0x0C. Sockets

HOLBERTON SCHOOL TUNISIA

Overiew

- In this project, we will learn about sockets - a fundamental concept in network programming. Sockets provide an interface for communication between different processes over a network. By understanding how sockets work, we can create applications that can send and receive data over a network, enabling us to build powerful distributed systems. We will explore how to create sockets, bind them to specific addresses and ports, listen for incoming connections.

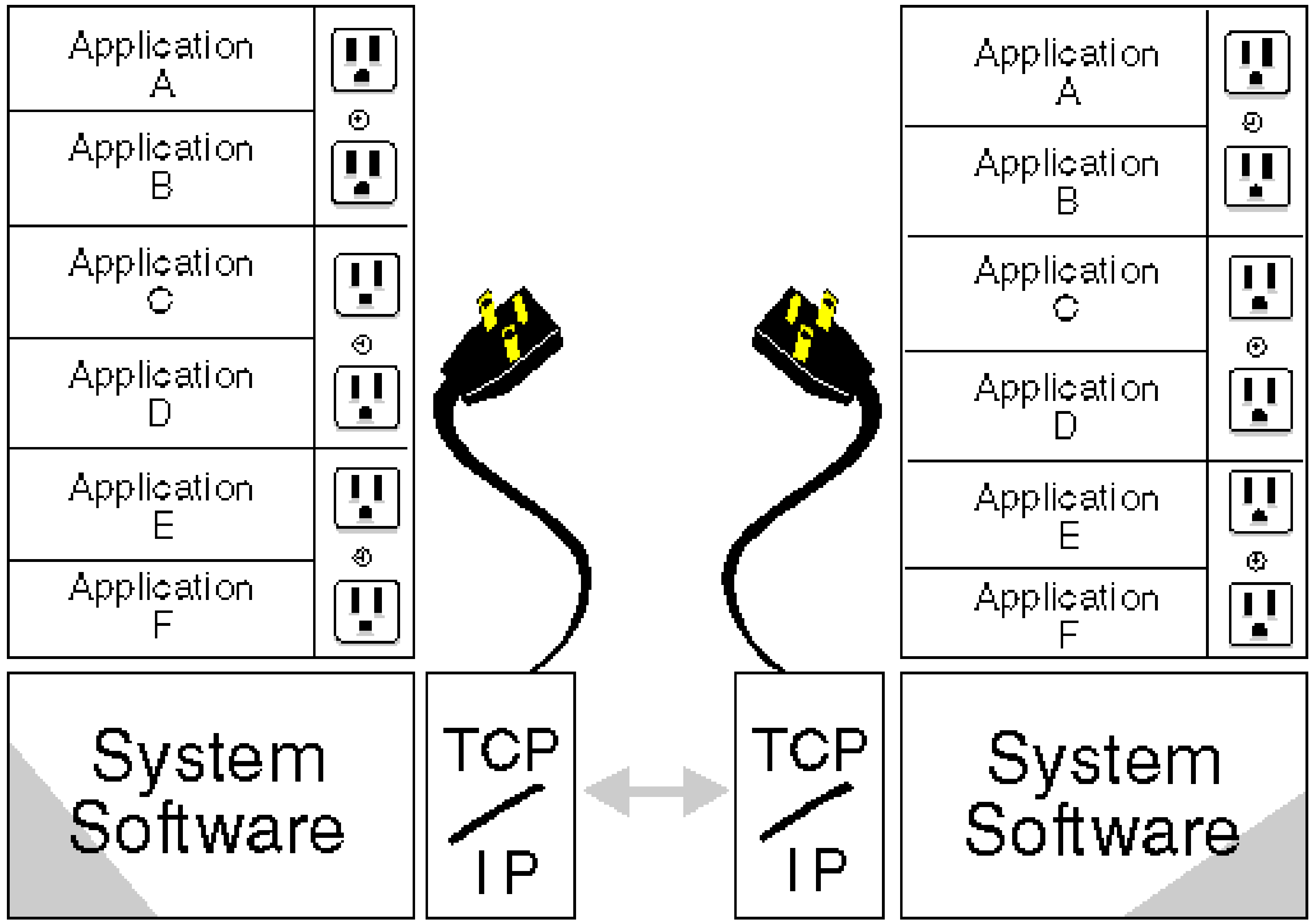
Let's get started

Our Plan and Things to discuss:

