1`



```
[308624.167021] ata4: hard resetting link
[308624.907018] ata4: SATA link up 6.0 Gbps (SStatus 133 SControl 300)
[308624.909281] ata4.00: supports DRM functions and may not be fully accessible
[308624.915010] ata4.00: disabling queued TRIM support
[308624.921248] ata4.00: supports DRM functions and may not be fully accessible
[308624.926896] ata4.00: disabling queued TRIM support
[308624.932732] ata4.00: configured for UDMA/133
[308624.932808] ata4: EH complete
[308624.933474] ata4.00: exception Emask 0x10 SAct 0xc000000 SErr 0x4880000 action 0xe frozen
[308624.933479] ata4.00: irq_stat 0x0c000040, interface fatal error, connection status changed
[308624.933481] ata4: SError: { 10B8B LinkSeq DevExch }
[308624.933483] ata4.00: failed command: READ FPDMA QUEUED
[308624.933485] ata4.00: cmd 60/98:d0:e0:3e:e4/03:00:04:00:00/40 tag 26 ncq dma 471040 in
                         res 40/00:d8:38:3a:e4/00:00:04:00:00/40 Emask 0x10 (ATA bus error)
[308624.933488] ata4.00: status: { DRDY }
[308624.933489] ata4.00: failed command: READ FPDMA QUEUED
[308624.933491] ata4.00: cmd 60/20:d8:38:3a:e4/04:00:04:00:00/40 tag 27 ncq dma 540672 in
                         res 40/00:d8:38:3a:e4/00:00:04:00:00/40 Emask 0x10 (ATA bus error)
[308624.933493] ata4.00: status: { DRDY }
[308624.933496] ata4: hard resetting link
[308625.656953] ata4: SATA link up 6.0 Gbps (SStatus 133 SControl 300)
[308625.657220] ata4.00: supports DRM functions and may not be fully accessible
[308625.662770] ata4.00: disabling queued TRIM support
[308625.668674] ata4.00: supports DRM functions and may not be fully accessible
[308625.674291] ata4.00: disabling queued TRIM support
[308625.679999] ata4.00: configured for UDMA/133
[308625.680030] ata4: EH complete
[308955.361073] ata4.00: exception Emask 0x10 SAct 0xe00000 SErr 0x4890000 action 0xe frozen
[308955.361082] ata4.00: irq_stat 0x0c400040, interface fatal error, connection status changed
[308955.361086] ata4: SError: { PHYRdyChg 10B8B LinkSeq DevExch }
[308955.361091] ata4.00: failed command: READ FPDMA QUEUED
[308955.361097] ata4.00: cmd 60/d0:a8:00:a8:12/09:00:08:00:00/40 tag 21 ncq dma 1286144 in
                         res 40/00:b8:00:b8:12/00:00:08:00:00/40 Emask 0x10 (ATA bus error)
[308955.361103] ata4.00: status: { DRDY }
[308955.361106] ata4.00: failed command: READ FPDMA QUEUED
[308955.361111] ata4.00: cmd 60/30:b0:d0:b1:12/06:00:08:00:00/40 tag 22 ncq dma 811008 in
                         res 40/00:b8:00:b8:12/00:00:08:00:00/40 Emask 0x10 (ATA bus error)
[308955.361117] ata4.00: status: { DRDY }
[308955.361119] ata4.00: failed command: READ FPDMA QUEUED
[308955.361124] ata4.00: cmd 60/00:b8:00:b8:12/04:00:08:00:00/40 tag 23 ncq dma 524288 in
                         res 40/00:b8:00:b8:12/00:00:08:00:00/40 Emask 0x10 (ATA bus error)
[308955.361130] ata4.00: status: { DRDY }
```

Testing

- `make qemu-nox BROKEN_DISK=1 SEED=100`
  - This will destroy D1 from fs.img using random buffer with seed of 100
  - If unspecified, -1 is used and disk will not be destroyed
  - BROKEN_DISK is defined as a macro in C files



Filled with random buffer (fixed seed)

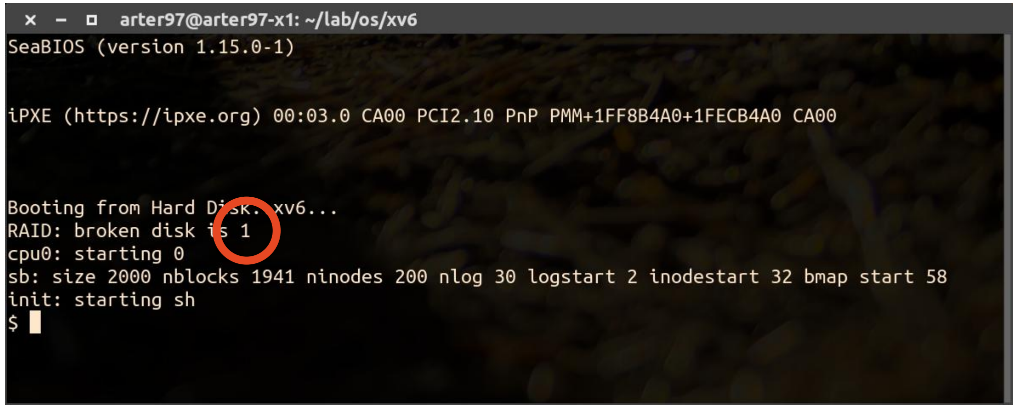
Disk

D0

D1

# of disks: 2

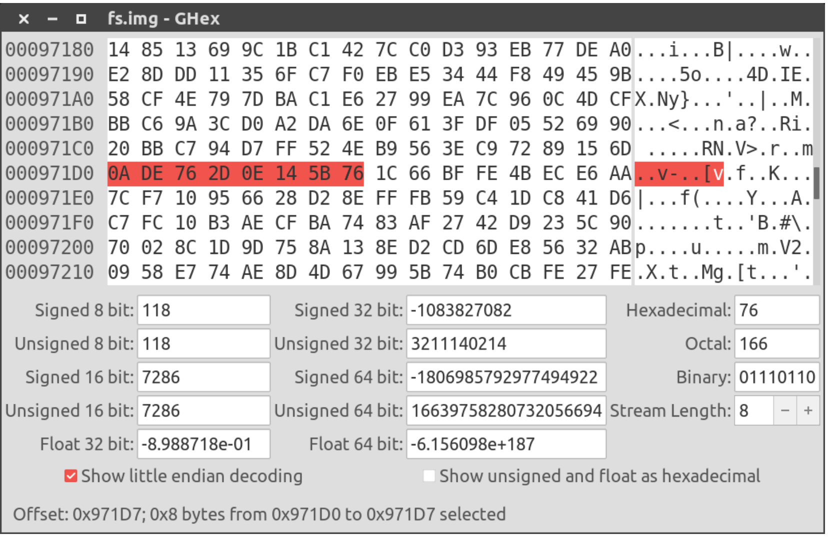
- BROKEN_DISK: 0,1,2
- SEED=anything
  - You may want to test with a fixed-seed to debug a specific issue
  - TAs will use seeds of our own for testing



```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
RAID: broken disk is 1
cpu0: starting 0
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ 
```

