

COMS 3157: Advanced Programming Networking and Sockets

May 3, 2025

Patrick Shen

pts2125@columbia.edu

1. What are the steps in order to send and receive data from a client to a server?
2. Explain the each parameter in `socket(int domain, int type, int protocol)` (3 marks)
 - (a) `int domain`
 - (b) `int type`
 - (c) `int protocol`
3. Syntax for establishing UDP/TCP sockets (2 marks)
 - (a) What is the syntax to establish a UDP socket?
 - (b) What is the syntax to establish a TCP socket?
4. What is the purpose of binding in networking (used in both UDP/TCP)?
5. Explain the parameters of `bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen)` (1 mark)
 - (a) `int sockfd`
 - (b) `struct *my_addr`
 - (c) `socklen_t addrlen`
6. Given that the `struct sockaddr` in the `bind()` method is designed to work for all networking connections i.e. outside of only just UDP and TCP, explain the significance of each field for `struct sockaddr` (1 mark)
 - (a) `sa_family_t sa_family`
 - (b) `char sa_data[14]`

```
1 struct sockaddr {  
2     sa_family_t sa_family; // unsigned short  
3     char sa_data[14]; // blob  
4 }
```

Listing 1: struct sockaddr

7. Below is the syntax for declaring a struct that holds information for constructing an IPv4 address. How can we pass this in as an argument for the aforementioned `bind()` function? (1 mark)

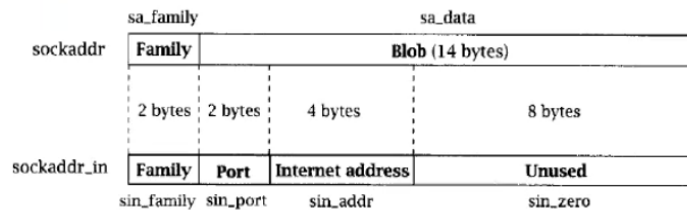
```

1 struct sockaddr_in {
2     sa_family_t sin_family;    // unsigned short, address family:
    AF_INET
3     in_port_t sin_port;        // unsigned short, port in network
    byte order
4     struct in_addr sin_addr;   // internet address
5     char sin_zero[8];         // not used}

```

Listing 2: IPv4 `sockaddr_in` struct

Hint below!



8. If I gave you an 32-bit integer with value 1, write the C code to determine if the value is stored as little-endian or big-endian format. The integer is stored in a variable named `x`. (1 mark)
9. Write C code to (2 marks)
- convert little endian to big endian format
 - convert big endian to little endian format
10. What is `htons()` and `noths()` used for? (2 marks)
11. What is the relationship between `htons()` and `noths()` in relation to bit and little endianness? (1 mark)
12. What does `connect()` do and explain each parameter field in `connect()` (2 marks)

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- `sockfd`
- `addr`
- `addrlen`

13. What does `listen()` do and explain each parameter field in `listen()` (2 marks)

```
int listen(int sockfd, int backlog)
```

- (a) `sockfd`
- (b) `backlog`

14. What does `accept()` do and explain each parameter field in `accept()` (2 marks)

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
```

- (a) `sockfd`
- (b) `addr`
- (c) `addrlen`

15. What does `recv()` do and explain each parameter field in `recv()`? (2 marks)

```
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- (a) `sockfd`
- (b) `buf`
- (c) `len`
- (d) `flags`

16. What does `listen()` do and explain each parameter field in `listen()` (2 marks)

```
int listen(int sockfd, int backlog)
```

- (a) `sockfd`
- (b) `backlog`

17. What does `select()` do and explain each parameter field (4 marks)

```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,  
struct timeval *timeout)
```

- (a) `int nfds`
- (b) `fd_set *readfds`
- (c) `struct timeval *timeout`

18. Why is `select()` considered a *destructive* function? (1 mark)

19. Write a complete C code block that performs the following steps in a TCP client:

- Creates a socket using IPv4 and TCP.
- Initializes a `sockaddr_in` struct with an IP address and port from `argv[1]` and `argv[2]`.
- Connects the socket to the server.
- Sends the message `argv[3]` to the server.
- Receives a response into a buffer.

Assume `BUFFSIZE` is predefined. You may use `inet_addr()`, `htons()`, `strlen()`, and standard system calls like `socket()`, `connect()`, `send()`, and `recv()`. Do not include the full `main()` function or headers.

20. Write a complete C code block that performs the following steps in a TCP server:

- Creates a socket using IPv4 and TCP.
- Binds the socket to port `argv[1]` on all local interfaces.
- Listens for incoming client connections with a maximum pending queue.
- Accepts a client connection.
- Receives data into a buffer and echoes it back to the client.

Assume `BUFFSIZE` and `MAXPENDING` are predefined. You may use system calls such as `socket()`, `bind()`, `listen()`, `accept()`, `recv()`, and `send()`, along with `htonl()`, `htons()`, and `memset()`. Do not include the full `main()` function or headers.

21. Write a C code block that uses `select()` to implement a concurrent TCP server. Your code should:

- Loop through all file descriptors to check for incoming connections or data.
- Accept new connections on the listening socket and add them to the active file descriptor set.
- Handle input on already-connected sockets using a function `read_from_client(int fd)`.
- Remove a socket from the set and close it if the client disconnects.

Assume the sets `active_fd_set` and `read_fd_set` are already declared, and that `sock` is the listening socket. The function `read_from_client()` is already defined. Do not write the full `main()` function or include headers.