- What is a socket and how it is represented on a Linux/UNIX system
- What are the different types of sockets
- What are the different socket domains
- How to create a socket
- How to bind a socket to an address/port
- How to listen and accept incoming traffic
- How to connect to a remote application
- What is the the HTTP protocol
- How to create a simple HTTP server

Introduction

A socket can be thought of as an endpoint in a two-way communication channel. Socket routines create the communication channel, and the channel is used to send data between application programs either locally or over networks. Each socket within the network has a unique name associated with it called a socket descriptor—a fullword integer that designates a socket and allows application programs to refer to it when needed.



This figure shows many application programs running on a client and many application programs on a server. When the client starts a socket call, a socket connection is made between an application on the client and an application on the server.

You can read more here in this <u>link</u>

What is a socket and how it is represented on a Linux/UNIX system

On Linux/UNIX systems, a socket is represented by a file descriptor, which is a non-negative integer used to identify an open file or socket. A socket can be created using the socket system call, which takes three arguments: the address family (e.g., AF_INET for IPv4 or AF_INET6 for IPv6), the socket type (e.g., SOCK_STREAM for TCP or SOCK_DGRAM for UDP), and the protocol (usually 0, which allows the system to choose the appropriate protocol).

Once a socket is created, it can be bound to a specific IP address and port number using the bind system call. This allows other programs to connect to the socket and initiate a network connection. The socket can also be put into a listening state using the listen system call, which allows it to accept incoming connections from other programs.

When a connection is established, the socket can be used for data exchange using read and write system calls, which allow data to be sent and received over the network. Finally, when the connection is no longer needed, the socket can be closed using the close system call, which releases the file descriptor and frees up system resources.

What are the different types of sockets

There are several types of sockets that can be used in computer networking, including:

- **Stream sockets**: These provide a reliable, stream-oriented connection between two programs. Stream sockets are commonly used with the Transmission Control Protocol (TCP) in the Internet Protocol (IP) suite.
- **Datagram sockets**: These provide an unreliable, message-oriented connection between two programs. Datagram sockets are commonly used with the User Datagram Protocol (UDP) in the IP suite.
- **Raw sockets**: These allow programs to send and receive IP packets directly without the transport-layer protocol (TCP or UDP) encapsulating them. Raw sockets are often used for low-level network programming and for network security applications.

Each type of socket has its own advantages and disadvantages, and the choice of socket type will depend on the specific requirements of the application.

What are the different socket domains

In computer networking, a socket domain (also known as an address family) is a set of protocols and data structures used to identify network addresses and endpoints. Different socket domains are used to support different types of network connections and communication. Here are some common socket domains:

- AF_INET: This is the Internet Protocol version 4 (IPv4) address family. It is used for communication over IP networks and supports both stream (TCP) and datagram (UDP) sockets.
- AF_INET6: This is the Internet Protocol version 6 (IPv6) address family. It is used for communication over IPv6 networks and also supports both stream and datagram sockets.
- AF_UNIX: This is the Unix domain socket address family. It is used for communication between processes running on the same system and supports both stream and datagram sockets.
- AF_BLUETOOTH: This is the Bluetooth socket address family. It is used for communication over Bluetooth networks and supports both stream and datagram sockets.

