ATTRIBUTION

- These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
- These slides incorporate material from:
  - Practical TCP/IP Sockets in C, 2nd ed., by Donahoo and Calvert
  - Computer Networks: A Systems Approach, 5e, by Peterson and Davie
  - CMU’s 15-441 Computer Networks, Xi Liu
ANNOUNCEMENTS

HW 1 due today at 5
Will have an opportunity to resubmit if you didn’t fully get it the first time

Project 1 out

HW 2 going out shortly
Outline

1. Server socket API
2. Demo: writing a simple server
3. Socket internals
**SERVER OVERVIEW**

### Steps

1. Create network socket
2. Bind socket to an interface
3. Tell the socket to listen for incoming connections
4. Accept an incoming connection:
5. Read/write to the socket
6. Close the socket

### Socket API used

1. `socket()`
2. `bind()`
3. `listen()`
4. `accept()`
5. `send/recv()`
6. `close()`
SERVER BIND

- Only server need to bind
  
  - `int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);`

- `sockfd`
  
  - file descriptor socket() returned

- `my_addr`
  
  - `struct sockaddr_in` for IPv4

  - `cast (struct sockaddr_in*) to (struct sockaddr*)`

```c
struct sockaddr_in {
    short       sin_family;  // e.g. AF_INET
    unsigned short  sin_port;  // e.g. htons(3490)
    struct in_addr sin_addr;  // see struct in_addr, below
    char       sin_zero[8];  // zero this if you want to
};
struct in_addr {
    unsigned long s_addr;  // load with inet_aton()
};
```
Now we can listen

`int listen(int sockfd, int backlog);`

`sockfd`

again, file descriptor socket() returned

`backlog`

number of pending connections to queue

For example,

`listen(sockfd, 5);`
SERVER ACCEPT

- Server must explicitly accept incoming connections
  - int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)

- sockfd

- addr
  - pointer to store client address, (struct sockaddr_in *) cast to (struct sockaddr *)

- addrlen
  - pointer to store the returned size of addr, should be sizeof(*addr)

- For example
  - int isock=accept(sockfd, (struct sockaddr_in *) &caddr, &clen);
1. Server socket API
2. Demo: writing a simple server
3. Socket internals
COMPANION CODE

https://github.com/gmporter/cse124-lec06-webserver
Outline

1. Server socket API
2. Demo: writing a simple server
3. Socket internals
rv = connect(s,...);
.
.
rv = send(s, buffer0, 1000, 0);
.
.
rv = send(s, buffer1, 2000, 0);
.
.
rv = send(s, buffer2, 5000, 0);
.
.
close(s);
AFTER 3 SEND() CALLS

- **SendQ**: 6500 bytes
- **RecvQ**: 1500 bytes
- **Delivered**: 0 bytes

1. First send call (1000 bytes)
2. Second send call (2000 bytes)
3. Third send call (5000 bytes)
AFTER FIRST RECV()
AFTER ANOTHER `RECV()`
WHEN DOES BLOCKING OCCUR?

- SendQ size: SQS
- RecvQ size: RQS
- `send(s, buffer, n, 0);`
  - `n>SQS`: blocks until `(n – SQS)` bytes xfered to RecvQ
  - If `n>(SQS+RQS)`, blocks until receiver calls `recv()` enough to read in `n-(SQS+RQS)` bytes
- How does this lead to deadlock?
  - Trivial cause: both sides call `recv()` w/o sending data
• SendQ size = 500; RecvQ size = 500