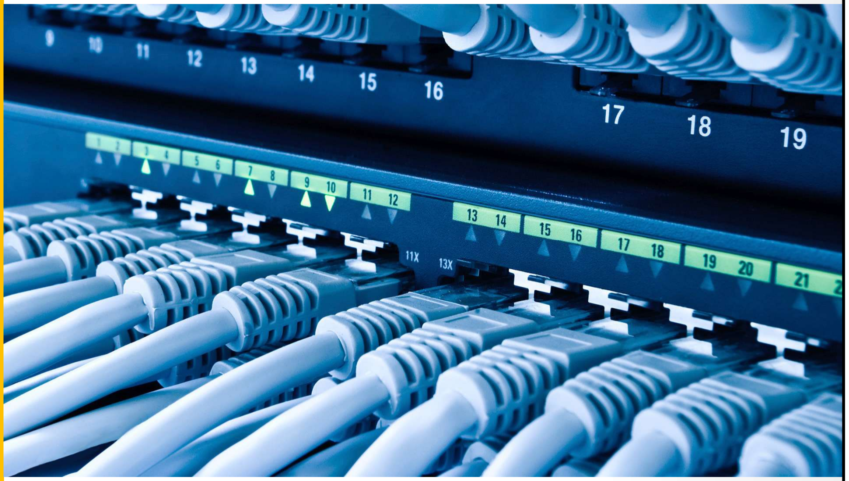
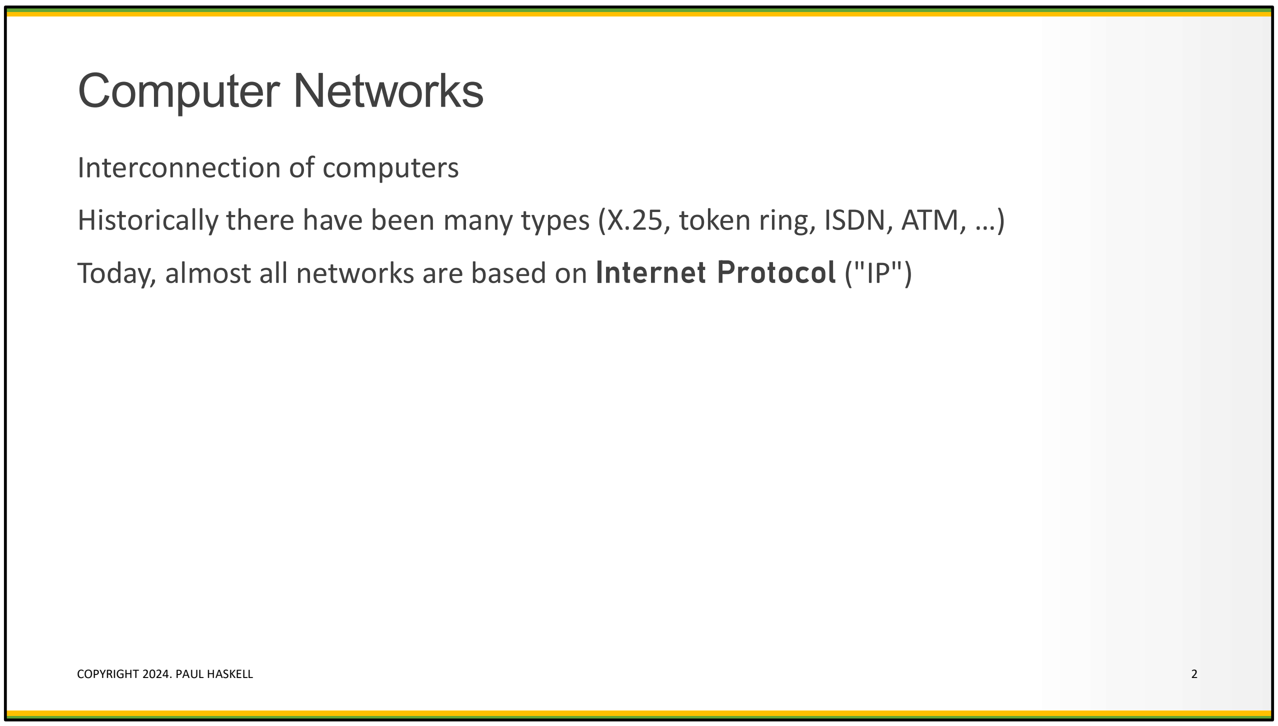
CS221
C and Systems Programming



