CS353 Linux Kernel

Chentao Wu吴晨涛
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

上海交通大學

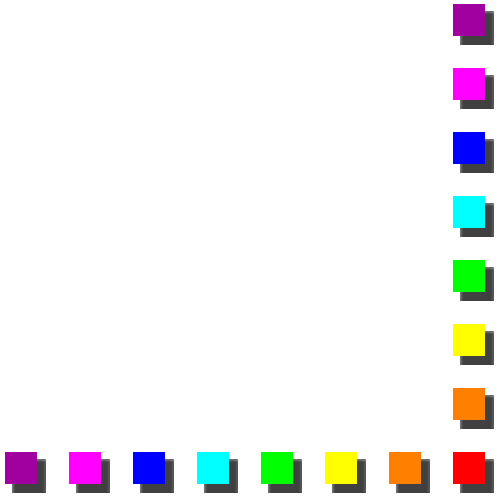
7A. Virtual File System

Chentao Wu
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

上海交通大學

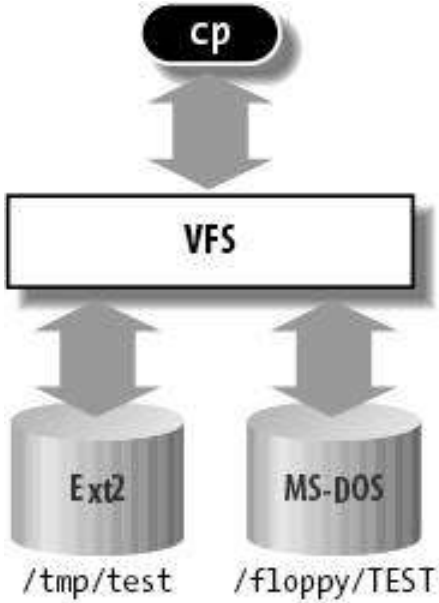
Outline

- Role of VFS
- VFS Data Structures
- Filesystem Types
- Filesystem Handling
- Pathname Lookup
- Implementation of VFS System Calls
- File Locking



Role of VFS (1)

- A common interface to several kinds of filesystems
  - Ex: cp /floppy/TEST /tmp/test



