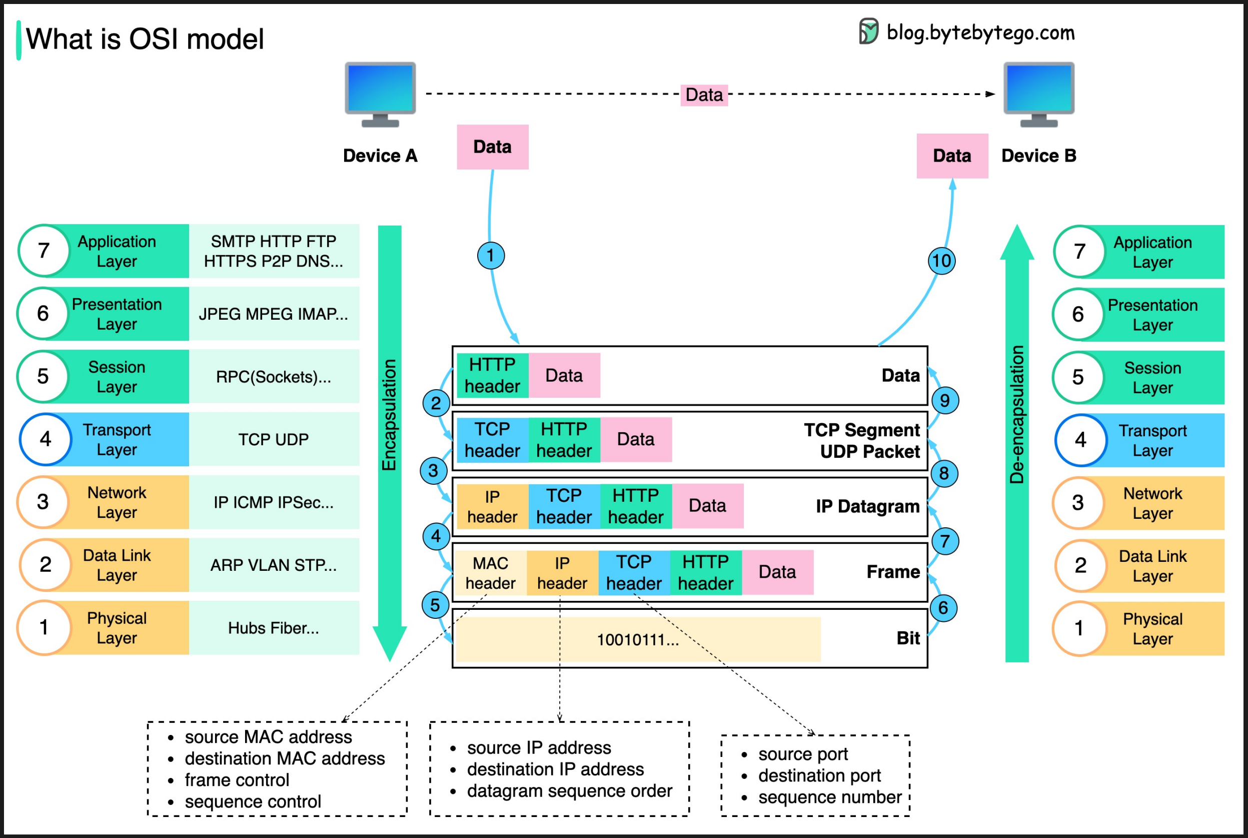
webserv

OSI Model



Webserv:

- http server (layer 7)
- interacts with the tcp/ip stack (layer 4) via socket system calls