1



Computer Networks

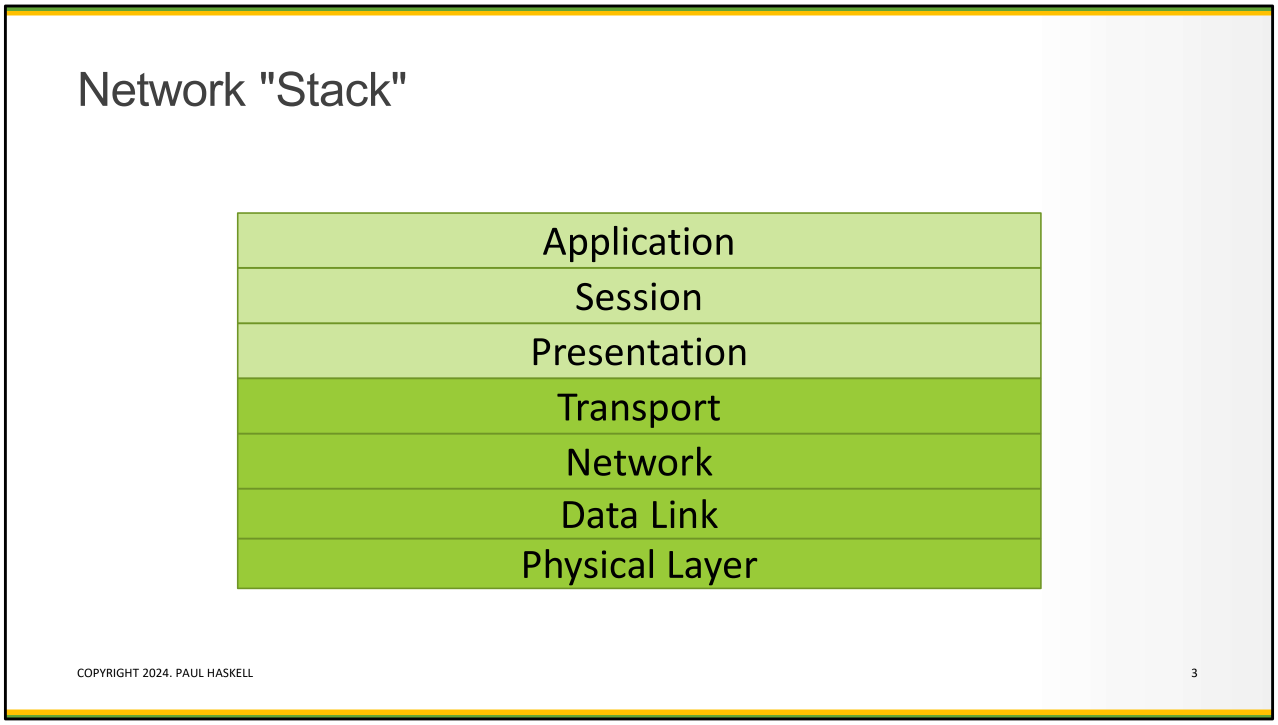
Interconnection of computers

Historically there have been many types (X.25, token ring, ISDN, ATM, …)

Today, almost all networks are based on **Internet Protocol** ("IP")

2

What kind of applications run across computer networks? Brainstorm:
email, web browsing, multiplayer games, media streaming, Git, videoconferencing, time-of-day, file sharing, virtual computing, remote system monitoring, …