Testing - fs.img

- The Xv6 disk is fs.img from the host
- Feel free to dissect fs.img to debug issues
  - Recommended tools: dd, xxd, ghex



```
00097180 14 85 13 69 9C 1B C1 42 7C C0 D3 93 EB 77 DE A0...i...B|....w..
00097190 E2 8D DD 11 35 6F C7 F0 EB E5 34 44 F8 49 45 9B....5o....4D.IE.
000971A0 58 CF 4E 79 7D BA C1 E6 27 99 EA 7C 96 0C 4D CFX.Ny}...'..|..M.
000971B0 BB C6 9A 3C D0 A2 DA 6E 0F 61 3F DF 05 52 69 90...<...n.a?..Ri.
000971C0 20 BB C7 94 D7 FF 52 4E B9 56 3E C9 72 89 15 6D .....RN.V>.r..m
000971D0 0A DE 76 2D 0E 14 5B 76 1C 66 BF FE 4B EC E6 AA..v-..[v.f..K...
000971E0 7C F7 10 95 66 28 D2 8E FF FB 59 C4 1D C8 41 D6|...f(....Y...A.
000971F0 C7 FC 10 B3 AE CF BA 74 83 AF 27 42 D9 23 5C 90.......t..'B.#\.
00097200 70 02 8C 1D 9D 75 8A 13 8E D2 CD 6D E8 56 32 ABp....u.....m.V2.
00097210 09 58 E7 74 AE 8D 4D 67 99 5B 74 B0 CB FE 27 FE.X.t..Mg.[t...'.
```

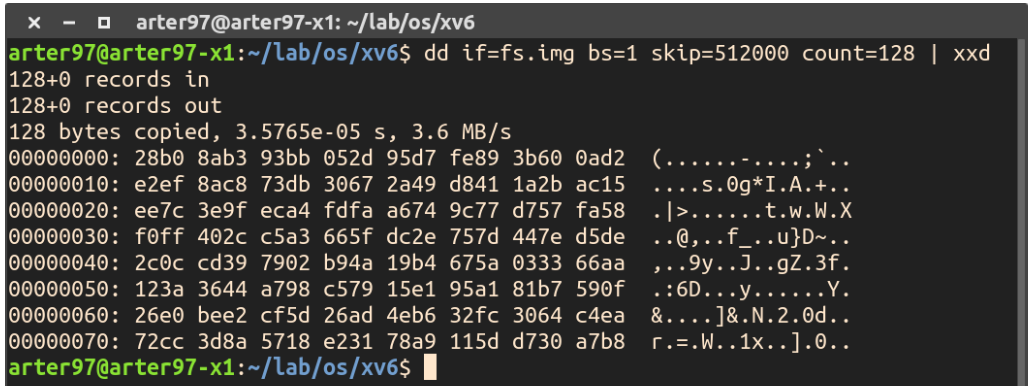
Signed 8 bit: 118
Unsigned 8 bit: 118
Signed 16 bit: 7286
Unsigned 16 bit: 7286
Float 32 bit: -8.988718e-01

Signed 32 bit: -1083827082
Unsigned 32 bit: 3211140214
Signed 64 bit: -1806985792977494922
Unsigned 64 bit: 16639758280732056694
Float 64 bit: -6.156098e+187

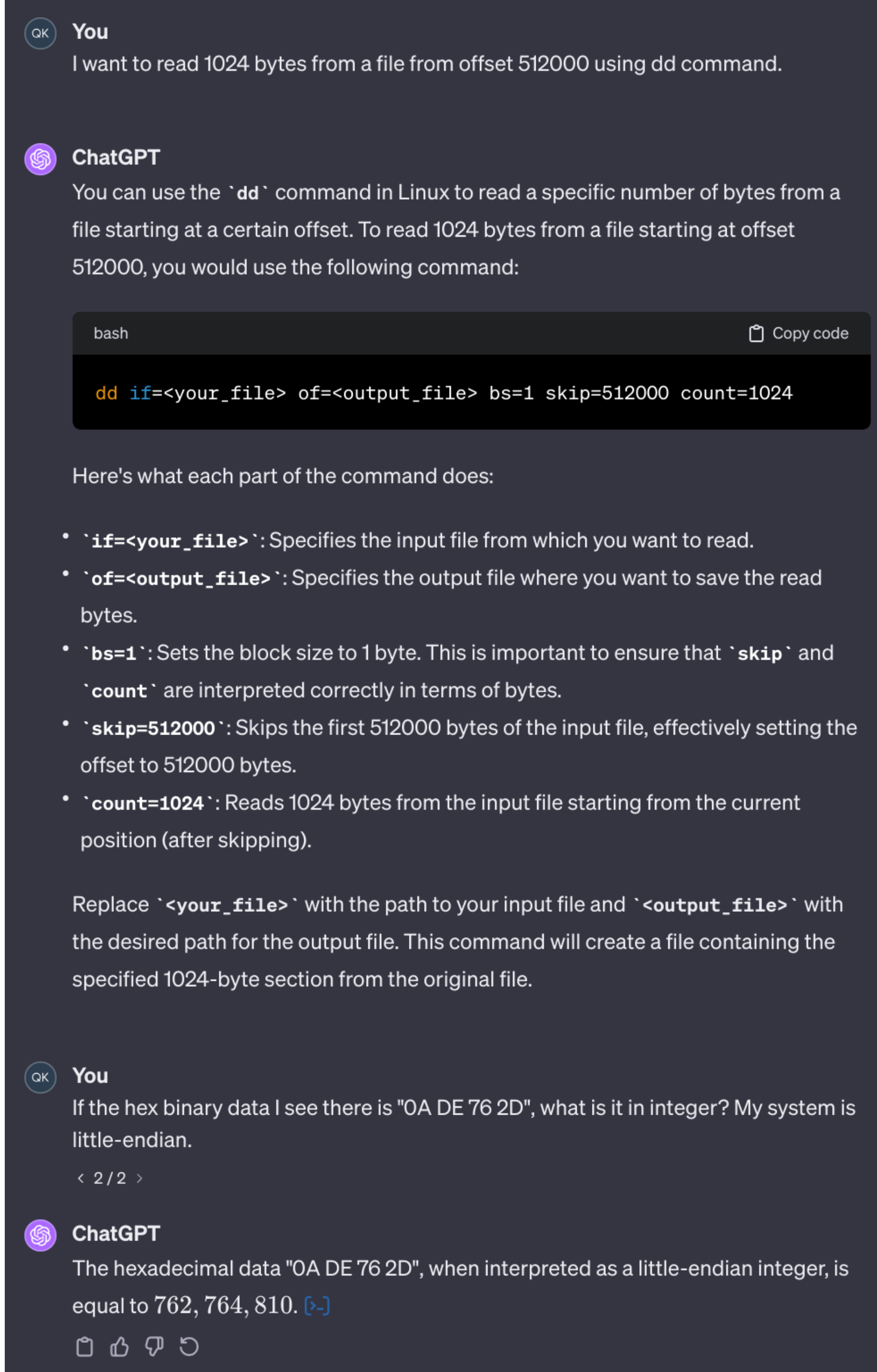
Hexadecimal: 76
Octal: 166
Binary: 01110110
Stream Length: 8

