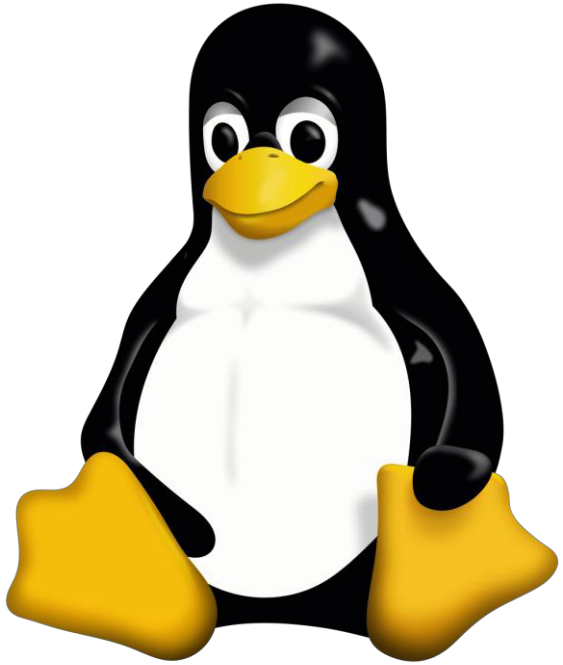


Linux Character Device Driver

Sunbeam Infotech

Using kfifo in char device driver

- Instead of using fixed sized buffer, it is recommended to use kfifo for better memory utilization.
- Can be allocated statically (using DEFINE_FIFO) or dynamically (using kfifo_alloc()).
- kfifo is commonly used to store data
  - from hardware device (before sending to user space).
  - from user space (before sending to hardware device).
- Copying data from or to user space directly is possible using kfifo_from_user() and kfifo_to_user().
- If synchronization is needed, kfifo_in_spinlocked() and kfifo_out_spinlocked() can be used.

ioctl() operation

- read(), write() are typical IO operations on the device.
- ioctl() system call: #include <sys/ioctl.h>
  - int ioctl(int fd, unsigned long cmd, ...);
- ioctl() is special ad-hoc operation that can be used for arbitrary purposes.
  - Manipulating device state directly.
  - Monitoring device state (debugging).
  - Direct hardware control operations.
- Example: handling CD-ROM using ioctl().
  - https://www.kernel.org/doc/Documentation/ioctl/cdrom.txt
  - e.g. ioctl(fd, CDROMEJECT, 0);
- Newer kernel version replace ioctl() with unlocked_ioctl() implementation.
  - long (*unlocked_ioctl)(struct file *pfile, unsigned int cmd, unsigned long param);

cdrom_test.c

```
#include <sys/ioctl.h>
int main() {
    int fd = open("/dev/sr0", O_RDONLY|O_NONBLOCK);
    ioctl(fd, CDROMEJECT, 0);
    close(fd);
    return 0;
}
```

ioctl() operation

- ioctl() operation – 2nd argument *cmd*:
  - Should be unique value throughout the kernel.
  - Old kernel version: 16 bit cmd = 8 bit device magic no (type) + 8 bit sequential value.
  - Refer Documentation/ioctl-number.txt for list of magic numbers used in kernel.
  - New kernel version: 32 bit cmd = 8 bit type (magic) + 8 bit ordinal number + 2 bit direction (NONE, READ, WRITE, READ|WRITE) + 13-14 bit width of data transfer (arch depend)
  - *cmd* argument is created using _IO(), _IOR(), _IOW(), _IOWR()
- pseudo char device driver ioctl() implementation:
  - pchar_ioctl.h: define struct for data transfer and define ioctl commands.
    - #define FIFO_CLEAR _IO('x', 1)
    - #define FIFO_GET_INFO _IOR('x', 2, info_t*)
    - #define FIFO_RESIZE _IOW('x', 3, long)

    } create the cmds. (32-bit nums).
  - Implement each cmd in device ioctl operation.
  - Use copy_to_user() / copy_from_user() to transfer data from 3rd argument (if pointer).

