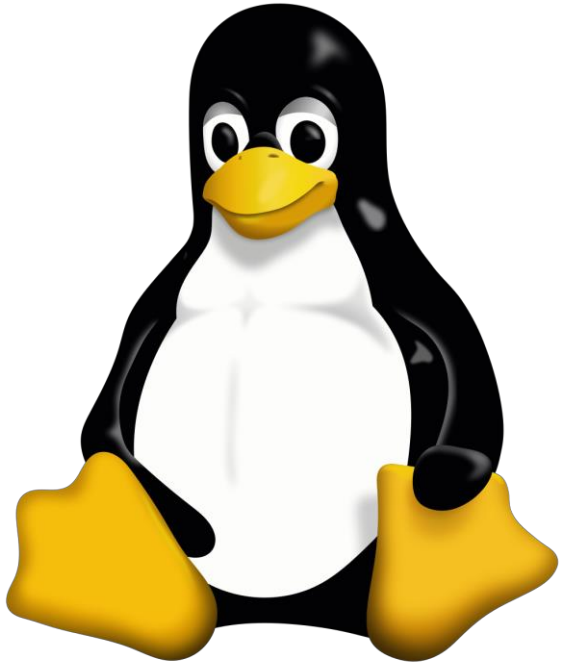
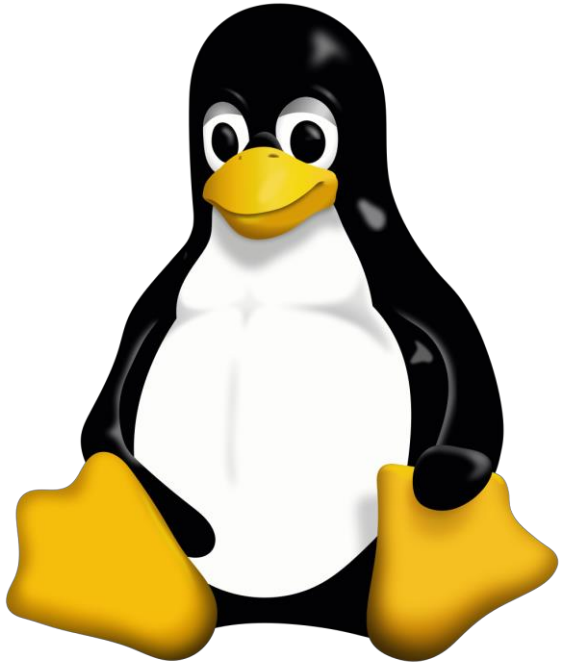