☑ Show little endian decoding
☐ Show unsigned and float as hexadecimal

Offset: 0x971D7; 0x8 bytes from 0x971D0 to 0x971D7 selected



```
arter97@arter97-x1:~/lab/os/xv6$ dd if=fs.img bs=1 skip=512000 count=128 | xxd
128+0 records in
128+0 records out
128 bytes copied, 3.5765e-05 s, 3.6 MB/s
00000000: 28b0 8ab3 93bb 052d 95d7 fe89 3b60 0ad2  (......-....;`..
00000010: e2ef 8ac8 73db 3067 2a49 d841 1a2b ac15  ....s.0g*I.A.+..
00000020: ee7c 3e9f eca4 fdfa a674 9c77 d757 fa58  .|>......t.w.W.X
00000030: f0ff 402c c5a3 665f dc2e 757d 447e d5de  ..@,..f_..u}D~..
00000040: 2c0c cd39 7902 b94a 19b4 675a 0333 66aa  ,..9y..J..gZ.3f.
00000050: 123a 3644 a798 c579 15e1 95a1 81b7 590f  .:6D...y......Y.
00000060: 26e0 bee2 cf5d 26ad 4eb6 32fc 3064 c4ea  &....]&.N.2.0d..
00000070: 72cc 3d8a 5718 e231 78a9 115d d730 a7b8  r.=.W..1x..].0..
arter97@arter97-x1:~/lab/os/xv6$
```



You

I want to read 1024 bytes from a file from offset 512000 using dd command.

ChatGPT

You can use the `dd` command in Linux to read a specific number of bytes from a file starting at a certain offset. To read 1024 bytes from a file starting at offset 512000, you would use the following command:

```bash
dd if=<your_file> of=<output_file> bs=1 skip=512000 count=1024
```

Here's what each part of the command does:

- `if=<your_file>`: Specifies the input file from which you want to read.
- `of=<output_file>`: Specifies the output file where you want to save the read bytes.
- `bs=1`: Sets the block size to 1 byte. This is important to ensure that `skip` and `count` are interpreted correctly in terms of bytes.
- `skip=512000`: Skips the first 512000 bytes of the input file, effectively setting the offset to 512000 bytes.
- `count=1024`: Reads 1024 bytes from the input file starting from the current position (after skipping).

Replace `<your_file>` with the path to your input file and `<output_file>` with the desired path for the output file. This command will create a file containing the specified 1024-byte section from the original file.

You

If the hex binary data I see there is "0A DE 76 2D", what is it in integer? My system is little-endian.

ChatGPT

The hexadecimal data "0A DE 76 2D", when interpreted as a little-endian integer, is equal to 762, 764, 810.

Xv6 – RAID 1

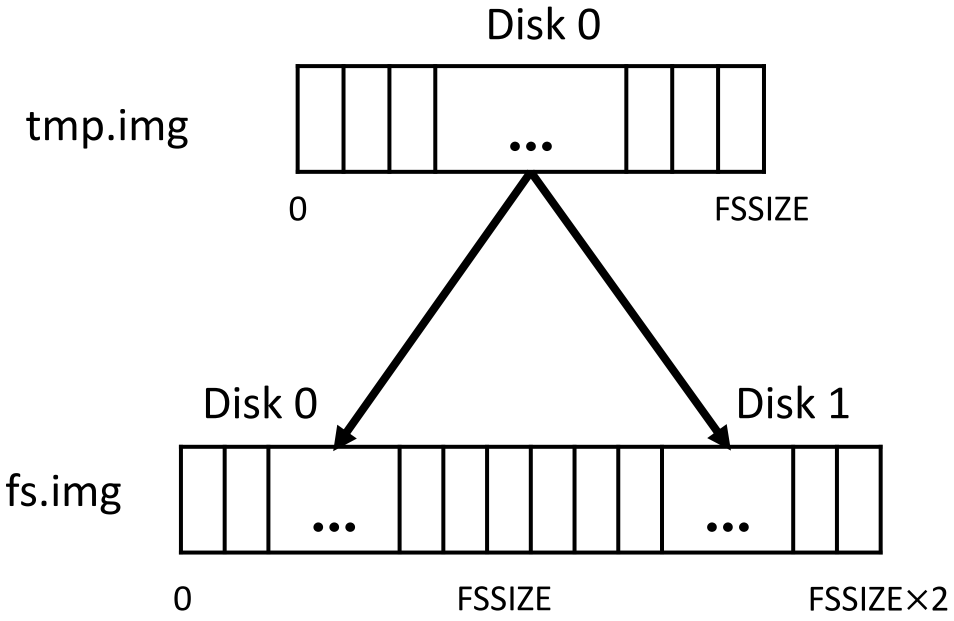
- You can change the file system size (the range of logical blocks) using the variable *FSSIZE* in *param.h*
  - Default : 1000
  - RAID 1 : 1000 (same as default)
- Also, you can set the range of physical blocks by changing *ide.c*
  - Default : *FSSIZE* (1000)
  - RAID 1 : *FSSIZE* × 2

