Recitation 7: TCP Socket Programming

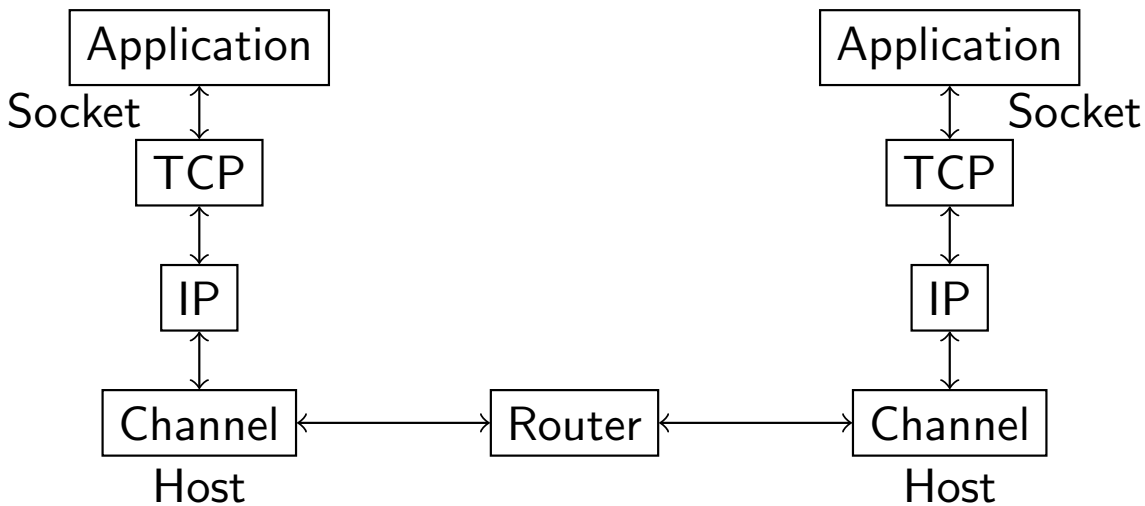
CSCI 4016: Introduction to Operating Systems

Overview

- TCP/IP
- Socket programming
- Exercises
  - Simple message transfer
  - Struct message transfer

TCP

- Protocol suite for data transfer in network
  - TCP - Transmission Control Protocol
  - IP - Internet Protocol
  - UDP - User Datagram Protocol
- TCP/IP



TCP

- Reliable byte-stream channel
  - Detect and recover from the losses, duplications and other errors experienced by IP
- Connection-oriented protocol
  - Programs must establish connection using handshake mechanism

Sockets

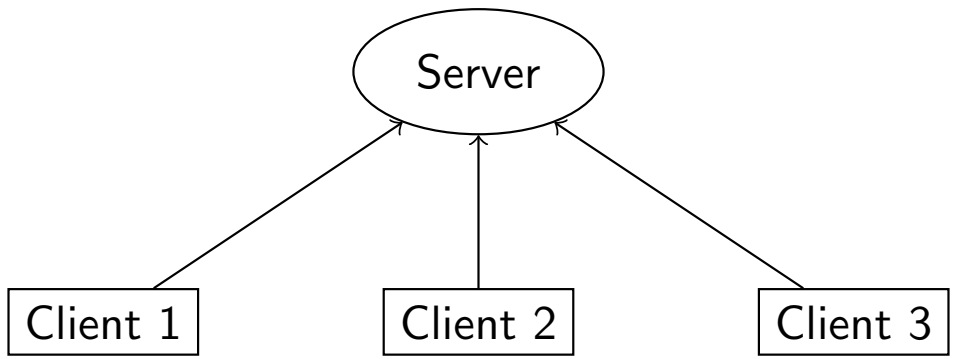
- Abstraction/File descriptor used by applications for sending and receiving data
- Uniquely identified by
  - An internet address (IPv4/IPv6)
  - An end-to-end protocol (TCP/UDP)
  - A port number (communication endpoint number from 0-65535)
- Types
  - Stream socket - TCP
  - Datagram socket - UDP

