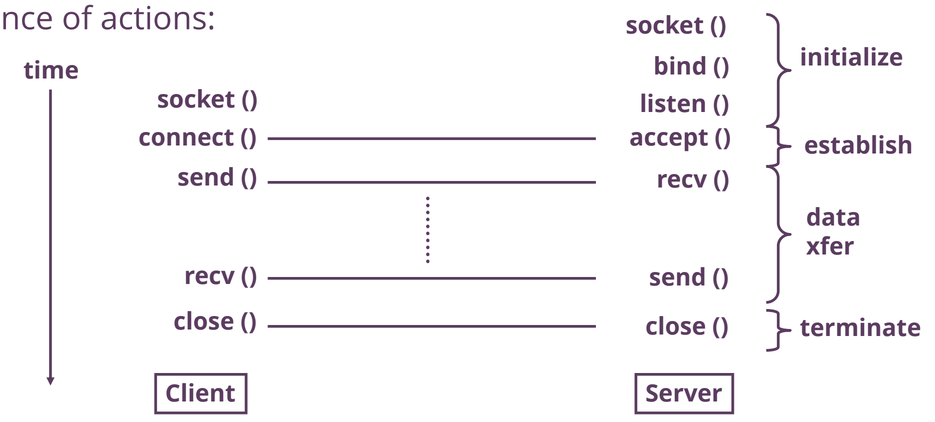
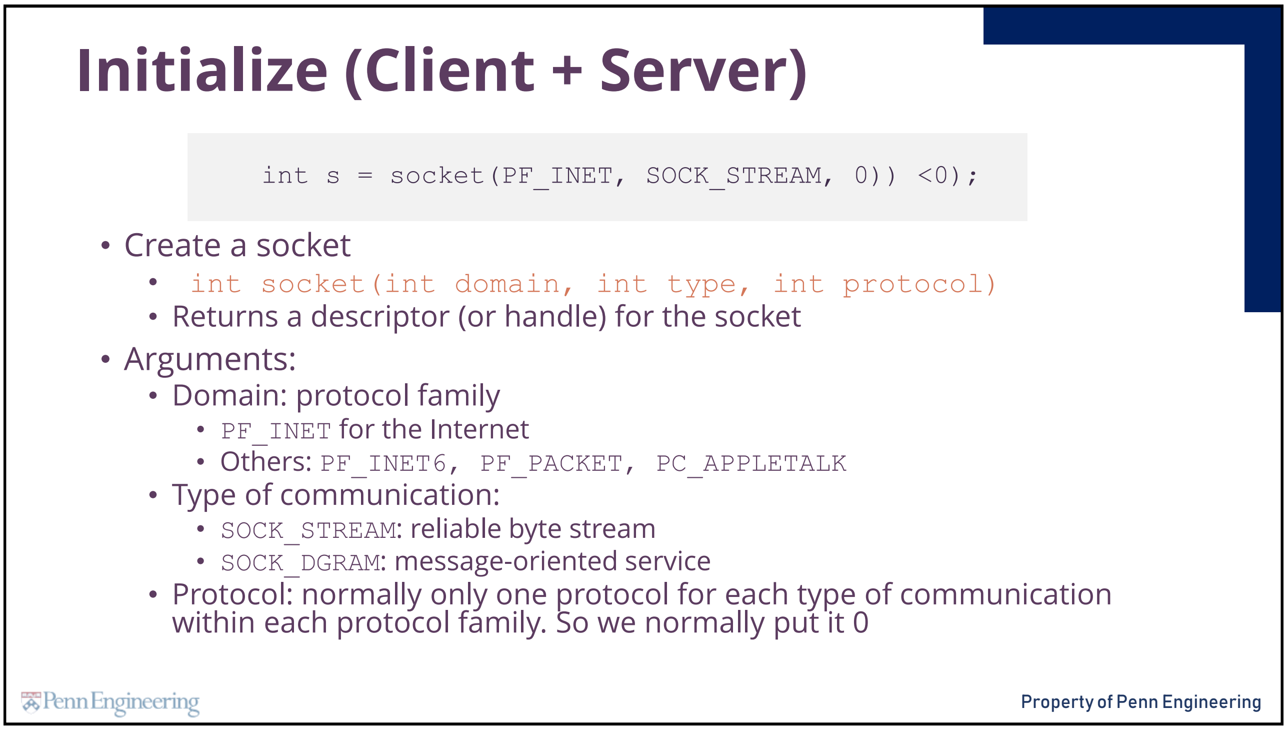
Network Programming Part 4

MCIT 595

Stream Sockets

- Implements Transmission Control Protocol (TCP)
- Does NOT set up virtual-circuit!
- Sequence of actions:

