USAGE

to start server

```
./ircserv <port> <password>
```

to start client using irssi

```
/set nick <name>
/set user_name <user name>
 /set real_name <real_name
/connect localhost <port> <password>
```

to start client in docker (change name irssi to irssi1 etc for each client)

```
docker run -it --rm --name irssi --network host irssi
/set nick <name>
/set user_name <user name>
 /set real_name <real_name
/connect host.docker.internal <port> <password>
```

Commands irssi

```
/oper 127.0.0.1 pa$$word
(will show @ operator name)
/kill <nick> reason
/join <channel>{,<channel>} [<key>{,<key>}]
(above means list channels seperated by commas and the list their keys. Can add # for regular or & for local before channel)
/msg <nick> <message>
/kick channel, channel user, user reason
/part <channel>{,<channel>} [<reason>]
/topic <channel> : [<topic>]
/invite <nickname> <channel>
/mode<target>[<mode string> [<mode arguments>...]]
```

Mode strings can be i, t, k, o or l as below and can have +(set) or – (remove)
· i: Set/remove Invite-only channel
· t: Set/remove the restrictions of the TOPIC command to channel operators
· k: Set/remove the channel key (password)
· o: Give/take channel operator privilege
· l: Set/remove the user limit to channel
/quit
/disconnect

To start client locally

```
nc localhost <password>
```

```
PASS <pass>
NICK <nick>
USER <nick> 0 <localhost> :<first_name last_name>
```

Commands locally

```
OPER 127.0.0.1 pa$$word
kill <nick> reason
PRIVMSG <nick to be sent to> : message
JOIN <channel>{,<channel>} [<key>{,<key>}]
above means list channels seperated by commas and the list their keys
KICK channel, channel user, user reason
PART <channel>{,<channel>} [<reason>]
TOPIC <channel> : [<topic>]
INVITE <nickname> <channel>
MODE <target>[<mode string> [<mode arguments>...]]
```

Mode strings can be i, t, k, o or l as below and can have +(set) or – (remove)
· i: Set/remove Invite-only channel

· t: Set/remove the restrictions of the TOPIC command to channel operators
· k: Set/remove the channel key (password)
· o: Give/take channel operator privilege
· l: Set/remove the user limit to channel
QUIT
DISCONNECT
USING VALGRIND

make docker

make re

make leaks (this starts server also)

for client

in another terminal

make docker

irssi

see using irssi above

or local

nc localhost <port>

```
docker run -it --cap-add=SYS_PTRACE --security-opt
seccomp=unconfined --security-opt apparmor=unconfined --
name 42-valgrind1 --network host --rm -v
"$PWD:/home/vscode/src" valgrind "/bin/zsh"
```

EXTERNAL FUNCTIONS

socket:

Purpose: Creates a new socket.
Parameters:
domain: The protocol family (e.g., AF_INET for IPv4, AF_INET6 for IPv6).

type: The socket type (e.g., SOCK_STREAM for TCP, SOCK_DGRAM for UDP).
protocol: Specific protocol to be used (typically 0 for default).
Return: Returns a socket descriptor, or -1 on error.

close:

Purpose: Closes a file descriptor (including sockets).
Parameters:
fd: The file descriptor to close.
Return: Returns 0 on success, -1 on error.

setsockopt:

Purpose: Sets options for a socket.
Parameters:
socket: The socket descriptor.
level: The protocol level at which the option resides.
option_name: The specific option to set.
option_value: Pointer to the option value.
option_len: Size of the option value.
Return: Returns 0 on success, -1 on error.

getsockname:

Purpose: Retrieves the local address of a socket.
Parameters:
socket: The socket descriptor.
address: Pointer to a sockaddr structure to store the address.
address_len: Pointer to the size of the sockaddr structure.
Return: Returns 0 on success, -1 on error.

getprotobyname:

Purpose: Retrieves protocol information based on its name.
Parameters:
name: The name of the protocol.
Return: Returns a pointer to a protoent structure or NULL on error.

gethostbyname:

Purpose: Resolves a hostname to an IPv4 address.
Parameters:
name: The hostname to resolve.

Return: Returns a pointer to a hostent structure or NULL on error.

getaddrinfo:

Purpose: Resolves a hostname and service into address structures.
Parameters:
node: The hostname or IP address.
service: The service name or port number.
hints: A pointer to a addrinfo structure with hints.
res: A pointer to the result.
Return: Returns 0 on success, non-zero on error.

The struct addrinfo structure is part of the <netdb.h> header in C and is typically used with functions like getaddrinfo. Here's how the struct addrinfo is defined:

```
struct addrinfo {
   int            ai_flags;      // AI_PASSIVE, AI_CANONNAME, etc.
   int            ai_family;     // AF_INET, AF_INET6, AF_UNSPEC
   int            ai_socktype;   // SOCK_STREAM, SOCK_DGRAM
   int            ai_protocol;   // IPPROTO_TCP, IPPROTO_UDP
   socklen_t      ai_addrlen;    // Length of ai_addr
   struct sockaddr *ai_addr;      // Actual socket address
   char           *ai_canonname;  // Canonical name for the node
   struct addrinfo *ai_next;      // Pointer to the next structure in the list
};
```

