

UNIX Tutorial 1: File Systems

SOEN321

Revision 1.3

Date: October 2, 2003

Contents

- UNIX/Linux FS
- UNIX Permissions
- Intro to NFS
- suid programs, setuid(), etc

Support For Different File Systems

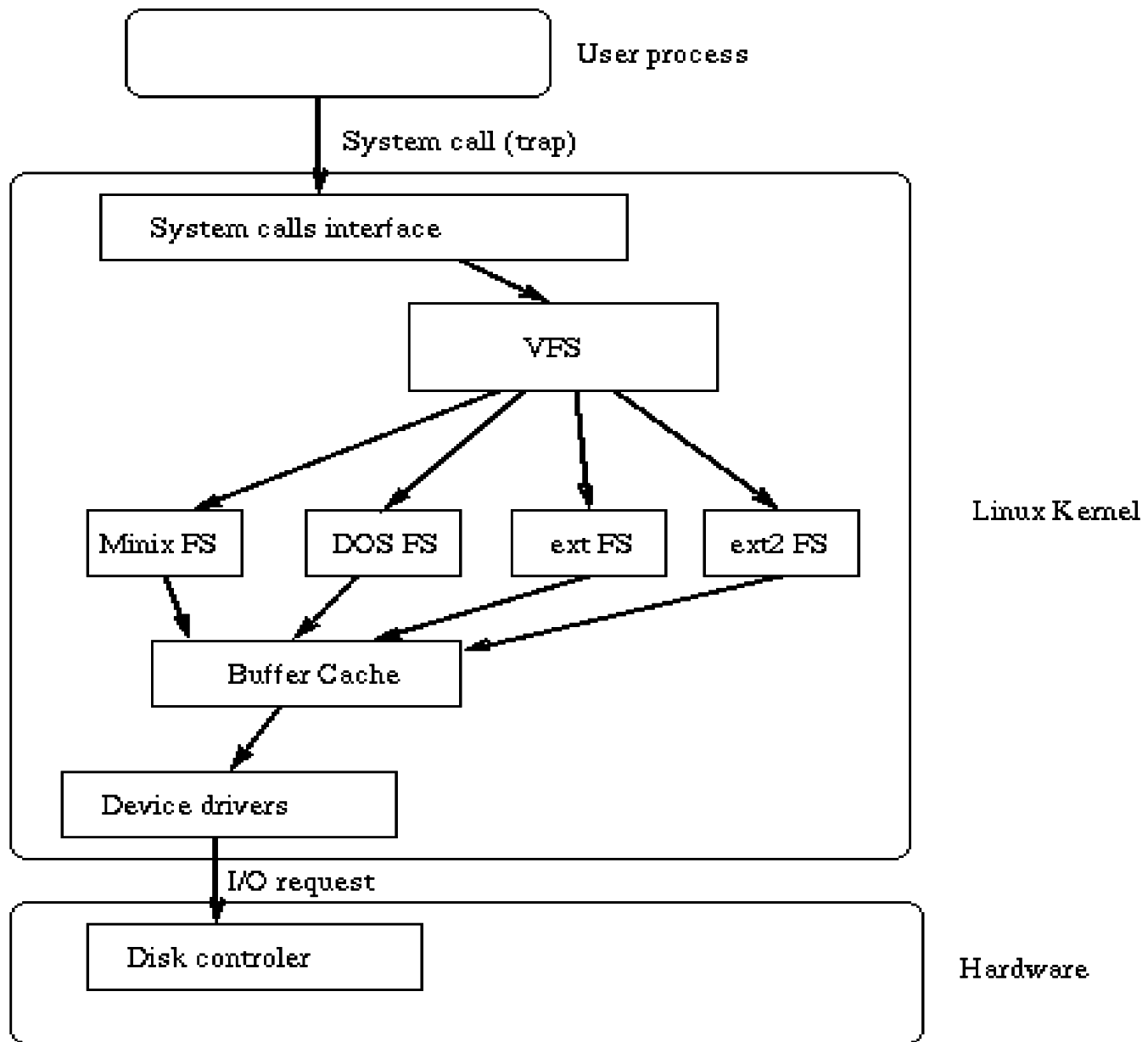
- Linux supports the following file systems: minix, ext, ext2, xia, msdos, umsdos, vfat, proc, **nfs**, iso9660, hpfs, sysv, smb, ncpfs
- man fs