In ide.c

```
static void
idestart(struct buf *b)
{
  …
  if(b->blockno >= FSSIZE*2)
    panic("incorrect blockno");
  …
}
```

mkfs_2.c

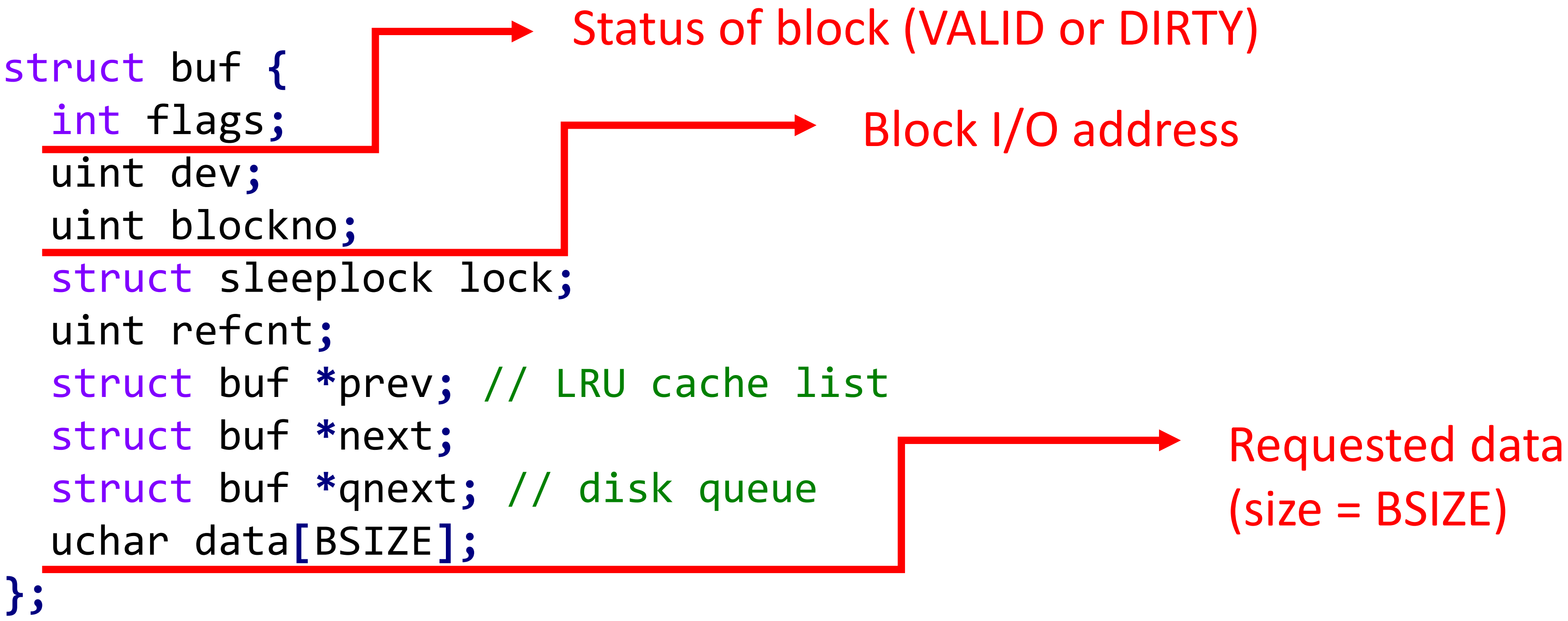
- Post-processor for mkfs
- Executed from the host, not Xv6
  - You can use C library functions, debugging techniques, etc
- The provided code **mirrors** existing fs.img and **doubles** it for RAID 1



- You should modify it for RAID 4

Xv6 – RAID 1

- Xv6 reads and writes the data to the disk using the disk buffer.
- Structure of the disk buffer



```c
struct buf {
  int flags;
  uint dev;
  uint blockno;
  struct sleeplock lock;
  uint refcnt;
  struct buf *prev; // LRU cache list
  struct buf *next;
  struct buf *qnext; // disk queue
  uchar data[BSIZE];
};
```

Status of block (VALID or DIRTY)

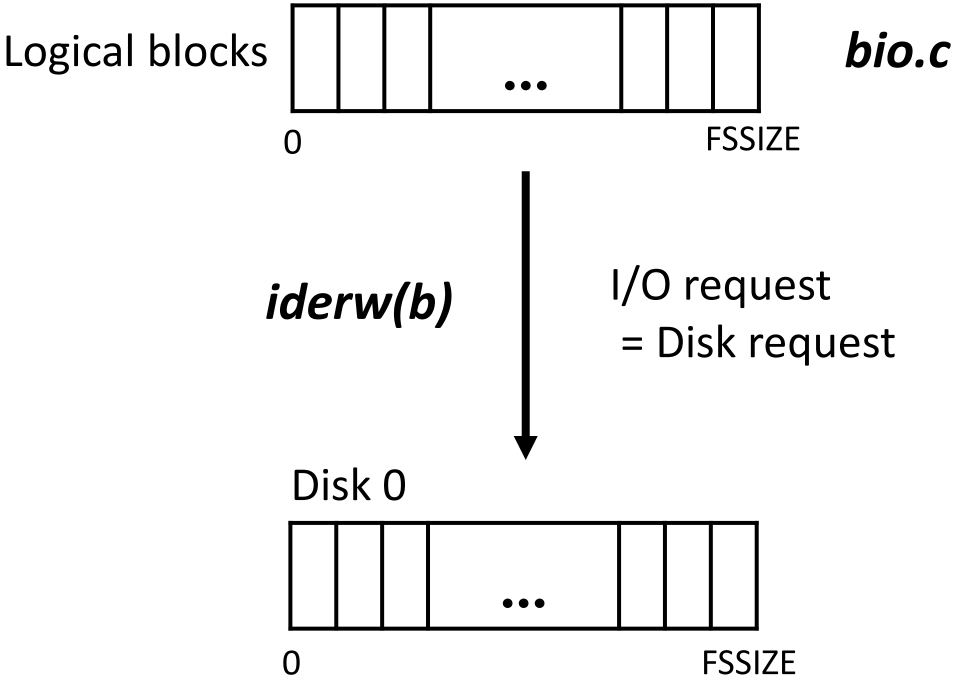
Block I/O address

Requested data
(size = BSIZE)