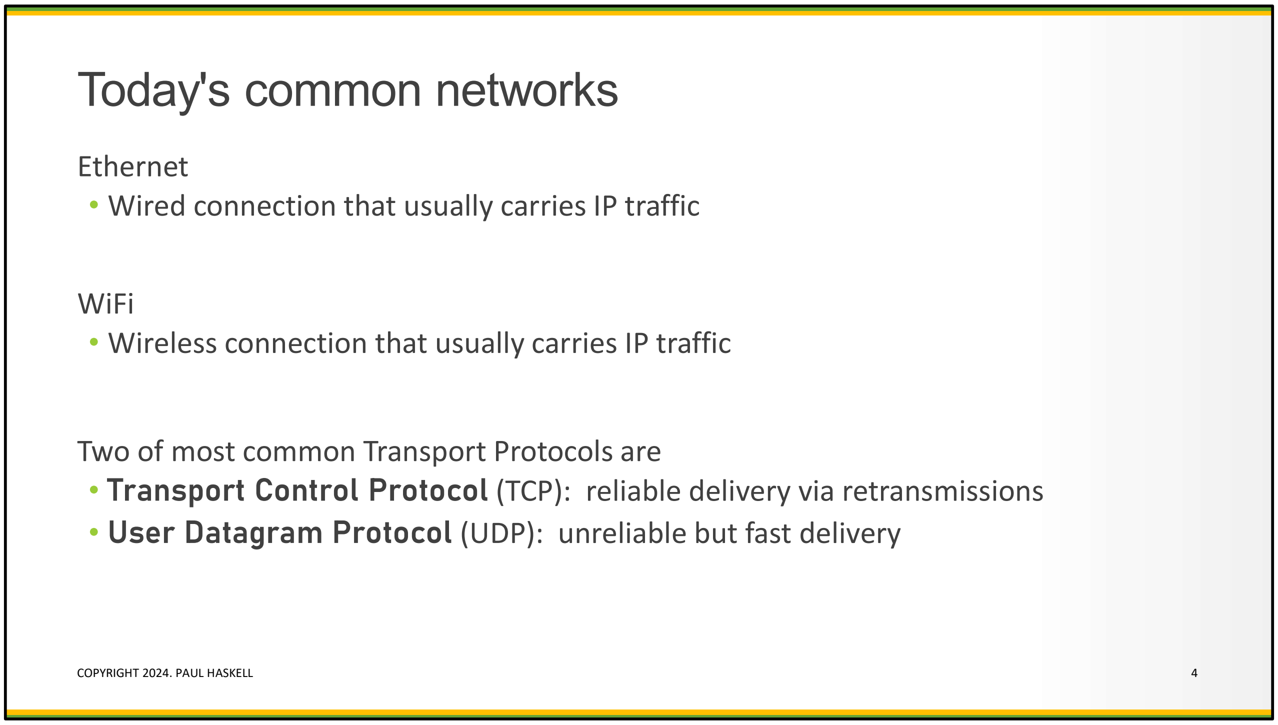


Network "Stack"

Application
Session
Presentation
Transport
Network
Data Link
Physical Layer

3

OSI model: Open Systems Interconnection
Physical is wires and connectors
Data Link is voltages, frequencies, timing, "access to shared medium", etc
Network is addressing, routing, "traffic control"
Transport is reliability, segmentation and reassembly, app multiplexing
Higher layers are more application-specific (e.g. language, character sets, etc)



Today's common networks

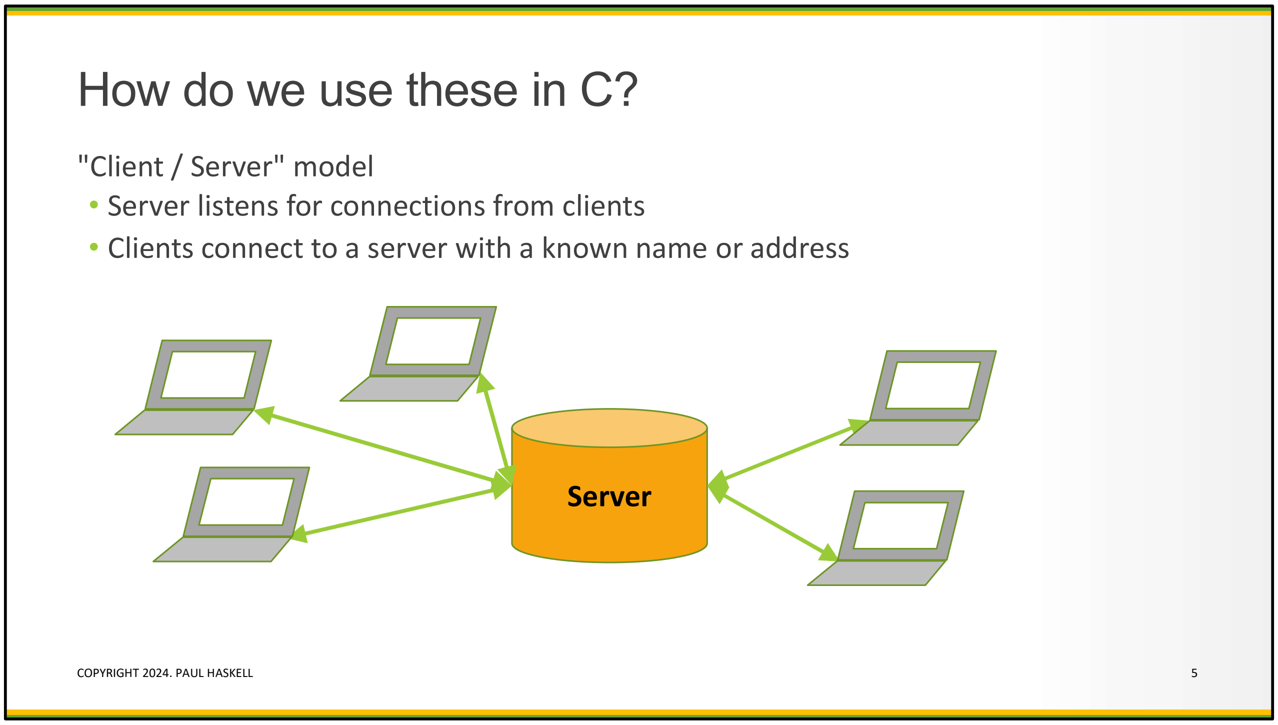
Ethernet
- Wired connection that usually carries IP traffic

