Real-time Operating Systems and Systems Programming

File IO, file and directory management

ARVUTITEADUSE INSTITUUT
DEPARTMENT OF COMPUTER SCIENCE

Topics

- Files
- Unix IO (system calls)
- Standard IO
- File commands

Files in UNIX

- Everything is a file:
  - File is a file
  - Directory is a file
  - FIFO special file (named pipe)
  - Character special file (subtype of a device file)
  - Block special file (subtype of a device file)
  - Symbolic link file

File attributes

Attribute	**Value meaning**
File type	Type of file (regular, directory, fifo, …)
Access permission	File access permissions for different users
Hard link count	Number of hard links of a file
UID	User ID of file owner
GID	Group ID of file
File size	File size in bytes
Last access time	Time the file was last accessed
Last modification time	Time the file was last modified
Last change time	Time the file attribute was last changed
Inode number	Inode number of the file
File system ID	File system ID where the file is stored

NB: File name is not an attribute!

Inode

- File data is held in a structure called inode.
- File-system keeps them in a table called *inode table*.
- Inode number uniquely defines a file

File status

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

- Returns information about the file node
- fstat() is analogous to stat(), but takes the file descriptor as the first argument.
- lstat() is like stat(), but returns the data about symbolic link in stead of linked file

Stat structure

```
#include <sys/stat.h>, <sys/types.h>
struct stat
{
    dev_t         st_dev;      /* device */
    ino_t         st_ino;      /* inode */
    mode_t        st_mode;     /* protection */
    nlink_t       st_nlink;    /* number of hard links */
    uid_t         st_uid;      /* user ID of owner */
    gid_t         st_gid;      /* group ID of owner */
    dev_t         st_rdev;     /* device type (if inode device) */
    off_t         st_size;     /* total size, in bytes */
    unsigned long st_blksize;  /* blocksize for filesystem I/O */
    unsigned long st_blocks;   /* number of blocks allocated */
    time_t        st_atime;    /* time of last access */
    time_t        st_mtime;    /* time of last data modification */
    time_t        st_ctime;    /* time of last inode change */

};
```

There are macros to manipulate the data more easily!

Directory

- In UNIX a directory is a file which contains pairs:
  - inode: name
- It contains the files "." and "..", which correspond to the same directory and to the one above the current, correspondingly. They contain their inodes.
- Root directory / has .. file which points to itself
- If inode is marked as 0, the entry is free to write

Unix IO

- Why necessary to know
  - Helps to understand other concepts such as process creation anomalies
  - Sometimes necessary to use: file metadata, network programming risks

File Descriptor

- File for a process is a small positive integer named "File Descriptor"
- Sometimes the word "channel" is used.
- Predefined descriptors:
  - 0 standard input
  - 1 standard output
  - 2 standard error
- Descriptor is a file table index

File sharing

- Open files in kernel are in 3 structures:
  - Descriptor Table: unique for a process points to:
  - File Table: shared by processes, holds file position, reference count, points to V-node table:
  - V-node Table: shared by processes, basically holds most of stat() information
- Fork() copies Descriptor Table

Unix IO

- File: sequence of bytes
- Open files: open()
- Change current file position: seek()
- Read and write: read(), write()
- Close: close()

Notes about read() and write()

- They return how many bytes were moved.
- Sometimes these calls return before you have sent all of the data (network reading would be a prime example)
- Buffering happens only on file system level

fopen(), fread(), fwrite() etc

- Wrap open(), read(), write() to create streams
- They are buffered and therefore preferred.
- Stream from fileno:

```
FILE *fdopen (int fd, const char *mode);
```

- File number from stream:

```
int fileno(FILE *stream);
```

dup(), dup2(), dup3

- Copy the open file descriptor
- dup() - new descriptor is the lowest one
- dup2(int oldfd, int newfd) – close new;
  copy old to new
- dup3(int oldfd, int newfd, int flags) –
  close new; copy old to new
  flags : O_CLOEXEC – prevents race conditions on threaded programs

Example of output redirection

- Open file to get descriptor
- Use dup2() to replace descriptor of stdout (1)
- Printf sends data to a file now
- If you exec a program; output also sent to file

What to use

- Use Standard IO calls if you can
- Use Unix IO if you must
  - Meaning: mostly for networking & control/speed

Hardlink

```
int link(const char *oldpath, const char *newpath);
```

- link() hardlinks oldpath to newpath.
- When newpath exists, it is not overwritten
- The new file acts as a pointer to data. The files are identical as they point to the same inode.
- Returns 0 on success, -1 on error and sets an errno

Softlink

```
int symlink(const char *oldpath, const char
  *newpath);
```

- Creates a softlink (or Symbolic link)
- Acts like a Windows shortcut and can be either:
  - Relative: ../text.txt
  - Absolute: /home/irve/text.txt

unlink()

```
int unlink(const char *pathname);
```

- Changes the inode of pointed file to 0, then decreases the reference counter to the file. If the counter reaches 0, deallocates the data.
- PS! In reality the data gets replaced only when the last process which uses the file is stopped. You can open a temporary file, unlink it and still access it!

File permissions

```
int access(const char *pathname, int mode);
```

- access() checks if the processes have the privilege to access the file for either writing, reading or verifying its existence.
- mode is a bitmask of permissions R_OK, W_OK, X_OK, ja F_OK (latter the existence of the file)
- Returns 0, when the access is granted, -1 otherwise.

Granting access to files

```
int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);
```

- Defines the file permissions to the file
- mode is the result of OR with the following constants:

```
#define S_IRWXU 0000700    /* RWX mask for owner */
#define S_IRUSR 0000400    /* R for owner */
#define S_IWUSR 0000200    /* W for owner */
#define S_IXUSR 0000100    /* X for owner */
#define S_IRWXG 0000070    /* RWX mask for group*/
#define S_IRGRP 0000040    /* R for group */
#define S_IWGRP 0000020    /* W for group */
/* jne... */
```

Setting the owner

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

- Change owner and/or group of the file pointed by its path or file descriptor
- Owner can only be changed by the root user.
- User can set the group of the file to the groups to which it belongs.
- The root can set the group as needed

Permissions to new files

```
mode_t umask(mode_t mask);
```

- Umask sets permissions to the new files
  umask = mask & 0777
- The bits set in umask are *removed.*
- Permission bits = mode & ~(umask)
- Widely used umask is 022, which creates files with permissions of 0666 & ~022 = 0644 = rw-r--r--
- The function always succeeds and it returns the previous umask value.

Directory management

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

- mkdir() creates a new directory

```
int rmdir(const char *path);
```

- rmdir() removes the directory
- The directory must be empty and can only contain the files . and ..

Working directory

```
int chdir(const char *path);
int fchdir(int fildes);
```

- chdir() changes the working directory to the pointed one.
- Returns a pointer to current working directory

```
char *getcwd(char *buf, size_t size);
```

- buffer size must be at least 1 larger than directory name.
- When buf is NULL, the function allocates size bytes, otherwise stores it to pointer

Directory Content Stream

- You can open directory streams

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *dirname);
```

- Opens dirname as a stream. Positioned on the first entry.

```
int closedir(DIR *dirp);
```

- Close the stream *stream*

Listing stream entries

```
struct dirent *readdir(DIR *dirp);
```

- Return the structure to dirent pointer, which contains the current entry
- NULL is returned after the last entry
  - <dirent.h> file describes the data structure
  - readdir() overwrites the data after running
  - POSIX standard states that dirent contains the field char d_name[], which has no determined length, to a maximum of NAME_MAX chars, null terminated. Other fields are not portable, while d_namelen field is often present

If you can read this the lecture is over