Socket primitives

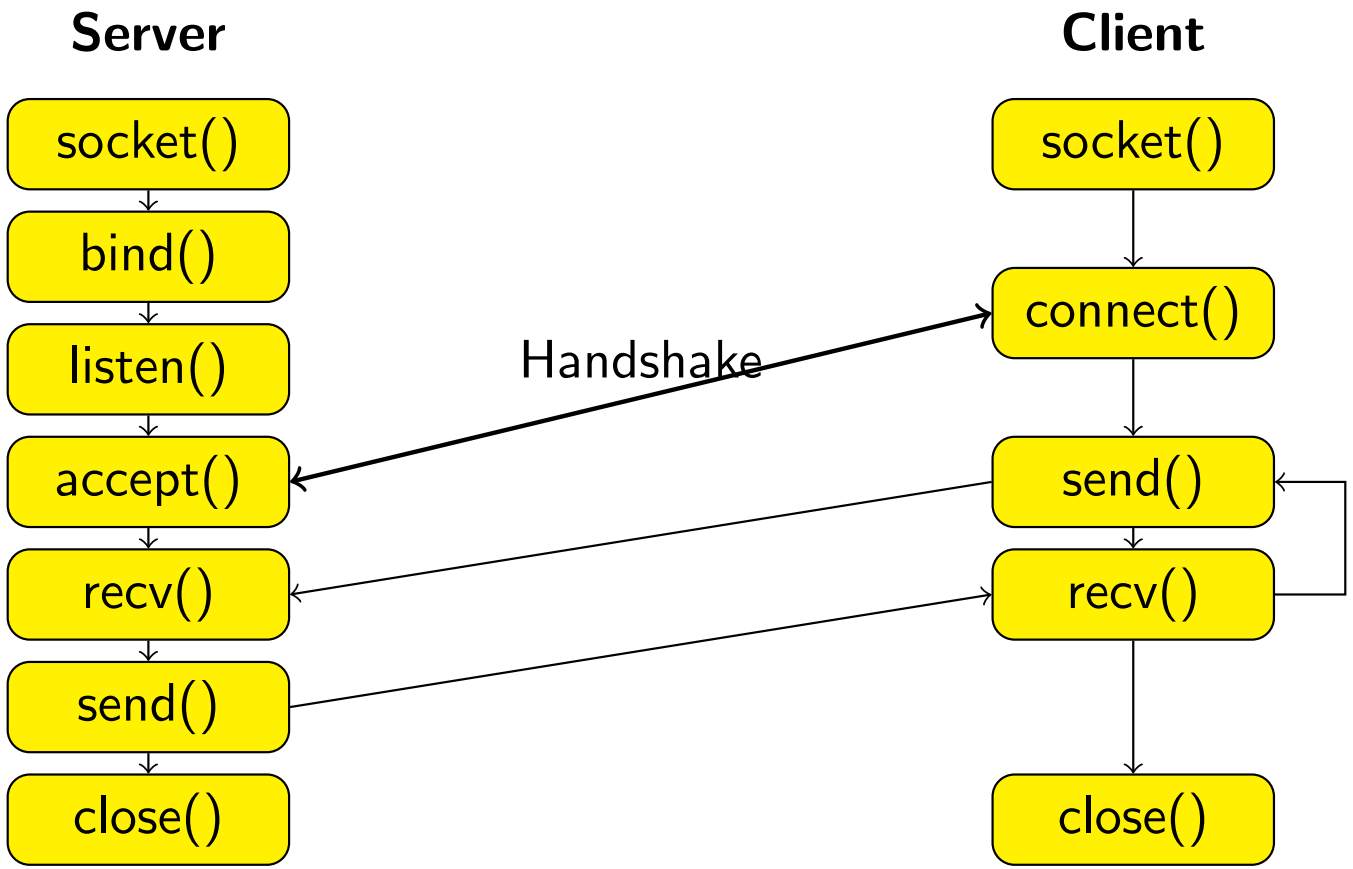
Primitives	**Usage**
socket	Create a new communication endpoint
bind	Attach an address to socket
listen	Listen on created socket for requests
accept	Accepts connection request and creates a new socket for the connection
connect	Attempts to establish a connection
send	Send data over connection
recv	Receive data over connection
close	Close the socket connection

Client-Server Model

- Server
  - Waits and responds to clients
  - Passive socket
- Client
  - Initiates communication with server
  - Should know the server address and port number
  - Active socket



Client - Server Communication - TCP



socket

```
1  #include <sys/types.h>
2  #include <sys/sockets.h>
3
4  int socket(int domain, int type, int protocol);
```

- **domain**: AF_INET - IPv4 protocols, our focus (check man socket)
- **type**: type of socket
  - SOCK_STREAM - reliable connection-oriented (our focus - tcp)
  - SOCK_DGRAM - unreliable, connectionless
- **protocol**: protocol type
  - Set to 0, default protocol TCP

Returns socket descriptor on success and -1 on failure

bind

```
1  #include <sys/types.h>
2  #include <sys/sockets.h>
3  #include <arpa/inet.h>
4
5  int bind(int sockfd, const struct sockaddr *addr,
6           socklen_t addrlen);
```

sockfd: socket descriptor returned by socket()
addr: address to which socket should be bound

```
1  struct sockaddr {
2      sa_family_t sa_family; /* AF_INET */
3      char sa_data[14]; /*Family specific address*/
4  }
5
6  struct sockaddr_in {
7      sa_family_t sin_family; /* AF_INET */
8      in_port_t sin_port; /* Port number */
9      struct in_addr sin_addr; /* IPv4 address */
10 };
```

addrlen: size of addr in bytes

Returns 0 on success, -1 on failure

Endianness

```
1  int x = 0x76543210;
2  char *c = (char*) &x;
```

Big endian format:

Byte address	0x01	0x02	0x03	0x04
Byte content	0x76	0x54	0x32	0x10

Little endian format:

Byte address	0x01	0x02	0x03	0x04
Byte content	0x10	0x32	0x54	0x76

- Host byte order - little endian
- Network byte order - big endian
- We should convert host to network bytes when storing data in sockaddr

Endianness

```
1  #include <arpa/inet.h>
2
3  // host to network byte order
4  uint32_t htonl(uint32_t hostlong);
5  uint16_t htons(uint16_t hostshort);
6
7  // network to host byte order
8  uint32_t ntohl(uint32_t netlong);
9  uint16_t ntohs(uint16_t netshort);
```

Example for sockaddr_in

```
1  int sockid = socket(AF_INET, SOCK_STREAM, 0);
2
3  struct sockaddr_in addr;
4  addr.sin_family = AF_INET;
5  addr.sin_port = htons(5100);
6  addr.sin_addr.s_addr = htonl(INADDR_ANY);
7
8  bind(sockid, (struct sockaddr *) &addr, sizeof(addr));
```

- **INADDR_ANY**: bind to any network interface
- Usually if we know the IP address, we can replace htonl(INADDR_ANY) with inet_addr(IP)
- Most of the time the IP is dynamic, hence using INADDR_ANY should resolve the IP

```
1  in_addr_t inet_addr(const char *ip);
```

Converts IP address to binary form and returns it

listen

```
1  #include <sys/types.h>
2  #include <sys/sockets.h>
3
4  int listen(int sockfd, int backlog);
```

- **sockfd**: socket descriptor from socket()
- **backlog**: maximum number of requests that can be queued. If a client request comes after queue is full, an error is raised at client side
- Non-blocking call, immediate returns after enabling listening on the socket with backlog queue length
- sockfd is only for listening and not for sending and receiving data
- New socket descriptors are created per connection by accept()

Returns 0 on success and -1 on failure

connect

```
1  #include <sys/types.h>
2  #include <sys/sockets.h>
3  #include <arpa/inet.h>
4
5  int connect(int sockfd, const struct sockaddr *addr,
6              socklen_t addrlen);
```

- **sockfd**: socket descriptor created by socket()
- **addr**: address information of the server or the party to which connection should be established
- **addrlen**: size of addr
- connect is a blocking call

Returns 0 on establishing connection and -1 on failure

accept

```
1  #include <sys/types.h>
2  #include <sys/sockets.h>
3  #include <arpa/inet.h>
4
5  int accept(int sockfd, struct sockaddr *addr,
6             socklen_t *addrlen);
```

- **sockfd**: socket descriptor created by socket() and passed to listen()
- **addr**: initially empty and populated during the call with the address information of the client/party sending request
- **addrlen**: size of addr
- Accept is a blocking call

Returns a new socket descriptor that can be used for sending and receiving data and -1 on failure

send, recv

```
1  #include <sys/types.h>
2  #include <sys/sockets.h>
3
4  ssize_t send(int sockfd, const void *buf, size_t len,
5               int flags);
6  ssize_t recv(int sockfd, void *buf, size_t len,
7               int flags);
```

- **sockfd**: socket descriptor returned by accept()
- **buf**:
  - send: data to be send to the receiving party
  - recv: buffer to receive data from the sending party (preallocate memory)
- **len**: length of the buffer
- **flags**: sets the behavior of send and recv (check man page), by default it is 0
- send and recv are blocking calls

Returns number of bytes sent or received and -1 on failure

close

```
1  #include <sys/types.h>
2  #include <sys/sockets.h>
3
4  int close(int sockfd);
```

- closes sockfd
- Returns 0 on success and -1 on failure

Exercise

1. In samples/eg1, we have provided you with a client and server code for one way communication from client to server. In this exercise, you will create a chat messenger between server and client. The client will initiate the chat with the server and the communication continues till client sends "END" to the server. Then both server and client exits. (check output1.mkv for expected output)
2. In samples/eg2, we have provided you with a client-server code similar to samples/eg1. However, a struct is passed between client and server during communication (PA2 - phase 2). In this exercise, you will create a single server and multiple clients, where each client will communicate in parallel with the server. The server is a daemon, ie, it keeps waiting for new requests similar to real servers. Hint: use threads for each client connection. (check output2.mkv for expected output)

Deliverables

- Use the deliver target given in exercises/Makefile to create the required zip
- Please check what deliver does before running it
- Submit the zip to Gradescope by July 25th, 2025, 11:59 pm

References

- Introduction to Sockets Programming in C using TCP/IP
- TCP/IP Sockets in C: Practical Guide for Programmers by Michael, J. Donahoo, and Kenneth L. Calvert