WiFi
- Wireless connection that usually carries IP traffic

Two of most common Transport Protocols are
- **Transport Control Protocol** (TCP): reliable delivery via retransmissions
- **User Datagram Protocol** (UDP): unreliable but fast delivery

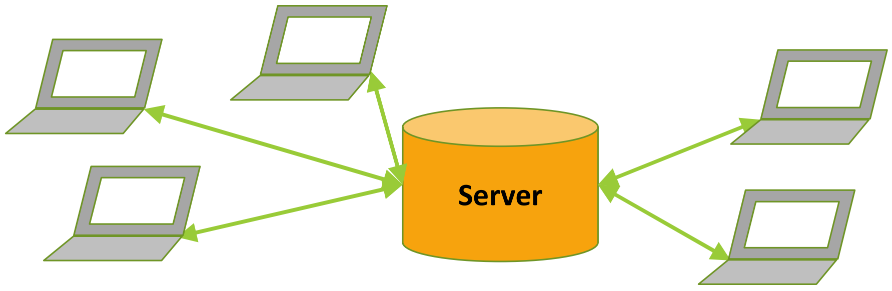
when would we use UDP?

Ethernet and WiFi are Network Layer
TCP and UDP are Transport Layer

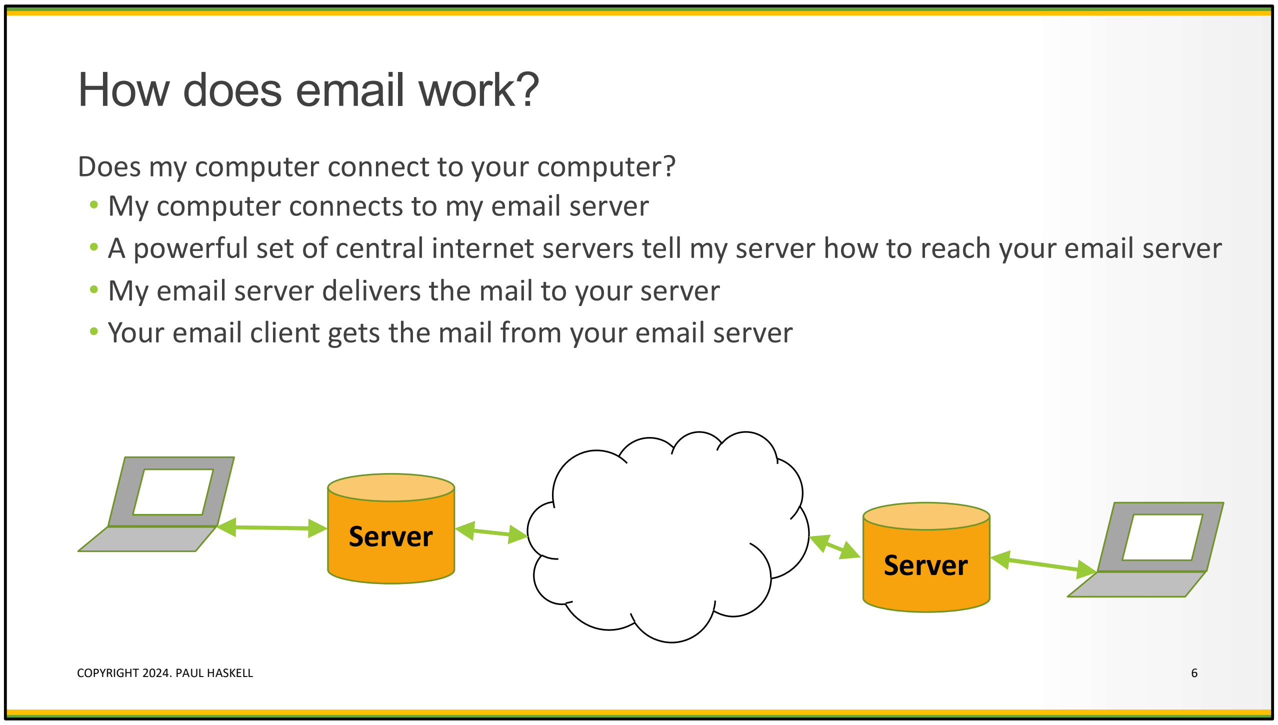


How do we use these in C?