Webserv handles http protocol

OS Kernel handles TCP/IP implementation

http

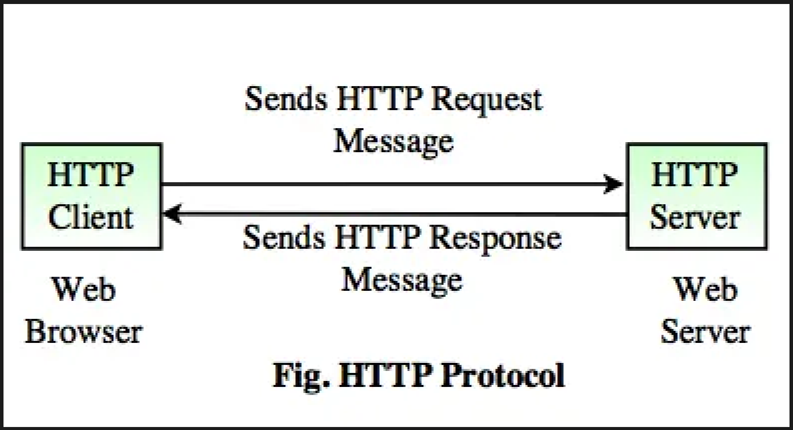


Fig. HTTP Protocol

Client

- Web browser etc
- Sends http request message to the server

Server

- Webserv, nginx etc
- Responds to the client request message

http messages

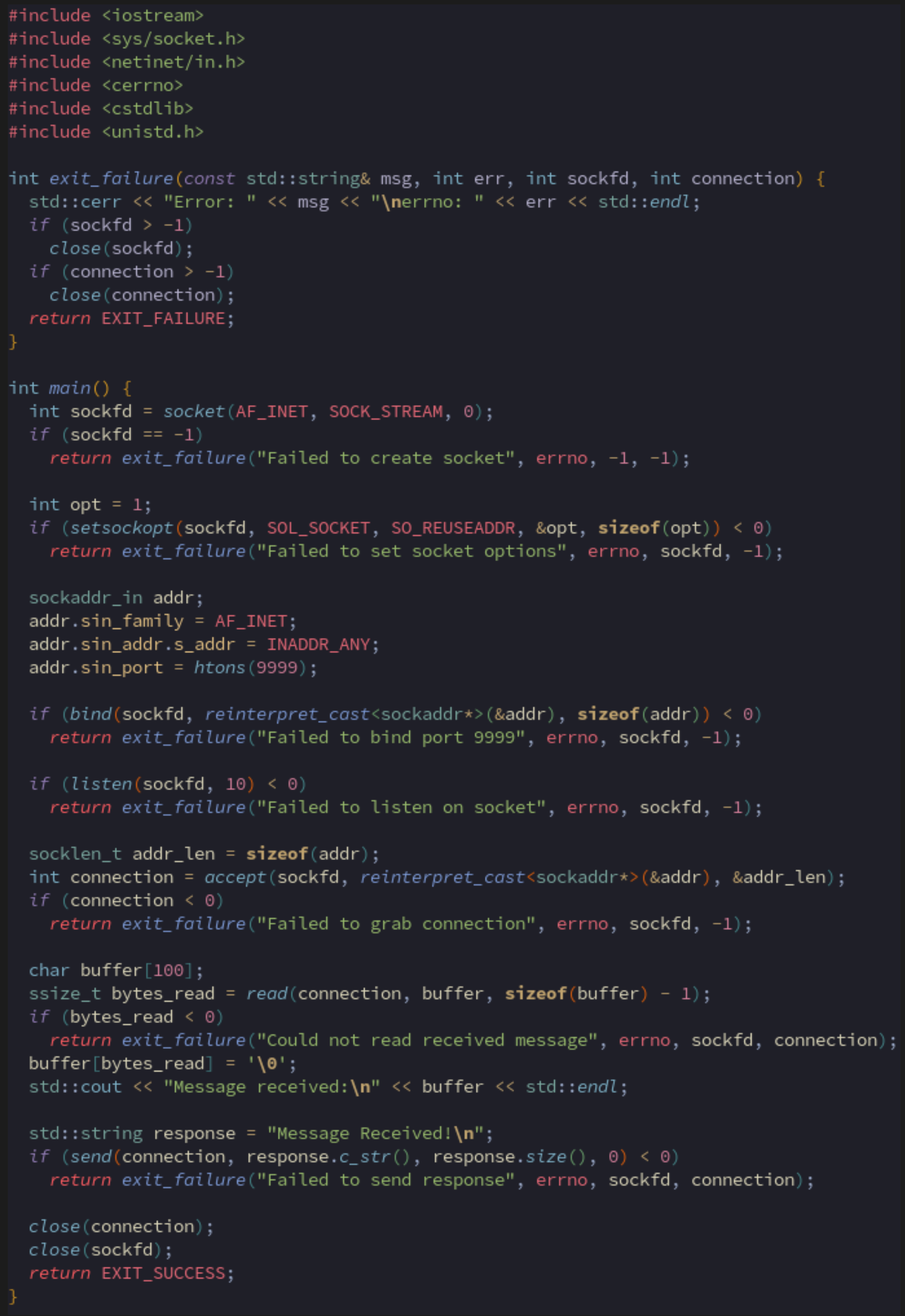
Client

- Web browser etc
- Sends http request to the server

Server

- Webserv, nginx etc
- Responds to the client request

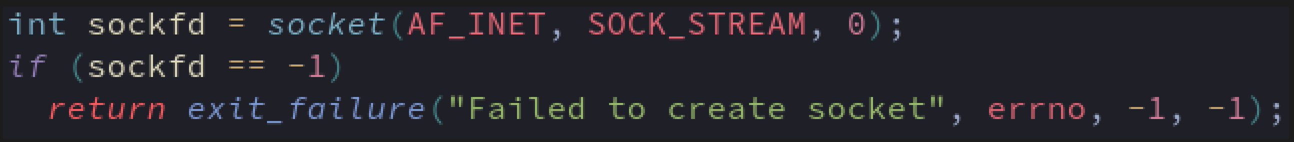
Basic Server



```cpp
#include <iostream>
#include <sys/socket.h>
#include <netinet/in.h>
#include <cerrno>
#include <cstdlib>
#include <unistd.h>

int exit_failure(const std::string& msg, int err, int sockfd, int connection) {
  std::cerr << "Error: " << msg << "\nerrno: " << err << std::endl;
  if (sockfd > -1)
    close(sockfd);
  if (connection > -1)
    close(connection);
  return EXIT_FAILURE;
}

int main() {
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd == -1)
    return exit_failure("Failed to create socket", errno, -1, -1);

  int opt = 1;
  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0)
    return exit_failure("Failed to set socket options", errno, sockfd, -1);

  sockaddr_in addr;
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(9999);

  if (bind(sockfd, reinterpret_cast<sockaddr*>(&addr), sizeof(addr)) < 0)
    return exit_failure("Failed to bind port 9999", errno, sockfd, -1);

  if (listen(sockfd, 10) < 0)
    return exit_failure("Failed to listen on socket", errno, sockfd, -1);

  socklen_t addr_len = sizeof(addr);
  int connection = accept(sockfd, reinterpret_cast<sockaddr*>(&addr), &addr_len);
  if (connection < 0)
    return exit_failure("Failed to grab connection", errno, sockfd, -1);

  char buffer[100];
  ssize_t bytes_read = read(connection, buffer, sizeof(buffer) - 1);
  if (bytes_read < 0)