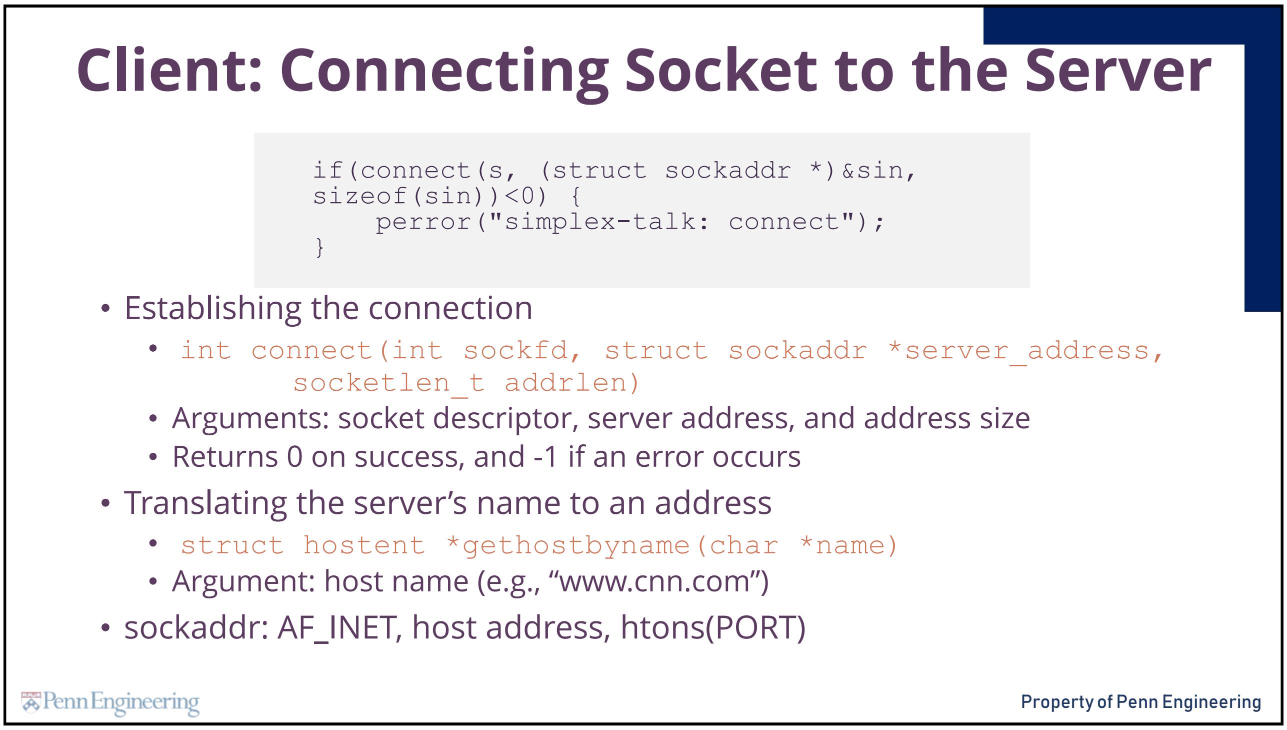




Initialize (Client + Server)

```
int s = socket(PF_INET, SOCK_STREAM, 0)) <0);
```

- Create a socket
  - `int socket(int domain, int type, int protocol)`
  - Returns a descriptor (or handle) for the socket
- Arguments:
  - Domain: protocol family
    - `PF_INET` for the Internet
    - Others: `PF_INET6, PF_PACKET, PC_APPLETALK`
  - Type of communication:
    - `SOCK_STREAM`: reliable byte stream
    - `SOCK_DGRAM`: message-oriented service
  - Protocol: normally only one protocol for each type of communication within each protocol family. So we normally put it 0



Client: Connecting Socket to the Server

```
if(connect(s, (struct sockaddr *)&sin,
sizeof(sin))<0) {
    perror("simplex-talk: connect");
}
```

- Establishing the connection
  - `int connect(int sockfd, struct sockaddr *server_address,`
    `socketlen_t addrlen)`
  - Arguments: socket descriptor, server address, and address size
  - Returns 0 on success, and -1 if an error occurs
- Translating the server's name to an address
  - `struct hostent *gethostbyname(char *name)`
  - Argument: host name (e.g., “www.cnn.com”)
- sockaddr: AF_INET, host address, htons(PORT)

Client: Sending and Receiving Data

```
while(fgets(buf, sizeof(buf),stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf)+1;
    send(s, buf, len, 0);
  }
```

- Sending data
  - `ssize_t send(int sockfd, void *buf, size_t len, int flags)`
  - Arguments: socket descriptor, pointer to data buffer, buffer size, and flags
  - Returns the number of characters written, and -1 on error
- Receiving data
  - `ssize_t read(int sockfd, void *buf, size_t len, int flags)`
  - Arguments: socket descriptor, pointer to data buffer, buffer size, and flags
  - Returns the number of characters read (0 = “end of file”), and -1 on error

(See example code for complete implementation)

Server: Preparing its Socket

```
if((s = socket(PF_INET, SOCK_STREAM, 0)) <0) {
    perror("simplex-talk: socket");
}

if((bind(s, (struct sockaddr*)&sin, sizeof(sin)))<0){
    perror("simplex-talk: bind");
}
```

- Server creates a socket
  - `int socket(int domain, int type, int protocol)`
- Bind socket to the local address and port number
  - `int bind (int sockfd, struct sockaddr *my_addr,`
    `socklen_t addrlen)`
  - Arguments: socket descriptor, server address, address length
  - Returns 0 on success, and -1 if an error occurs

Server: Accepting Client Connection

```
listen(s, MAX_PENDING);

if((new_s = accept(s, (struct sockaddr *)&sin, &len)) <0) {
        perror("simplex-talk: accept");
}
```

- Server blocks and wait for requests to arrive
- Accept a new connection from a client
  - `int accept(int sockfd, struct sockaddr *addr, socketlen_t *addrlen)`
  - Arguments: socket descriptor, structure that will provide client address and port, and length of the structure
  - Returns descriptor for a new socket for this connection

(See example code for complete implementation)

Datagram Sockets

- Similar to stream sockets, except:
  - Sockets created using `SOCK_DGRAM` instead of `SOCK_STREAM`
  - No need for connection establishment and termination
  - Uses **`recvfrom()`** and **`sendto()`** in place of **`recv()`** and **`send()`** respectively
  - Data sent in packets, not byte-stream oriented
- Use “man pages”

Explore man pages for more information

Try: **`man recvfrom`**