Linux Device Driver

Sunbeam Infotech



Linux Character Device Driver

Sunbeam Infotech

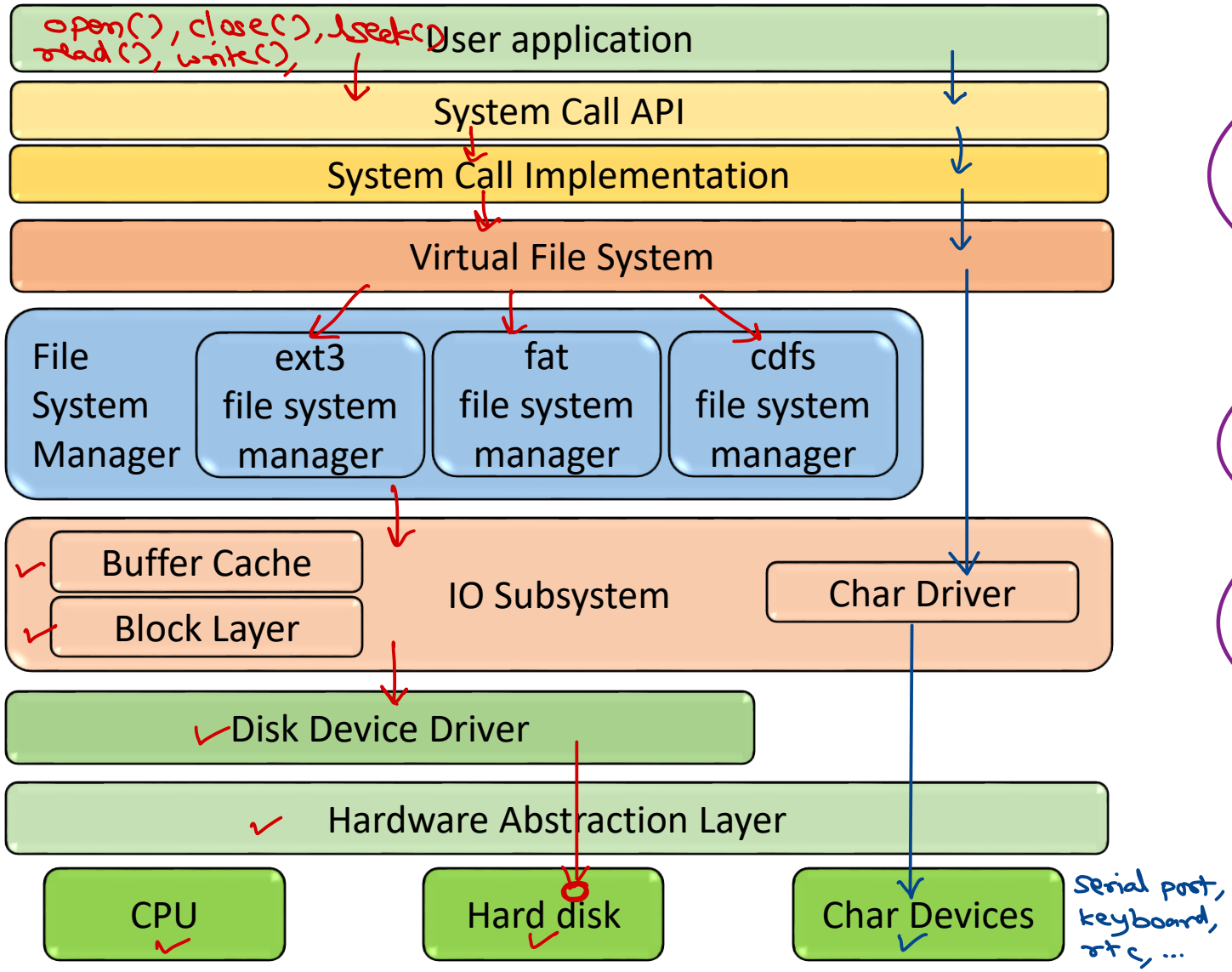
File

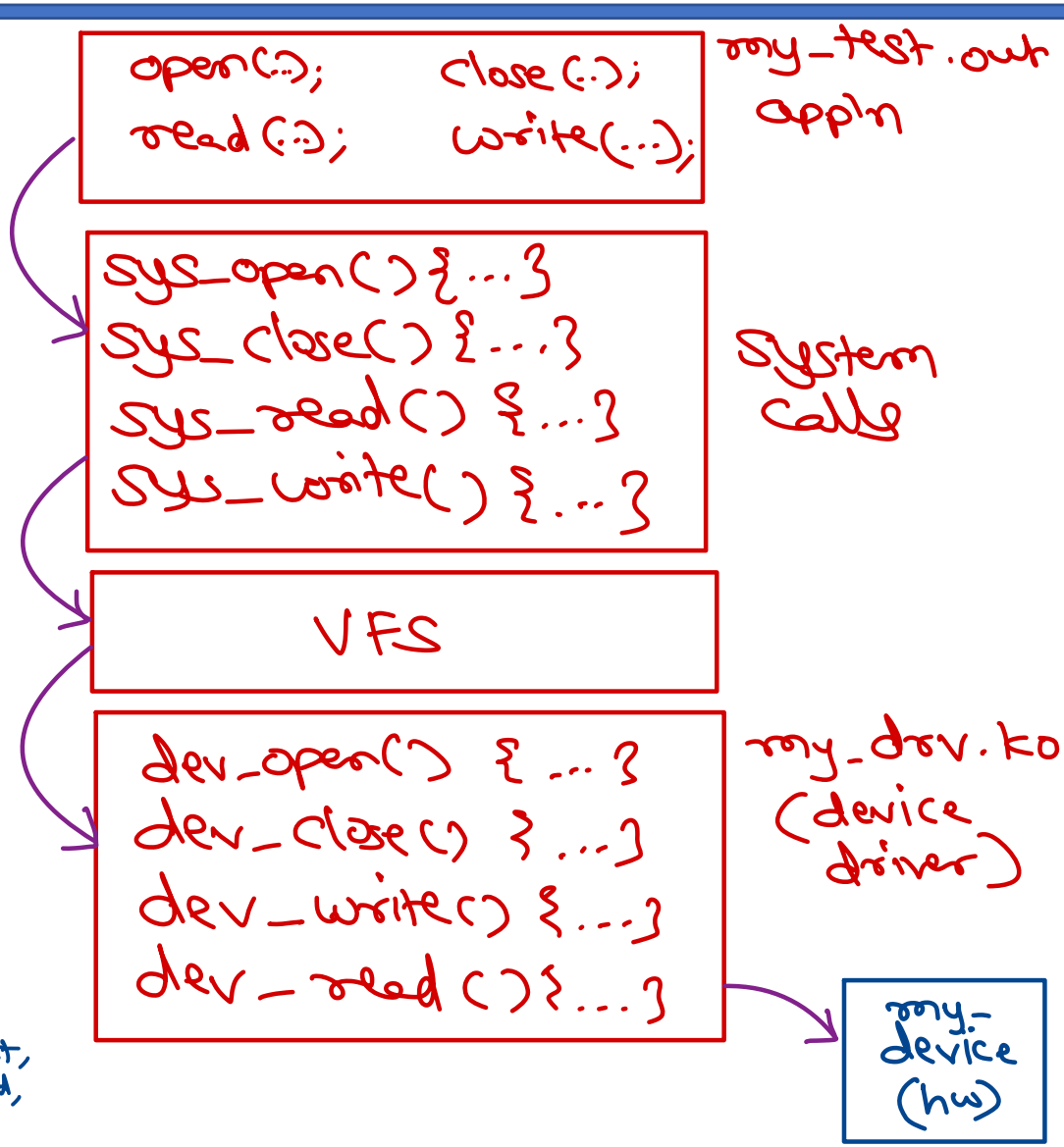
- File is collection of data and information on storage device.
- File = Data (Contents) + Metadata (Information)
- File metadata is stored in inode (FCB).
  - Type, Mode, Size, User & Group, Links.
  - Timestamps, Info about data blocks.
- File data is stored in data block(s).
- File types: regular, directory, link, pipe, socket, character and block.
- Directory file contains directory entries for each sub-directory and file in it. Each dentry contains inode number & data block.

File system

- File system is way of organizing files on the disk.
- File systems have
  - Boot block
  - Super block
  - Inode list
  - Data blocks
- Content/arrangement of these components will change FS to FS.
- Different FS use different algorithms/ data structures to allocate disk.
- File system layout & operations are handled by file system manager using IO subsystem and device driver.

File System architecture





File IO System Calls

- File system operations deals with metadata (inode and dentry).
  - stat(), link(), symlink(), unlink(), mkdir(), …
- File IO operations deals with data/contents.
  - open(), close(), read(), write(), lseek(), ioctl(), …
- fd = open("filepath", flags, mode);
  - Open or create a file.
- close(fd);
  - Close opened file.
- ret = read(fd, buf, nbytes);
  - Read from file into user buffer.
- ret = write(fd, buf, nbytes);
  - Write from file into user buffer.
- newpos = lseek(fd, offset, whence);
  - Change file current position.

fd = open("filepath", flags, mode);

- Convert path name into inode number.
- Load inode into in-memory inode table.
- Make an entry into open file table (OFT).
- Add pointer to OFT into open file descriptor table (OFDT).
- Return index of OFDT to the program, known as file descriptor (FD).

