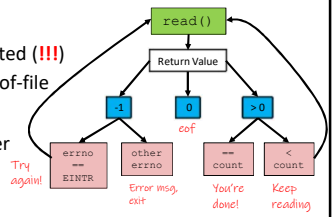
Reading from a File

Stores read result in buf

Number of bytes

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

- Returns the number of bytes read
  - Might be fewer bytes than you requested (!!!)
  - Returns 0 if you're already at the end-of-file
  - Returns -1 on error (and sets errno)
  - Advances forward in the file by number of bytes read



- There are some surprising error modes (check errno)
  - EBADF: bad file descriptor
  - EFAULT: output buffer is not a valid address
  - EINTR: read was interrupted, please try again (ARGH!!!! 😵😠)
  - And many others…

Defined in errno.h

Poll Everywhere

pollev.com/cse33320su

- Let's say we want to read 'n' bytes. Which is the correct completion of the blank below?

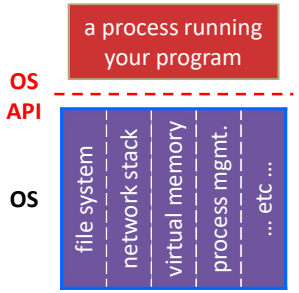
```
char* buf = ...;  // buffer of size n
int bytes_left = n;
int result;       // result of read()

while (bytes_left > 0) {
  result = read(fd, ______, bytes_left);
  if (result == -1) {
    if (errno != EINTR) {
      // a real error happened,
      // so return an error result
    }
    // EINTR happened,
    // so do nothing and try again
    continue;  Keyword that jumps to beginning of loop
  }
  bytes_left -= result;
}
```

A. buf
B. buf + bytes_left
C. buf + bytes_left - n
D. buf + n - bytes_left
E. We're lost…

OS: Abstraction Provider

- The OS is the "layer below"
  - A module that your program can call (with system calls)
  - Provides a powerful OS API – POSIX, Windows, etc.



File System
- open(), read(), write(), close(), …

Network Stack
- connect(), listen(), read(), write(), …

Virtual Memory
- brk(), shm_open(), …

Process Management
- fork(), wait(), nice(), …

OS: Protection System

- OS isolates process from each other
  - But permits controlled sharing between them
    - Through shared name spaces (*e.g.* file names)
- OS isolates itself from processes
  - Must prevent processes from accessing the hardware directly
- OS is allowed to access the hardware
  - User-level processes run with the CPU (processor) in unprivileged mode
  - The OS runs with the CPU in privileged mode
  - User-level processes invoke system calls to safely enter the OS

There are special cases where "super-user" permissions granted