CSE 511: Operating Systems Design

Lecture 15

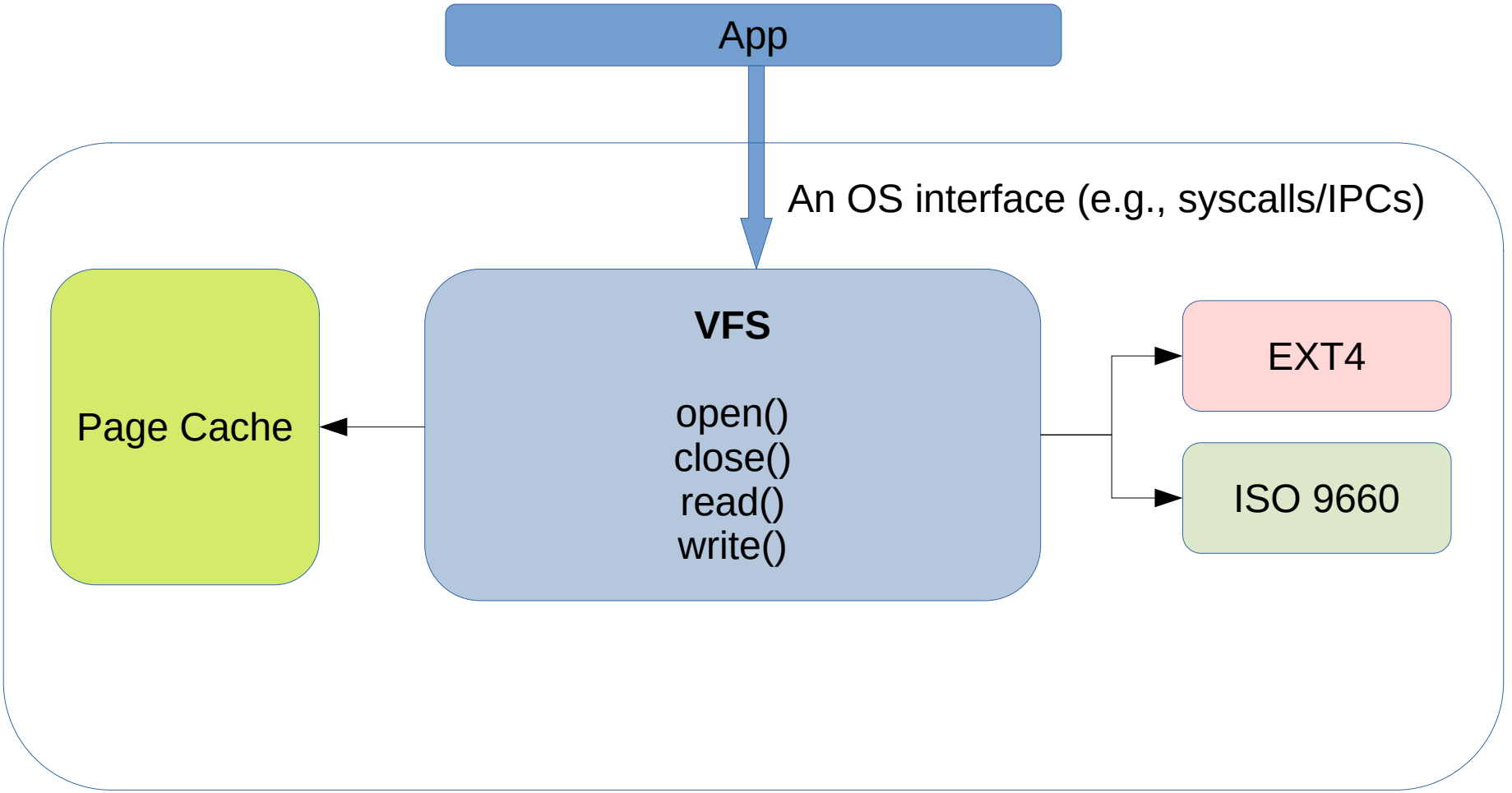
Virtual File System (VFS)

Virtual File System (VFS)