- struct inode

- struct file

ret = read(fd, buf, nbytes);

- Get current file position from OFT entry.
- Calculate file logical block number and block byte offset.
- Convert file logical block to disk block using data blocks information in inode.
- Check if disk block is available in buffer cache.
- In not available, instruct disk driver to load the block from disk.
- Disk driver initiate disk IO and block the current process.
- Read desired part of block from buffer cache into user buffer. Update file position into OFT.
- Repeat for subsequent file blocks, until desired number of bytes are read.
- Return number of bytes allocated.

ret = write(fd, buf, nbytes);

- Get current file position from OFT entry.
- Calculate file logical block number and block byte offset.
- Convert file logical block to disk block using data blocks information in inode.
- Check if disk block is available in buffer cache.
- In not available, instruct disk driver to load the block from disk.
- Disk driver initiate disk IO and block the current process.
- Overwrite on intended part of block in buffer cache (from user buffer). Update file position into OFT. Mark the block in buffer cache as dirty, so that it will be flushed on the disk.
- Repeat for subsequent file blocks, until desired number of bytes are written.
- Return number of bytes allocated.

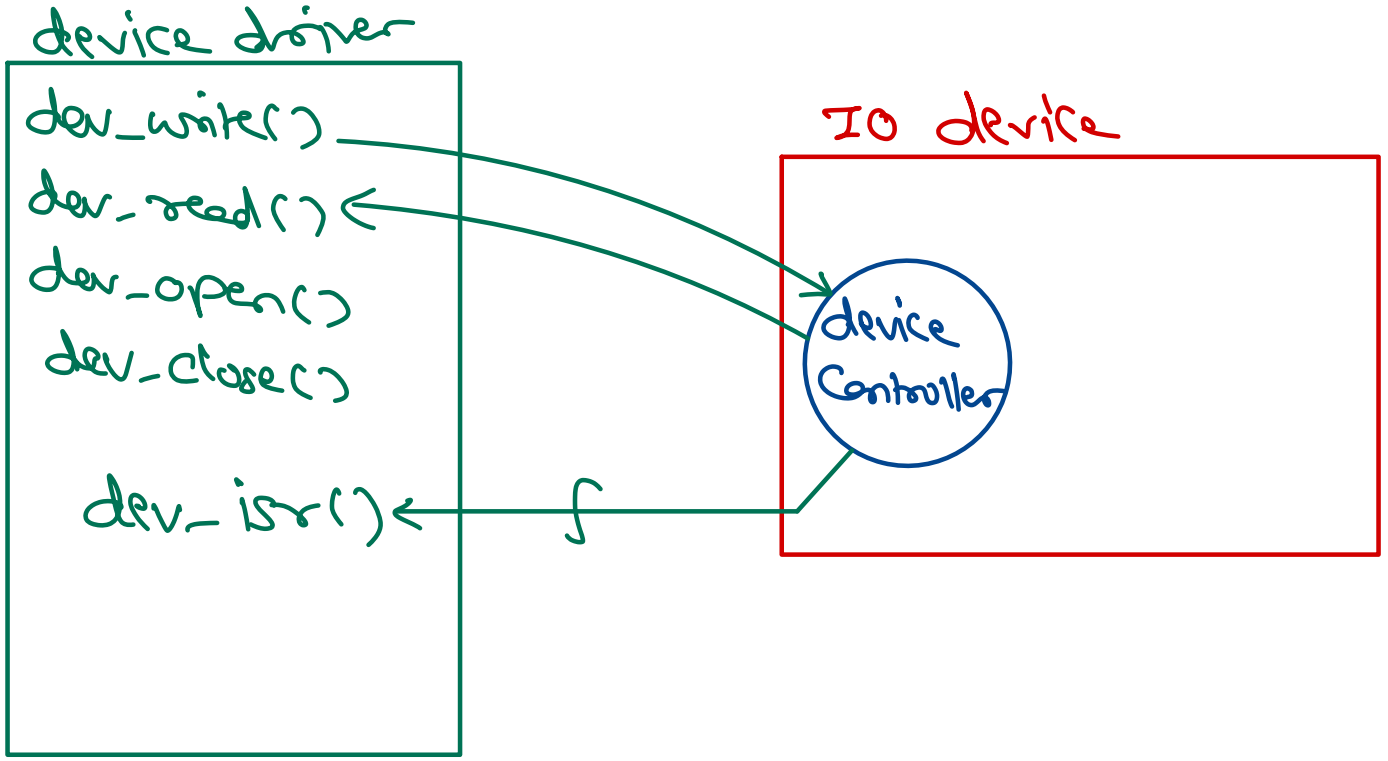
close(fd) and newpos = lseek(fd, offset, whence);

- close(fd);
  - Decrement reference count in OFT entry.
  - If count is zero, release the file/resources.

- newpos = lseek(fd, offset, whence);
  - Get current file position from OFT entry.
  - Calculate new file position.
  - Update file position in OFT entry.

Linux device drivers

- Device driver is a kernel module that instructs device controllers to perform the operations and also handles interrupts generated from it.



Linux device drivers

- Character device drivers → also called as "raw" devices.
  - Char devices transfer data in byte by byte manner. So device drivers are implemented to read/write data as stream of bytes. They support four major operations i.e. open(), close(), read() and write(). Example: Serial port, parallel port, keyboard, tty, etc.
- Block device drivers
  - Block devices transfer data as bunch of bytes i.e. block by block. Size of block is typically 512 Bytes. Support major operations open(), close(), read(), write() and lseek(). Example: All mass storage devices.
- Network device drivers
  - Network drivers are responsible for packets transmit and receive, however network protocols are implemented up in network stack. Unlike character and block devices network device entry is not done under /dev.

