



CSE 511: Operating Systems Design

Lecture 15

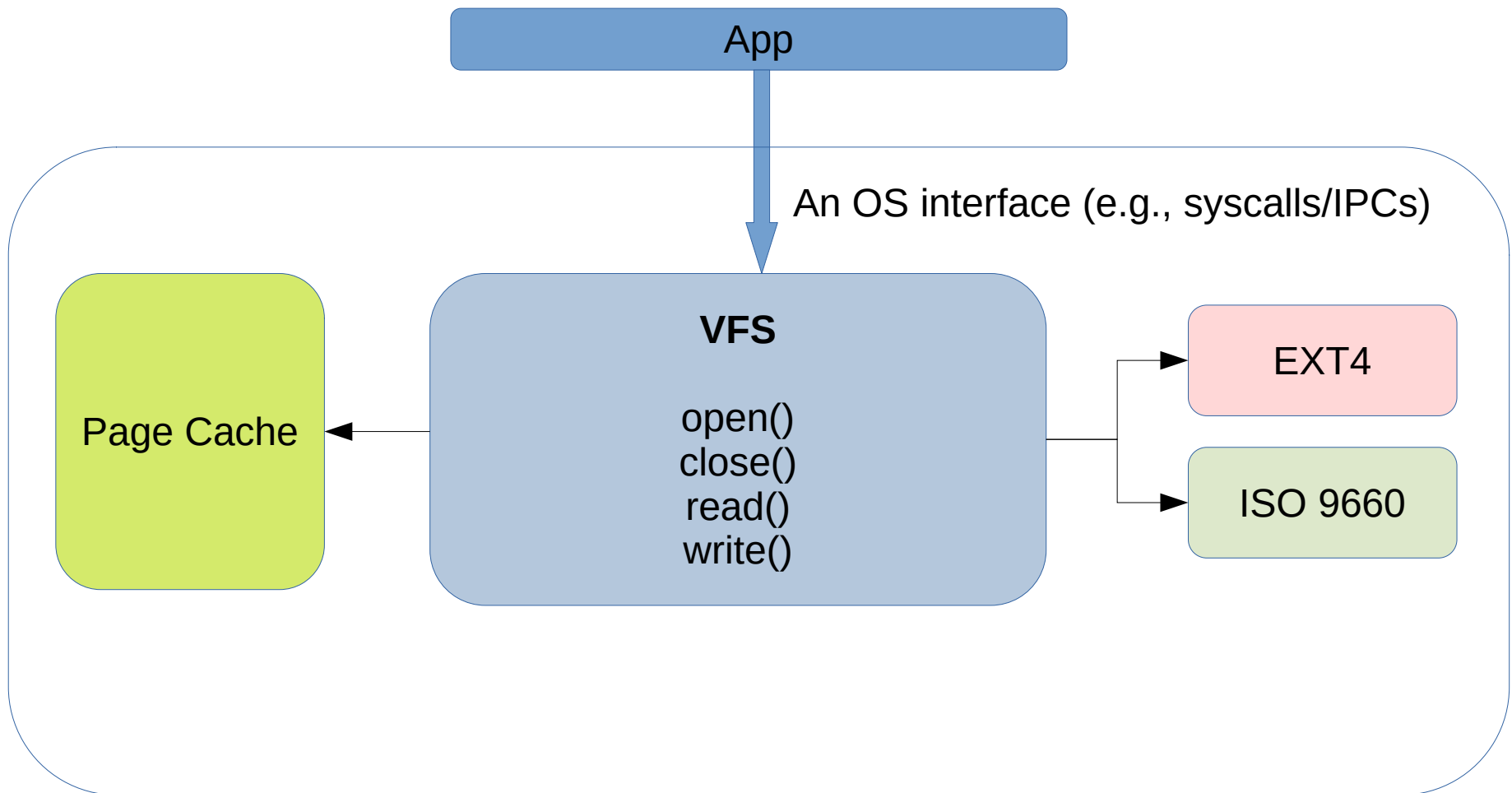
Virtual File System (VFS)

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

```
ruslan@ruslan-ThinkPad-T470p: ~/linux-5.9.12
/*
 * directories can handle most operations...
 */
const struct inode_operations ext4_dir_inode_operations = {
    .create      = ext4_create,
    .lookup      = ext4_lookup,
    .link        = ext4_link,
    .unlink      = ext4_unlink,
    .symlink     = ext4_symlink,
    .mkdir       = ext4_mkdir,
    .rmdir       = ext4_rmdir,
    .mknod       = ext4_mknod,
    .tmpfile     = ext4_tmpfile,
    .rename      = ext4_rename2,
    .setattr     = ext4_setattr,
    .getattr     = ext4_getattr,
    .listxattr   = ext4_listxattr,
    .get_acl     = ext4_get_acl,
    .set_acl     = ext4_set_acl,
    .fiemap      = ext4_fiemap,
};

const struct inode_operations ext4_special_inode_operations = {
    .setattr     = ext4_setattr,
    .getattr     = ext4_getattr,
    .listxattr   = ext4_listxattr,
    .get_acl     = ext4_get_acl,
    .set_acl     = ext4_set_acl,
};

4095,10-16 Bot
```

