CSE333 – SECTION 3

Non-STDIO POSIX Functions

Contents

- STDIO vs. POSIX Functions
  - read/write Behavior and Errors
- Working with Directories
- Section Exercise

STDIO vs. POSIX Functions

- Recall the exercise from section 1
  - fopen(), fread(), fwrite(), fclose() from stdio.h
- fopen() returns a FILE*
  - Used for buffered IO
- Under the hood, these contain a file descriptor
  - An integer that indexes a table in the OS that keeps track of any state associated with open files
- POSIX system calls
  - open(), read(), write(), close()
  - Very low level
  - Uses file descriptors instead of FILE* for unbuffered IO
  - The STDIO functions are implemented using these functions

read()/write() Behavior and Errors

- read() returns the number of bytes read
  - May be less than you asked for
- It's the same with write()
- Furthermore, there is an error that isn't really an error!
  - EINTR indicates that the call was interrupted by a signal handler
  - This just means that you should try again
- So how do you get all N bytes you asked for?
  - We still need one more piece

Checking for Errors

- If the POSIX functions encounter an error, they return -1
- In addition to this, errno, a global variable, is set to give more information about what went wrong
  - Many C library functions use this variable as well
- #include <errno.h> to access errno and check it against various error codes
- EINTR is the only one we'll worry about right now
  - See the man pages for other errors you could encounter
  - `man 3 errno` for more general info on this variable

Reading N Bytes From a File

```
#include <errno.h>
#include <unistd.h>  // for POSIX functions
…
char buf[N+1];
int bytes_read = 0;
while (bytes_read < N) {
  int result = read(fd, buf + bytes_read, N - bytes_read);
  if (result == -1) {
    if (errno != EINTR)
      // real error, handle appropriately
    else
      result = 0;
  }
  bytes_read += result;
}
buf[N] = '\0';
```

Working with Directories

- First, let's get some information about the file with stat()
  - `man 2 stat`
- Once we know that a file is a directory, we can try to open it with opendir()
  - `man 3 opendir`
- Assuming we were able to open it, we'll use readdir() to get its contents, one at a time
  - `man 3 readdir`
- Lastly, we'll use closedir() when we're done with it
  - `man 3 closedir`

Section Exercise

- Find a partner if you wish
- Part I: (required)
  - 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.
  - Similar to cat
  - You must use the POSIX functions, but printf is fine for output
- Part II: (optional)
  - Given a command line argument, if it is a directory name, list the names of all the files in the directory.
  - A very basic ls