Lab 11 - Socket Programming I

Nov 20, 2023

CSCI 4061

Introduction to Operating Systems

University of Minnesota
Driven to Discover℠

Overview

1. Activity - Server Client Chatting
2. TCP/IP Socket Programming
    a. TCP/IP
    b. Socket APIs
3. Expected output

Activity: Server Client Chat

Given a server and a client

Start server followed by client

The client and server will chat back and forth, with the client initiating the chat

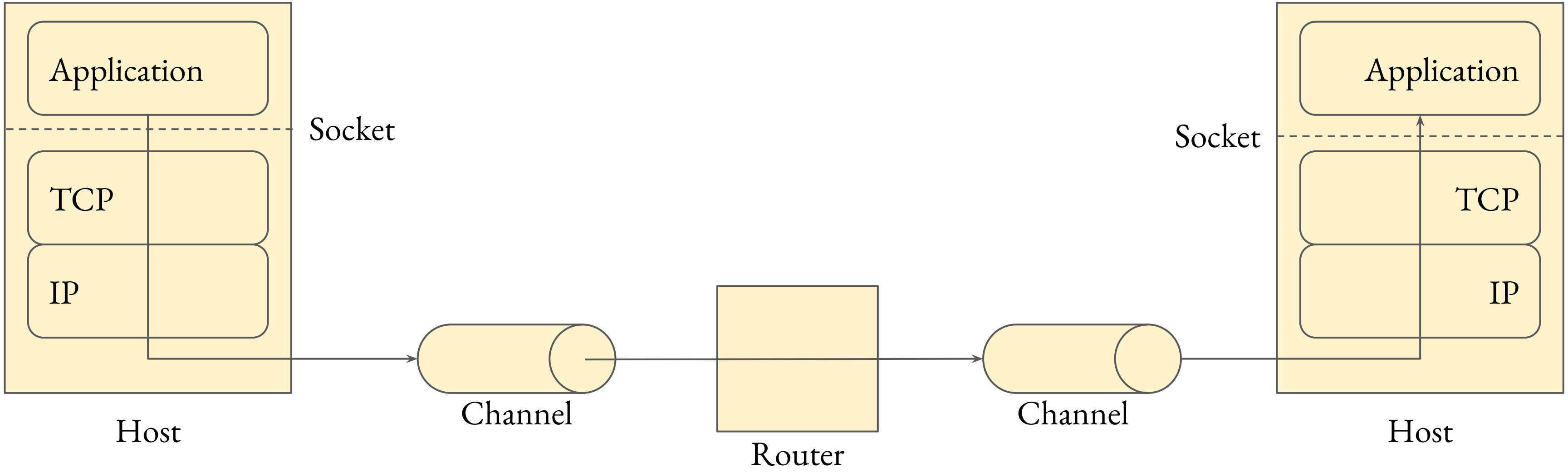
The chat should continue until the client sends END message to the server

The server and client will run on the same machine, but on different terminals

You can use the sample server and client code as starting point. You just need to account for the back and forth send and recv in the code. Rest of the code is almost the same.

TCP/IP

- 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
- For more information, refer TCP/IP Sockets in C

Sockets

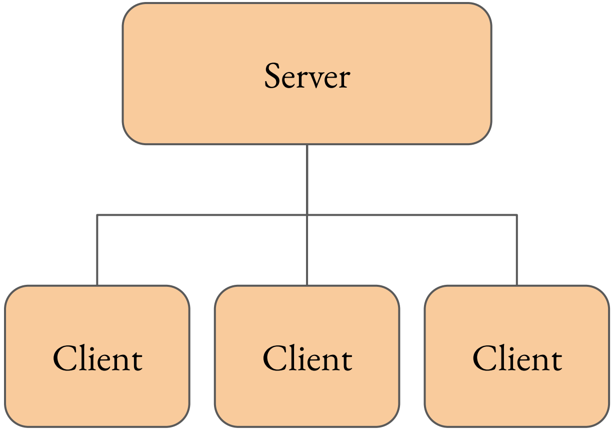
- Abstraction/File descriptor used by applications for sending and receiving data
- Uniquely identified by
  - An internet address (IPv4/IPv6)
  - An end-to-end protocol (TCP/UDP)
  - A port number (communication endpoint number from 0-65535)
- Types
  - Stream socket - TCP
  - Datagram socket - UDP