What are the different socket domains

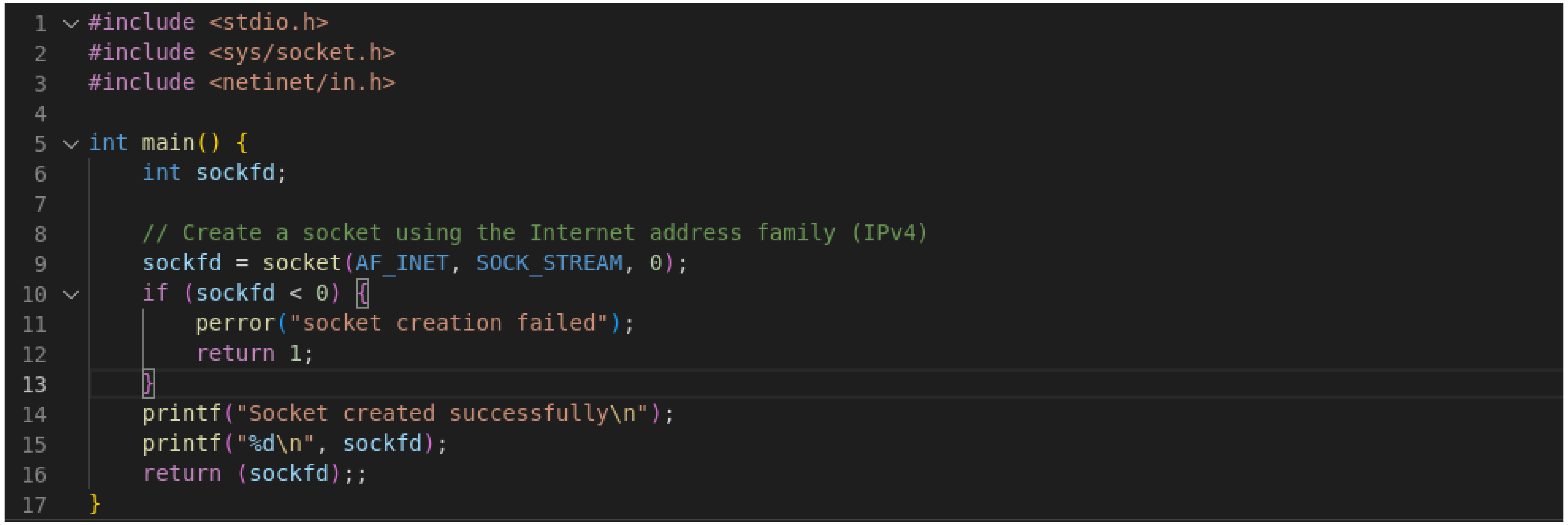
In computer networking, a socket domain (also known as an address family) is a set of protocols and data structures used to identify network addresses and endpoints. Different socket domains are used to support different types of network connections and communication. Here are some common socket domains:

- AF_PACKET: This is the packet socket address family. It is used for low-level access to network packets and supports raw sockets for receiving and sending packets directly.

Each socket domain has its own set of protocols, data structures, and options that can be used to configure and control network communication. The choice of socket domain depends on the specific requirements of the application.

How to create a socket in c

To create a socket in C, you can use the socket() system call provided by the operating system. Here is a basic example of how to create a socket:



