UNIX System Programming
Lecture 19: IP Sockets

• Outline
  - IP Sockets
• Reference
  - BLP: Chapter 15
  - man pages: socket, bind, connect, listen, accept, ip(7), ipv6(7), getaddrinfo, getnameinfo

Lecture 19: IP Sockets
Review of UNIX Sockets

• On the server, we first call `socket()` (with ________). A file descriptor is returned.

• We fill in the address of the named socket and `bind()` the file descriptor to the address (creating the socket on the file system.)

• We then call `listen()` to set up a listen queue and `accept()` to accept clients. `accept()` returns a file descriptor that we use to communicate with the client.

Lecture 19: IP Sockets
Review of UNIX Sockets

• On the client, we call `socket()` to get a file descriptor. We then `connect()` our socket to the server socket address.

• We used the `recv()` and `send()` routines to send data across the socket.

• Sockets are ________, we read and write to the same file descriptor receiving and sending data written and read from the other endpoint of the connection.

Lecture 19: IP Sockets
IP Sockets

• IP or Internet sockets can be used to communicate between two processes on two different machines connected by a network. (Or for IPC on a single machine.)

• An IP socket is an address/port-number pair and uniquely identifies an endpoint.

• A IP socket _______ consists of both the server and client sockets. Every TCP connection is uniquely associated with a socket pair.

Lecture 19: IP Sockets
IP Sockets

• Just as for local sockets the server will call `socket()`, `bind()`, `listen()`, and `accept()`.

• First call `socket()` to create the socket:
  ```c
  sfd = socket(domain, type, proto);
  ```
  The `domain` is `AF_INET`. The `type` is usually either `SOCK_DGRAM` (UDP) or `SOCK_STREAM` (TCP). (We will use `SOCK_STREAM`.) The `proto` should be 0.
Lecture 19: IP Sockets

**IP Sockets**

- `bind()` is used to assign an address (TCP port) to the socket:

```c
struct sockaddr_in in_addr;
struct sockaddr *p_addr =
    (struct sockaddr *)&in_addr;
in_addr.sin_family = AF_INET;
in_addr.sin_port = htons(2400);
in_addr.sin_addr.s_addr =
    htonl(INADDR_ANY);
int len = sizeof(struct sockaddr_in);
bind(sfd, p_addr, len);
```

- Use `getaddrinfo()` to form local ________. Then call `socket()` and `bind()`.

```c
// Use getaddrinfo to form address
struct addrinfo hints, *res;
memset(&hints, 0, sizeof(hints));
// IPv4 or V6
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
// Fill in my IP (INADDR_ANY)
hints.ai_flags = AI_PASSIVE;
getaddrinfo(NULL, "2400", &hints, &res);
```

- On the server we ________ our socket to a particular IP address and port number.

- Use of `AI_PASSIVE` in `ai_flags` will bind to all local IP addresses. We could instead bind to a single IP address on a single interface.

- Our server will bind to port 2400 in this example. The port number is passed as a string to `getaddrinfo()`. We do not need to use `htons()` (no big/little endian problems).

- Instead of binding to a particular port, if we specify a port number of 0 in the address we will be ________ assigned an available port number. The `getsockname()` routine can then be used to find the port number:

```c
getsockname(sfd, p_addr, &addrlen);
cout << "Port #: " << ntohs(in_addr.sin_port) << endl;
```

- The server then uses `listen()` and `accept()` just as for UNIX sockets.

```c
result = listen(sfd, qlength);
nfd = accept(sfd, p_addr, &len);
```

- The `p_addr` parameter is a pointer to a `sockaddr` structure. This will be filled with connecting ________ info (IP address and port number).
**Lecture 19: IP Sockets**

**IP Sockets**

- On the client we need only `socket()` and `connect()`. We use the server IP and port numbers in the address:

  ```c
  cfd = socket(AF_INET, SOCK_STREAM, 0);
  struct sockaddr *address = (struct sockaddr *) malloc(sizeof(struct sockaddr_in));
  in.sin_family = AF_INET;
  in.sin_port = htons(2400);
  in.sin_addr.s_addr = inet_addr("10.10.0.9");
  int len = sizeof(struct sockaddr_in);
  res = connect(cfd, address, len);
  ```

- First, form the address:

  ```c
  struct addrinfo hints, *res;
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags =.AI_NUMERICHOST;
  hints.ai_flags |= AI_NUMERICSERV;
  getaddrinfo("10.10.0.9", "2400", &hints, &res);
  ```

- Since our socket file descriptor is not bound to a local endpoint, `connect()` will do this for us. This is usually what we want. You could use `bind()` to bind to a particular local number before the call to `connect()`.

- `getsockname()` is used to get local address information for a file descriptor. It is useful when using dynamic port assignment.
- `getpeername()` is used to get remote address information.
- `getaddrinfo()` can be used to translate addresses (IP and port) back to names. It is the inverse of `getaddrinfo()`.

**In Class Exercise**

- Download `ntcp_server` and `ntcp_client`.
  - Run both locally. Try connecting to both 127.0.0.1 and actual IP address.
  - Connect to a classmates’ server.
  - Copy programs to cserver. Try running server on cserver and local client then try local server and client on cserver.
  - Try connections between virtual machine and host machine (depends on VM config).
  - Update the client to use `getaddrinfo()` to form server address for use with `connect`.