Overview of character device driver

- Char device driver is a kernel module.
- Driver initialization includes
  - Allocate device number
  - Create device class & file
  - Init cdev object & add it.
- Driver de-initialization includes
  - Release cdev
  - Destroy device file & ~~close~~ class
  - Release device number
- Implement minimal device operations
  - open() and release()
  - read() and write()

Pseudo character device driver

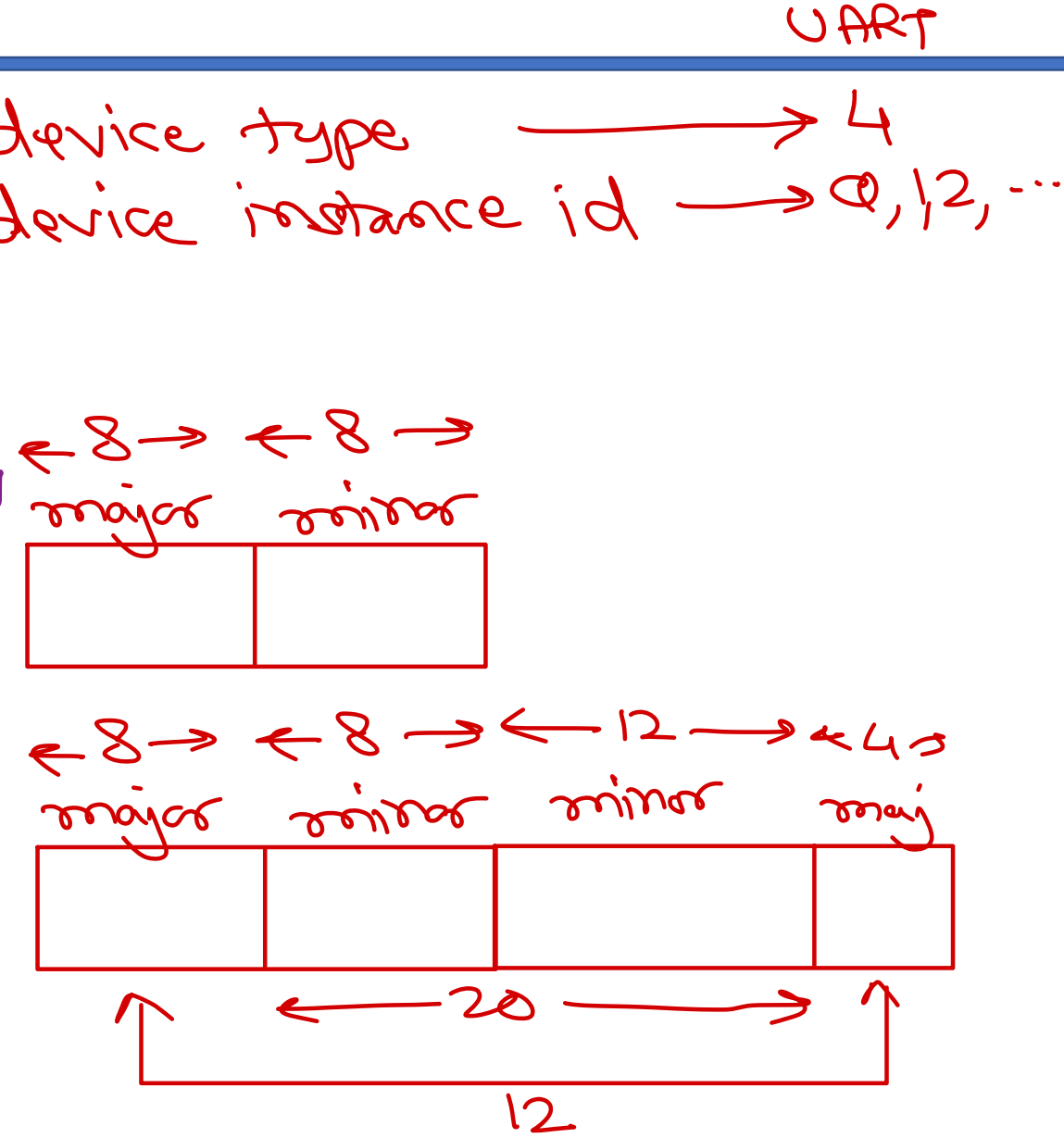
- A memory buffer is treated as device.

char buf[32];

- open() operation – do nothing.
- release() operation – do nothing.
- write() operation
  - Take data from user buffer and write into device (memory) buffer.
  - Update current file position.
  - Return number of bytes successfully written.
- read() operation
  - Take data from device (memory) buffer and write to user buffer.
  - Update current file position.
  - Return number of bytes successfully read.

Device numbers

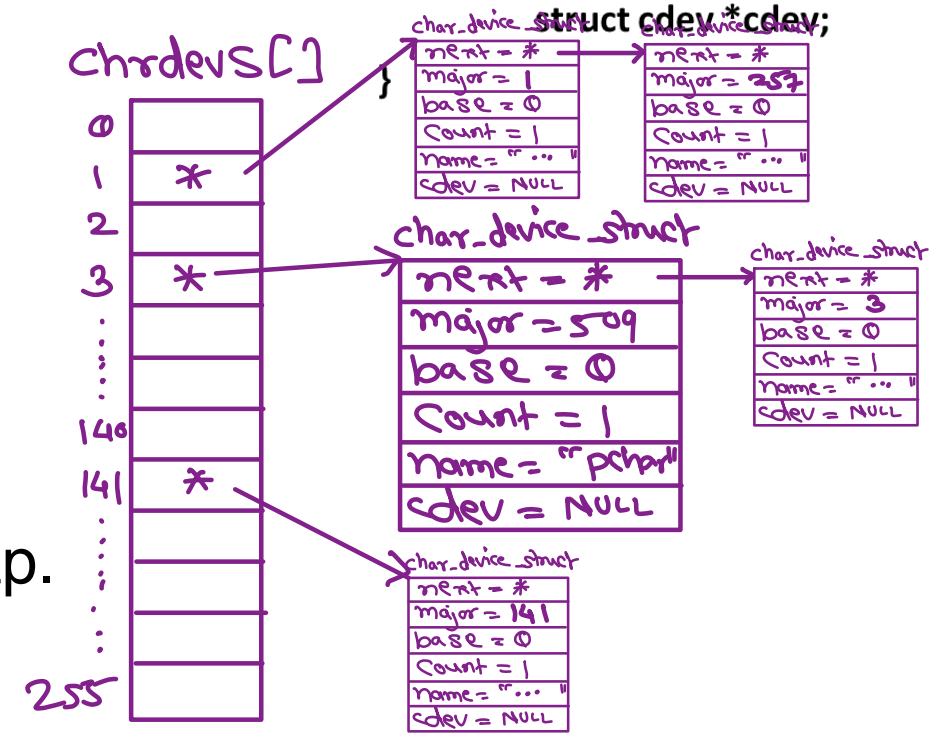
- Each device is uniquely identified by a major number and minor number.
- (Kernel < 2.6): 16 bit device number
  - 8-bit major + 8-bit minor.
- (Kernel >= 2.6): 32 bit device number
  - 12-bit major + 20-bit minor.
  - 32 bit = 8 bit + ~~12~~ 20 bit + ~~12~~ 4 bit
- Device number represented as dev_t.
  - MKDEV(major, minor)
  - MAJOR(devno)
  - MINOR(devno)