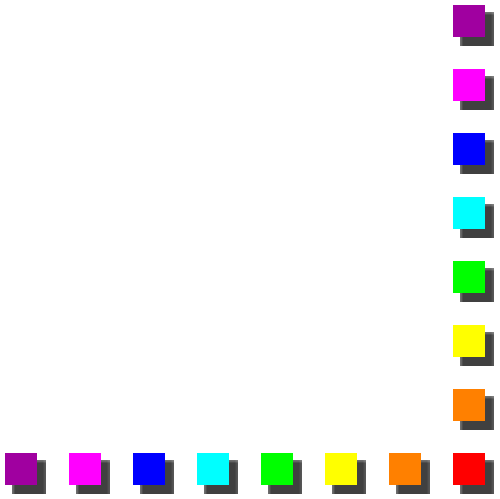
(a)

```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

(b)

Role of VFS (2)

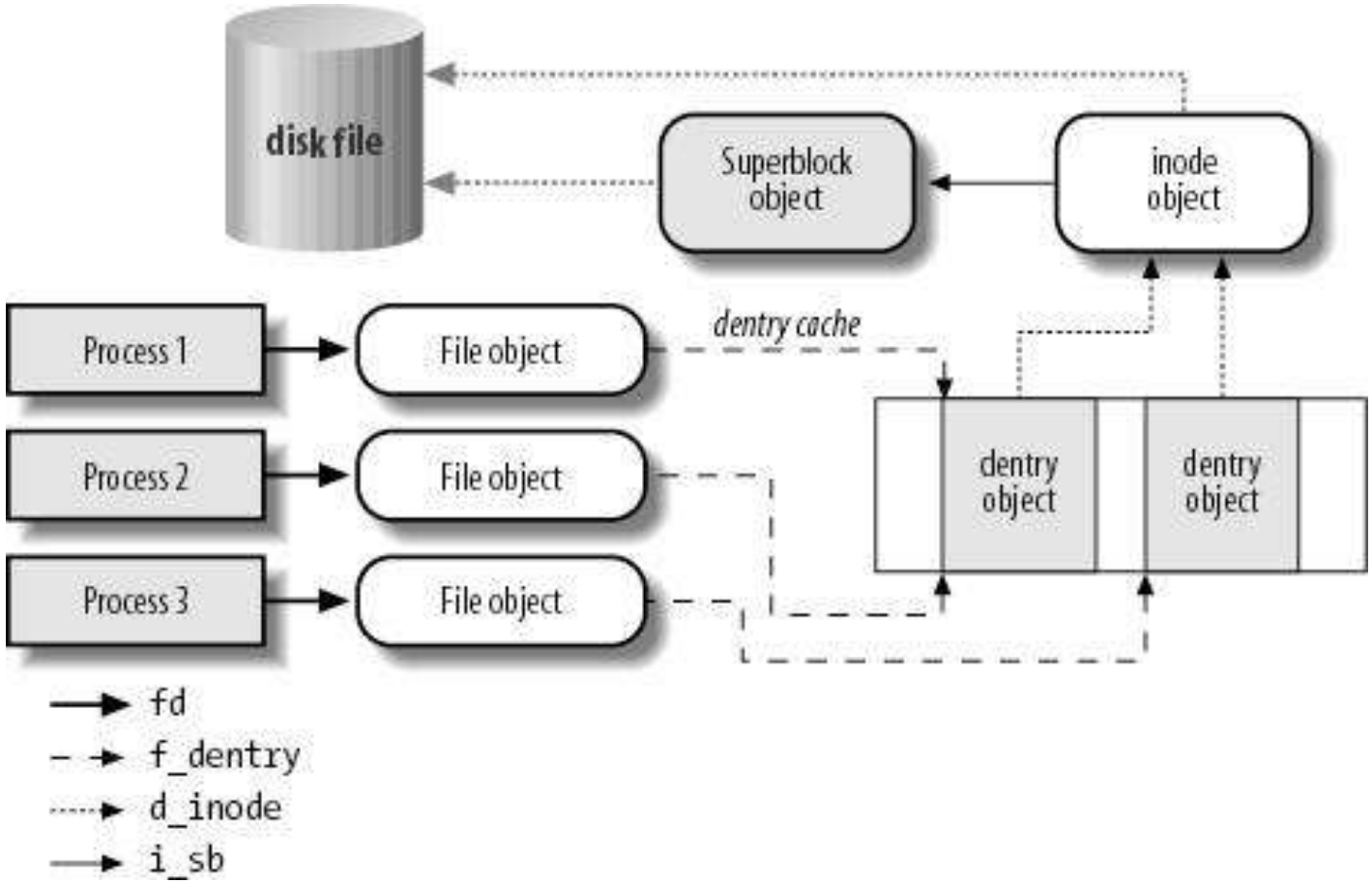
- Filesystems supported by the VFS
  - Disk-based filesystems
    - Ext2, ext3, ReiserFS
    - Sysv, UFS, MINIX, VxFS
    - VFAT, NTFS
    - ISO9660 CD-ROM, UDF DVD
    - HPFS, HFS, AFFS, ADFS,
  - Network filesystems
    - NFS, Coda, AFS, CIFS, NCP
  - Special filesystems
    - E.g. /proc



The Common File Model (1)

- Capable of representing all supported filesystems
  - Each specific filesystem implementation must translate its physical organization into VFS's common file model
  - E.g.: read(…): file->f_op->read(…);
- Object-oriented: data structures and associated operations
  - Superblock object: a mounted filesystem
  - Inode object: information about a file
  - File object: interaction between an open file and a process
  - Dentry object: directory entry

The Common File Model (2)



Some System Calls Handled by the VFS (1)

- Filesystem
  - Mount(), umount(), umount2()
  - Sysfs()
  - Statfs(), fstatfs(), statfs64(), fstatfs64(), ustat()
- Directories
  - Chroot(), pivot_root()
  - Chdir(), fchdir(), getcwd()
  - Mkdir(), rmdir()
  - Getdents(), getdents64(), readdir(), link(), unlink(), rename(), lookup_dcookie()
- Links
  - Readlink(), symlink()

Some System Calls Handled by the VFS (2)

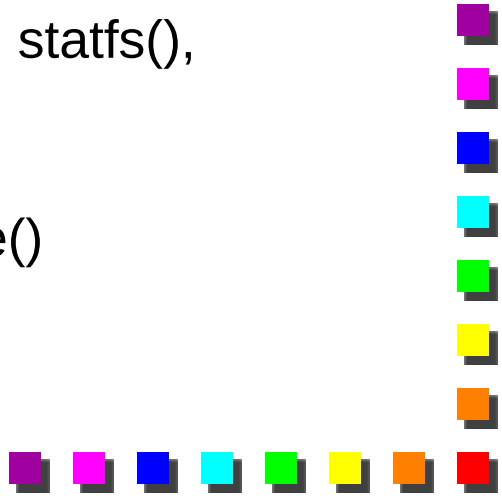
- Files
  - Chown(), fchown(), lchown(), chown16(), fchown16(), lchown16()
  - Chmod(), fchmod(), utime()
  - Stat(), fstat(), lstat(), acess(), oldstat(), oldfstat(), oldlstat(), stat64(), lstat64(), fstat64()
  - Open(), close(), creat(), umask()
  - Dup(), dup2(), fcntl(), fcntl64()
