CSE333 Section 4: Non-STDIO POSIX Functions

Aryan Naraghi

What Are We Talking About?

- In class we talked about STDIO functions:
  - fopen(), fread(), fwrite(), etc.
- These make use of file handles with types like FILE* or DIR*
- But those are just pretty wrappers for the real deal…

What’s in a POSIX FILE?

From /usr/include/libio.h:

```
struct _IO_FILE {
    int _flags;
    ...
    int _fileno;
    ...
};
```

File Descriptors

- Under the hood of those stdio functions are *file descriptors*:
  - Integer handles to files; stdio functions wrap them with usermode buffers
  - In the kernel, literally an index into a per-process array of OS file structures
  - Used in place of FILE* or DIR* for unbuffered IO (sort of: the kernel does some buffering of its own)

open()

- int open(const char* pathname, int flags)
- Returns a file descriptor
- Uses flags like O_RDONLY, O_WRONLY, O_RDWR instead of “r” etc.

read()

- ssize_t read(int fd, void *buf, size_t count)
- Reads up to *count* bytes into address *buf* from the file with handle *fd*, and returns # bytes read
- Has some surprising failure modes…

read() Returns

On success, **the number of bytes read is returned** (zero indicates end of file), and the file position is advanced by this number. **It is not an error if this number is smaller than the number of bytes requested;** this may happen for example because fewer bytes are actually available right now (maybe because we were close to end-of-file, or because we are reading from a pipe, or from a terminal), or because read() was interrupted by a signal. **On error, -1 is returned, and errno is set appropriately.** In this case it is left unspecified whether the file position (if any) changes.

From “man 2 read”

What Errors Might Read Encounter?

- EBADF – Bad file descriptor
- EFAULT – Output buffer is outside your address space
- EINTR – A signal was encountered, and no data was read
  - This is not an error!
- And more…

How to Really Get N bytes with read()

```
#include <errno.h>
#include <unistd.h>
...
    char *buf = ...;
    int bytes_left = n;
    int result = 0;
    while(bytes_left > 0) {
        result = read(fd, buf + (n-bytes_left), bytes_left);
        if (result == -1 && errno != EINTR) {
            // Real error, return error result
        } else if (result == -1) {
            result = 0;
        }
        bytes_left -= result;
    }
```

write()

- ssize_t write(int fd, const void* buf, size_t count);
- Similar to read(), with similar funny success modes:
  - Can return that it only wrote part of the buffer
  - Can return -1 with error EINTR, which is not an error

close()

- int close(int fd);
- Closes a file descriptor
- Can return -1 for errors
  - Could also set errno to EINTR, which means you need to try again!!!

opendir(), readdir(3)

- DIR* opendir(const char *name);
- struct dirent *readdir(DIR *dirp);
- opendir() opens a directory the way fopen opens a regular file
- readdir() returns a pointer to the next (statically allocated) directory entry
- Docs in “man opendir” and “man 3 readdir”
- Like the other stdio functions, file handles lurk under the hood…

readdir(2)

- open() *can be used on directories*
- read() can’t, so we use readdir(2)
- int readdir(unsigned int fd, struct old_linux_dirent *dirp, unsigned int count);
- The man page (man 2 readdir) will tell you this was superceded by getdents()…

getdents()

- int getdents(unsigned int fd, struct linux_dirent *dirp, unsigned int count);
- Slightly better behaved than readdir(2)
- Still one of the most painful syscalls to use
  - Not declared in a header; must call directly with the syscall() function
  - The man page (man getdents) contains an example in all its horror

readdir(3) vs. readdir_r(3)

- readdir() returns a DIR* for the next directory entry
  - Subsequent calls may return the *same memory*!
  - Means if you iterate through multiple directories at the same time, bad things can happen…
- See “man readdir_r” for details if you use readdir from multiple threads, or need to see the dirent for more than one file at a time.

fsync()

- int fsync(int fd);
- Flushes the contents of a file out of the OS’s cache, all the way to disk
  - Crucial for databases
  - Assumes the OS doesn’t lie to you
    - Assumes the HDD doesn’t lie to the OS…

Using File Descriptors with STDIO

- There are conversion functions
- FILE* fdopen(int fd, const char* mode);
- DIR* fdopendir(int fd);