Register character device

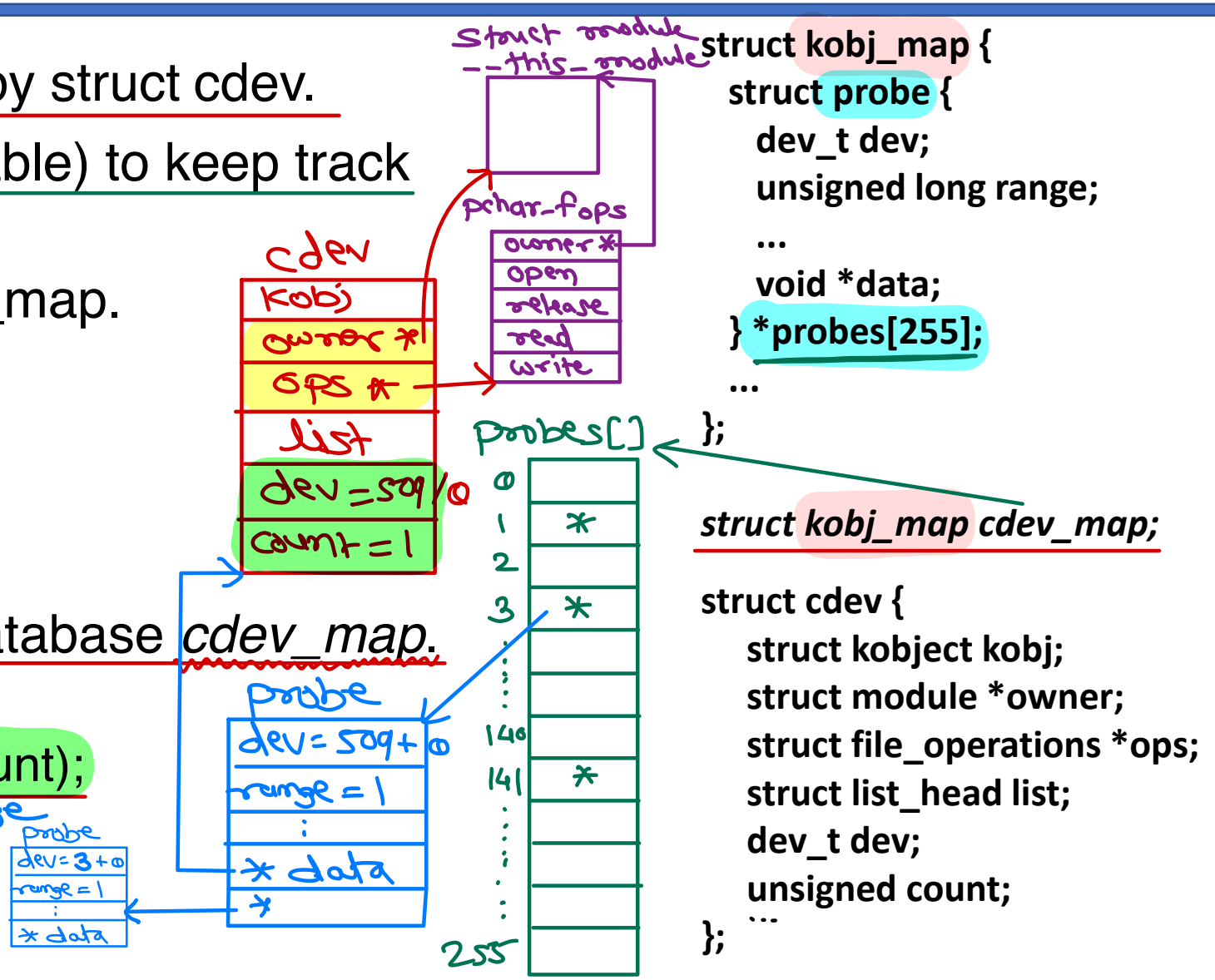
- (1) Register/allocate character device number.
  - register_chrdev_region(devno, count, “name”);
  - alloc_chrdev_region(dev, baseminor, count, “name”);
- Makes entry into
  - struct char_device_struct *chrdevs[…];
  - hashed by major number of device.
- Allocated device numbers can be seen under /proc/devices.
- Allocated device number can be released using
  - unregister_chrdev_region (devno, count);
- Char device should also be added into cdev_map.

```
struct char_device_struct {
    struct char_device_struct *next;
    unsigned major;
    unsigned baseminor;
    int minorct;
    char name[...];
    struct cdev *cdev;
};
```



Register character device

- Each char device is represented by struct cdev.
- *cdev_map* is global array (hash table) to keep track of all char devices.
- cdev_map is object of struct kobj_map.
  - key = device major number
  - hash function = major % 255
  - value = struct probe
    - void *data = struct cdev
- (2) Add device into char device database *cdev_map*.
  - cdev_init(&cdev, &fops);
  - cdev_add(&cdev, devno, dev_count);
- Added device can be removed.
  - cdev_del(&cdev);



