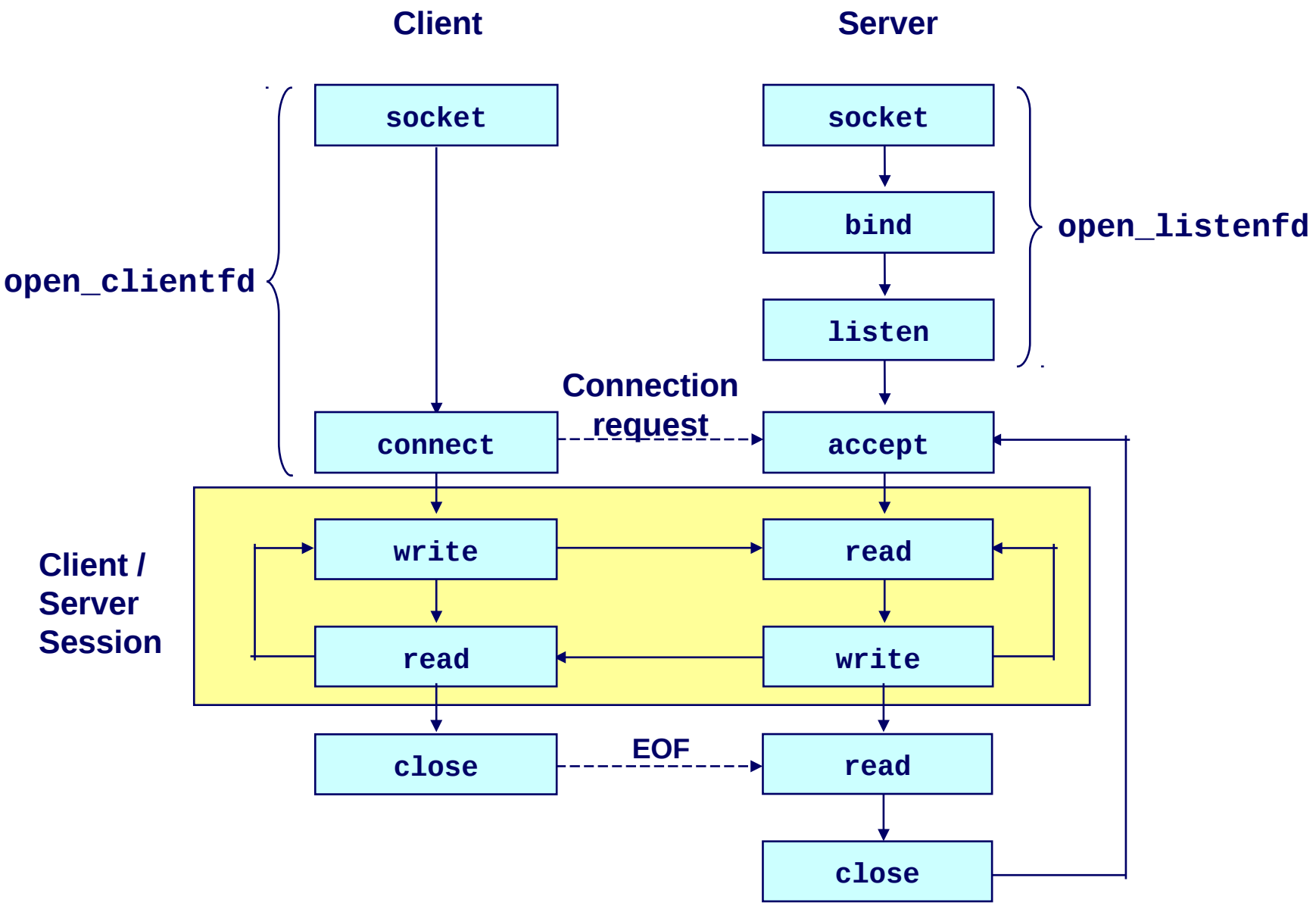
Introduction to Socket Programming CSL343

Sockets

- How to use sockets
  - Setup socket
    - Where is the remote machine (IP address, hostname)
    - What service gets the data (port)
  - Send and Receive
    - Designed just like any other I/O in unix
    - send -- write
    - recv -- read
  - Close the socket

Overview


Client
Server
socket
socket
bind
listen
open_clientfd
open_listenfd
Connection request
connect
accept
Client / Server Session
write
read
read
write
close
EOF
read
close

Step 1 – Setup Socket

- **Both client and server need to setup the socket**
  - *int socket(int domain, int type, int protocol);*
- *domain*
  - AF_INET -- IPv4 (AF_INET6 for IPv6)
- *type*
  - SOCK_STREAM -- TCP
  - SOCK_DGRAM -- UDP
- *protocol*
  - 0
- For example,
  - *int sockfd = socket(AF_INET, SOCK_STREAM, 0);*

Step 2 (Server) - Binding

- **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*)

```
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()
};
```

Step 2 (Server) - Binding contd.

- *addrlen*
  - size of the sockaddr_in

```
struct sockaddr_in saddr;
int sockfd;
unsigned short port = 80;

if((sockfd=socket(AF_INET, SOCK_STREAM, 0) < 0) {// from back a couple slides
     printf(“Error creating socket\n”);
     ...
}

saddr.sin_family = AF_INET;                 // match the socket() call
saddr.sin_addr.s_addr = htonl(INADDR_ANY);  // bind to any local address
saddr.sin_port = htons(port);               // specify port to listen on

if((bind(sockfd, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) { // bind!
     printf(“Error binding\n”);
     ...
}
```

What is htonl(), htons()?

- Byte ordering
  - Network order is big-endian
  - Host order can be big- or little-endian
    - x86 is little-endian
    - SPARC is big-endian
- Conversion
  - *htons(), htonl()*: host to network short/long
  - *ntohs(), ntohl()*: network order to host short/long
- What need to be converted?
  - Addresses
  - Port
  - etc.

Step 3 (Server) - Listen

- **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);*

Step 4 (Server) - Accept

- **Server must explicitly accept incoming connections**
  - *int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)*
- *sockfd*
  - again... file descriptor socket() returned
- *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);*

Telnet

- Telnet (short for TErminal NETwork) is a network protocol.
- Telnet protocol has you log on to a server as if you were an actual user, granting you direct control and all the same rights to files and applications as the user that you're logged in as.
    - Telnet localhost port_number
- Telnet is rarely used to connect to devices or systems anymore.

What about client?

- Client need not bind, listen, and accept
- **All client need to do is to connect**
  - *int connect(int sockfd, const struct sockaddr *saddr, socklen_t addrlen);*
- For example,
  - *connect(sockfd, (struct sockaddr *) &saddr, sizeof(saddr));*

We Are Connected

- Server accepting connections and client connecting to servers
- Send and receive data
  - *ssize_t read(int fd, void *buf, size_t len);*
  - *ssize_t write(int fd, const void *buf, size_t len);*
- For example,
  - *read(sockfd, buffer, sizeof(buffer));*
  - *write(sockfd, “hey\n”, strlen(“hey\n”));*

Close the Socket

- Don’t forget to close the socket descriptor, like a file
  - *int close(int sockfd);*

- Now server can loop around and accept a new connection when the old one finishes

For reference use the following link:

https://www.freebsd.org/doc/en/books/developers-handbook/sockets-essential-functions.html