CSci 4061
Introduction to Operating Systems

File Systems: Basics
Chapter 5

File as Abstraction

- Container for related information
- Named
- Associated attributes
- Persistent

Naming a File

```
creat/open (“path/name”, …);
```

Links: files with multiple names

Each name is an alias

```
#include <unistd.h>
int link (const char *original_path,
          const char *new_path)
```

cannot exist as a file already

```
link ("foo", "bar");  // "bar" refers to file "foo"
unlink ("bar");       // remove name "bar"
// if file is open by someone, will not actually get deleted until
// all fd's to it are closed
```

File Attributes: Access to metadata

```
#include <sys/stat.h>
int fstat (int filedes, struct stat *buf)

int stat (const char *pathname,
          struct stat *buf)
```

Structure contains file/directory info:

```
off_t st_size;     // file size
nlink_t st_nlink;  // links
mode_t st_mode;    // type + permission
time_t st_mtime;   // last modification time
```

`fcntl` can also be used to set or get lower-level attrs

Exercise: Metadata

- Write a program that monitors a given file every minute [`sleep(60)`] and if the file size has changed, it outputs the new size to stdout

```
#include <sys/stat.h>
int stat (const char *pathname,
          struct stat *buf)
```

Structure contains file/directory level info:

```
off_t st_size;      // file size
nlink_t st_nlink;   // links
…
```

Example

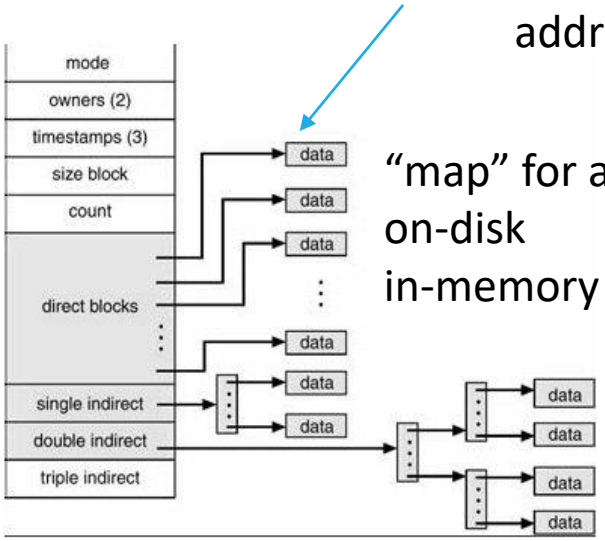
```
void mon_size (char *fname) {
      struct stat sb;
      off_t psize = -99;

      while (1) {




        sleep (60);
      }
}
```

Storing File Meta-data: Unix inode

inode

disk sector address

disk block/sector size typically 4K



"map" for a file
on-disk
in-memory

Set of 12 data pointers – With 4KB blocks => max size of ?

Indirect block pointer – pointer to disk block of data pointers
– 4KB block size => 1K data blocks ptrs => ?

Doubly indirect block pointer
– Doubly indirect block => 1K indirect block ptrs => ?

Triply indirect block pointer
=> 1K doubly indirect blocks => ?

Filesystem

- Directory is a file as well
  - it has an inode
  - what are file contents?

- Filesystem
  - Files
  - Directories
  - Free disk sectors (free list)
  - Root dir

Filesystem (cont'd)

- On-disk organization
  - inode for root dir of filesystem "/" stored in well-known sector on the disk
  - inode for disk sector *free-list* also stored in a well-known sector on the disk
  - inode table or file (inode #, sector)
  - These are stored in the *superblock*

Unix file types/modes

- Indicated by the first character in ls -l
  - - regular file
  - d directory
  - **c character special file**
  - **b block special file**
  - p pipe
  - s socket
  - l symbolic link

File types

- Within `stat` structure:

```
struct_t stat st;
stat ("foo", &st);
```

Macros:

```
int S_ISDIR (st.st_mode);
int S_ISREG (st.st_mode);
int S_ISSOCK (st.st_mode);
…
```

see P. 158 (Table 5.1)

Another look at ls -l

Example:

drwx-xr-x 3 jon fac 4066 Nov 2 09:14 st

file type

hard links

allocation size

File permissions

Operations (r, w, x): read, write, execute

Subjects (u: user/owner, g: group, o: others)

Users may belong to any number of groups (type `groups` at the shell)

File permissions (cont'd)

shell> `ls -l`

drwxr-xr-w 3 jon fac 46

u g o owner group

When a file is created it is given a restricted permission and a default group

You can broaden or further restrict permissions

File permissions (cont'd)

0077
each octal digit is rwx (or 1) for ugo
000, 111, 111

```
#include <sys/types.h>
#include <sys/stat.h>
int chmod (char *path, mode_t mode);
```

prefer symbolic flags: `man fstat, e.g. S_IRGRP : GRouP has Read`)

Also at the command-line (absolute)

```
chmod 0077 st.txt
```

Also at the command-line (relative)

```
chmod go-xr st.txt
chmod u+xrw st.txt
```

Filesystem semantics: Unix

- Two processes open the same file
- Reader sees most recent write
- One reader and one writer – run together
  - File “foo” contains “aaaaaaaaaaaaaaaaaaaaaaaa”

Filesystem semantics (cont'd)

```
// reader.c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

void main () {
   int fd, n;

   char c, buf[100];
   read (0, &c, 1);
   fd = open ("foo", O_RDONLY);
   n = read (fd, buf, 10);
   buf[n] = '\0';
   printf ("buf=%s\n", buf);
   …
}
```

```
// writer.c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

void main () {
   int fd, n;

   char c, buf[100] = "bbbbbbbbbb…";
   read (0, &c, 1);
   fd = open ("foo", O_WRONLY);
   write (fd, buf, 10);
   read (0, &c, 1);
   close (fd) }
```

Power of IDs

- Real user-id: user that actually initiated a process
  - Not executable owner!

  -r-x … 1 jon fac 203 Feb 10 10:47 test

  bill> `/usr/jon/test`

- Effective user-id: user that system associates with the process for purposes of protection
  - Usually the same as the real user-id: this would be?
  - Sometimes want effective user-id to that of the file owner and not the user … why?

Power of IDs (cont'd)

- How do to it?

  jon> chmod u+s test

  -r-s... 1 jon fac 203 Feb 10 10:47 test

bill> `/usr/jon/test`

has priviledge of 'jon'

jon> `chmod g+s`

has privilege of group 'fac'

Masks

```
creat ("my_file", 0777);
```

- Expectation:
  shell> `ls -l`
  -rwxrwxrwx jon … my_file
- Instead:
  -rw-------- …. … my_file

- What happened?
  - To prevent against accidental exposure, Unix sets a default **mask** with your process (type `umask`)
  - Typically: 077 (1 means mask out)

  `creat ("name", PERM & (~mask));` AND
  regular files also mask out execute

Masks (cont'd)

```
creat ("name", PERM & (~mask));
```

`umask` is 022: what is this one?

To change the mask:

```
mode_t umask (mode_t newmask);
```