    return exit_failure("Could not read received message", errno, sockfd, connection);
  buffer[bytes_read] = '\0';
  std::cout << "Message received:\n" << buffer << std::endl;

  std::string response = "Message Received!\n";
  if (send(connection, response.c_str(), response.size(), 0) < 0)
    return exit_failure("Failed to send response", errno, sockfd, connection);

  close(connection);
  close(sockfd);
  return EXIT_SUCCESS;
}
```

- Creates socket
- Sets options to allow address reuse
- Binds to port 9999
- Listens for connections on port 9999
- Reads message from port 9999
- Sends response to client
- Closes connection

int socket(int domain, int type, int protocol)



```c
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1)
  return exit_failure("Failed to create socket", errno, -1, -1);
```

Create socket. Returns a file descriptor for socket or -1 on error

Domain
The protocol the socket will use for communication:
- AF_UNIX / AF_LOCAL = Local communication
- AF_INET = IPv4 protocol
- AF_INET6 = IPv6 protocol
- AF_IPX: IPX Novell protocol

Type
Specifies if communication will be connectionless or persistent. Not all types are compatible with all domains:
- SOCK_STREAM = Two-way reliable communication (TCP)
- SOCK_DGRAM = Connectionless, unreliable (UDP)

Protocol
Normally there is only one protocol for each type, so the value zero can be used

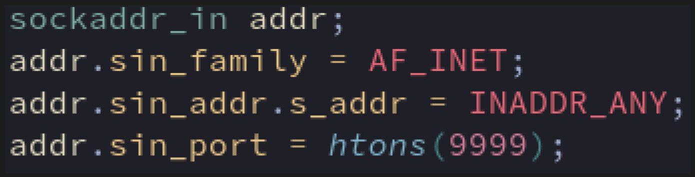
```c
int setsockopt(int sockfd, int level, int optname,
    const void optval[.optlen], socklen_t optlen)
```

```c
int opt = 1;
if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0)
  return exit_failure("Failed to set socket options", errno, sockfd, -1);
```

Set socket options. Used to allow immediate restarts after server stop. Without, OS keeps the the port in TIME_WAIT state for ~60 seconds. Returns 0 on success or -1 on error.