Defined in *buf.h*

Xv6 – Default

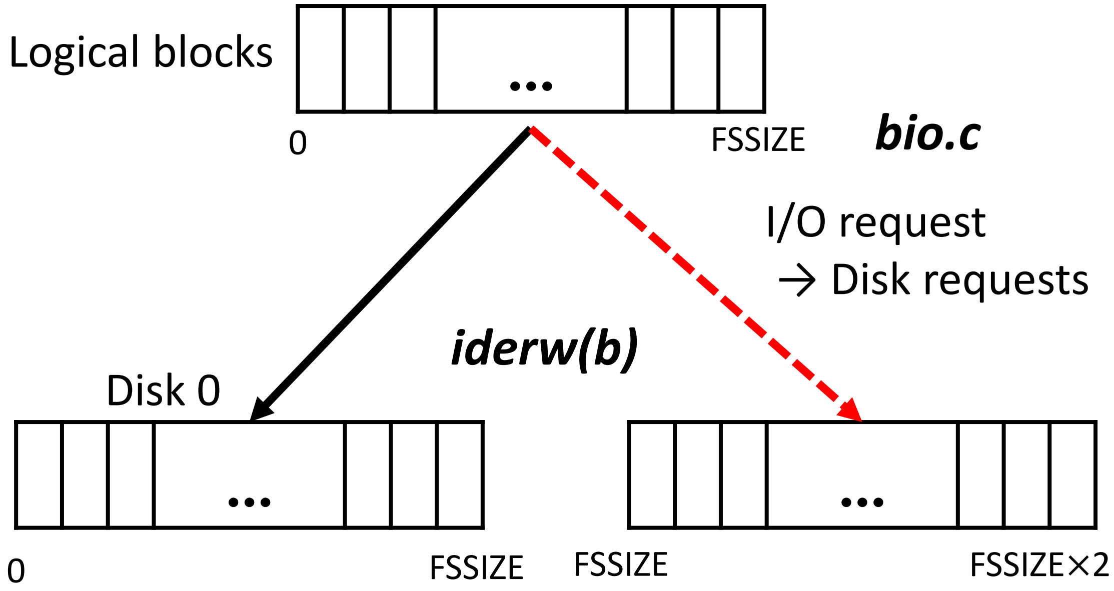
- The *bio.c* file contains block I/O functions such as *bread()*, *bwrite()* and *bget()*.
- By default, the address (*blockno*) is used directly by the device driver (*iderw()*).



```
struct buf*
bread(uint dev, uint blockno) {
  struct buf *b;
  b = bget(dev, blockno);
  if((b->flags & B_VALID) == 0) {
    iderw(b);
  }
  return b;
}

void
bwrite(struct buf *b) {
  …
  b->flags |= B_DIRTY;
  iderw(b);
}
```

Xv6 – RAID 1

- The RAID system is implemented **in the bio functions, before the iderw()** function
  - iderw() : disk read/write



```
struct buf*
bread(uint dev, uint blockno) {
  struct buf *b;
  b = bget(dev, blockno);
  if((b->flags & B_VALID) == 0) {
    iderw(b);
  }
  return b;
}

void
bwrite(struct buf *b) {
  …
  b->flags |= B_DIRTY;
  iderw(b);
}
```

Xv6 – RAID 1 : Read

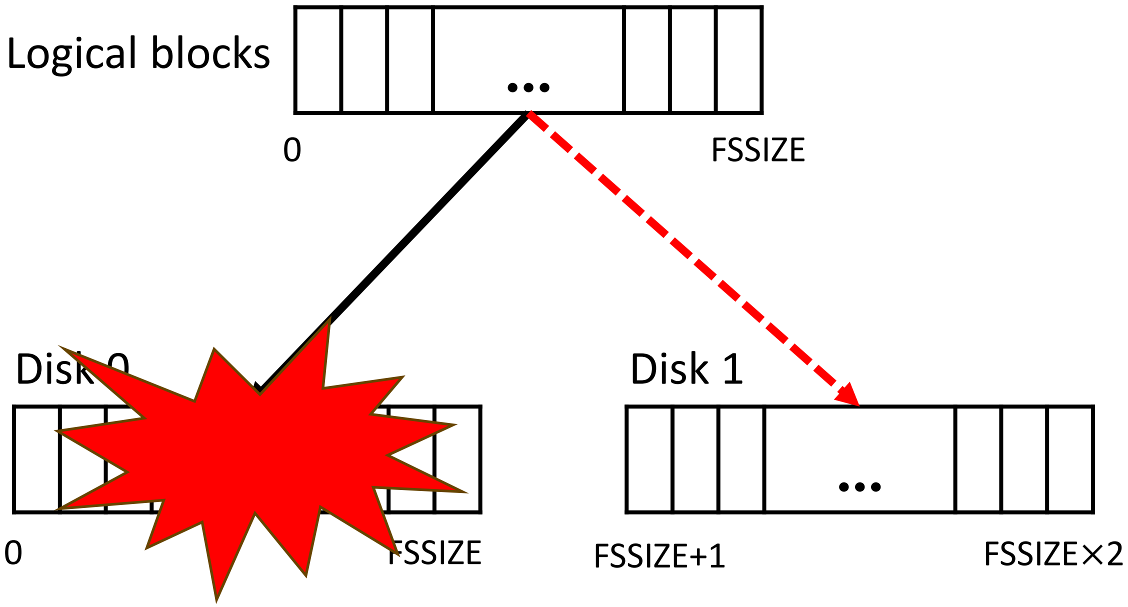
- The *bio.c* file
  - bread(uint dev, uint blockno)
  - *FSSIZE* = 1000
- *bget(uint dev, uint blockno)*
  - Look through the buffer cache for block
  - Return *struct buf*
  - *If cache hit:*
    - The *buf* contains the valid data
    - *buf->flags* set to *B_VALID*
  - *If cache miss:*
    - You have to read the data on the disk

```
#define BROKEN_DISK 0

bread(uint dev, uint blockno)
{
  struct buf *b, *b2;

  b = bget(dev, blockno);
  if (b->flags & B_VALID)
    return b;

  if (BROKEN_DISK == 0) {
    b2 = bget_direct(b->dev, b->blockno +
FSSIZE);
    iderw(b2); //read
    for (int i=0; i< BSIZE/sizeof(int); i++)
      b->udata[i] = b2->udata[i];
    brelse(b2);
    b->flags |= B_VALID;
  } else
    iderw(b);

  return b;
}
```