Representation of File Systems

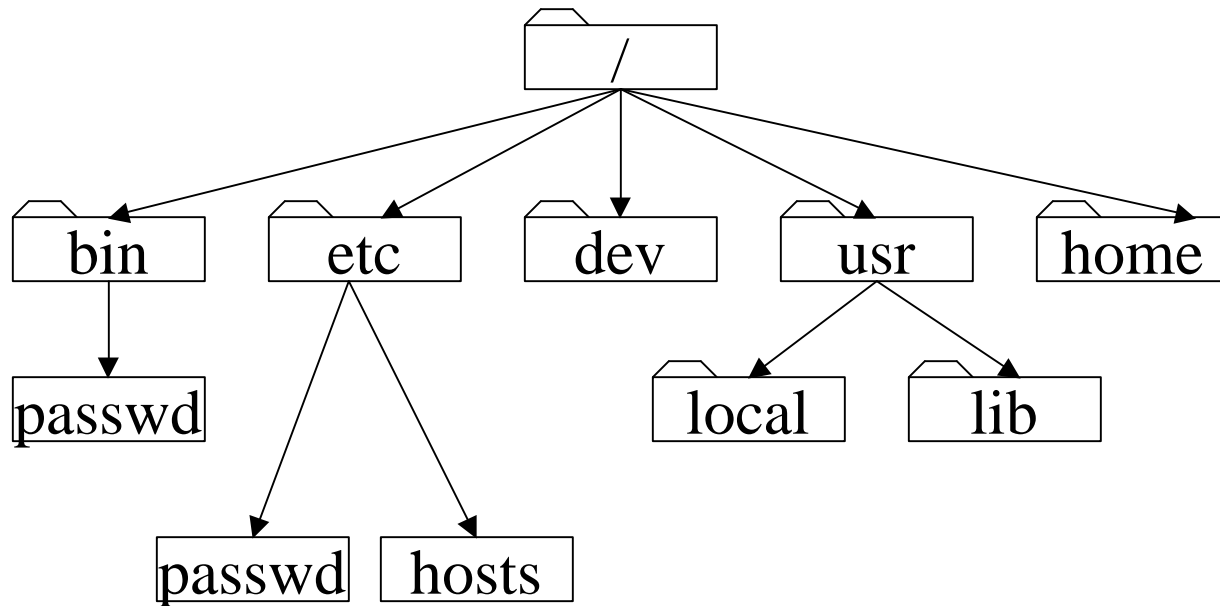
- The separate file systems the system may use are not accessed by device identifiers (such as a drive number or a drive name) .
- They are combined into a single hierarchical tree structure that represents the file systems as one whole single entity.
- UNIX/Linux adds each new file system into this single file system tree as it is *mounted*. All file systems, of whatever type, are mounted onto a directory and the files of the mounted file system cover up the existing contents of that directory

Virtual File System (VFS)

- VFS allows UNIX/Linux to support many, often very different, file systems, each presenting a common software interface to the VFS.
- Details of the file systems are translated by software so that all file systems appear identical to the rest of the kernel
- Virtual File System layer allows you to transparently mount the many different file systems at the same time.