- **sockfd** = socket file descriptor
- **SOL_SOCKET** = option level (socket level)
- **SP_REUSEADDR** = which option to set. This option allows us to bind a port that is in the TIME_WAIT status
- **&opt** = pointer to the value (1 = enable, 0 = disable)
- **sizeof(opt)** = size of the value

sockaddr_in addr



```c
sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
addr.sin_port = htons(9999);
```

Struct defined in netinet/in.h that specifies where the socket binds and listens to. The “address” of the server

sin_family = AF_INET;

- The address family (IPv4 in this case)
- Tells the OS that this is an IPv4 structure

sin_addr.s_addr = INADDR_ANY;

- Equates to 0.0.0.0 (listen on all interfaces)
- If device has multiple connections, we can specify a particular one
- To specify address use htonl(*address*):
  - uint32_t address = (192 << 24) | (168 << 16) | (1 << 8) | 116;
    sin_addr.s_addr = htonl(address);

addr.sin_port = htons(9999);

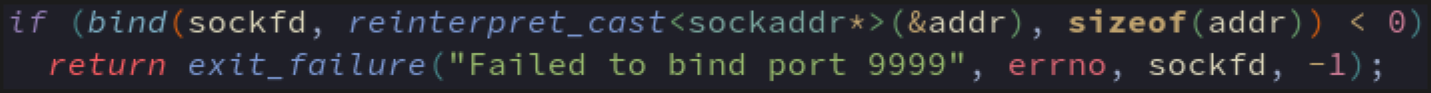
- The port number to bind to

Network / host conversions

- htons() - Host TO Network Short (16-bit) - for ports
- htonl() - Host TO Network Long (32-bit) - for IP addresses
- ntohs() - Network TO Host Short - reverse conversion
- ntohl() - Network TO Host Long - reverse conversion
- x86 processors are little-endian (multi-byte numbers stored 'backwards') whereas network protocols are big-endian.
- htons() converts from host byte order to network byte order:

```
Input: 9999 → 0x270F
Memory Before (host/little-endian): [0x0F] [0x27]
Memory After (network/big-endian):  [0x27] [0x0F] (Bytes swapped)
```

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



```
if (bind(sockfd, reinterpret_cast<sockaddr*>(&addr), sizeof(addr)) < 0)
  return exit_failure("Failed to bind port 9999", errno, sockfd, -1);
```

Assign an IP address and port to the socket. Returns 0 on success or -1 on error

sockfd
File descriptor to assign address to (the fd of our socket)

addr
Struct used to specify the address to assign to the socket

- sockaddr_in = IPv4 struct (sockaddr_in6 = IPv6, sockaddr_un = unix domain socket)
- Bind function works with all, so takes the generic base struct (reinterpret cast)

addrlen
size of addr

int listen(int s, int backlog)

```c
if (listen(sockfd, 10) < 0)
  return exit_failure("Failed to listen on socket", errno, sockfd, -1);
```

Marks a socket as passive - the socket will be used to accept connections. Return 0 for success, -1 for error

sockfd
File descriptor of the socket

Backlog
The maximum number of connections that will be queued before connections start being refused

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

```
socklen_t addr_len = sizeof(addr);
int connection = accept(sockfd, reinterpret_cast<sockaddr*>(&addr), &addr_len);
if (connection < 0)
  return exit_failure("Failed to grab connection", errno, sockfd, -1);
```

extracts an element from a queue of connections (created by listen) for a socket. Returns fd for connection or -1 on error

sockfd
File descriptor of the socket

reinterpret_cast<sockaddr*>(&addr)
Needs to be cast to sockaddr* similarly to bind()

addrlen
Unlike bind() this needs to be a pointer, it will be set to the size of the peer address

ssize_t send(int s, const void *msg, size_t len, int flags)

```
if (send(connection, response.c_str(), response.size(), 0) < 0)
  return exit_failure("Failed to send response", errno, sockfd, connection);
```

Transmits a message to another socket. Returns the number of octets sent on success, -1 on error

connection
File descriptor of the connection created by accept

Response.c_str()
The message to be sent

response.size()
The length of the message to be sent

flags
None used here, possible values:
- MSG_OOB: process out-of-band data
- MSG_DONTROUTE: bypass routing, use direct interface
- MSG_EOR: data completes record
- MSG_DONTWAIT: do not block
- MSG_EOF: data completes transaction
- MSG_NOSIGNAL: do not generate SIGPIPE on EOF