  - Select(), poll()
  - Truncate(), ftruncate(), truncate64(), ftruncate64()
  - Lseek(). _llseek()
  - Read(), write(), readv(), writev(), sendfile(), sendfile64(), readahead()

Some System Calls Handled by the VFS (3)

- Others
  - Io_setup(), io_submit(), io_getevents(), io_cancel(), io_destroy()
  - Pread64(), pwrite64()
  - Mmap(), mmap2(), munmap(), madvise(), mincore(), remap_file_pages()
  - Fdatasync(), fsync(), sync(), msync()
  - Flock()
  - Setxattr(), lsetxattr(), fsetxattr(), getxattr(), lgetxattr(0, fgetxattr(), listxattr(), llistxattr(), flistxattr(), removexattr(), lremovexattr(), fremovexattr()

VFS Data Structures (1)

- Superblock objects: super_block structure
  - (Table 12-2)
  - S_op: superblock operations – super_operations structure
    - Alloc_inode(), destroy_inode()
    - Read_inode(), dirty_inode(), write_inode(),
    - Put_inode(), drop_inode(), delete_inode()
    - Put_super(), write_super()
    - Sync_fs(), write_super_lockfs(), unlokcfs(), statfs(), remount_fs()
    - Clear_inode(), umount_begin()
    - Show_options(), quota_read(), quota_write()

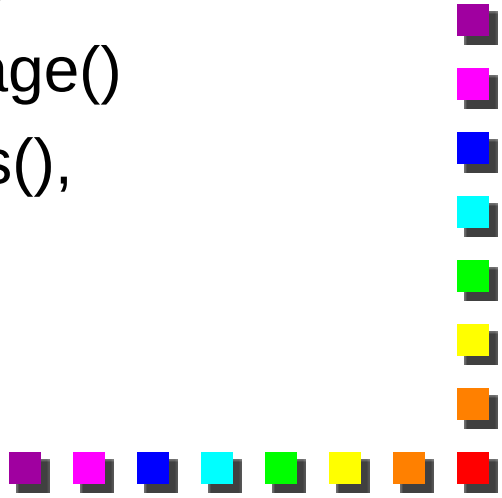


VFS Data Structures (2)

- Inode objects: inode structure
  - i_op: inode operations – inode_operations structure
    - Create(), lookup(), link(), unlink(), symlink()
    - Mkdir(), rmdir(), mknod(), rename()
    - readlink(), follow_link(), put_link()
    - Truncate(), permission(),
    - Setattr(), getattr(), setxattr(), getxattr(), listxattr(), removexattr()

VFS Data Structures (3)

- File objects: file structure (Table 12-4)
  - File operations
    - Llseek(), read(), aio_read(), write(), auio_write()
    - Readdir(), poll(), ioctl(), unlocked_ioctl(), compat_ioctl()