"Client / Server" model
- Server listens for connections from clients
- Clients connect to a server with a known name or address



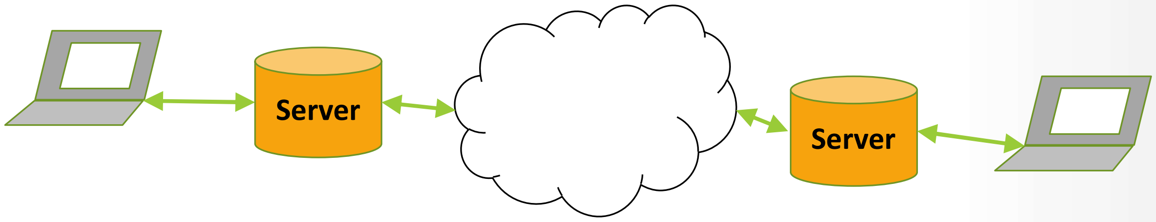
CS336 is Networking



How does email work?

Does my computer connect to your computer?

- My computer connects to my email server
- A powerful set of central internet servers tell my server how to reach your email server
- My email server delivers the mail to your server
- Your email client gets the mail from your email server



COPYRIGHT 2024. PAUL HASKELL

6

How do computers find each other?

DHCP – Dynamic Host Configuration Protocol

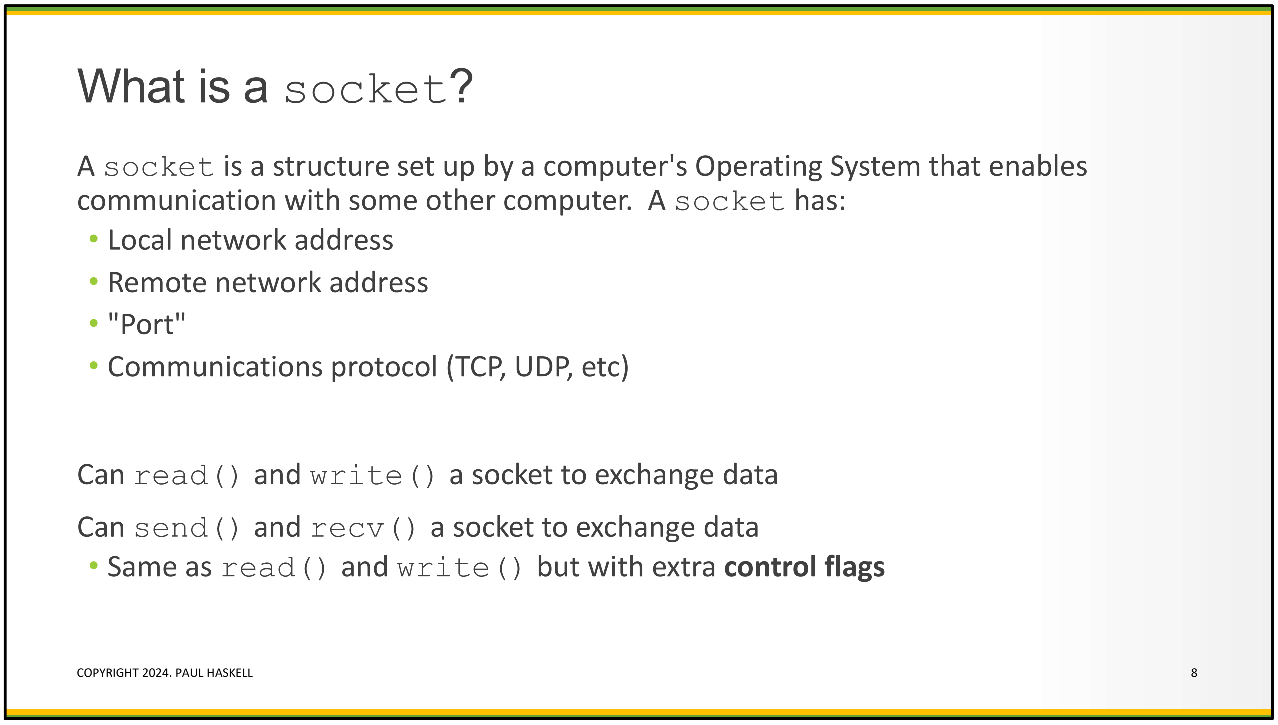
- Assigns an IP address to computers on a local network

RIP – Routing Information Protocol (and others)

- Enable IP routers to send traffic to the "next" router

DNS – Domain Name System

- A distributed service on the internet to let hosts look each other by name rather than by IP address (e.g. **stargate.cs.usfca.edu**)



What is a `socket`?