Multiple device instances

- Pseudo char device driver → for one device
  - Operations: open(), close(), read(), write().
  - Device file: /dev/pchar0
  - Kernel presence: struct cdev
  - Data hold in: kernel fifo
- For multiple devices of same type (i.e. multiple device instances) single driver is used. Driver should be capable of handling multiple device data independently with the same code base (i.e. driver operations).
  - Operations: open(), close(), read(), write().
  - Device file: /dev/pchar0, /dev/pchar1, /dev/pchar2, …
  - Kernel presence: struct cdev s
  - Data hold in: kernel fifo s
- Each device operation, should be able to access respective kernel fifo. Device file should be associated with it's own data (cdev & kfifo) i.e. private_data.

Multiple device instances

multiple device instances e.g. 4

1. multiple device buffers → 4 kfifo
2. multiple cdevs → 4 cdev
3. multiple minor nums → 4 minors
4. multiple device files → 4 files
   /dev/pchar0, /dev/pchar1, ...

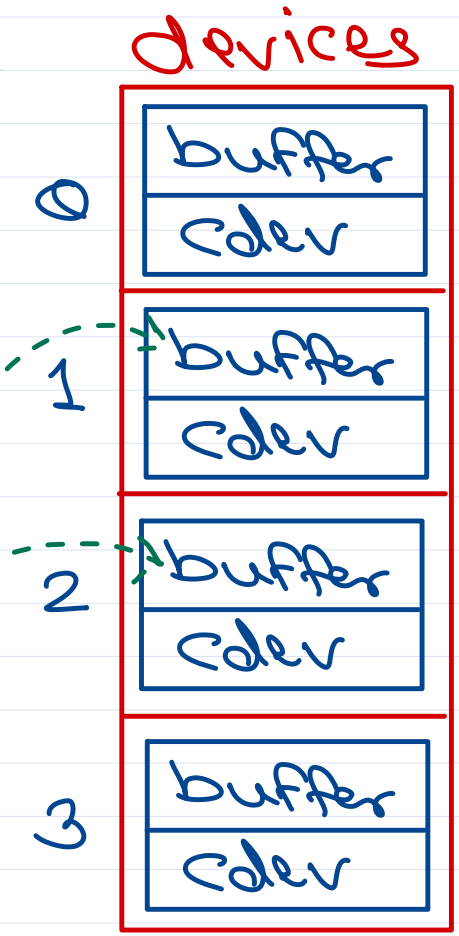
```
struct pchar_device {
    kfifo buffer;
    cdev cdev;
};
```