    - Mmap(), open(), flush(), release()
    - Fsync(), aio_fsync(), fasync(), lock()
    - Readv(), writev(), sendfile(), sendpage()
    - Get_unmapped_area(), check_flags(), dir_notify(), flock()

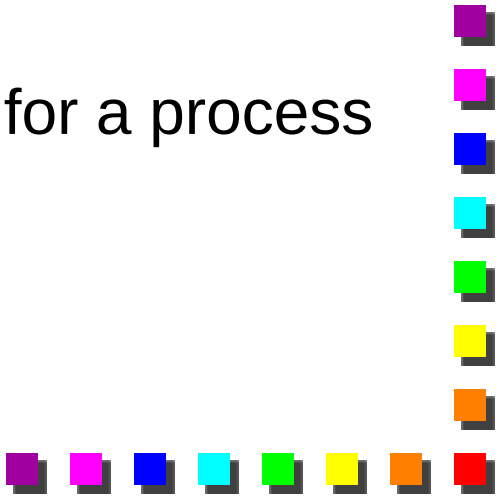


VFS Data Structures (4)

- Dentry objects: (Table 12-5)
  - States: free, unused, in use, negative
  - Dentry operations
    - D_revalidate()
    - D_hash()
    - D_compare()
    - D_delete()
    - D_release()
    - D_input()
- Dentry cache
  - A set of dentry objects
  - A hash table

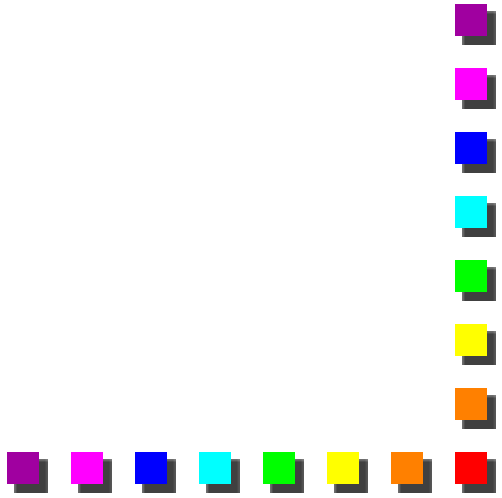
Files Associated with a Process

- fs field: fs_struct structure
- files field: files_struct structure
  - fd: file descriptors
    - fd[0]: stdin
    - fd[1]: stdout
    - fd[2]: stderr
  - NR_OPEN: max # of file descriptors for a process
    - Usually 1,048,576



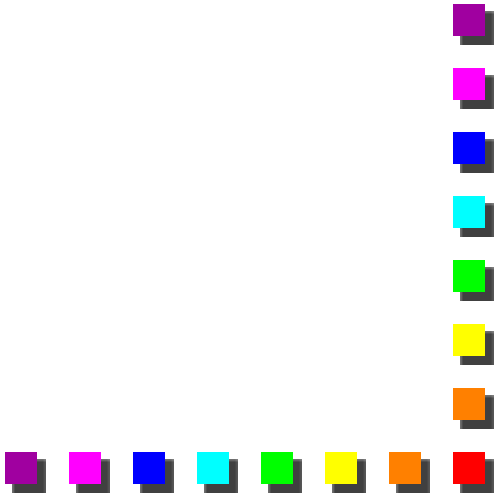
Special Filesystems

- /dev/pts: pseudo terminal support
- /proc: general access point to kernel data structures
- /sys: general access point to system data
- /proc/bus/usb: USB devices
- …



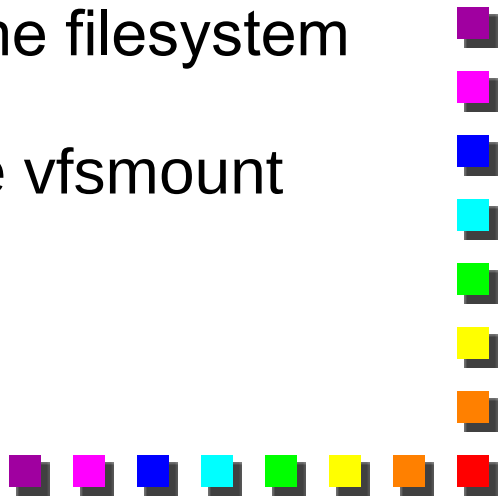
Filesystem Type Registration

- File_system_type object
- Fs_flags



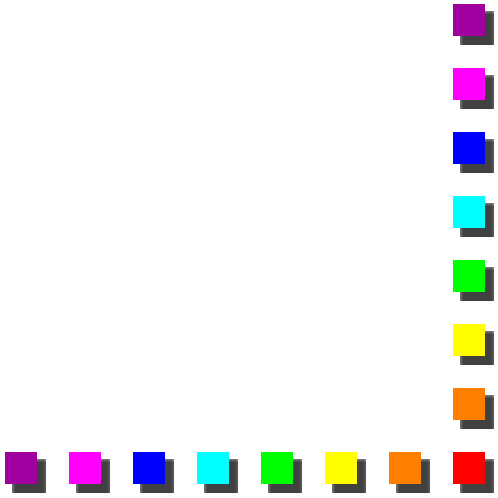
Filesystem Handling (1)

- Root filesystem
- Mount point
- Namespaces: in Linux 2.6, each process might have its own tree of mounted filesystems
  - Namespace structure (Table 12-11)
- Filesystem mounting
  - It's possible in Linux to mount the same filesystem several times
  - Mounted filesystem descriptor: of type vfsmount (Tabel 12-12)
  - Mounting/unmounting the filesystem



Filesystem Handling (2)

- Pathname lookup
  - Pathname -> inode
  - Pathlookup(): return the nameidata structure (Table 12-15)