A `socket` is a structure set up by a computer's Operating System that enables communication with some other computer. A `socket` has:

- Local network address
- Remote network address
- "Port"
- Communications protocol (TCP, UDP, etc)

Can `read()` and `write()` a socket to exchange data

Can `send()` and `recv()` a socket to exchange data

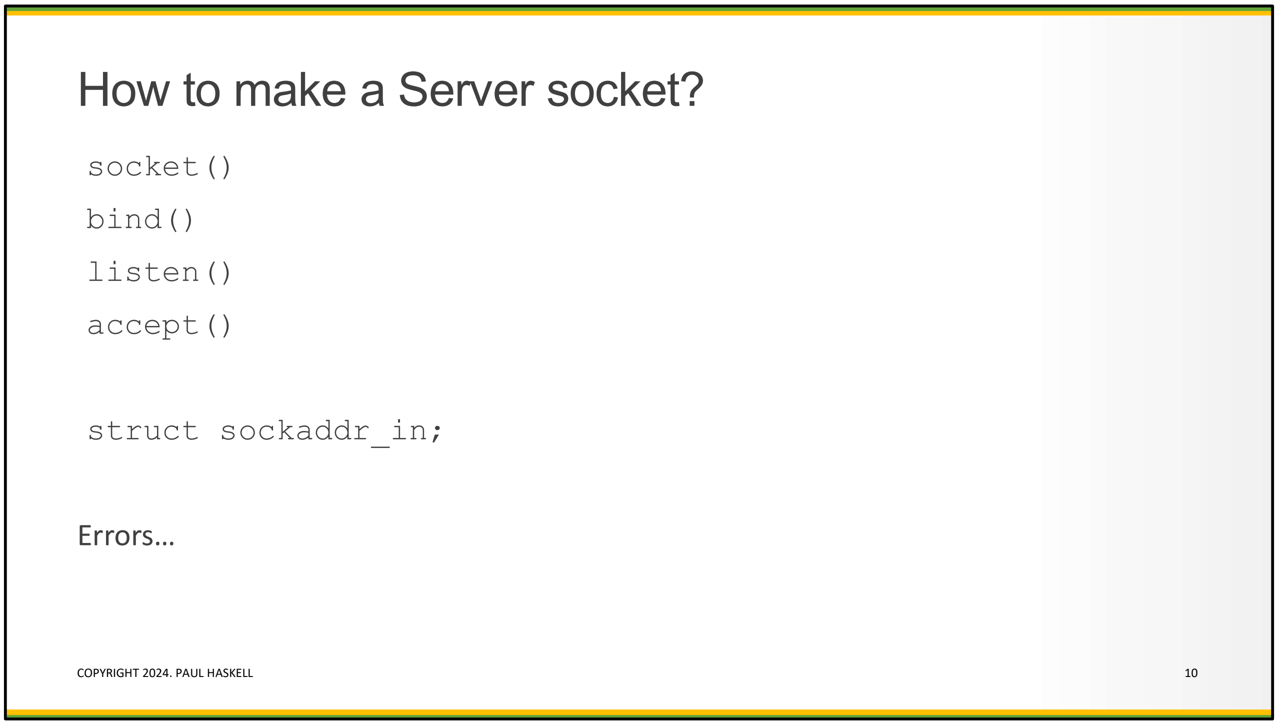
- Same as `read()` and `write()` but with extra **control flags**

We might talk about the control flags, depending on time

What is a `port`?

https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

A port# identifies a particular service



How to make a Server socket?

