**Lecture 18: Sockets**

**Client/Server Architecture**
- Network applications are divided into *client* and *server* parts. A particular server usually communicates with a single server. While a server may communicate with several clients.

```
client
server
```

**TCP and UDP Ports**
- Clients use *ephemeral* or short-lived ports. These port numbers are usually automatically assigned to the client.
- The routines `getport()` and `getservbyname()` can be used to translate between port names and numbers.
- The program `getsrv.cpp` illustrates the use of these two routines.

**Sockets**
- A socket provides yet another mechanism for IPC. Although we have been focusing on TCP/IP networks, a socket can be used for IPC across other networks (IPX, AppleTalk) and even for local communication.
- The different transport mechanisms are known as ________. We will concentrate on local (or UNIX) domain communication via sockets today and return to network sockets next time.
Lecture 18: Sockets

**Sockets**

- There is a useful ______ for establishing a TCP connection and using a telephone:
  - CLIENT
    - socket() | Purchase a telephone.
    - connect() | Dial the server.
  - SERVER
    - socket() | Purchase a telephone.
    - bind() | Publish your number.
    - listen() | Turn on the ringer.
    - accept() | Answer the phone.

- To create a socket use the `socket()` routine:
  
  ```c
  clifd = socket(domain, type, protocol);
  ```

  A file descriptor is returned. The `domain` parameter selects the __________. It is PF_UNIX for a local (filesystem) socket. (See “man 7 unix”).

Lecture 18: Sockets

**Client Routines**

- The `type` is either SOCK_STREAM for a stream-oriented socket or SOCK_DGRAM for a datagram-oriented socket that preserves message boundaries. (Linux also supports SOCK_SEQPACKET for a connection-oriented socket that preserves message boundaries.)

- We will only discuss __________ types today. We can use the standard `send()` and `recv()` routines for socket I/O.

Lecture 18: Sockets

**Client Routines**

- The `protocol` parameter is used to select a subprotocol within the protocol family and is 0 for PF_UNIX.

- A client uses the `connect()` routine to establish a connection to a ______ server.

  ```c
  res = connect(clifd, addr, sizeof(addr));
  ```

Lecture 18: Sockets

**Server Routines**

- The server will call `socket()`, `bind()`, ______( ), and `accept()`.

- First call `socket()` to create the socket:

  ```c
  srvfd = socket(domain, type, protocol);
  ```

The `domain`, `type` and `protocol` parameters are the same as for the client (PF_UNIX, SOCK_STREAM, and 0).
Lecture 18: Sockets Server Routines

- **bind()** is used to assign the local ________ to the socket:

```c
struct sockaddr_un unaddr;
struct sockaddr *address =
    (struct sockaddr *)&unaddr;
unaddr.sun_family = AF_UNIX;
strcpy(unaddr.sun_path,"/tmp/tmpsock");
int len = sizeof(un_family) + strlen(unaddr.sun_path);
bind(srvfd, address, len);
```

- The **listen()** routine sets up a ________ for client connections:

```c
res = listen(srvfd, qlength);
```

- A queue length of 5 is typical. Sockets are designed for client/server communication. There may be several clients trying to connect to a single server. They are held in the queue until accepted by the server.

Lecture 18: Sockets Server Routines

- Finally, the server calls the **accept()** routine to wait for a client connection:

```c
newfd =
    accept(sockfd, &ptr_addr, &len);
accept() ________ until a connection is made.
```

- A new socket file descriptor is returned. The server uses this file descriptor to communicate with the client.

Lecture 18: Sockets Socketpair

- The **socketpair** routine creates a pair of connected, ________ sockets (like *pipe*, but bidirectional). It can be used for IPC between related processes.

```c
err =
    socketpair(PF_UNIX, type, 0, int *sv);
```

- **type** can be **SOCK_STREAM**, etc. **sv** is a two element array containing the file descriptors.

Lecture 18: Sockets In-Class Exercise

- Download the socket server and client routines from the web site. The server is complete, but the client is not. Complete the client so that it can communicate with the server.