Device class and device file

- Char device added in kernel space should be accessible from user space.
- Traditionally device is created using *mknod*. → older kernels
  - mknod /dev/devname c major minor
- In newer kernel device is accessible from *sysfs* and *devfs*.
- Device class is created under /sys/class → pchar_class
  - pclass = class_create(module, “class_name”);
- Device file is create under /sys/devices/ virtual/class_name/devname and /dev/devname → pchar0
  - pdev = device_create(pclass, NULL /*parent*/, devno, NULL /*drvdata*/, “devname“, …);
- Device class and device file are destroyed using.
  - device_destroy(pclass, devno);
  - class_destroy(pclass);

→ inode is created and dentry is created } devfs ↓ /dev

Char device operations

- Char device driver provide char device operations and register with the system.
- Driver device operations can be do one of the following:
  - Overwrite default function
  - Provide additional functionality
  - Minimal placeholder
  - NULL – use default/no implementation
- These operations are specified as callback functions in struct file_operations and associated with struct cdev. All functions have pre-defined prototype and purpose.
  - int (*open)(struct inode *, struct file *);
  - int (*release)(struct inode *,struct file *);
  - ssize_t (*write)(struct file *, const char __user *, size_t, loff_t *);
  - ssize_t (*read)(struct file *, char __user *, size_t, loff_t *);
  - loff_t (*llseek)(struct file *, loff_t, int);
  - long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
  - …

char device operations
struct file_operations
* 27 operations (*f)()
↳ 4 ops - common