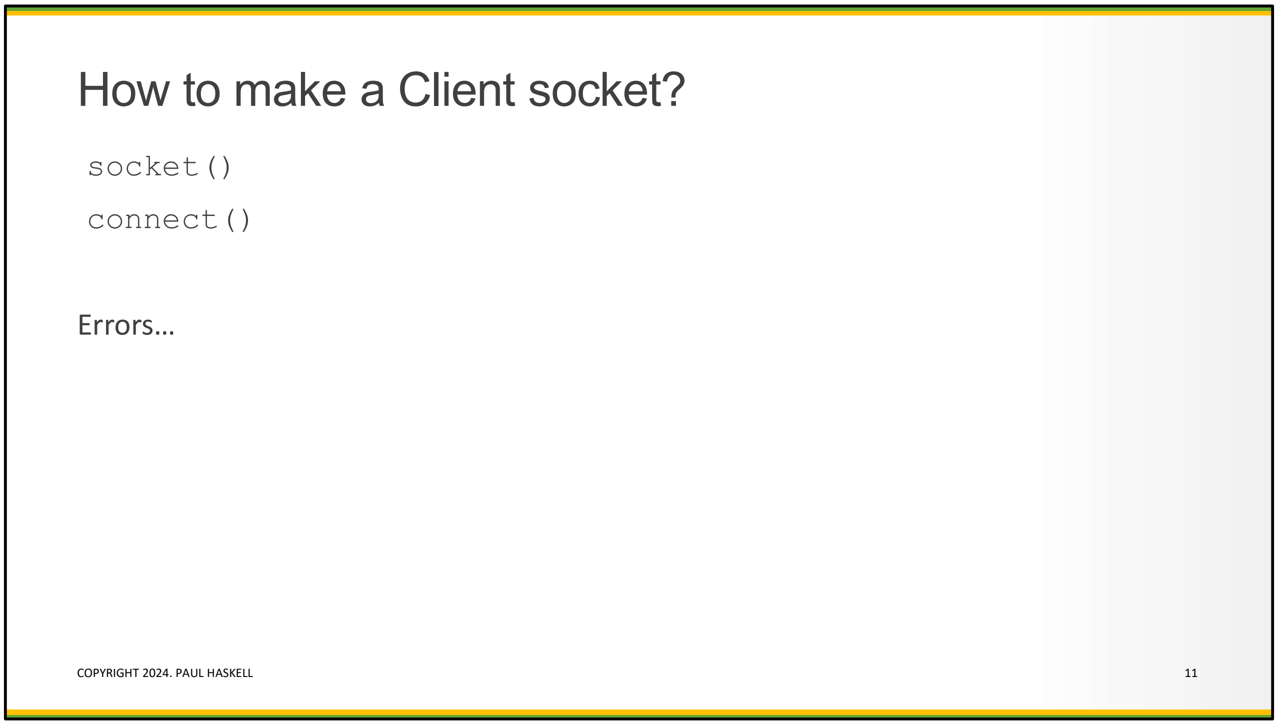
```
socket()
bind()
listen()
accept()

struct sockaddr_in;
```

Errors…

Look at man pages for these commands
Look at server.c

errno and perror() !!! man errno and man perror



How to make a Client socket?

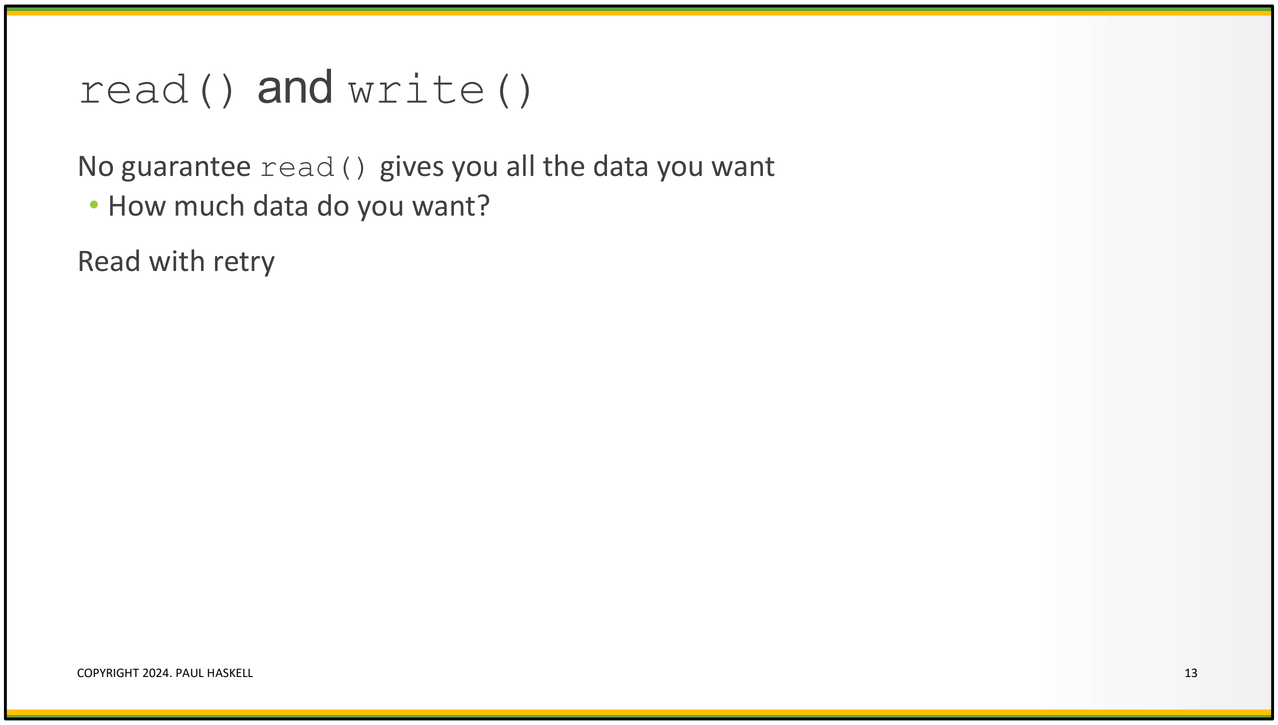
```
socket()
connect()
```

Errors…

Look at man connect
Look at client.c

Two issues…

12



`read()` and `write()`

No guarantee `read()` gives you all the data you want

- How much data do you want?