- An OS kernel handles multiple file systems (FAT, ext4, NTFS, ISO 9660, etc)
- The same interface must be provided to user-space programs (POSIX's read, write, etc)
- Need to interact with an OS page cache

```
include/linux/fs.h
```

Virtual File System (VFS)



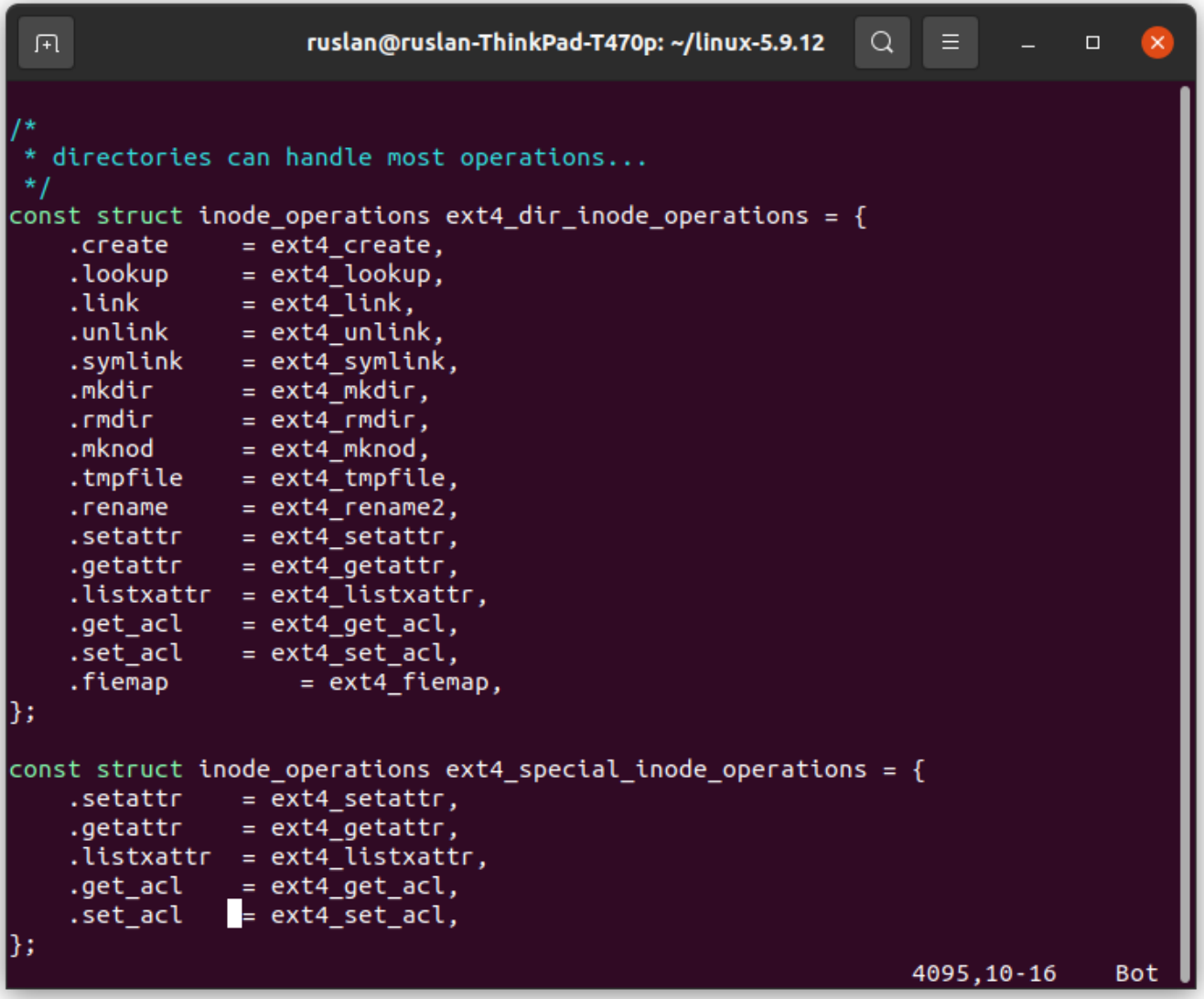
Directory Entry Cache (dentry cache)

