

# CSE 333

## Lecture 8 - low-level I/O

**Hal Perkins**

Department of Computer Science & Engineering

University of Washington



# Lower-level file access

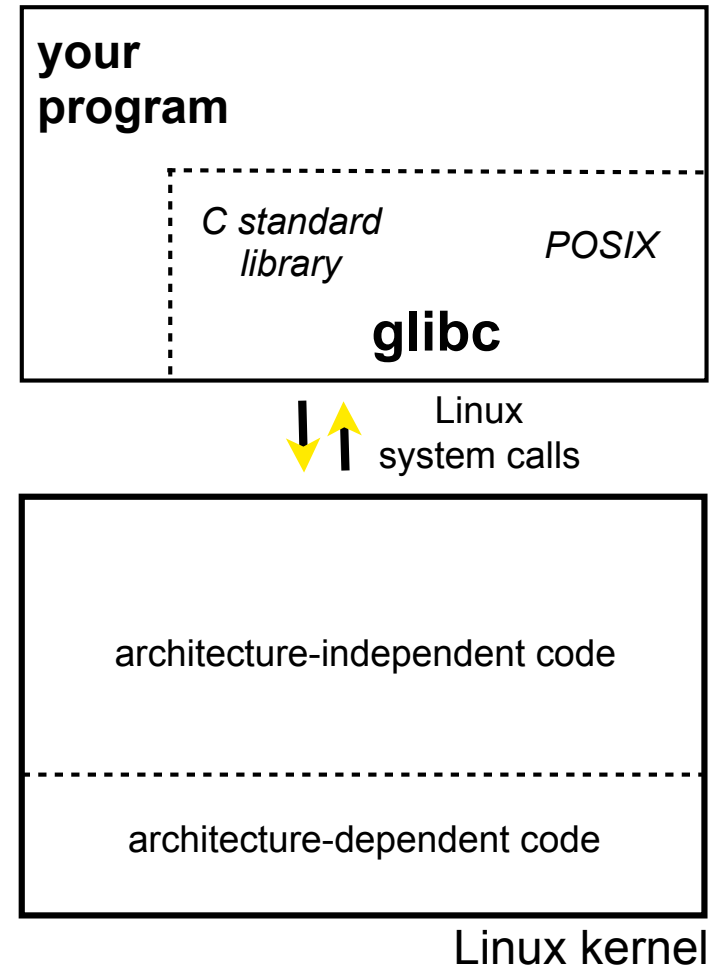
Remember this picture?

your program can access  
many layers of APIs

C standard library

POSIX compatibility API

underlying OS system calls



# So far...

You've used the C standard library to access files

specifically, `fopen`, `fread`, `fwrite`, `fclose`, `fseek`

these provide a (`FILE *`) stream abstraction

These are convenient and portable...

but, they are *buffered*

and, they are implemented by using lower-level OS calls

# Lower-level file access

Most UNIX-en support a common set of lower-level file access APIs

- open, read, write, close, lseek

- similar in spirit to their fopen (etc.) counterparts

- but, lower-level and unbuffered

- (well, unbuffered from user's perspective; OS does its own buffering at least for disk blocks)

- and, less convenient

- you will have to use these for network I/O, so we might as well learn them now

# open / close

To open a file...

pass in the filename and access mode, similar to fopen

get back a “file descriptor”

similar to a (FILE \*) from fopen, but is just an int

```
#include <fcntl.h>

...

int fd = open("foo.txt",
              O_RDONLY);
if (fd == -1) {
    perror("open failed");
    exit(EXIT_FAILURE);
}

...

close(fd);
```

# Reading from a file

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

returns the # of bytes read

might be fewer bytes than you requested (!!!)

returns 0 if you're at end-of-file

return -1 on error

warning: read has some very surprising error modes!

# read( ) error modes

On error, the “errno” global variable is set

you need to check it to see what kind of error happened

What errors might read( ) encounter?

EBADF -- bad file descriptor

EFAULT -- output buffer is not a valid address

EINTR -- read was interrupted, please try again

argh!!!

and many others

# How to read( ) n bytes

```
#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) {
        if (errno != EINTR) {
            // a real error happened, return an error result
        }
        // EINTR happened, do nothing and loop back around
        continue;
    }
    bytes_left -= result;
}
```

# Other low-level functions

Read the man pages to learn about:

**write()** -- write data

**fsync()** -- flush data to the underlying device

**opendir()**, **readdir()**, **closedir()** -- get a directory listing

make sure you read the section 3 version, e.g.:

man 3 opendir

kind of painful to use

# A useful cheat-sheet

From a CMU systems programming course:

<http://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture24.pdf>

See you on Friday!

But wait! There's more!!