Linux File I/O

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Introduce Linux File I/O

Buffered I/O vs system calls

Opening Files

File I/O in Linux

- Files are opened via file descriptors (fd)
  - represented by an integer
- Kernel maintains a Per-process list of files - file table
  - List indexed by fds

File I/O in Linux

- Every process has at least 3 file descriptors open
  - stdin (fd 0 or STDIN_FILENO)
    - terminal input device
  - stdout (fd 1 or STDOUT_FILENO)
    - terminal display
  - stderr (fd 2 or STDERR_FILENO)
    - terminal display

Opening and Accessing Files

- open() (possibly with O_CREAT)
- read()
- write()
- close()
- What about fopen, fwrite, fread?

Opening and Accessing Files

- What about fopen, fwrite, fread?
  - “f” prefix relate to buffered I/O, not system calls
  - non “f” prefix versions listed here are the underlying system calls.

fopen vs open()

```
FOPEN(3)                                                Linux Programmer's Manual

NAME
       fopen, fdopen, freopen - stream open functions

SYNOPSIS
       #include <stdio.h>

       FILE *fopen(const char *pathname, const char *mode);
```

```
OPEN(2)                                                 Linux Programmer's Manual

NAME
       open, openat, creat - open and possibly create a file

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

       int open(const char *pathname, int flags);
       int open(const char *pathname, int flags, mode_t mode);
```

- Buffered IO uses FILE*
- syscall open() uses file descriptor

http://man7.org/linux/man-pages/man3/fopen.3.html
http://man7.org/linux/man-pages/man2/open.2.html

fopen vs open()

```
FOPEN(3)                                                Linux Programmer's Manual

NAME
       fopen, fdopen, freopen - stream open functions

INTRO(3)                                Linux Programmer's Manual                                INTRO(3)

NAME
       intro - introduction to library functions

DESCRIPTION
       Section  3  of the manual describes all library functions excluding the library functions (system call wrappers)
       described in Section 2, which implement system calls.
```

```
OPEN(2)                                                 Linux Programmer's Manual

NAME
       open, openat, creat - open and possibly create a file
```

```
INTRO(2)                                                Linux Programmer's Manual

NAME
       intro - introduction to system calls

DESCRIPTION
       Section  2  of  the manual describes the Linux system calls.  A system call is an
       are not invoked directly: instead, most system calls have corresponding C library
       trapping  to kernel mode) in order to invoke the system call.  Thus, making a sys
       tion.
```

- Located in different man page sections

http://man7.org/linux/man-pages/man2/intro.2.html
http://man7.org/linux/man-pages/man2/intro.3.html

fopen vs open()

- openat() is eventually called from fopen()

```
extern FILE *_IO_new_fopen (const char*, const char*);
#   define fopen(fname, mode) _IO_new_fopen (fname, mode)
```

```
FILE *
_IO_new_fopen (const char *filename, const char *mode)
{
  return __fopen_internal (filename, mode, 1);
}
```

```
FILE *
__fopen_internal (const char *filename, const char *mode, int is32)
{
```

```
  _IO_no_init (&new_f->fp.file, 0, 0, &new_f->wd, &_IO_wfile_jumps);
  _IO_JUMPS (&new_f->fp) = &_IO_file_jumps;
  _IO_new_file_init_internal (&new_f->fp);
  if (_IO_file_fopen ((FILE *) new_f, filename, mode, is32) != NULL)
    return __fopen_maybe_mmap (&new_f->fp.file);
```

```
FILE *
_IO_file_open (FILE *fp, const char *filename, int posix_mode, int prot,
               int read_write, int is32not64)
{
  int fdesc;
  if (__glibc_unlikely (fp->_flags2 & _IO_FLAGS2_NOTCANCEL))
    fdesc = __open_nocancel (filename,
                             posix_mode | (is32not64 ? 0 : O_LARGEFILE), prot);
  else
    fdesc = __open (filename, posix_mode | (is32not64 ? 0 : O_LARGEFILE), prot);
  if (fdesc < 0)
    return NULL;
  fp->_fileno = fdesc;
```

```
return SYSCALL_CANCEL (openat, AT_FDCWD, file, oflag, mode);
```

https://github.com/bminor/glibc/blob/master/include/stdio.h
https://github.com/bminor/glibc/blob/master/libio/iofopen.c
https://github.com/bminor/glibc/blob/master/libio/fileops.c
https://github.com/bminor/glibc/blob/master/sysdeps/unix/sysv/linux/open.c

fopen vs open()

- What's the difference between fopen() and open()?
  - f prefix functions from <stdio.h> provide platform-independent user-buffering.
- What is file buffering?
  - Rather than reading/writing a full block, read from or write to copies in memory.
  - Improves performance.

fopen vs open()

- What do we mean by user-buffering?
  - Happens in userspace, rather than in the kernel.
  - Reduce the amount of system calls into the kernel.

Opening Files

```
OPEN(2)                                          Linux Programmer's Manual

NAME
       open, openat, creat - open and possibly create a file

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

       int open(const char *pathname, int flags);
       int open(const char *pathname, int flags, mode_t mode);
```

- open()
  - maps pathname to file descriptor
  - Include open flags argument
    - access - (read only, read/write create, append, synchronous I/O, etc)
  - mode argument used when creating
    - read/write/execute permissions for user/group/other

Linux System Programming Chapter 2
http://www.gnu.org/software/libc/manual/html_mono/libc.html#Opening-and-Closing-Files