Socket primitives

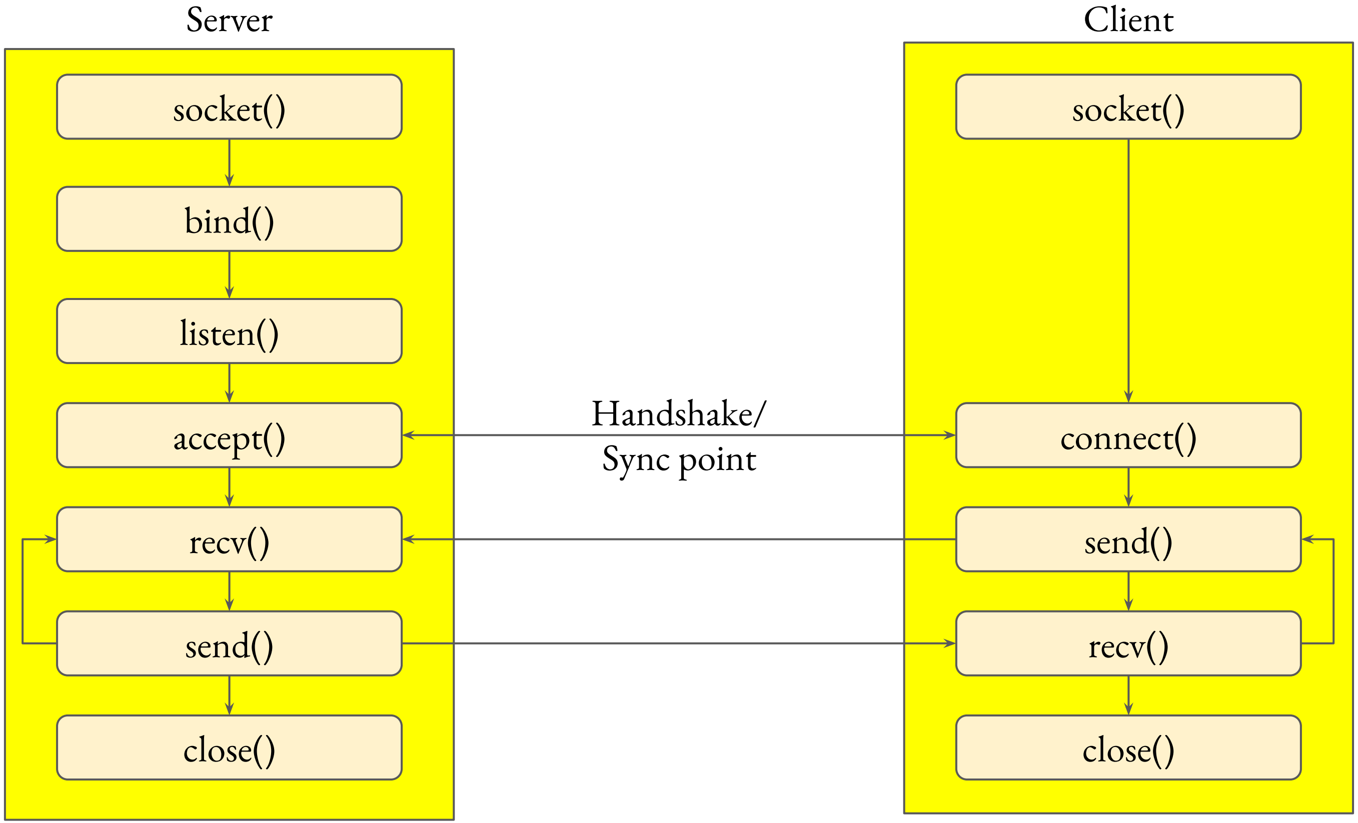
Primitive	**Usage**
socket	Create a new communication endpoint
bind	Attach an address to socket
listen	Listen on created socket for requests
accept	Accepts first request and creates a new socket for the connection
connect	Attempts to establish a connection
send	Send data over connection
recv	Receive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



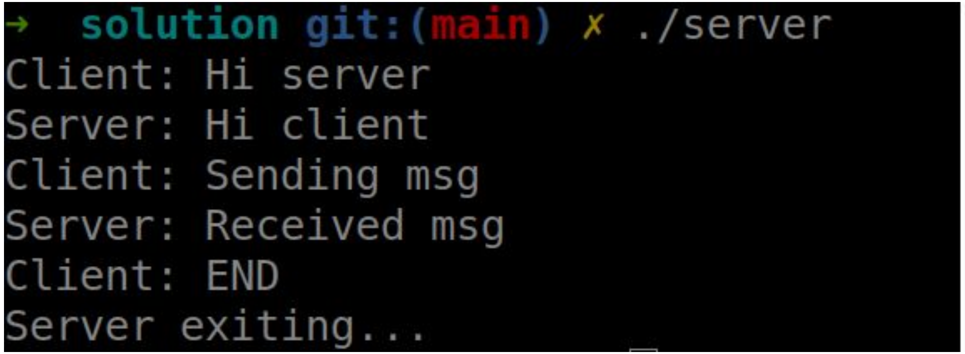
Client - Server Communication (TCP)