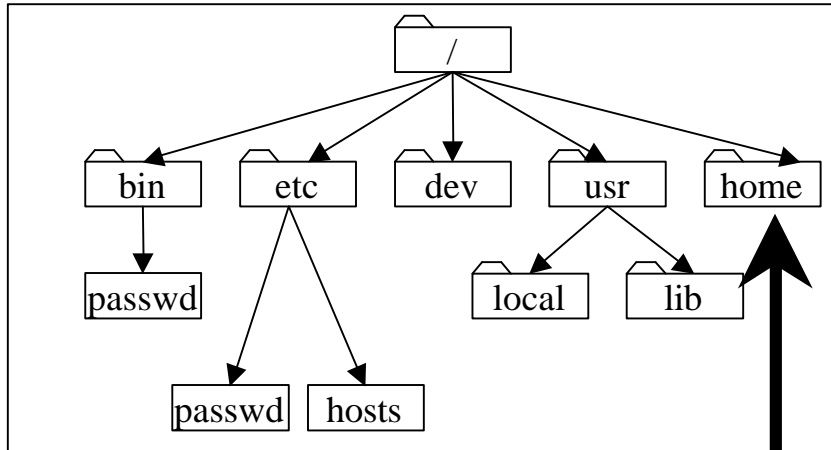


FS Tree Hierarchy

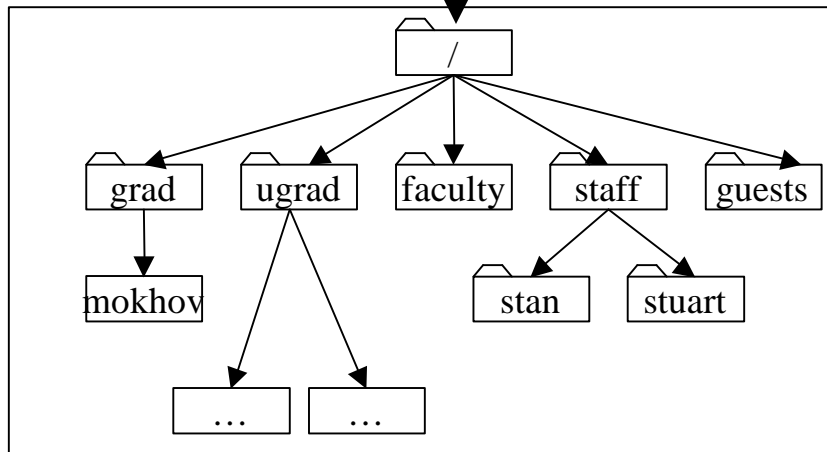


FS Tree Hierarchy: Mounted FS

fs0



fs1



File Permissions

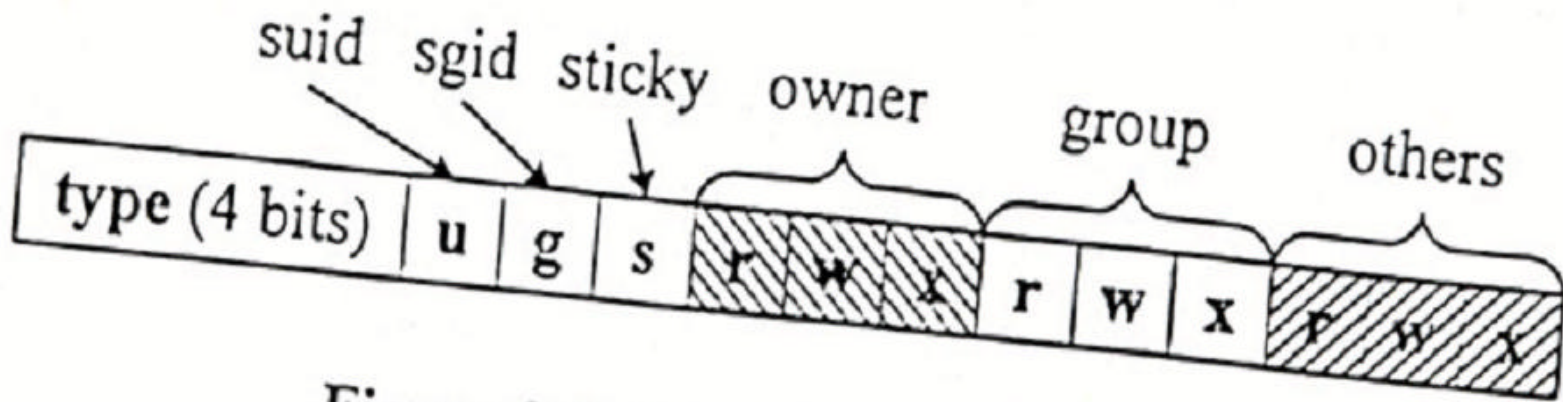


Figure 9-3. Bit-fields of `di_mode`.