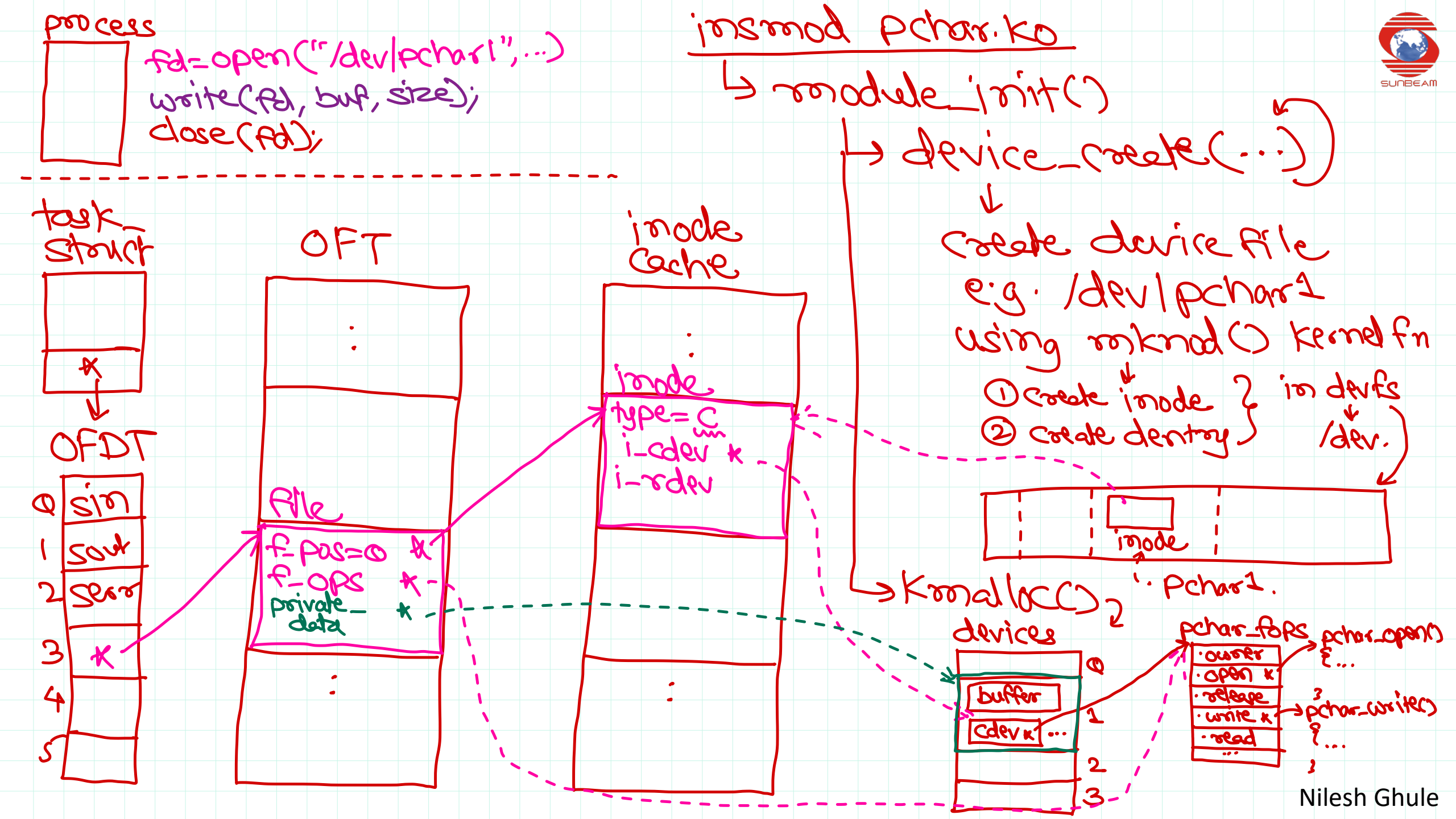
write() → /dev/pchar2

read() → /dev/pchar1