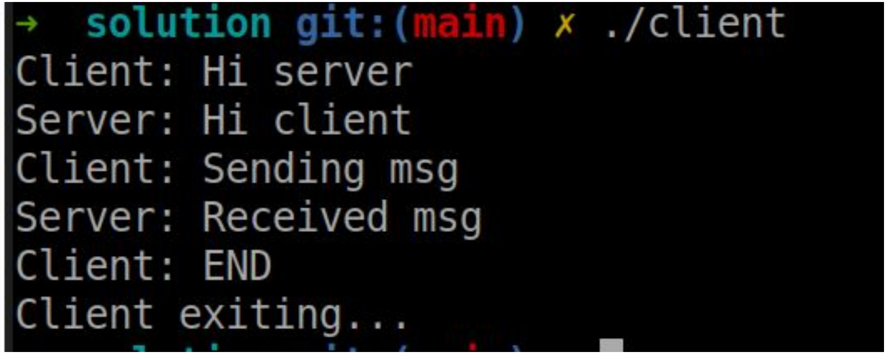
Have samples/server.c and samples/client.c open

Socket APIs are given towards end of the presentation

Expected Output



```
→  solution git:(main) ✗ ./server
Client: Hi server
Server: Hi client
Client: Sending msg
Server: Received msg
Client: END
Server exiting...
```



```
→  solution git:(main) ✗ ./client
Client: Hi server
Server: Hi client
Client: Sending msg
Server: Received msg
Client: END
Client exiting...
```

- mkv video file present in the zip for the same.

Deliverables

This time we have provided you with just the server.c and client.c file. You can use the sample server.c and client.c as template code and complete the back and forth communication between server and client.

Submit the deliverables to Gradescope as a zip by Nov 21, 11:59 pm.

- server.c
- client.c
- output.txt - copy output on the screen

socket

```
#include <sys/types.h>
#include <sys/sockets.h>

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)

          SOCK_DGRAM - unreliable, connectionless

     protocol: protocol type

          Set to 0, default protocol TCP

Returns socket descriptor on success and -1 on failure
```

bind

```
#include <sys/types.h>
#include <sys/sockets.h>
#include <arpa/inet.h>

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

      sockfd: socket descriptor returned by socket()

      addr: address to which socket should be bound, depends on the protocol
           Generic structure:
                struct sockaddr {
                      sa_family_t  sa_family;     /* AF_INET */
                      char         sa_data[14];   /*Family specific address*/
                }
           TCP/IP specific: can be casted to sockaddr
                struct sockaddr_in {
                      sa_family_t     sin_family;     /* AF_INET */
                      in_port_t       sin_port;       /* Port number */
                      struct in_addr  sin_addr;       /* IPv4 address */
                };

      addrlen: size of addr in bytes
Returns 0 on success, -1 on failure
```

Endianess

```
int x = 0x76543210;
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

Endianess

```
#include <arpa/inet.h>

// host to network byte order

uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

// network to host byte order

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);
```

Example for sockaddr_in

```
int sockid = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in addr;
Addr.sin_family             = AF_INET;
Addr.sin_port          = htons(5100);
Addr.sin_addr.s_addr  = htonl(INADDR_ANY);

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). However, most of the time the IP is dynamic, hence using INADDR_ANY should resolve the IP.

```
in_addr_t inet_addr(const char *ip);
     Converts IP address to binary form and returns it
```

listen

```
#include <sys/types.h>
#include <sys/sockets.h>

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

```
#include <sys/types.h>
#include <sys/sockets.h>
#include <arpa/inet.h>

int connect(int sockfd, const struct sockaddr *addr, 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

```
#include <sys/types.h>
#include <sys/sockets.h>
#include <arpa/inet.h>

int accept(int sockfd, struct sockaddr *addr, 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

```
#include <sys/types.h>
#include <sys/sockets.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);

ssize_t recv(int sockfd, void *buf, size_t len, 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

```
#include <sys/types.h>
#include <sys/sockets.h>

int close(int sockfd);

     closes sockfd



Returns 0 on success and -1 on failure
```

References

- Introduction to Sockets Programming in C using TCP/IP
- TCP/IP Sockets in C