Cache hit

Xv6 – RAID 1 : Read

- *broken_disk*
  - The index of broken disk (default : -1)
- bget_direct(uint dev, uint blockno)
  - Does not check the cache
  - Just return the unused buffer



```
#define BROKEN_DISK -1

bread(uint dev, uint blockno)
{
  struct buf *b, *b2;

  b = bget(dev, blockno);
  if (b->flags & B_VALID)
    return b;

  if (BROKEN_DISK == 0) {
    b2 = bget_direct(b->dev, b->blockno +
FSSIZE);
    iderw(b2); //read
    for (int i=0; i< BSIZE/sizeof(int); i++)
      b->udata[i] = b2->udata[i];
    brelse(b2);
    b->flags |= B_VALID;
  } else
    iderw(b); //read

  return b;
}
```

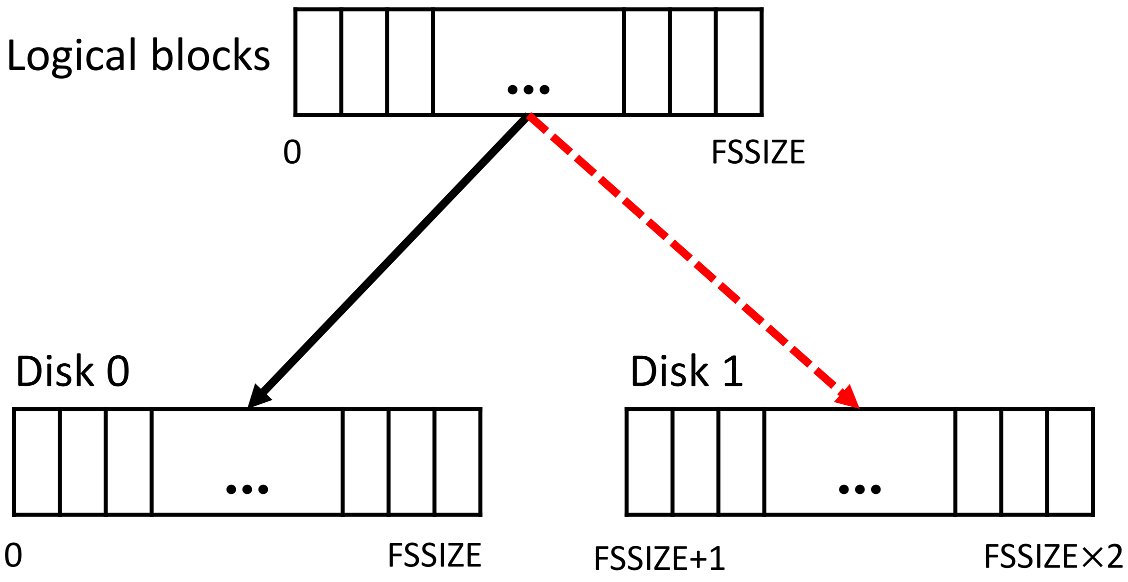
The index of broken disk

Cache hit

Disk 1

Xv6 – RAID 1 : Write

- The *bio.c* file
  - bwrite(struct buf *b)
- iderw(struct buf *b)
  - If b->flags & B_DIRTY : write data
  - Else : read data
  - Check the *ide.c* file



```
void
bwrite(struct buf *b)
{
  struct buf *b2;
  if(!holdingsleep(&b->lock))
    panic("bwrite");

  if (BROKEN_DISK != 1) {
    b2 = bget_direct(b->dev, b->blockno +
FSSIZE);

    for (int i=0; i<BSIZE/sizeof(int); i++)
      b2->udata[i] = b->udata[i];

    b2->flags |= B_DIRTY;
    iderw(b2); //write
    brelse(b2);
  }
  b->flags |= B_DIRTY;
  iderw(b); //write
}
```