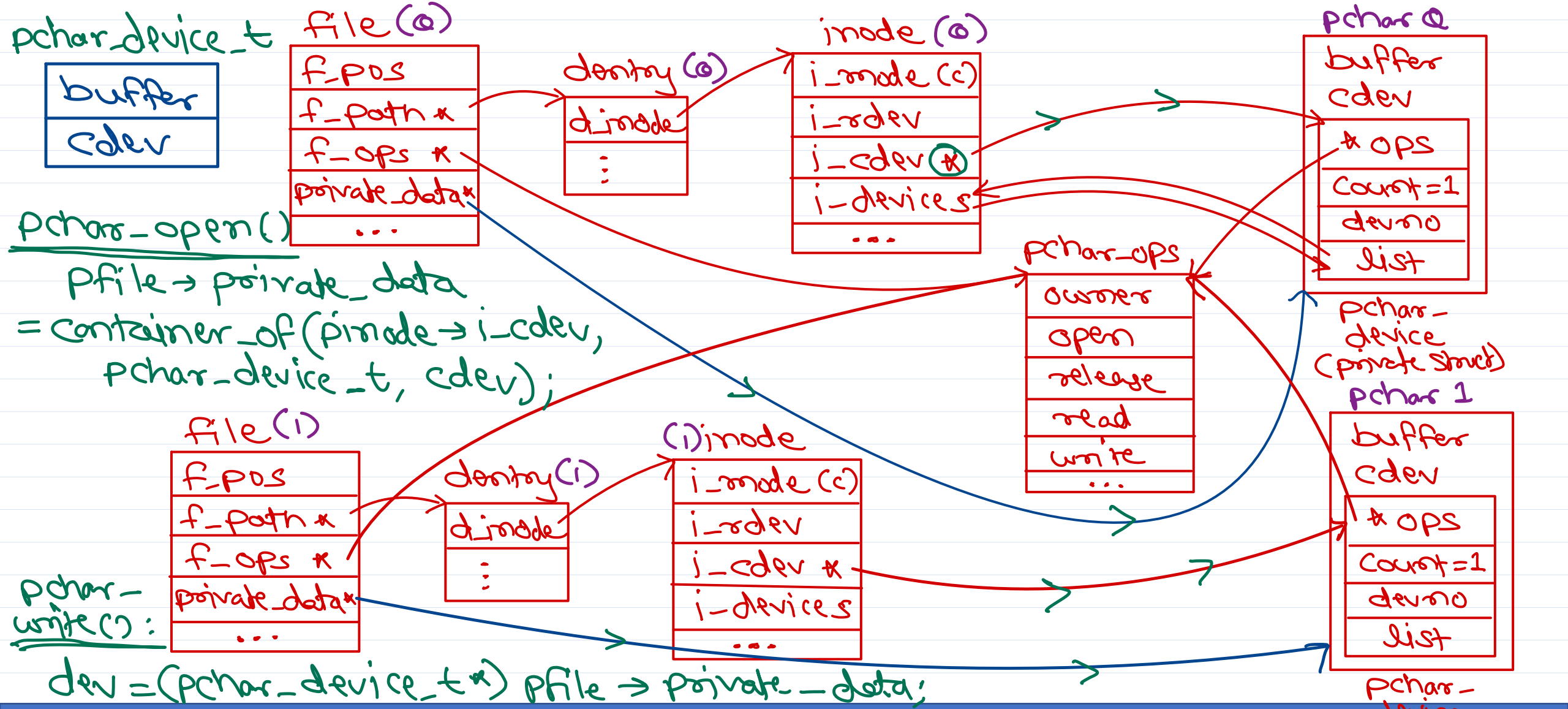
pchar_device devices[DEVCNT];