Inode Operations

```
ruslan@ruslan-ThinkPad-T470p: ~/linux-5.9.12
static int ext4_mkdir(struct inode *dir, struct dentry *dentry, umode_t mode)
{
    handle_t *handle;
    struct inode *inode;
    int err, err2 = 0, credits, retries = 0;

    if (EXT4_DIR_LINK_MAX(dir))
        return -EMLINK;

    err = dqquot_initialize(dir);
    if (err)
        return err;

    credits = (EXT4_DATA_TRANS_BLOCKS(dir->i_sb) +
               EXT4_INDEX_EXTRA_TRANS_BLOCKS + 3);
retry:
    inode = ext4_new_inode_start_handle(dir, S_IFDIR | mode,
                                       &dentry->d_name,
                                       0, NULL, EXT4_HT_DIR, credits);
    handle = ext4_journal_current_handle();
    err = PTR_ERR(inode);
    if (IS_ERR(inode))
        goto out_stop;

    inode->i_op = &ext4_dir_inode_operations;
    inode->i_fop = &ext4_dir_operations;
    err = ext4_init_new_dir(handle, dir, inode);
    if (err)
        goto out_clear_inode;
}
```

2788,26-29 68%

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

```
ruslan@ruslan-ThinkPad-T470p: ~/linux-5.9.12/fs/ext4
static int ext4_file_open(struct inode *inode, struct file *filp)
{
    int ret;

    if (unlikely(ext4_forced_shutdown(EXT4_SB(inode->i_sb))))
        return -EIO;

    ret = ext4_sample_last_mounted(inode->i_sb, filp->f_path.mnt);
    if (ret)
        return ret;

    ret = fscrypt_file_open(inode, filp);
    if (ret)
        return ret;

    ret = fsverity_file_open(inode, filp);
    if (ret)
        return ret;

    /*
     * Set up the jbd2_inode if we are opening the inode for
     * writing and the journal is present
     */
    if (filp->f_mode & FMODE_WRITE) {
        ret = ext4_inode_attach_jinode(inode);
        if (ret < 0)
            return ret;
    }

    filp->f_mode |= FMODE_NOWAIT;
```

842,40-46 92%

Inode Operations

```
ruslan@ruslan-ThinkPad-T470p: ~/linux-5.9.12/fs
/**
 * vfs_open - open the file at the given path
 * @path: path to open
 * @file: newly allocated file with f_flag initialized
 * @cred: credentials to use
 */
int vfs_open(const struct path *path, struct file *file)
{
    file->f_path = *path;
    return do_dentry_open(file, d_backing_inode(path->dentry), NULL);
}
```

927,3 67%

```
ruslan@ruslan-ThinkPad-T470p: ~/linux-5.9.12/fs
goto cleanup_all;

/* normally all 3 are set; ->open() can clear them if needed */
f->f_mode |= FMODE_LSEEK | FMODE_PREAD | FMODE_PWRITE;
if (!open)
    open = f->f_op->open;
if (open) {
    error = open(inode, f);
    if (error)
        goto cleanup_all;
}
f->f_mode |= FMODE_OPENED;
```

816,9-12 59%



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

```
ruslan@ruslan-ThinkPad-T470p: ~/linux-5.9.12
struct open_how how = build_open_how(flags, mode);
int err = build_open_flags(&how, &op);
if (err)
    return ERR_PTR(err);
return do_file_open_root(dentry, mnt, filename, &op);
}
EXPORT_SYMBOL(file_open_root);

static long do_sys_openat2(int dfd, const char __user *filename,
                          struct open_how *how)
{
    struct open_flags op;
    int fd = build_open_flags(how, &op);
    struct filename *tmp;

    if (fd)
        return fd;

    tmp = getname(filename);
    if (IS_ERR(tmp))
        return PTR_ERR(tmp);

    fd = get_unused_fd_flags(how->flags);
    if (fd >= 0) {
        struct file *f = do_filp_open(dfd, tmp, &op);
        if (IS_ERR(f)) {
            put_unused_fd(fd);
            fd = PTR_ERR(f);
        } else {
            fsnotify_open(f);
            fd_install(fd, f);
        }
    }
    putname(tmp);
    return fd;
}

long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
```

1165,0-1 84%