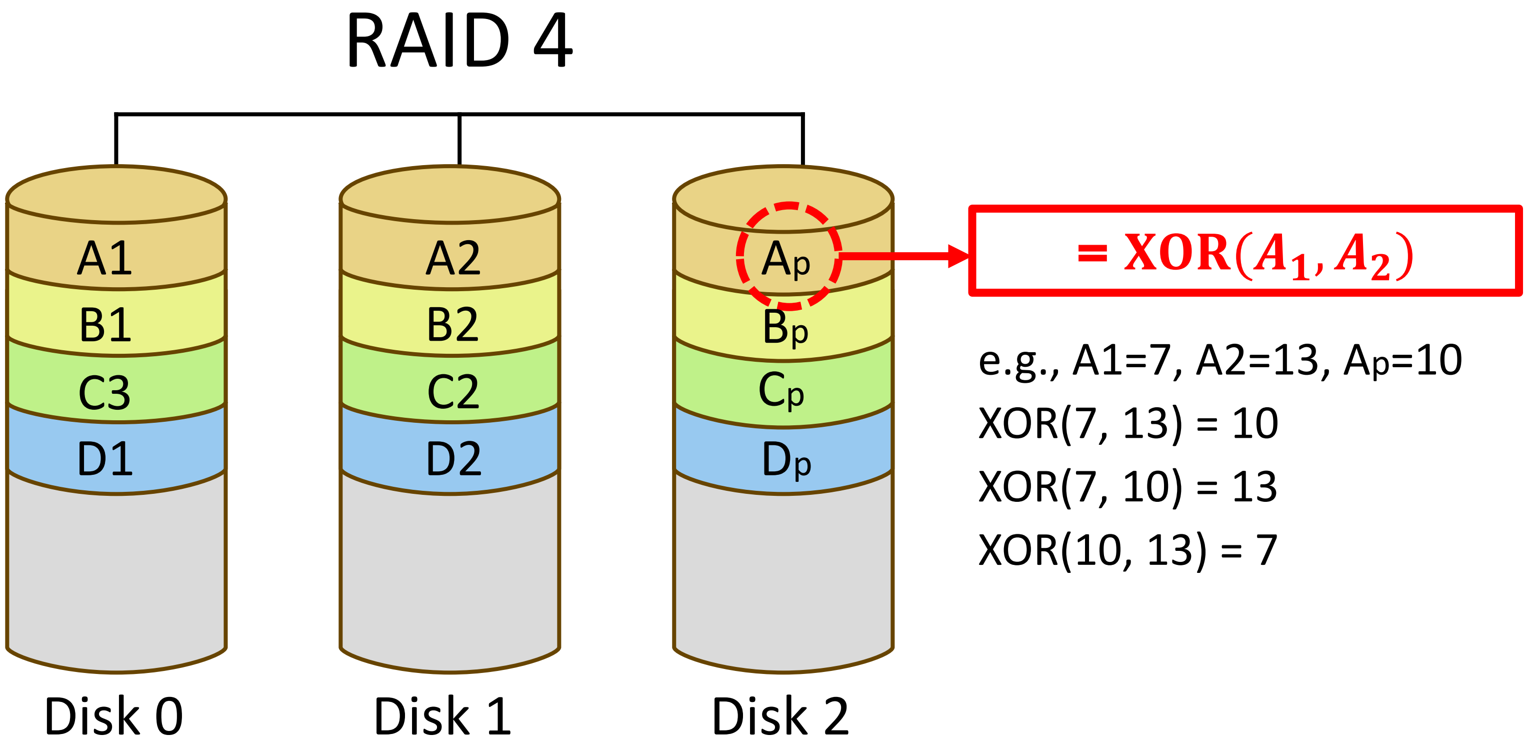
Disk 1

Xv6 – RAID 1

1. Change the range of logical blocks and physical blocks
   1. Range of logical blocks : *FSSIZE* in param.h
   2. Range of physical blocks : *idestart()* in ide.c
2. Recreate the file system image for the RAID system
   1. *main()* in mkfs_2.c
3. Modify the block I/O functions
   1. *bread()* and *bwrite()* in bio.c
4. **We already implemented RAID 1, passing every steps**
5. Test the RAID 1 system
   1. make clean; make qemu-nox BROKEN_DISK=[-1,0,1] SEED=*
   2. Just type "usertests" in xv6 console

Overview – RAID 4

- RAID 4 consists of block-level striping with parity block
  - Parity block : store the data that can recover the broken disk
  - Calculate parity block data using the XOR operation
- Parity information is dedicated in one drive

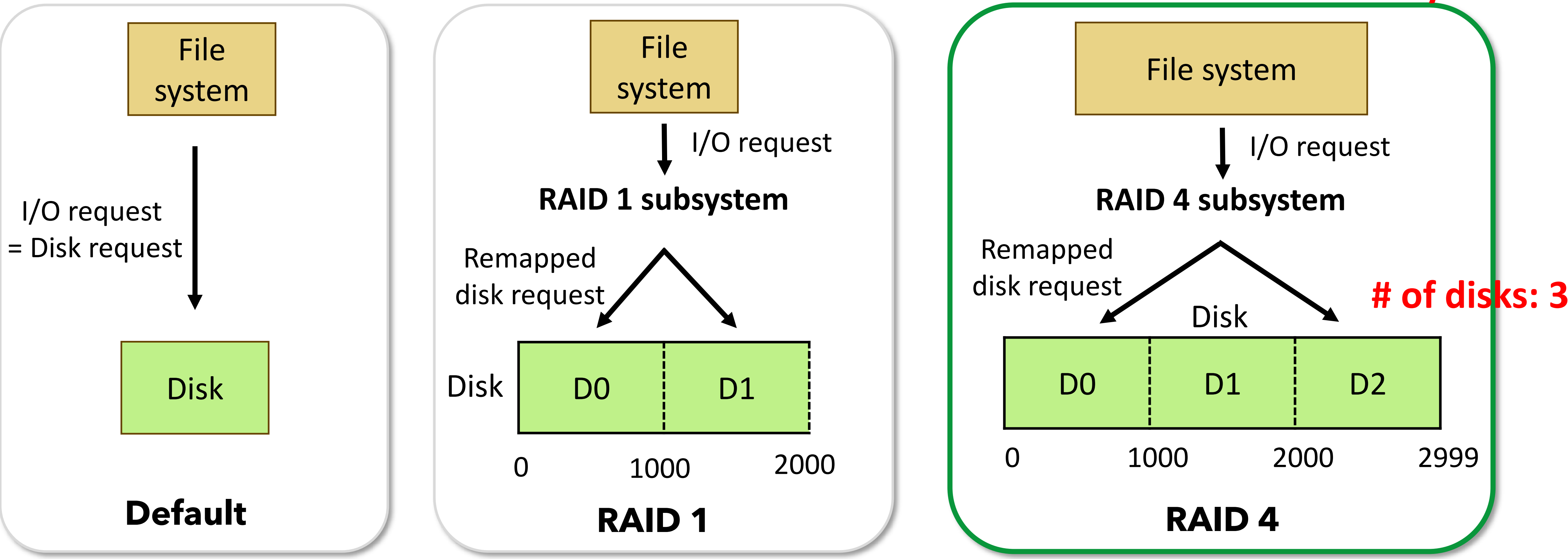


$= \mathbf{XOR}(A_1, A_2)$

e.g., A1=7, A2=13, Ap=10
XOR(7, 13) = 10
XOR(7, 10) = 13
XOR(10, 13) = 7

1. RAID 4

- We extend the size of disk to virtually implement the RAID 4.
  - Disk size: 3× (0 - 3000), usable file-system size: 2× (0 - 2000)
- Modify the ***param.h*** and ***ide.c*** for accepting new enlarged size