File Permissions (2)

```
man ls:
```

```
38     The mode printed under the -l option consists of ten charac-  
39     ters. The first character may be one of the following:
```

```
40
```

```
41     d     the entry is a directory;
```

```
42
```

```
43     D     the entry is a door;
```

```
44
```

```
45     l     the entry is a symbolic link;
```

```
46
```

```
47     b     the entry is a block special file;
```

```
48
```

```
49     c     the entry is a character special file;
```

```
50
```

```
51     p     the entry is a fifo (or "named pipe") special file;
```

```
52
```

```
53     s     the entry is an AF_UNIX address family socket;
```

```
54
```

```
55     -     the entry is an ordinary file;
```

File Permissions (3)

```
alpha.mokhov [~] % ls -agl /usr/bin/passwd
---s--x---  3 root      acmaint      85520 Dec 11  2001 /usr/bin/passwd
alpha.mokhov [~] % ls -agl /etc/passwd
-rw-r--r--  1 root      sys          450 Dec 18  2001 /etc/passwd
alpha.mokhov [~] % ls -al
...
-rwx-----  1 mokhov   13593 Aug 15  2000 settings.zip
drwx-----  3 mokhov    4096 Sep 28  2001 share
drwx-----  2 mokhov    4096 May 10  22:09 soft
drwx-----  3 mokhov    4096 Sep 15  18:59 tmp
-----  1 mokhov      0 Nov 30  2001 trace.txt
drwx-----  3 mokhov    4096 Sep  9  18:05 var
lrwxrwxrwx  1 mokhov      21 May  8  15:46 www -> /home/grad/www/mokhov
alpha.mokhov [~] %
```

File Permissions (4)

```
106      The permissions are indicated as follows:
107
108      r      the file is readable
109
110      w      the file is writable
111
112      x      the file is executable
113
114      -      the indicated permission is not granted
115
116      s      the set-user-ID or set-group-ID bit is on, and the
117              corresponding user or group execution bit is also on
118
119      S      undefined bit-state (the set-user-ID bit is on and the
120              user execution bit is off)
121
122      t      the 1000 (octal) bit, or sticky bit, is on (see
123              chmod(1)), and execution is on
124
125      T      the 1000 bit is turned on, and execution is off
```

UNIX Privilege Levels

username, account, password, userid

the /etc/passwd file;

entry = <uname:"phash":uid:gid:gcoss:home:shell>

Traditionally, a 13-character "phash"; newer Linux's differ
shadow password file;

```
haida.mokhov [soen321] % ls -alg /etc/shadow
-r----- 1 root root 720 May 30 11:38 /etc/shadow
```

interaction with **NFS**

users and groups, root (the superuser); uid is what counts
the superuser exerts nearly complete control over the operating system
principle of least privilege, compartments, "need to know"

su ("set/substitute user") command, su root
processes, command interpreters, real and effective uids
suid bits in access-control lists; suid files; suid-root files
on having your own process's privilege raised in a controlled way and why
this is dangerous

Remote File Systems

ref: Vahalia, ch 10

- Goals
 - Mount file systems of a remote computer on a local system
 - Mount any FS, not only UNIX
 - H/w independent
 - Transport independent
 - UNIX FS semantics must be maintained
 - Performance
 - Crash recovery
 - **Security**
- Example: Sun's NFS, 1985

NFS

- Network File System
 - Designed to operate with a wide range of operating systems
- Challenge
 - Simulate the semantics of the user O/S

NFS Server

- Server export available filesystems in `/etc/exports`
 - Only local filesystems
 - May not cross mount points
 - i.e client must mount each filesystem as it wishes, but it cannot mount a set of mounted filesystems

NFS Components

- NFS Protocol
- RPC (Remote Procedure Call interface)
- XDR (eXtended Data Representation)
- Server code
- Client code
- Mount Protocol
- Daemon processes – `nfsd`, `mountd`
- Network Lock Manager (NLM)

Statelessness

- Server maintains no state
 - e.g a read on the server opens, seeks, reads, and closes
 - A write is similar, but the buffer is flushed to disk before closing
- Server crash: client continues to try until server reboots – no loss
- Client crashes: client must rebuild its own state – no effect on server

RPC - XDR

- RPC:
 - Standard protocol for calling procedures in another machine.
 - Procedure is packaged with authorization and admin info.
- XDR:
 - standard format for data, because manufacturers of computers cannot agree on byte ordering.

