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, int mode);
```

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

`mode`: File protection mode. Ignored if O_CREAT is not specified.

[man 2 open]

System I/O Calls

```
ssize_t read(int fd, char *buffer, size_t bytes);
ssize_t write(int fd, char *buffer, size_t bytes);
```

`fd:` file descriptor.

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

`bytes:` 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]

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.

```
[man 3 errno]
[man 3 perror]
```

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`**
  - **`fwrite`**
  - **`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(char* dir_name);
```

  - Opens a directory given by `dir_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);
```

.

- returns NULL on reaching the end of the directory stream or
- if an error occurred

```
struct dirent {
 int_t d_ino; /* i-node number for the dir entry */
 u_short d_reclen; /* length of this record */
 off_t d_off; /* offset to the next dirent*/
 unsigned char d_type; /* type of file; not supported by all
                         file system types */
 char d_name[NAME_MAX+1] ; /* directory entry name */
};
```

[man 3 readdir] or
[man 3 readdir_r] but not
[man readdir]

Directories

- Reading a directory file.

```
int readdir_r(DIR *dirp, struct dirent *entry,
    struct dirent **result);
```

- returns 0 on success.
- A NULL pointer is returned in `*result` when the end of the directory is reached.

```
struct dirent {
 u_long d_ino; /* i-node number for the dir entry */
 u_short d_reclen; /* length of this record */
 off_t d_off; /* offset to the next dirent*/
 unsigned char d_type; /* type of file; not supported by all
                          file system types */
 char d_name[NAME_MAX+1] ; /* directory entry name */
};
```

[man 3 readdir] or
[man 3 readdir_r] but not
[man readdir]

Section Exercise

- Find a partner if you wish.
- Write a C program that does the following:
  - Given a command line argument, if it is an ordinary file, print its contents to stdout.
  - If not, or some other error occurs, print an informative error message using perror().
  - Similar to cat.
  - You must use the POSIX functions to open, close, read and write.