- Many system calls use file paths, e.g., open("/home/user/file.txt", O_RDWR)
- All files are stored in "inodes"
  - An inode contains file metadata, e.g., permissions, modification date/time, and a **list of data blocks** (e.g., a tree of blocks)
  - Each inode has an address
- Need to translate a path to an inode address
- May need to traverse directories for new dcache entries

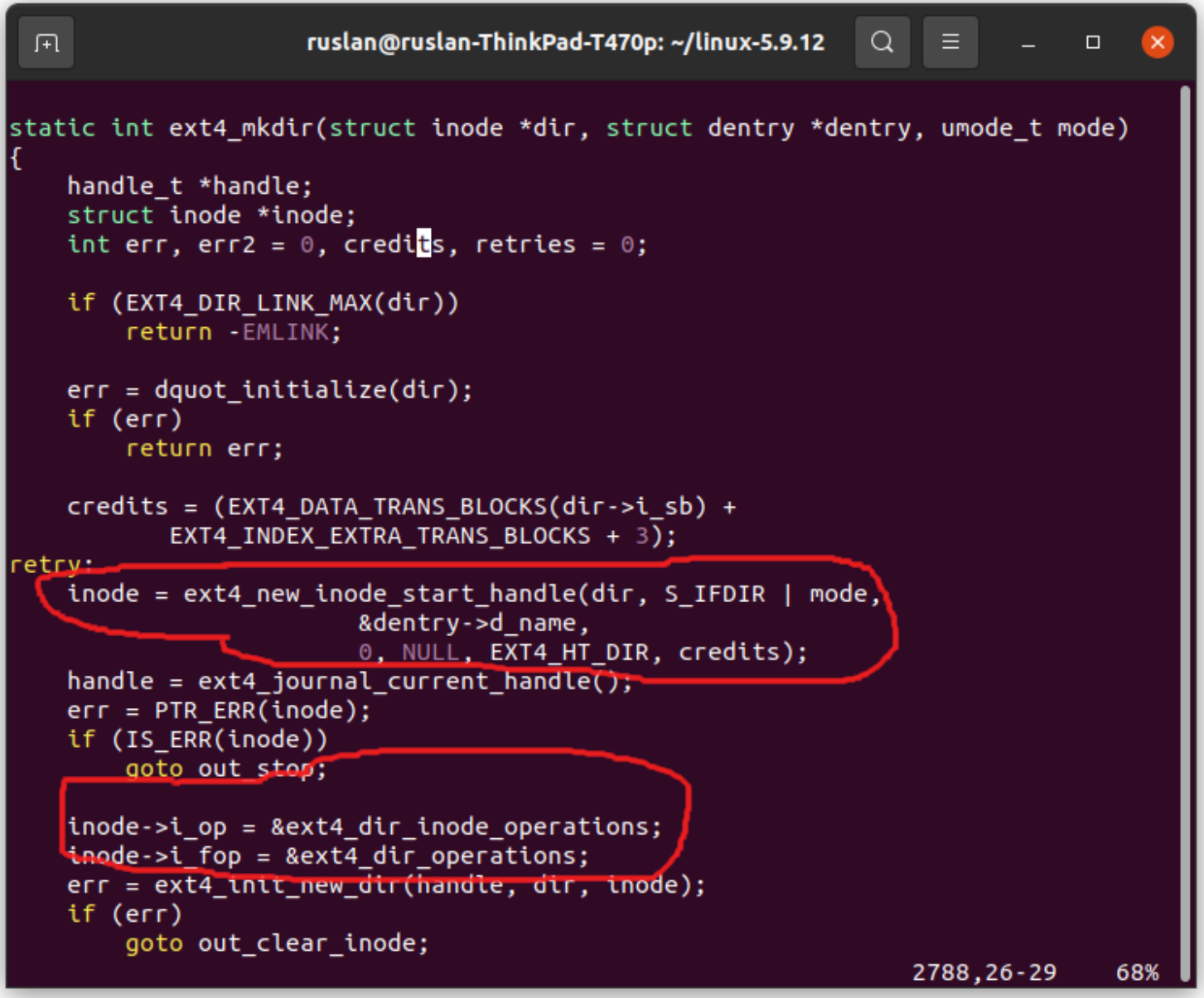
Inode Cache

- Creates a hash table of inode objects
  - An inode can be looked up by an address or some other criteria
- If you open a file, VFS allocates and reads an inode

