

CSE 333 – SECTION 3

POSIX I/O Functions

Basic File Operations

- Open the file
- Read from the file
- Write to the file
- Close the file / free up resources

System I/O Calls

```
int open(char* filename, int flags);
```

Returns an integer which is the file descriptor.

Returns -1 if there is a failure.

filename: A string representing the name of the file.

flags: An integer code describing the access.

O_RDONLY -- opens file for read only

O_WRONLY -- opens file for write only

O_RDWR -- opens file for reading and writing

O_APPEND --- opens the file for appending

O_CREAT -- creates the file if it does not exist

O_TRUNC -- overwrite the file if it exists

System I/O Calls

```
ssize_t read(int fd, void *buf, size_t count);  
ssize_t write(int fd, const void *buf, size_t count);
```

fd: file descriptor.

buf: address of a memory area into which the data is read.

count: the maximum amount of data to read from the stream.

The return value is the actual amount of data read from the file.

```
int close(int fd);
```

Returns 0 on success, -1 on failure.

[man 2 read]
[man 2 write]
[man 2 close]

Reading a file

```
#include <errno.h>
#include <unistd.h>

char *buf = ...;
int bytes_read = 0;
int result = 0;

int fd = open("filename", O_RDONLY);

// BUG: if filename is smaller than N, infinite loop!
while (bytes_read < N) {
    result = read(fd, buf + bytes_read, N - bytes_read);
    if (result == -1) {
        if (errno != EINTR) {
            // a real error happened, return an error result
        }
        // EINTR happened, do nothing and loop back around
        continue;
    }
    bytes_read += result;
}
```

Errors

- When an error occurs, the error number is stored in `errno`, which is defined under `<errno.h>`
- View/Print details of the error using `perror()` and `errno`.
- POSIX functions have a variety of error codes to represent different errors.
- Some common error conditions:
 - **EBADF** - *fd* is not a valid file descriptor or is not open for reading.
 - **EFAULT** - *buf* is outside your accessible address space.
 - **EINTR** - The call was interrupted by a signal before any data was read.
 - **EISDIR** - *fd* refers to a directory.

Why learn these functions?

- They are unbuffered. You can implement different buffering/caching strategies on top of read/write.
- More explicit control since read and write functions are system calls and you can directly access system resources.
- There is no standard higher level API for network and other I/O devices.

STDIO vs. POSIX Functions

- User mode vs. Kernel mode.
- STDIO library functions – *fopen*, *fread*, *fwrite*, *fclose*, etc. use FILE* pointers.
- POSIX functions – *open*, *read*, *write*, *close*, etc. use integer file descriptors.
- Think about levels of abstraction

Standard I/O Calls

- Read the man pages!
 - `[man 3 stdio]` for a full list of functions declared in `<stdio.h>`
- The most important (for you):
 - `fopen`
 - `fclose`
 - `fread` (!! the return value is probably not what you think !!)
 - `fwrite` (!! the return value is probably not what you think !!)
 - `fseek`
 - Be sure to check out some of the others though! You might just find something interesting and/or useful!

Directories

- Accessing directories:
 - Open a directory
 - Iterate through its contents
 - Close the directory
- Opening a directory:

```
DIR *opendir(const char* name) ;
```

- Opens a directory given by `name` and provides a pointer `DIR*` to access files within the directory.
- Don't forget to close the directory when done:

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

[man 0P dirent.h]

[man 3 opendir]

[man 3 closedir]

Directories

- Reading a directory file.

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

```
struct dirent {
    ino_t      d_ino;      /* inode number for the dir entry */
    off_t      d_off;      /* not necessarily an offset */
    unsigned short d_reclen; /* length of this record */
    unsigned char d_type;   /* type of file (not what you think);
                             not supported by all file system types */
    char        d_name[NAME_MAX+1]; /* directory entry name */
};
```

[man 3 readdir] but not
[man readdir]

Section Exercise

- Find a partner if you wish.
1. Write a C program that does the following:
 - Given two files on the command line, print the contents of the first file to the second.
 - On error, print an informative error message.
 - Similar to cat.
 - You must use the POSIX functions to open, close, read and write.
 2. Given a directory name as an argument, print the name of the directory entries to stdout.