# CU CS 4995-03: Internet Systems Programming

## Unix System Calls and Libraries for File I/O

©1999 Columbia University

# Overview

- system services
- file creation
- hard and soft links
- directories

# System Services

- operating system (with wrapper): (2), e.g.,
- library: (3?), e.g., `stdio`
- model: $status = f(in, out)$;
- status $< 0 \Rightarrow$ trouble
- `errno`: cause of trouble

Read: Stevens, ch. 1, 2

# System Services

```
if ((n = read(f, buf, sizeof(buf))) < 0) {
  if (errno == EBADF) {
    /* do something special */
  }
  goto error;
}
return 0;

error:
  perror("where"); /* print message */
  /* clean up */
```

## Blocking vs. Non-Blocking

- blocking: wait until finished or interrupt (EINTR) $\Rightarrow$ threads, `select()`
- non-blocking, asynchronous

# File I/O

What operations are needed?

- (almost) everything a file: `/dev/audio`, `/dev/zero`, `/dev/cdrom/`, directories
- not quite true in Windows
- logical stream of bytes
- simultaneous access
- positioning

"file object": permanent ("persistent"), non-typed byte array

# File I/O: open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

fd = open(pathname, oflag, [mode]);
```

| | | |
|---|---|---|
| | `O_RDONLY` | reading only (why?) |
| `oflag =` | `O_WRONLY` | writing only |
| | `O_RDWR` | reading and writing |

# File I/O: open

Flags:

| | |
|---|---|
| create if needed | `O_CREAT` |
| test-and-set | `O_EXCL` |
| file position | `O_APPEND`, `O_TRUNC` |
| synchronization | `O_SYNC` |

# File I/O: Positioning

```
off_t lseek(int fd, offset_t, int whence);
offset_t llseek(int fl, offset_t, int whence);
```

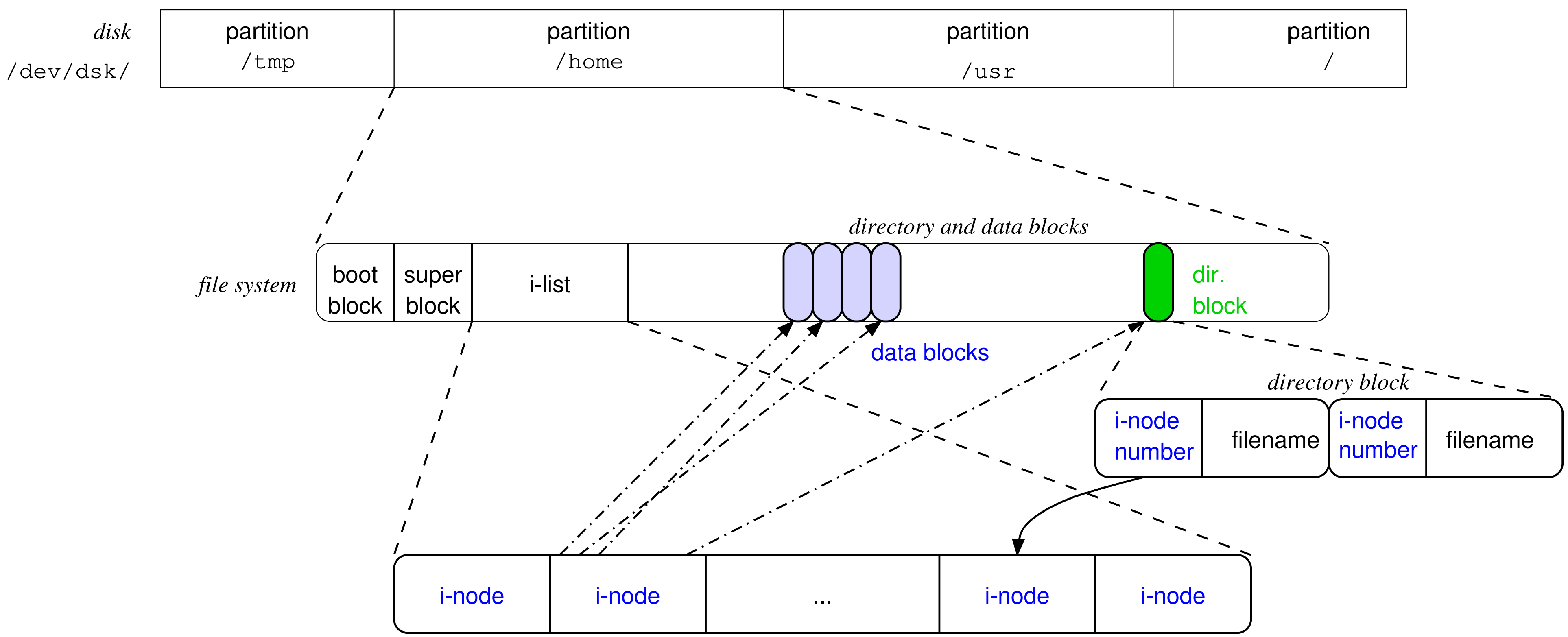
| | |
|---|---|
| from beginning of file | SEEK_SET |
| from current value | SEEK_CUR |
| from end of file | SEEK_END |

- seekable: |?, stdin?
- “holes”
- get current

# File System Implementation and Sharing

- process table $\longrightarrow$ file table
- file table $\longrightarrow$ v-node table
- sharing: append
- atomic creation (`O\_EXCL|O\_CREAT`)
- delete on close only ("private" file: create and unlink)

# File System


disk
/dev/dsk/
partition
/tmp
partition
/home
partition
/usr
partition
/
directory and data blocks
file system
boot
block
super
block
i-list
dir.
block
data blocks
directory block
i-node
number
filename
i-node
number
filename
i-node
i-node
...
i-node
i-node

# File Descriptor Duplication

