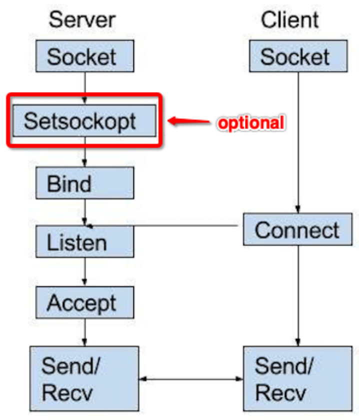
CISC 4615 — Lab 1

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

As we discussed in the lecture, the following figure presents the states of the server and client.



Workflow for a server

1. Socket creation: **int sockfd = socket(domain, type, protocol)**
   - **sockfd**: socket descriptor, an integer (like an ID for a specific socket)
   - **domain**: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
   - **type**: communication type: SOCK_STREAM or SOCK_DGRAM.
     SOCK_STREAM: TCP(reliable, connection oriented)
     SOCK_DGRAM: UDP(unreliable, connectionless)
   - **protocol**: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)
2. **Setsockopt**: **int setsockopt(SOCKET s, int level, int optname, const char *optval, int optlen**); If no error occurs, setsockopt returns zero.
   - **s**: A descriptor that identifies a socket.
   - **level**: The level at which the option is defined (in this course, we use SOL_SOCKET).

- **optname**: The socket option for which the value is to be set (in this course, we use SO_BROADCAST). The optname parameter must be a socket option defined within the specified level, or behavior is undefined.
- **optval**: A pointer to the buffer in which the value for the requested option is specified.
- **optlen**: The size, in bytes, of the buffer pointed to by the optval parameter.

3. **Bind**: **int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)**;
   - **socket**: The socket descriptor returned by a previous socket() call.
   - **address**: The pointer to a sockaddr structure containing the name that is to be bound to socket. In our course, sockaddr is the same as sockaddr_in.
   - **address_len**: The size of address in bytes (e.g. sizeof(a_sockaddr_in_struct)).
4. **Listen**: **int listen(int sockfd, int backlog)**;

   It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.
5. **Accept**: **int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);**

   The accept() system call is used with connection-based socket types (SOCK_STREAM, SOCK_SEQPACKET). It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket sockfd is unaffected by this call.
   - **sockfd**: The argument sockfd is a socket that has been created with socket, bound to a local address with bind, and is listening for connections after a listen.
   - **addr**: The argument addr is a pointer to a sockaddr structure.
   - **address_len**: It must initially point to an integer that contains the size in bytes of the storage pointed to by address.

Workflow for a client

1. **Socket connection**: Exactly same as that of servers socket creation.
2. **Connect**: **int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);**

- The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Servers address and port is specified in addr.

Send and Receive Data

1. **send**: **int send(int socket, char *buffer, int length, int flags)**;

   The send() function sends data on the socket with descriptor socket. The send() call applies to all connected sockets.

   - **socket**: The socket descriptor.
   - **buffer**: The pointer to the buffer containing the data to transmit.
   - **length**: The length of the message pointed to by the buffer parameter.
   - **flags**: The flags parameter (we use 0 in this course).
   - Similar function: **write(int fd, const void *buf, size_t count)**;

2. **recv**: **int recv(int socket, char *buffer, int length, int flags)**;

   The recv() function receives data on a socket with descriptor socket and stores it in a buffer. The recv() call applies only to connected sockets.

   - **socket**: The socket descriptor.
   - **buffer**: The pointer to the buffer containing the data to transmit.
   - **length**: The length of the message pointed to by the buffer parameter.
   - **flags**: The flags parameter (we use 0 in this course).
   - Similar function: **read(int fd, const void *buf, size_t count)**;

Lab 1 Assignment: Part 1 and 2

The given programs can make a connection between client and server. The client can send a word to the server, which will send back the same word with all letters in uppercase.

Lab 1 consists of the following two parts.

1. Make your own client and server programs and let them connect with each other.

   - Make your client and server programs to let them connect with each other through TCP.

   - After the connection, client sends message 1, ”Hello Fordham”, to server, the server prints out the message, send back a message, ”Bye”, to client, then, the client prints out, ”Bye”, and close the connection.

2. Make your client and server talk to each other and store the messages.
   - After the connection, client sends message 1, ”Hello Fordham”, server replies message 2, ”Hello CIS Students”, client replies message 3, ”Hello CISC4615” and then, server replies ”Bye” and close the connections.
   - Upon receiving each message, the client needs to store the message. After closing the connection, the client has to print out the message list.

Grading Rubric

We will try to complete the part 1 in class and you have one week to complete part 2. In addition, you should prepare a short report about what you have learned and what are the challenges.

(75%) Part 1;
(20%) Part 2;
(5%) Report;

Submission

On Thursday, January 31st, each group has to demonstrate the lab with part 1 to me in class and email me the code along with your part 2 and report by Sunday, February 3rd.