Inode Operations



Inode Operations



Inode Operations

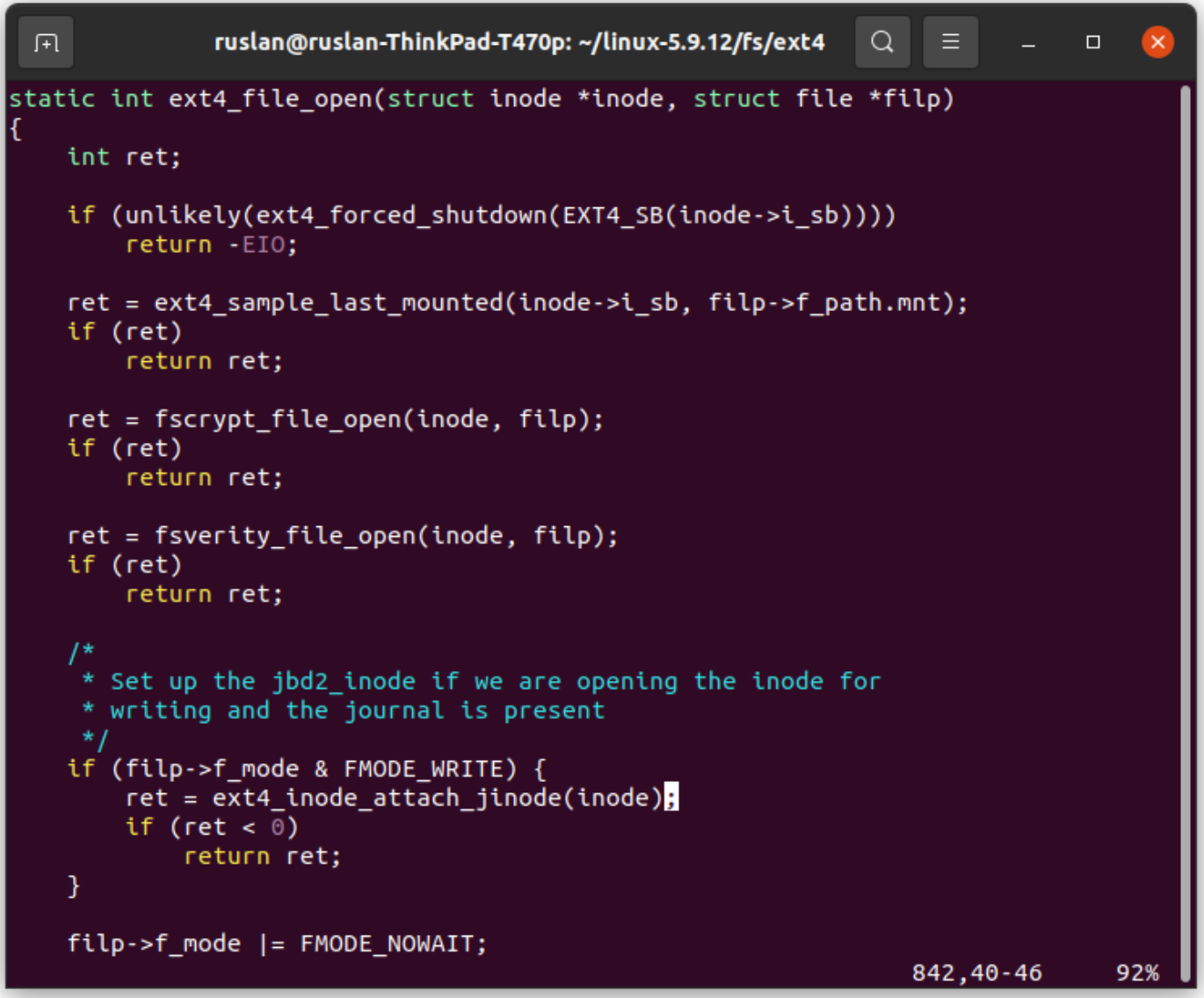
ruslan@ruslan-ThinkPad-T470p: ~/linux-5.9.12/fs/ext4

```
}

const struct file_operations ext4_file_operations = {
    .llseek     = ext4_llseek,
    .read_iter  = ext4_file_read_iter,
    .write_iter = ext4_file_write_iter,
    .iopoll     = iomap_dio_iopoll,
    .unlocked_ioctl = ext4_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl   = ext4_compat_ioctl,
#endif
    .mmap       = ext4_file_mmap,
    .mmap_supported_flags = MAP_SYNC,
    .open       = ext4_file_open,
    .release    = ext4_release_file,
    .fsync      = ext4_sync_file,
    .get_unmapped_area = thp_get_unmapped_area,
    .splice_read    = generic_file_splice_read,
    .splice_write   = iter_file_splice_write,
    .fallocate  = ext4_fallocate,
};

const struct inode_operations ext4_file_inode_operations = {
    .setattr    = ext4_setattr,
    .getattr    = ext4_file_getattr,
    .listxattr  = ext4_listxattr,
    .get_acl    = ext4_get_acl,
    .set_acl    = ext4_set_acl,
    .fiemap     = ext4_fiemap,
};
                                                    911,2-5       99%
```