NFS Control Flow

- Vahalia, 10-4
- State maintained on client
 - Vnodes
 - File table with offsets, open modes, protection
 - Nfs_vops map into nfs client code, which packages RPC's

Operations

- Every operation is independent:
 - server opens file for every operation
- File identified by handle; no state information retained by server
- Client maintains:
 - mount table, vnode, offset in file table etc.

NFS Protocol

- Client - server: server holds file systems
- Stateless protocol: The server does not retain the state of the client operations
- Server exports filesystems (/etc/exports)
- Client (re)mounts filesystems
- File handle: used to uniquely identify a file on the server

Operations

- client sends mount (pathname)
 - server returns handle
- lookup(dirfh, name) one file at a time
 - server returns (handle, status, attributes)
- open: local to client
- read (handle, offset, count, totcount)
 - server: (status, attributes, data)

Mount

- Local:
 - add vfs structure,
 - send mount to server
- Server checks to see if the filesystem is exported, if OK, sends ok back to client with the handle.
- Client completes initialization of vfs structure.

lookup()

- One component at a time
- Very slow
- Client may cache directory info, but this is dangerous.

UNIX Semantics - Access

- UNIX owner may change permissions on an open file, but while the file is open user can continue to access.
- NFS file is opened on every access this conflicts with above – workaround owner can always access file.
- On open, server returns permissions.

UNIX Semantics - Deletion

- Deletion of open file-legal in UNIX, but file not deleted until closed
- NFS client code sends rename, and then deletes renamed file on close.

UNIX Semantics – Read/Write

- UNIX serializes reads, writes - atomic on system call granularity.
- Handled by client for reads from same machine
- Multiple machines: handled by NLM
 - Possible to bypass locks.

NFS Security Problems

- Two points of access control: mount and each RPC; no restriction on specific users
- NFS servers blindly trust NFS clients -> exported FS may be read by any machine
- Remember: clients maintain the state, including permissions.
- Anyone can write a malicious NFS client, and having UIDs and GIDs of users read the files.

NFS Security Problems (2)

- To strengthen NFS a bit...
 - UID and GID remapping
 - requirement for secure RPC (NFS v.3), which is not widespread
 - CERTificates
 - Use `nosuid` option
 - Untrusted client's root (`root_squash` option)
 - System binaries owned by root
- P. 86 – Few links

UIDs, setuid, suid

- Difference between the real uid, the effective uid, and the saved uid, and their use.
- Real UID is used to identify a real owner of a process and the signals it can send. Effective UID influence file creation and access permissions and is normally the same as real UID. Effective UID may change if one execs a setuid program, which acts on behalf of the owner of the program. When this happens the system saves the UID prior exec and allows to restore it back by setuid.

setuid()

- *Is there a way a programmer could use a setuid() program to penetrate the security of UNIX/Linux?*
- Normally, no. Good intentions of this call in user mode are just set it's effective UID to real. The superuser can set any UID to whatever s/he wants. However, on an unpatched UNIX/Linux by tracing a setuid program with ptrace and if the program invokes subsequent execs, one can modify its address space to exec a shell and gain unauthorized superuser's access to the system (p. 154, Vahalia).

NFS Crash Review

- *In the case where an NFS server crashes and reboots, how does it know what filesystems its clients have mounted? Does it care? Does the client care?*
- The server knows about file systems when the clients reconnect back again and it doesn't really care of what were they because of the stateless nature (and the clients can only connect to exported FS's anyway). The clients do care because they have a state, and they attempt to reconnect every once in a while to get the mount table back, but no data loss occurs.

NFS and Performance

- *Why is client side-caching desirable in NFS? Why is it dangerous? Use VOP_LOOKUP as an example.*
- Client-side NFS caching is desirable because it reduces network traffic (i.e. # of collisions) and, more importantly, pathname lookup time will be reduced greatly. The caveat is, however, a possibility of going out of sync with the server if the server decides to fool around with the mounted FS's by clients (see some examples in Q#1), so the cache is periodically invalidated.

References

- COMP444 Slides S. Winnikoff, T. Fancott
- Vahalia: UNIX Internals
- DK. Probst Notes, Fall 2001
- man fs
- man chmod
- man set[f]acl/get[f]acl
- man ls
- man nfsd
- man mountd