```
#include <unistd.h>
int dup(int fd);
int dup2(int fd, int new_fd);
```

Motivation?

# File Control

- standard non-OO "pattern":
  `Xcntl(object, parameter, value)`

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int fcntl(fd, cmd, arg, ...);
```

| | |
|---|---|
| `F_DUPFD` | `dup()` |
| `F_GETFD`, `F_SETFD` | get/set flags |
| `F_GETOWN`, `F_SETOWN` | async. owner |
| `F_GETLK`, `F_SETLK` | locks |

# Special Files

- `/dev/tcp`
- `/dev/fd/`$n$
- `/dev/zero`
- `/dev/null`
- `/proc/`

```
as   ctl  lpsinfo lwp/    pagedata root@  usage
auxv cwd@ lstatus map     psinfo   sigact watch
cred fd/  lusage  object/ rmap     status xmap
```

# Missing File System Services

- notification on change
- insertion
- data typing (`file` magic numbers)
- individual access rights ($\rightarrow$ ACLs)
- network ownership
- transactions (modify group files)
- LDAP for user, group ids

# File Types

- regular file (`S_ISREG()`)
- directory file (`S_ISDIR()`)
- character special file (`/dev/`)
- block special file
- FIFO (IPC)
- socket (IPC)
- symbolic link

# File Information

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(const char *pathname, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *pathname, struct stat *buf);
```

# struct stat

