Agenda for Today

1. Bug of the Day
2. Lab Q&A
3. New Assignment
4. Filesystems
Bug of the Day
What's wrong here?

Server

```c
int fd = server_socket_accept(server_socket_fd);
// Error check omitted
char buffer[MESSAGE_LENGTH];
read(fd, buffer, MESSAGE_LENGTH);
```

Client

```c
int fd = socket_connect(server_name, server_port);
// Error check omitted
char buffer[MESSAGE_LENGTH];
read(fd, buffer, MESSAGE_LENGTH);
```

Should this be write? We would need to initialize buffer first.
The log message in the example solution showed the port number. How can we display that with `ui_display` when it only takes a string? If you remember back to the memory allocator lab, we could use `snprintf` to format a string into a buffer instead of writing it to a stream. You can do the same here, then pass that string to `ui_display`.

Can we use a thread for each connected client to make sure the whole program isn’t blocked? Yes, that’s a good approach. It’s really close to the approach in the server-poll.c example from the event-based concurrency day last week.

Why does `ui_display` have an else case that just calls `printf`? I didn’t intend for you to use this path, but it’s there so if you forget to call `ui_run`, this function still does something.
P2P Chat Lab Q&A
P2P Chat Lab Q&A
New Assignment
Filesystems
File Basics

What is a file?

A file holds data, which is a sequence of bytes. A file is associated with one or more regions on disk that actually store its contents. Each file also has metadata that includes permission information that specifies who can access the file, and how.

What operations can you perform on a file?

- **open** - use this function to gain access to a file’s content through a file descriptor handle.
- **read** - transfer the bytes in the file’s storage to program memory.
- **write** - transfer bytes from program memory to the file’s stored content.
- **execute** - start a new program using a file as its executable.
- **close** - stop using the file, and stop using the entry in the file descriptor table.
- **delete** - not really an operation on the file, but it’s relevant to files.
- **metadata operations** - getting or setting metadata for the file, including permissions.
Directory Basics

What is a directory?

A directory is like a file, but it contains a list of links instead of a sequence of bytes. Each link is a name, and then a connection to a specific inode (which could be a file or directory, or something else).

What operations can you perform on a directory?

- **link** - add a link to a directory which adds an entry in that directory that reaches a file.
- **unlink** - remove a link from a directory
- **list** - get a list of the files in a directory
- **access** - check for a specific entry in a directory by trying to use it
- **modify permissions** - the same way we could set permissions for files, we can that for directories.
The UNIX Philosophy
Filesystem Basics

What is a filesystem's job?
Filesystems should provide access to files, connect them up with storage, and perform permissions checks for each file access. Some systems have a separate filesystem for each storage device (like Windows). In UNIX-like systems, there is one filesystem where storage devices are *mounted*.

What are the basic components of the simple filesystem we read about?

- **blocks** - a unit of storage that a filesystem uses to hold data or other state
- **bitmaps** - a specific use of a block that can track whether other blocks are free or not
- **inodes** - records associated with individual files/directories on the filesystem
- **data region** - a range of blocks that actually store file data
- **superblock** - a special block at the start of a filesystem that describes its structure