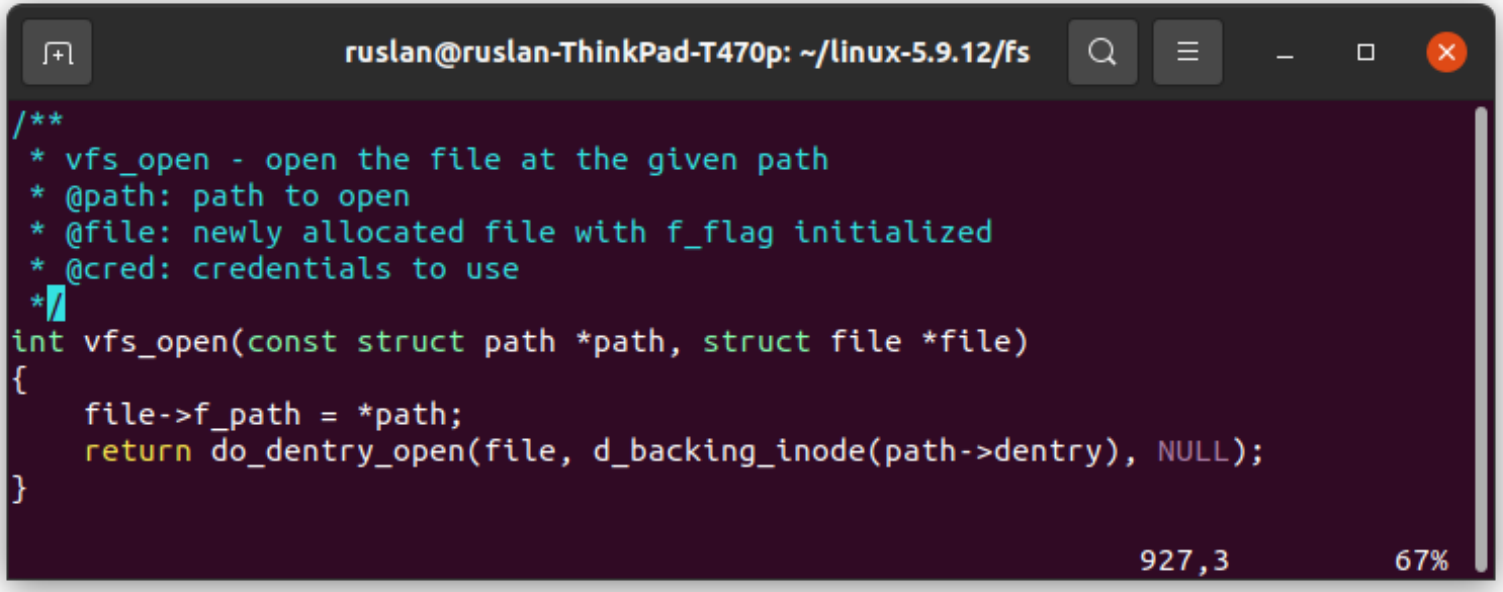
File Objects

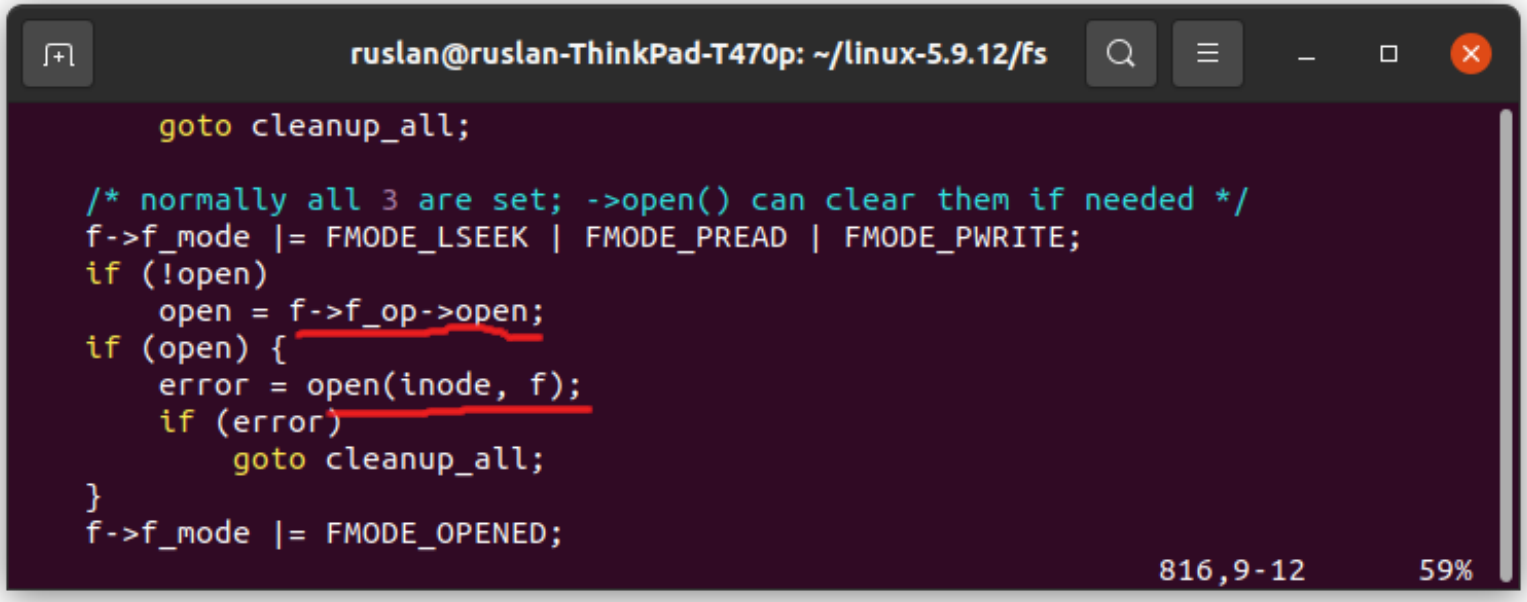
- The same inode can be accessed by multiple processes
- A pointer to dentry, a pointer to file operations (specific to each file system)
- Each process has to keep private information
  - Current file position
  - Access mode when the file is opened
  - UID/GID when the file is opened

Inode Operations



Inode Operations





File Descriptors

- For each process, Linux maintains integer identifiers for opened files
- An integer identifier is used to locate the corresponding ‘file’ object
- Each file object has a reference counter

File Descriptors