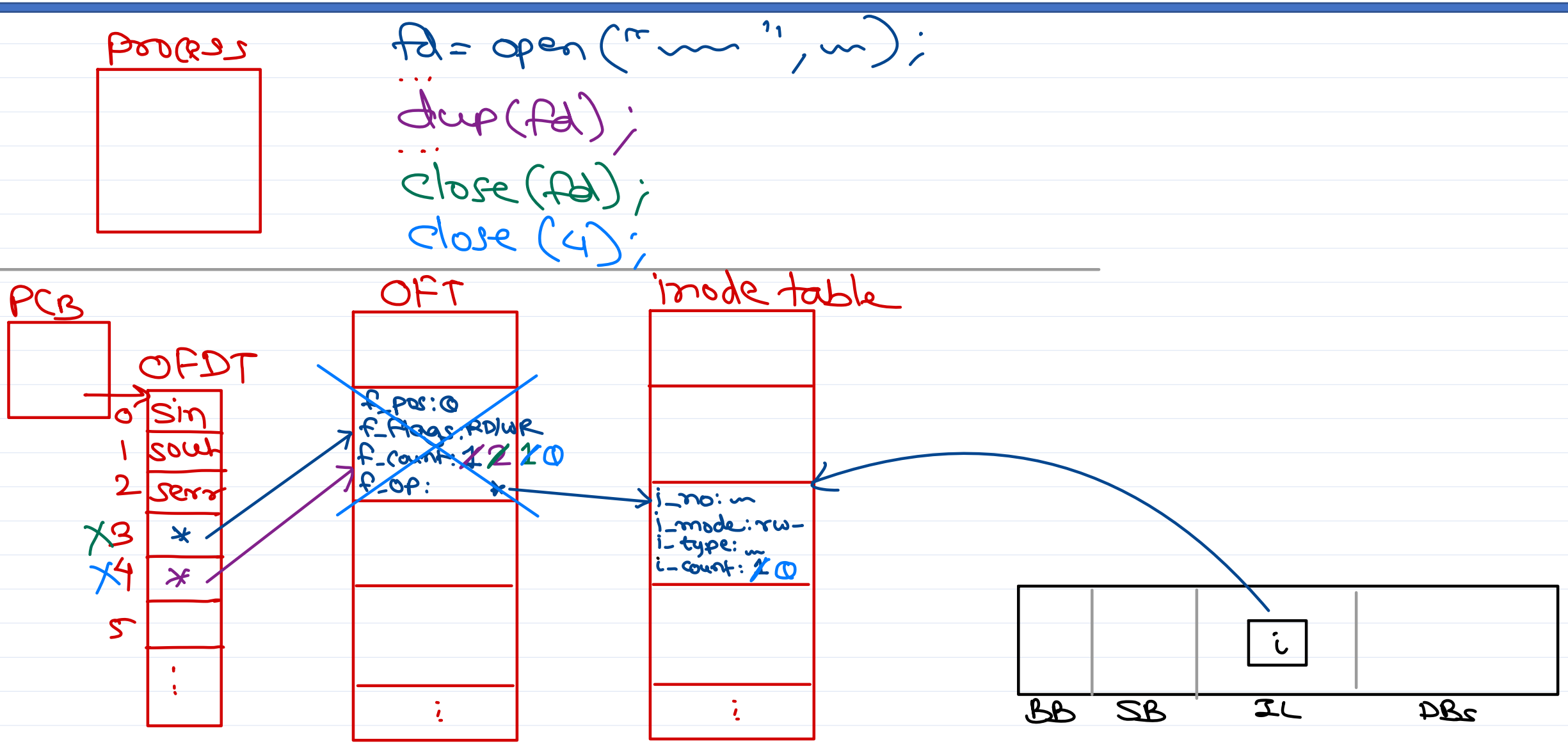
open() operation

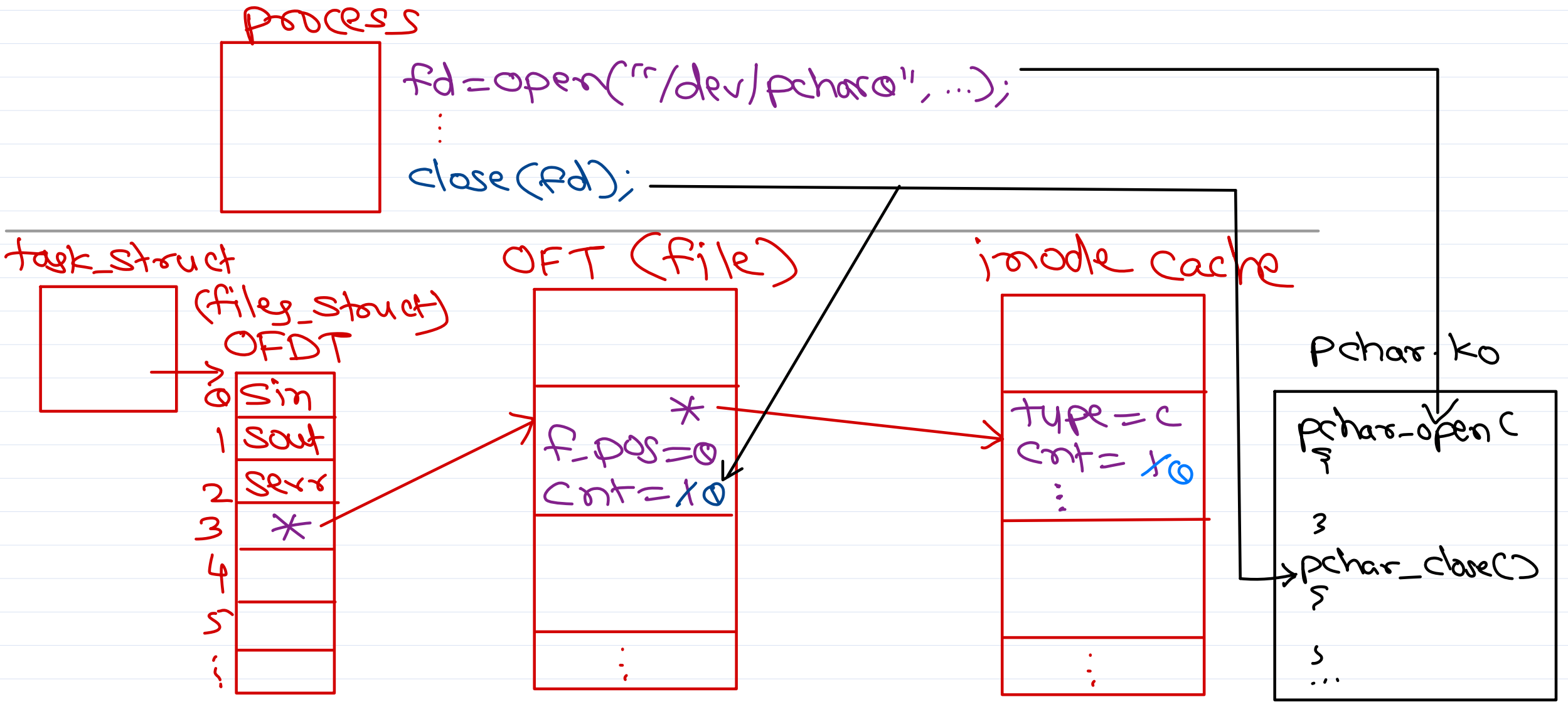
- Every driver/file system must have this function directly or indirectly.
- This opens a device and makes it ready for IO.
- The typical implementation contains initialization of hardware device, registering its IRQs, allocating internal buffers, etc.
- If unimplemented, kernel always give default function and never fails. Neither it notifies driver about this.
- In pseudo char device driver, there is no physical device involved and hence this function is kept empty.