Doubled file system

File system

I/O request = Disk request

Disk

Default

File system

I/O request

RAID 1 subsystem

Remapped disk request

Disk

D0 D1

0 1000 2000

RAID 1

File system

I/O request

RAID 4 subsystem

Remapped disk request

Disk

D0 D1 D2

0 1000 2000 2999

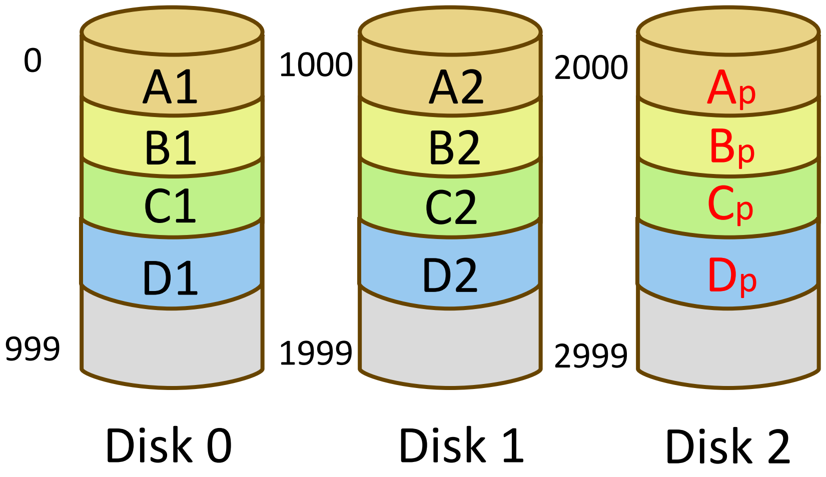
# of disks: 3

RAID 4

2. RAID 4 : MKFS

- The ***mkfs_2.c*** file
  - Read tmp.img file
  - Calculate the parity data
  - Calculate the location of parity blocks
  - Write blocks to fs.img

```
int main(int argc, char *argv[])
{
        ...
        //TODO implement here
        int rdfd = open(argv[1], O_RDONLY);
        int wrfd = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, 0644);
        ftruncate(wrfd, (FSSIZE * 2) * BSIZE);
        for (int blockno = 0; blockno < FSSIZE; blockno++) {
                …
        }
        close(rdfd);
        close(wrfd);
}
```



- **The range of block index of each disk** is specified
  - Disk 0 (0 – 999), Disk 1(1000 – 1999), Disk 2 (2000 – 2999)
- The parity block has to be **located in Disk 2**

3. RAID 4 : Read

- The ***bio.c*** file
  - bread(uint dev, uint blockno)
- If **the disk of the target block is broken**, you need to recover the data
  - Check the BROKEN_DISK value
  - Read parity block and raid block
  - Calculate the recovered data of the target block
    - Using XOR operation
  - Copy the recovered data to the target block buffer, then return the buffer

```
#define BROKEN_DISK 0

bread(uint dev, uint blockno)
{
  struct buf *b, *b2;

  b = bget(dev, blockno);
  if (b->flags & B_VALID)
    return b;
 //TODO implement here
  if (BROKEN_DISK == 0) {
    b2 = bget_direct(b->dev, b->blockno + FSSIZE);
    iderw(b2); //read
    for (int i=0; i< BSIZE/sizeof(int); i++)
      b->udata[i] = b2->udata[i];
    brelse(b2);
    b->flags |= B_VALID;
  } else
    iderw(b);

  return b;
}
```

4. RAID 4 : Write

- The ***bio.c*** file
  - bwrite(struct buf *b)
- You need to calculate the new parity blocks every time you write the blocks
  - Read raid block and calculate the parity block
- If there is a **broken disk**, you need to recover the block before you change the parity block data
  - Check the broken_disk value
  - Read the raid block and the parity block
  - **Recover the broken block**
  - Calculate the parity block

```
void
bwrite(struct buf *b)
{
  ...
  //TODO implement here
  if (BROKEN_DISK != 1) {
    b2 = bget_direct(b->dev, b->blockno + FSSIZE);

    for (int i=0; i<BSIZE/sizeof(int); i++)
      b2->udata[i] = b->udata[i];

    b2->flags |= B_DIRTY;
    iderw(b2); //write
    brelse(b2);
  }
  b->flags |= B_DIRTY;
  iderw(b); //write
}
```

Testing

- **To evaluate your implementation, you should perform usertests**
- You should capture the result of usertests
  - Just type "usertests" in xv6 console

- Your implementation should pass all possible breakage from RAID 4
  - make clean; make qemu-nox BROKEN_DISK=[-1,0,1,2] SEED=*
  - See "Testing" slide
  - You have to pass the whole usertests even when a disk is broken

- After you pass the usertests, we will check your implementations to verify that the parity blocks are located correctly

Summary

1. **Change the range of logical blocks and physical blocks**
    1. Range of logical blocks : *FSSIZE* in param.h
    2. Range of physical blocks : *idestart()* in ide.c
2. **Recreate the file system image for the RAID system**
    1. *main()* in mkfs_2.c
3. **Modify the block I/O functions**
    1. *bread()* and *bwrite()* in bio.c

- If needed, you can add functions
- **To evaluate your implementation, you should perform usertests**
- Fill in the code wherever it contains *//TODO*

Project #3 – RAID 4

- Deadline
  - ~ 2024.12.19 (Thu) 23:59
- Hand-in procedure
  - p3_202212345.patch
    - Run the following command and upload p3_202212345.patch
      - git diff > p3_202212345.patch
    - Check the patch file with Notepad and confirm your modifications are in the patch file
  - p3_202212345.pdf/docx
    - Submit a 1~3 pages report
      - Free format (Korean/English)
      - Description of your implementation in detail
        - Changed functions, added functions, etc
      - Include screenshot of usertests succeeding

Finally …

Do NOT hesitate to ask questions!

Mini Project #1, #2	Juhyung Park	*arter97@dgist.ac.kr*
Project #1	Minjae Kim	*jwya2149@dgist.ac.kr*
Project #2	Seonggyun Oh	*sungkyun123@dgist.ac.kr*
Project #3	**Jeeyun Kim**	***kimgyun@dgist.ac.kr***