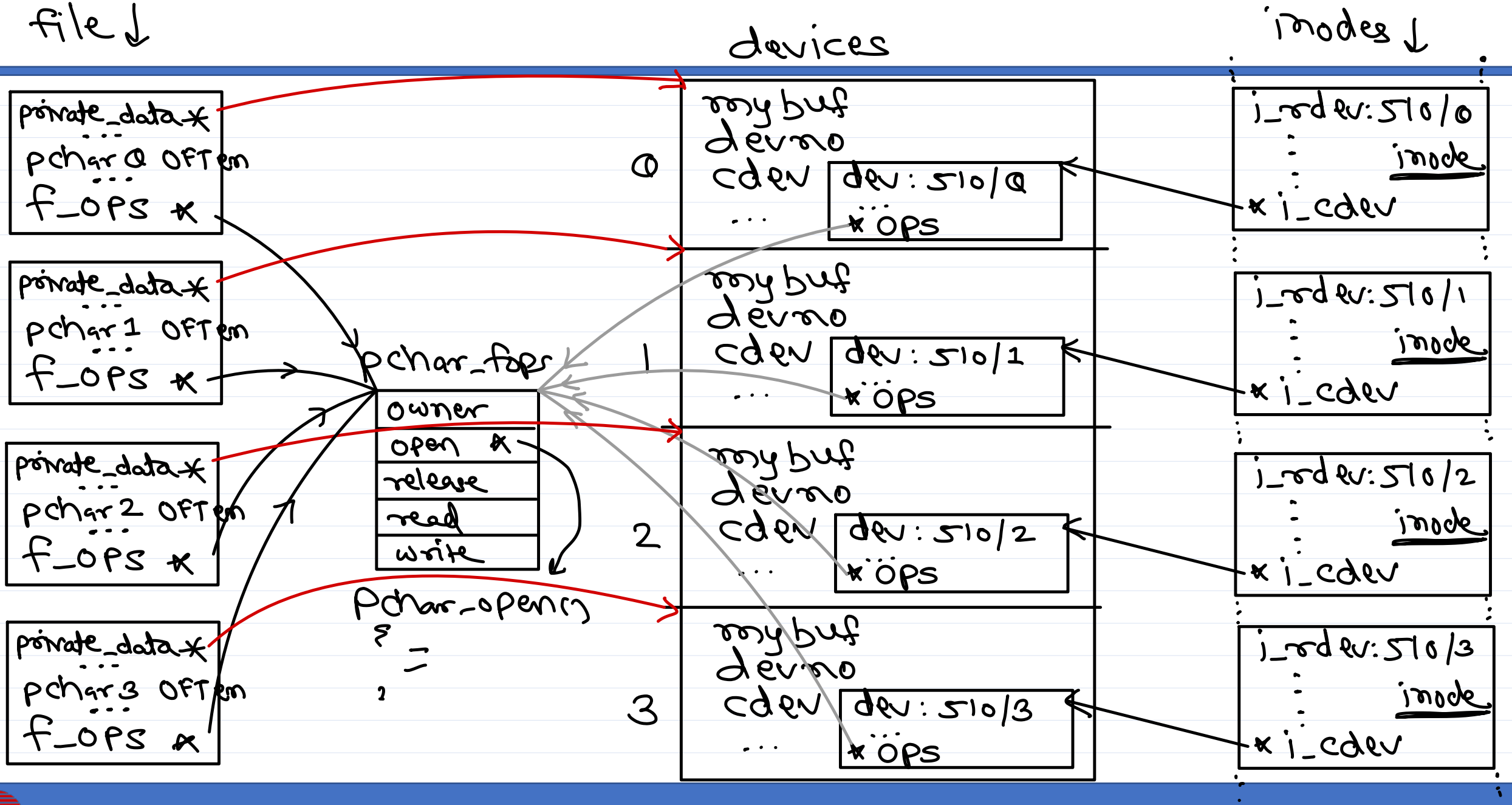
DEVCNT → 4



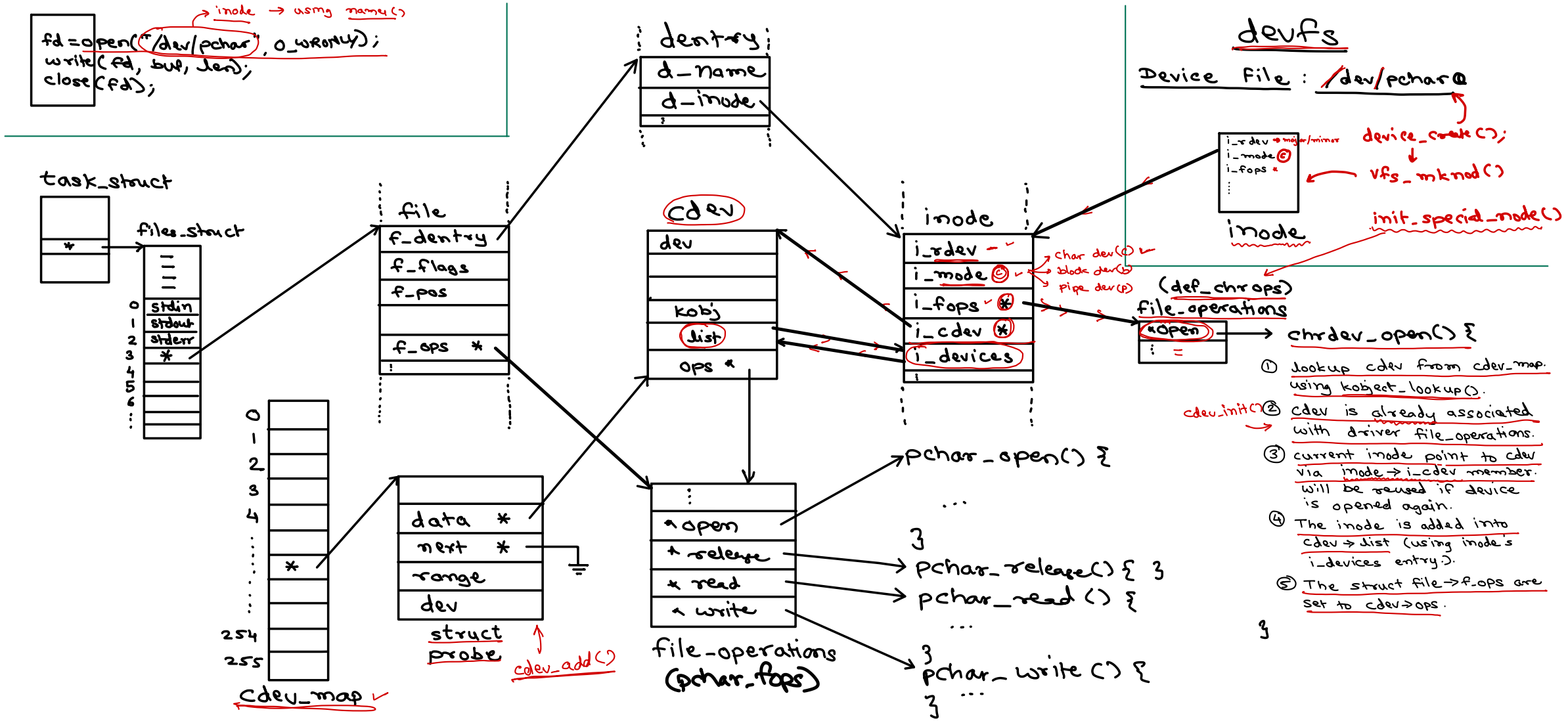

process
fd=open("/dev/pchar1",...)
write(fd, buf, size);
close(fd);
insmod pchar.ko
module_init()
device_create(...)
create device file
e.g. /dev/pchar1
using mknod() kernel fn
① create inode
② create dentry
in devfs
/dev.
task_struct
OFT
inode cache
OFDT
0 stdin
1 stdout
2 stderr
3
4
5
file
f_pos=0
f_ops
private_data
inode
type=c
i_cdev
i_rdev
inode
pchar1
Kernel(CC)
devices
buffer
cdev
pchar_fops
owner
open
release
write
read
pchar_open()
{...
pchar_write()
{...
}
Nilesh Ghule

Handling multiple device instances




file
devices
inodes
private_data *
pchar0 OFTem
f_ops *
private_data *
pchar1 OFTem
f_ops *
private_data *
pchar2 OFTem
f_ops *
private_data *
pchar3 OFTem
f_ops *
pchar_fops
owner
open *
release
read
write
pchar_open()
{
}
0
1
2
3
mybuf
devno
cdev
dev: 510/0
*ops
dev: 510/1
dev: 510/2
dev: 510/3
i_rdev: 510/0
inode
*i_cdev
i_rdev: 510/1
i_rdev: 510/2
i_rdev: 510/3

Execution Flow of Pseudo Char Device Driver



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>