  - Standard pathname lookup
  - Parent pathname lookup
  - Lookup of symbolic links



Implementation of VFS System Calls

- Open()
- Read()
- Write()
- Close()

File Locking

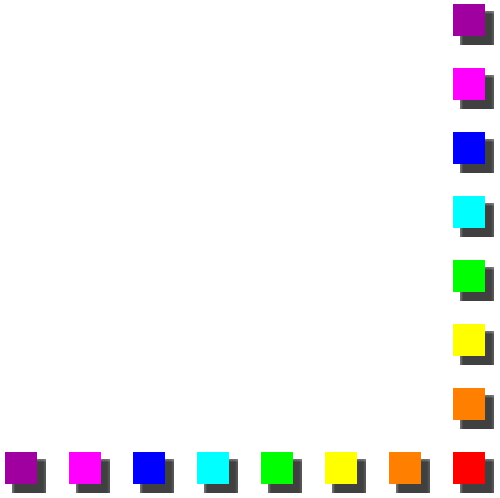
- Advisory locks: by POSIX
  - Based on fcntl() system call
  - Possible to lock an arbitrary region of a file
- Mandatory locks: by System V Release 3
  - The kernel checks every invocation of open(), read(), write() system calls does not violate a mandatory lock
- Linux supports both + fcntl() and flock() system calls

Project 4:
File System

上海交通大學

Source

- Inode.c/Makefile (kernel source of romfs)
- Test.img (a romfs image, you can mount it to a dir with ‘mount –o loop test.img xxx)
- Say test.img is mounted in t, ‘find t’ output
  - aa
  - bb
  - ft
  - fo
  - fo/aa

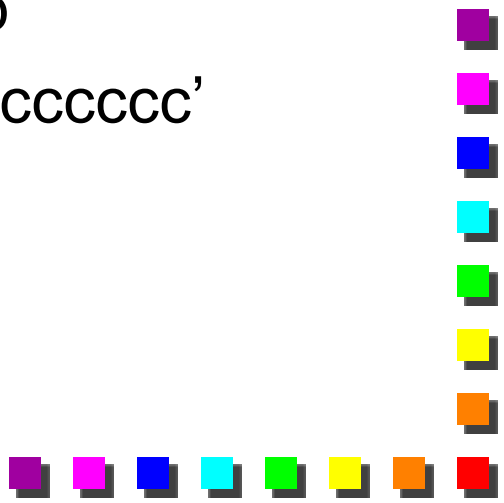


Practice 1

- Change romfs code to hide a file/dir with special name
- Test & result
  - insmod romfs hided_file_name="aa“
  - Mount –o loop test.img t
  - then ls t, ls t/fo, no "aa" and "fo/aa". found
  - ls t/aa, or ls fo/aa, no found
  - Without the code change, above two operations can find file ‘aa’

Practice 2

- change the code of romfs to correctly read info of an ‘encrypted’ romfs
- Test & result
  - insmod romfs hided_file_name=“bb“
  - Mount –o loop test.img t
  - Say bb’s original content is ‘bbbbbbb’
  - With the change, cat t/bb output ‘ccccccccc’



Practice 3

- change the code of romfs to add ‘x’ (execution) bit for a specific file
- Test & result
  - insmod romfs hided_file_name=“bb“
  - Mount –o loop test.img t
  - Without code changes ‘ls –l t’, output is ‘-rw-r--r--’
  - With the change, output is ‘-rwxr-xr-x’