Here's a brief explanation of the fields in the struct addrinfo:

ai_flags: Specifies additional options. Common flags include AI_PASSIVE, indicating that the returned socket addresses will be suitable for binding a socket to accept incoming connections, and AI_CANONNAME, requesting a canonical name for the node.

ai_family: Specifies the desired address family, such as AF_INET for IPv4, AF_INET6 for IPv6, or AF_UNSPEC to allow either.

ai_socktype: Specifies the socket type, such as SOCK_STREAM for TCP or SOCK_DGRAM for UDP.

ai_protocol: Specifies the protocol to be used, such as IPPROTO_TCP or IPPROTO_UDP.

ai_addrlen: Length of the ai_addr field.

ai_addr: A pointer to a struct sockaddr containing the actual socket address.

ai_canonname: A pointer to a null-terminated string containing the canonical name of the host.

ai_next: A pointer to the next structure in the linked list. getaddrinfo may return multiple struct addrinfo structures in a linked list.

When you call getaddrinfo, it dynamically allocates memory for a linked list of struct addrinfo structures based on the provided hints and stores
the information about possible addresses in these structures. After you finish using this linked list,
you should free the memory using freeaddrinfo.

freeaddrinfo:

Purpose: Frees the memory allocated for the result of getaddrinfo.
Parameters:
res: The result obtained from getaddrinfo.
Return: No return value.

bind:

Purpose: Associates a socket with a specific local address and port.
Parameters:
socket: The socket descriptor.
address: The local address to bind to.
address_len: The size of the local address structure.
Return: Returns 0 on success, -1 on error.

connect:

Purpose: Initiates a connection on a socket.
Parameters:
socket: The socket descriptor.
address: The address to connect to.
address_len: The size of the address structure.
Return: Returns 0 on success, -1 on error.

listen:

Purpose: Marks a socket for listening to incoming connections.
Parameters:
socket: The socket descriptor.
backlog: Maximum length of the pending connections queue.
Return: Returns 0 on success, -1 on error.

accept:

Purpose: Accepts a new incoming connection on a listening socket.
Parameters:
socket: The listening socket descriptor.
address: Pointer to a sockaddr structure to store the client's address.
address_len: Pointer to the size of the sockaddr structure.
Return: Returns a new socket descriptor for the accepted connection, or -1 on error.

htons, htonl, ntohs, ntohl:

Purpose: Convert between host and network byte order for 16-bit and 32-bit values.
Parameters:
hostshort, hostlong: Value in host byte order.
Return: Returns the value in network byte order.

inet_addr:

Purpose: Converts an IPv4 address from text to binary form.
Parameters:
cp: The string containing the IPv4 address.
Return: Returns the binary representation of the IPv4 address.

inet_ntoa:

Purpose: Converts an IPv4 address from binary to text form.
Parameters:
in: The binary representation of the IPv4 address.
Return: Returns a string containing the text representation of the IPv4 address.

send, recv:

Purpose: Send and receive data on a connected socket.
Parameters:
socket: The socket descriptor.
buf: Pointer to the data buffer.
len: The length of the data buffer.
flags: Additional flags.
Return: Returns the number of bytes sent/received, or -1 on error.

signal, sigaction:

Purpose: Set a function to handle a specific signal.
Parameters:

signum: The signal to handle.
handler: The function to be called when the signal occurs.
Return: Returns the previous signal handler or SIG_ERR on error.

lseek:

Purpose: Moves the file offset.
Parameters:
fd: The file descriptor.
offset: The offset value.
whence: The reference point for the offset.
Return: Returns the new file offset, or -1 on error.

fstat:

Purpose: Gets file status information.
Parameters:
fd: The file descriptor.
buf: Pointer to the stat structure to store the information.
Return: Returns 0 on success, -1 on error.

fcntl:

Purpose: Performs various operations on a file descriptor.
Parameters:
fd: The file descriptor.
cmd: The operation to perform.
arg: Additional argument depending on the operation.
Return: Returns the result of the operation, or -1 on error.

poll:

Purpose: Waits for events on a set of file descriptors.
The poll function is a system call in Unix-like operating systems that allows a program to monitor multiple file descriptors (sockets, pipes, etc.) to see if I/O is possible on any of them. It's commonly used in event-driven and asynchronous programming to efficiently handle multiple I/O operations without blocking.

```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

Here's a breakdown of the poll function parameters:

fds: An array of structures of type struct pollfd, each representing a file descriptor that the program wants to monitor.

```
struct pollfd {
   int   fd;        // File descriptor to be monitored
   short events;    // Events to watch for (input/output/priority exceptions)
   short revents;   // Events that actually occurred
};
```

fd: The file descriptor to be monitored.
events: The events the program is interested in. It can be a combination of the following flags:
POLLIN: Data to be read is available.
POLLOUT: Data can be written (without blocking).
POLLERR: An error occurred on the file descriptor.
POLLHUP: The file descriptor has been disconnected.
revents: The events that actually occurred will be filled by the poll function.
nfds: The number of file descriptors in the fds array.

timeout: The maximum time (in milliseconds) that poll should wait for an event to occur. A value of -1 means to wait indefinitely,
and a value of 0 means to return immediately.

The poll function returns the number of file descriptors that have events or errors, or one of the following values:

-1: An error occurred, and errno is set to indicate the error.
0: The timeout specified by the timeout parameter expired, and no file descriptors had events.