```
struct stat {
  mode_t st_mode;   /* file type, mode */
  ino_t  st_ino;    /* inode */
  dev_t  st_dev;    /* device number */
  dev_t  st_rdev;   /* device number */
  nlink_t st_nlink; /* number of links */
  uid_t  st_uid;    /* UID of owner */
  gid_t  st_gid;    /* group ID of owner */
  off_t  st_size;   /* size in bytes */
  time_t st_atime;  /* access time */
  time_t st_mtime;  /* time of last modification */
  time_t st_ctime;  /* time of last status change */
  long st_blksize;  /* best block size */
  long st_blocks;   /* blocks allocated */
}
```

# Set-User-ID and Set-Group-ID

| | |
|---|---|
| real user/group ID | login identity |
| effective user ID | file access checks |
| supplementary group IDs | |
| saved set-user/group-ID | saved by `exec` |

- set *process* effective uid/gid to owner of *file*
- usually, time-limited privileges
- `newgrp` to set group to alternate ones

# File Permissions

| permission bit | file | directory |
|---|---|---|
| r | reading | list filenames |
| x | run | search (needed for path), new file |
| w | writing | create, delete file |

Do not need `r` or `w` for file to delete it! Why?

# File Permissions

Order of checking:

1. superuser (uid 0) $\Rightarrow$ ok
2. process owns file $\Rightarrow$ must have permission
3. groups, supplementary IDs
4. check ’other’ bit

- test access using *real* uid with `access(file, R_OK);`
- usually, access “pyramid”

# Ownership of New Files

- user id = effective user ID
- group id = effective group ID *or* group id of directory

| | |
|---|---|
| SVR4 | set-group-id bit $\longrightarrow$ directory; propagates |
| BSD | directory |
| others | `mount` flag |

# Permissions of New File: `umask`

- file creation mask for process
- ugo: 1 = execute, 2 = write, 4 = read
- bits *on* in umask are *off* in file
- 077 = paranoia setting

```
$ umask
22
```

# Changing Permissions

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod(const char *path, mode_t);
int fchmod(int fd, mode_t);

$ chmod 22 foo
```

- must be owner of file
- set-uid bits turned off if not superuser on write
- works only on open – can't take rights away

## Sticky Bit

- originally, saved “text” in swap
- now, if set for directory, files in directory cannot be removed or renamed except by owner of file or directory
- example: `/tmp`

# chown

```
int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
```

- generally, can't give away files (why?)
- but: system-wide configuration for SVR4
- if changed, reset setuid bits
- changes file or link, not file linked

# Review: Unix Filesystem

- disk drive = $\sum$ partitions
- filesystem: boot block, super block, i-list, directory/data blocks
- i-list: i-nodes, with pointers to data blocks
- inode numbers unique within filesystem
- directory blocks = filename, i-node number (`ls -i`)
- one file $\geq 1$ directory entries (“hard links”, only files)
- symbolic link: file data contains name of file

## Creating and Deleting Links

```
#include <unistd.h>
int link(const char *path, const char *newpath);
int unlink(const char *path);
int remove(const char *path);
int rename(const char *old, const char *new);
```

Files are not deleted until closed (process terminates).

`/a/b` $\longrightarrow$ `a/b/c/d`?

# Symbolic (Soft) Links

- only `lstat`, `lchown`, `remove`, `rename`, `unlink` do not follow link
- symbolic links can dangle:

```
$ ln -s /doesntexist foo
$ ls -l foo
lrwxrwxrwx   1 hgs faculty 12 Sep 22 13:26 foo -> /doesntexist
$ cat foo
cat: foo: No such file or directory
```

- `symlink` creates link
- `readlink(path,buf,bufsize)` reads link

# File Times

| stat field | shows | modified | ls -l |
|---|---|---|---|
| st_atime | data access | read | -u |
| st_mtime | data modification | write | default |
| st_ctime | i-node status | chown, chmod | -c |

- change with utime(char *path, struct utimbuf)
- must be owner
- touch, tar

# Creating and Deleting Directories

```
int mkdir(const char *path, mode_t mode);
int rmdir(const char *path);
```

Can only delete empty directories.

# Traversing the Directory Hierarchy

```
#include <sys/types.h>
#include <dirent.h>
int main(int argc, char *argv[])
{
  DIR *dp;
  struct dirent *d;

  dp = opendir(".");
  while ((d = readdir(dp))) {
    printf("%s: %d\n", d->d_name, d->d_ino);
  }
  closedir(dp);
}
```

# Current Working Directory

Property of process for relative file names:

```
#include <unistd.h>
int chdir(const char *path);
int fchdir(int fd);
```

# Synchronizing Disk and Memory

```
void sync(void);
int fsync(int fd);
```

`fsync` waits for I/O to complete, but `sync` doesn't.

Also need to flush `stdio` buffers to disk: `fflush`; `gets` and similar flush before read. Set stdio buffering via `setbuffer`.