Read with retry

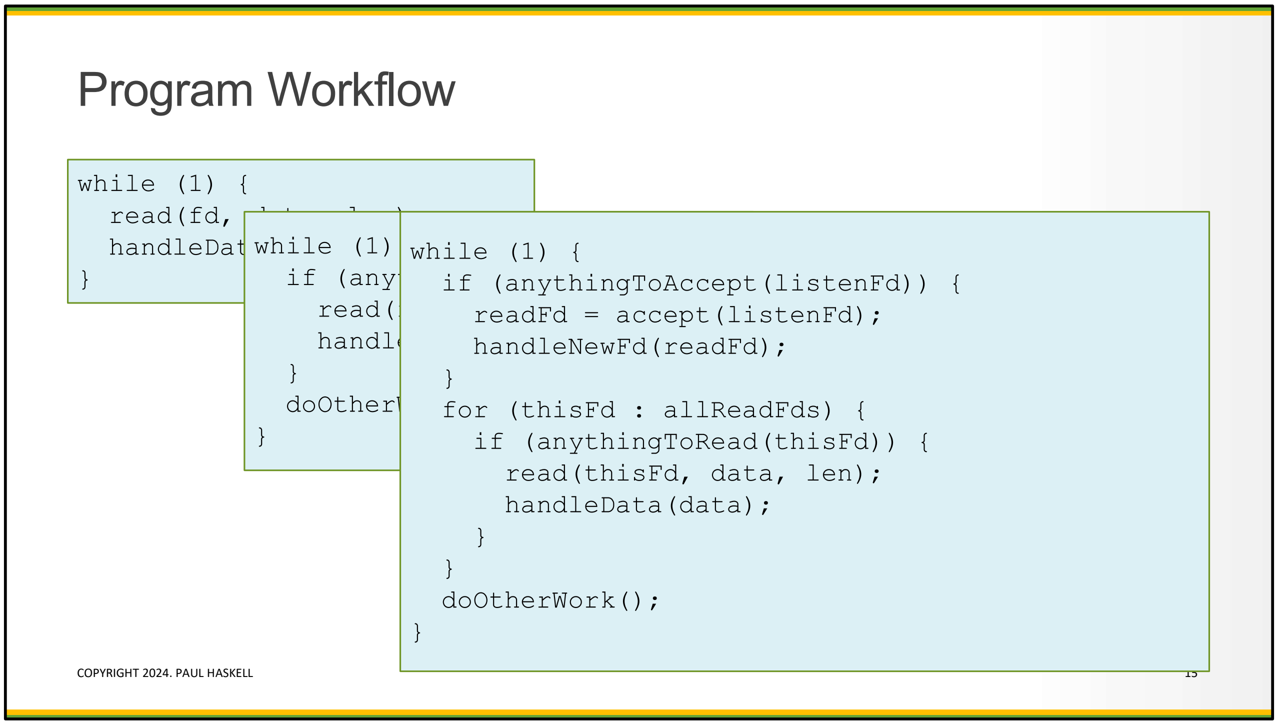
How much data do you want? Known fixed length, or length is first part of message
CODEALONG: add doRead() to client.c where 1st value read is length of rest of msg.

Same problem MIGHT affect write(), if you write lots of data to a slow device.

Program Workflow

Our **server.c** and **client.c** example is really simple

- Probably want many `read()` and `write()` calls
- Possibly want to `accept()` multiple client connections
- Probably want to do other work if nothing to read or accept



Program Workflow

```
while (1) {
  read(fd,
  handleDat
}
```

```
while (1)
  if (any
    read(
    handl
  }
  doOther
}
```

```
while (1) {
  if (anythingToAccept(listenFd)) {
    readFd = accept(listenFd);
    handleNewFd(readFd);
  }
  for (thisFd : allReadFds) {
    if (anythingToRead(thisFd)) {
      read(thisFd, data, len);
      handleData(data);
    }
  }
  doOtherWork();
}
```

sleep() rather than thrash the computer if nothing to read or accept

Several ways to implement these workflows: multiple threads, one for each reader and accepter. But then we still must communicate between threads.
Some tools built into the OS (poll(), select()) that we will not talk about
One solution next time.