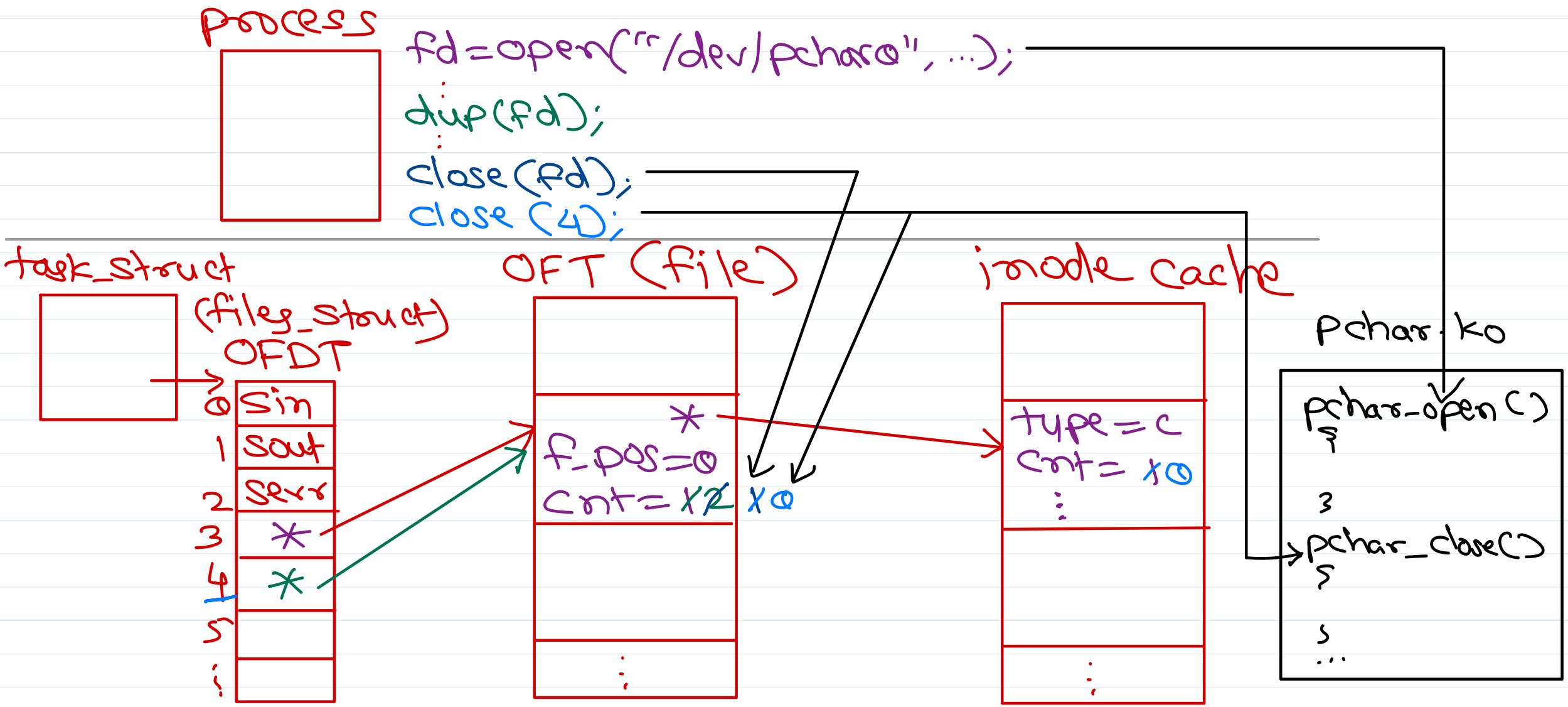
release() operation

- Closes the device, decrement reference count & releases memory synchronously.
- The typical implementation release all the resources allocated in open() operation.
- If unimplemented, these things are done by the kernel in its default function.
- Not every close() syscall causes release() operation to be invoked
  - When dup() or fork() is called, no new struct file is created; rather only increments reference count in it.
  - Each call to close() decrements the reference count.
  - Only if count drops to 0, the release() operation is invoked.
  - This ensures that only one release() is called per open().
- In pseudo char device driver, there is no physical device involved and hence this function is kept empty.

Reference counting

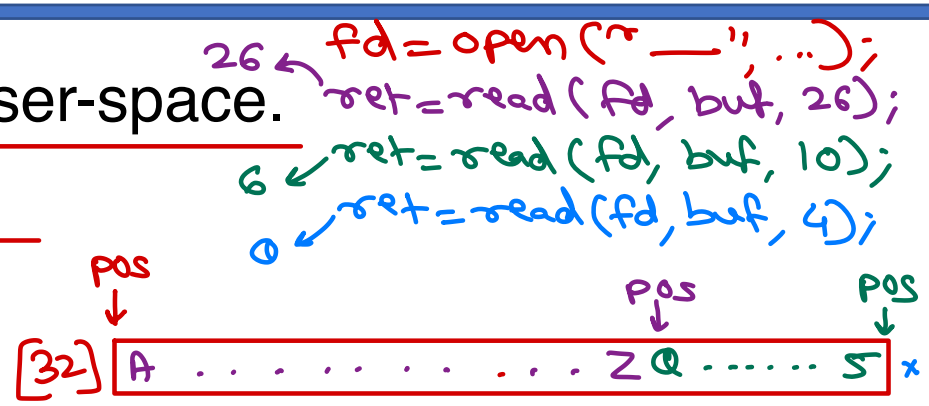



process
fd=open("/dev/pchar0", ...);
close(fd);
task_struct
(files_struct)
OFDT
0 Sin
1 Sout
2 Serr
3
4
5
OFT (file)
f_pos=0
cnt=1
inode cache
type=c
cnt=1
pchar.ko
pchar_open()
3
pchar_close()
5


process
fd=open("/dev/pchar0", ...);
dup(fd);
close(fd);
close(4);
task_struct
(files_struct)
OFDT
0 Sin
1 Sout
2 Serr
3
4
5
i
OFT (file)
f_pos=0
cnt=1 2
1 0
inode cache
type=c
cnt=10
pchar.ko
pchar_open()
3
pchar_close()
5
...

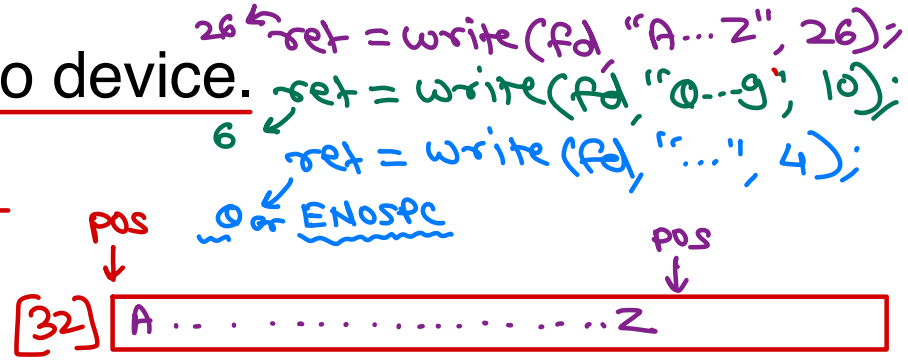
read() operation

- Reads the device and transfers data from device to user-space.
- If kept NULL, read() syscall (from user-space) will fail.
- pseudo char device driver read() implementation:
  - Get how many bytes of data is left in buffer.
  - Decide number of bytes to be copied in user space buffer (min of available and user buffer length).
  - If bytes to read is 0, then obviously it means no data left in buffer (EOD).
  - Copy bytes from device buffer to user buffer using copy_to_user() from current file position.
  - copy_to_user() returns number of bytes not copied. Calculate number of bytes successfully copied.
  - Modify the file position.
  - Return number of bytes successfully read.



write() operation

- Writes the device and transfers data from user-space to device.
- If kept NULL, write() syscall (from user-space) will fail.
- pseudo char device driver write() implementation:
  - Get how many bytes of empty space left in buffer.
  - Decide number of bytes to be copied from user space buffer (min of empty space and user buffer length).
  - If bytes to write is 0, then it means no space left in buffer (ENOSPC).
  - Copy bytes from user buffer to device buffer using copy_from_user() from current file position onwards.
  - copy_from_user() returns number of bytes not copied. Calculate number of bytes successfully copied.
  - Modify the file position.
  - Return number of bytes successfully written.



Test char device driver

- Implement user space program
  - fd = open(“/dev/pchar”, flags);
  - write(fd, buf, len);
  - lseek(fd, nbytes, SEEK_SET);
  - read(fd, buf, len);
  - close(fd);

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>