CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

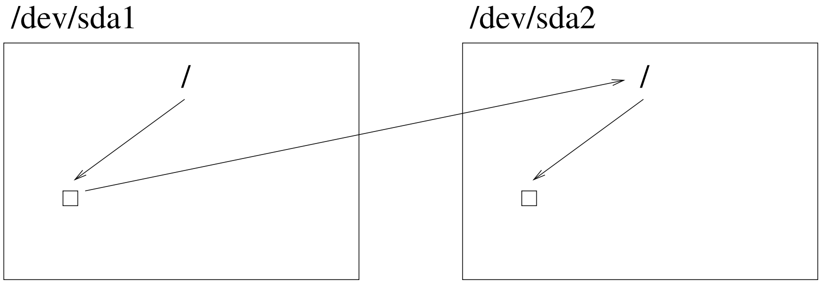
File Systems and Protection

Today's Topics

- Removable File Systems
- Special Files
- Links (Hard and Symbolic)
- FIFOs
- Protection
- Opening files via kernel calls!

Removable File Systems

- Only the / directory is known to Unix at boot time
- Any directory is capable of having a file system mounted on it
- Usually restricted to privileged accounts



Special Files

- Makes physical devices appear to be part of file system hierarchy
- Provides uniform I/O interface (Can read or write just like an ordinary file!)
- Read/Write maps onto direct I/O on the device
- `/dev/sda` or `/dev/sda0` or `/dev/hdb2` or . . .
- `/dev/tty`, `/dev/mem`, `/dev/kmem`, `/dev/null`

Other File Types

Links

- Either allows one file to point to another file (Symbolic Links)
- Or file is accessible via multiple names (Hard Links)

Sockets

- Sockets are interfaces to a network

FIFOs (Named Pipes)

- FIFOs are mechanisms for communication between unrelated processes

Symbolic Links Example

```
$ echo "This␣is␣a␣test␣file." > test_sl.txt
$ cat test_sl.txt
This is a test file.
$ ln -s test_sl.txt test_link
$ ls -la test_sl.txt
-rw-r--r-- 1 galolu cs-stu 21 May 16 15:11 test_sl.txt
$ ls -la test_link
lrwxrwxrwx 1 galolu cs-stu 11 May 16 15:12 test_link -> test_sl.txt
$ cat test_link
This is a test file.
$ echo "Appending␣data." >> test_link
$ cat test_link
This is a test file.
Appending data.
$ rm test_sl.txt
$ cat test_link
cat: test_link: No such file or directory
```

FIFO Example

```
$ mkfifo my_fifo
$ ls -l my_fifo
prw-r--r-- 1 galolu cs-stu 0 May 16 15:21 my_fifo
$ echo "Test␣String" > test_file
$ wc < my_fifo &
[1] 5972
$ cat test_file | tee my_fifo | grep e
Test String
1       2       12
```

Protection

- User and user's processes have associated *user id* and *group id*
- The Unix file system has 12 bits to specify protection
- These bits are specified by the creator of the file
- _ _ _ r w x r w x r w x
    - First three bits discussed later
    - Second group of 3 bits apply if the user is owner of the file
    - Third group of 3 bits apply if the user is not the owner, but is in the group of the owner
    - Last group of 3 bits apply if user is not owner, and user is not in group of owner
- For ordinary files: `r` is read, `w` is write and `x` is execute
- For directories: `r` allows `ls`, `w` allows file creation and deletion and `x` allows pass-through
- `chmod` allows the owner to change protection of file
- Protection bits are expressed as 4 *octal* digits

Protection

- Processes have *real* and *effective* user ids
- Effective user id is used in protection validation
- The leftmost protection bit is for `suid` (set user id)
- If a binary with the `suid` bit set is executed, it will execute with the privilege level of its owner, not the invoker
- The second bit is `sgid` (set group id), and does the same thing as `suid` for groups instead of users
- The third bit is the sticky bit.
  - If on an executable, it will tell the OS to keep the "text" segment of the file in swap so it can be re-executed rapidly
  - If on a directory, it means people who have write access to the directory can create files, but no one can delete files but the owner

Protection Example

```
$ mkdir newdir
$ ls -l
total 168
-rw-r--r-- 1 galolu cs-stu  4754 May 16 14:59 mount.pdf
prw-r--r-- 1 galolu cs-stu     0 May 16 15:21 my_fifo
drwxr-xr-x 2 galolu cs-stu    80 May 16 15:42 newdir
-rw-r--r-- 1 galolu cs-stu  2089 May 16 15:41 slides05.aux
...
$ umask
0022
$ echo "Test" > newdir/newFile
$ cat newdir/newFile
Test
$ ls newdir
newFile
$ chmod u-r newdir
$ ls newdir
ls: cannot open directory newdir: Permission denied
$ cat newdir/newFile
Test
$ chmod u-x newdir
$ cat newdir/newFile
cat: newdir/newFile: Permission denied
```

open() Kernel Call

- Read the manual!
- We have to tell the function what file we want opened, and with what intent.
- Some of the possible flags are:
  - `O_RDONLY` - Open for reading only
  - `O_WRONLY` - Open for writing only
  - `O_RDWR` - Open for reading and writing
  - `O_APPEND` - If set, the seek pointer will be set to the end of the file prior to each write.
  - `O_CREAT` - If the file exists, this flag has no effect. Otherwise, the file is created, owner ID of the file set to the *effective* user ID of the process
  - `O_TRUNC` - If the file exists and is a regular file, and the file is successfully opened via `O_RDWR` or `O_WRONLY`, its length is truncated to zero but the mode and owner are unchanged.

open() Kernel Call

- A successful open call returns a *file descriptor*
- File descriptors, when used to do I/O, access a *currency indicator* stored by the OS to figure out where to start the next I/O operation
- If you open the same file multiple times, you will end up with multiple file descriptors and currency indicators!
- Some sample calls:

-
  ```
  datafile = open("mydata", O_RDWR);

  newfile = open("mydata", O_RDWR|O_CREAT, 0644);
  ```

open() vs. fopen()

- `open()` is a Unix kernel call; `fopen()` is a standard I/O library function
- `open()` returns an int (file descriptor); `fopen()` returns a `FILE *` (an stdio stream)
- Unix I/O kernel calls `[read(),write(),lseek()]` operate on file descriptors; standard I/O functions `[fscanf(),fprintf(),fread(),fwrite(),fseek()]` operate on stdio streams
- User level application programs should use `fopen()` for conventional files; systems programs should use `open()`.
- Never mix and match the two calls!

Next Week

- First assignment!
- Other kernel call I/O functions
- File locking
- Processes