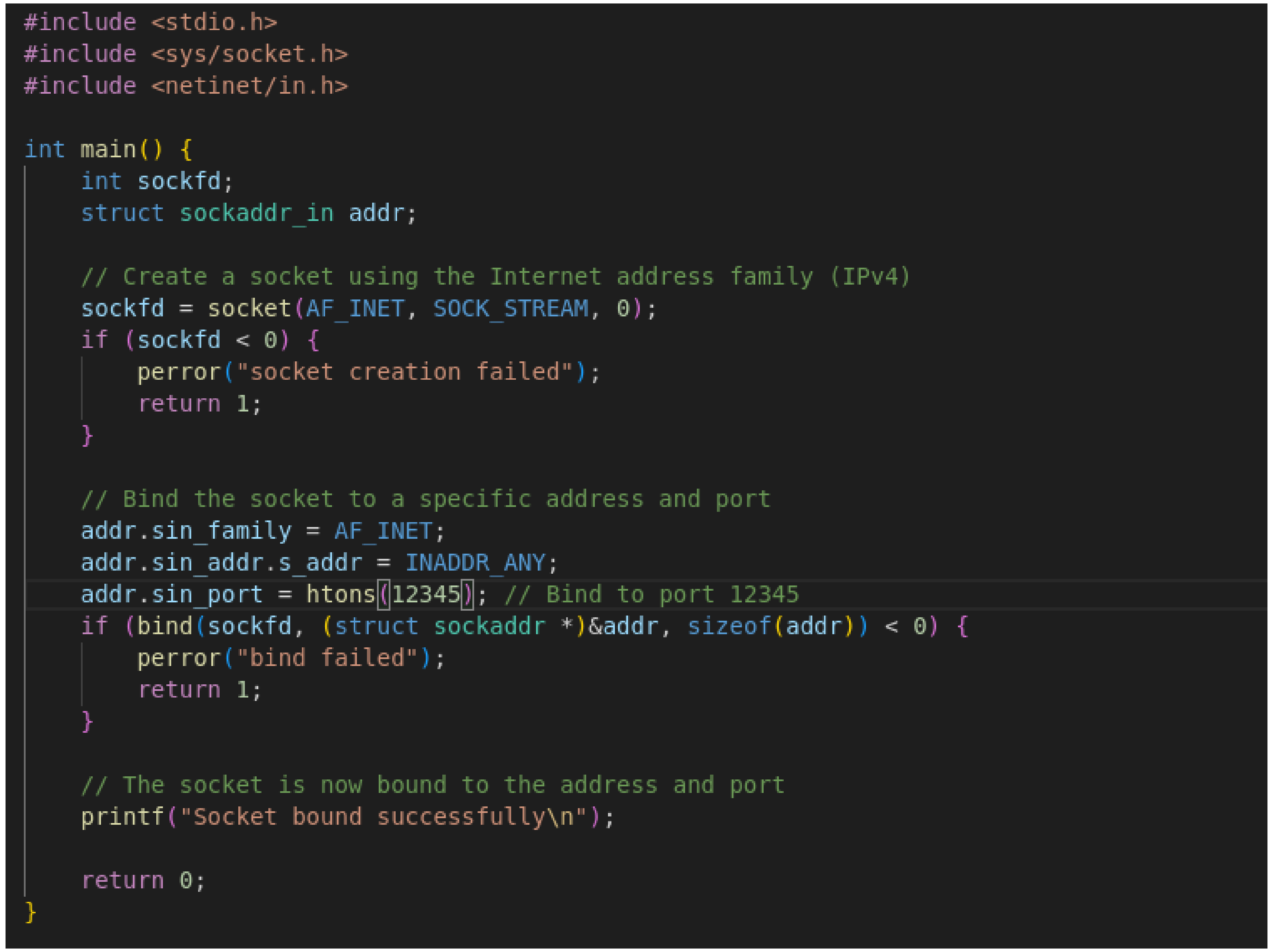
```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main() {
    int sockfd;

    // Create a socket using the Internet address family (IPv4)
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        return 1;
    }
    printf("Socket created successfully\n");
    printf("%d\n", sockfd);
    return (sockfd);;
}
```

In this example, we create a TCP stream socket using the socket() function. The first argument specifies the address family, which in this case is AF_INET for IPv4. The second argument specifies the socket type, which is SOCK_STREAM for TCP. The third argument is the protocol, which is 0 for the default protocol.

Once the socket is created, it can be used for sending and receiving data over the network.

How to bind a socket to an address/port

To bind a socket to a specific address and port in C, you can use the bind() system call. Here's an example of how to do it:

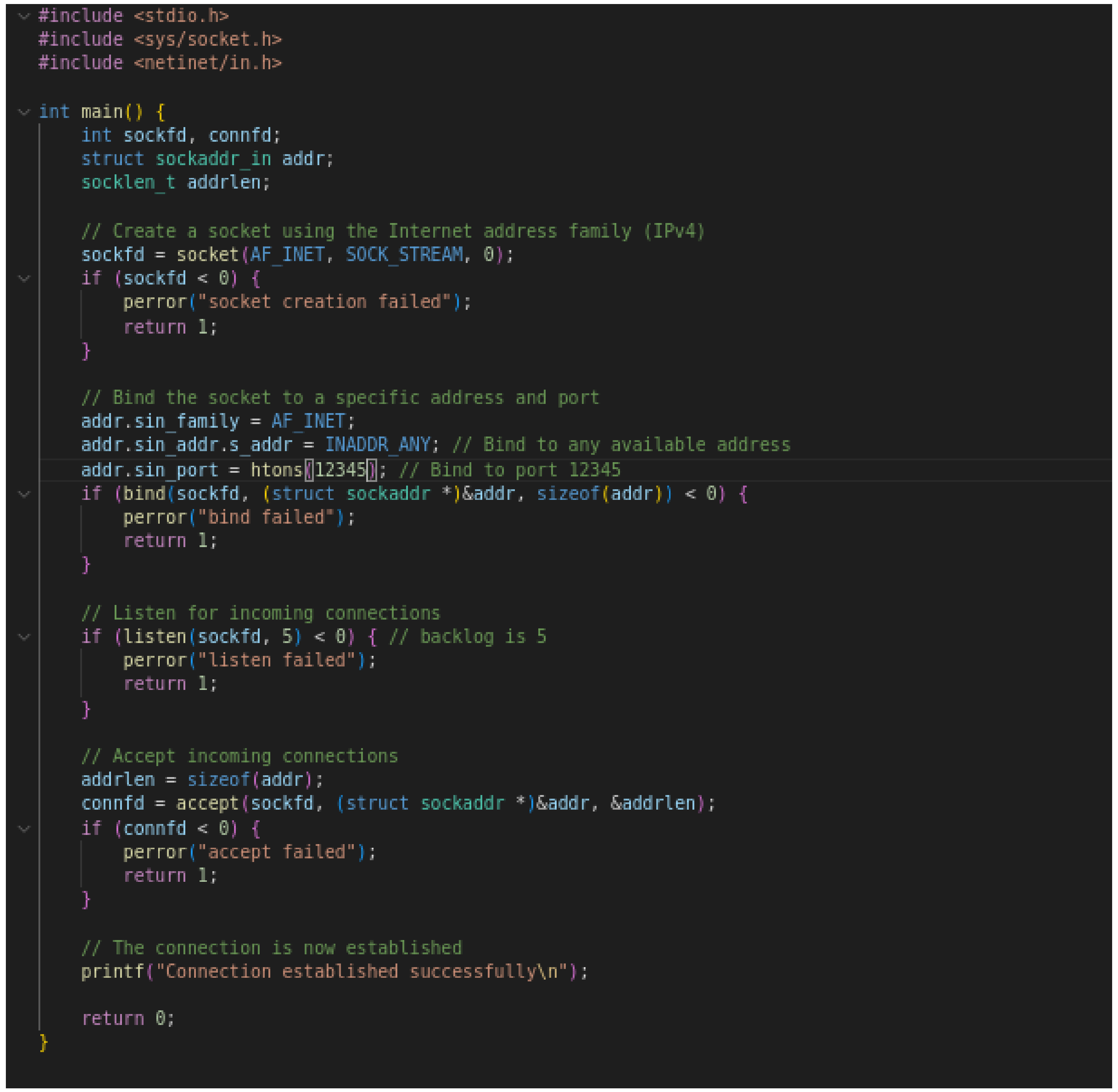


```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main() {
    int sockfd;
    struct sockaddr_in addr;

    // Create a socket using the Internet address family (IPv4)
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        return 1;
    }

    // Bind the socket to a specific address and port
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port = htons(12345); // Bind to port 12345
    if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("bind failed");
        return 1;
    }

    // The socket is now bound to the address and port
    printf("Socket bound successfully\n");

    return 0;
}
```

- In this example, we create a TCP stream socket using the socket() function as described in the previous answer. We then initialize a sockaddr_in structure called addr with the address and port we want to bind to.
- The sin_family field of the addr structure is set to AF_INET to indicate the Internet address family. The sin_addr.s_addr field is set to INADDR_ANY to bind the socket to any available network interface. Finally, the sin_port field is set to htons(1234) to bind the socket to port number 1234.
- Once the addr structure is initialized, we call the bind() function to bind the socket to the address and port. If the bind operation is successful, bind() returns 0. If it fails, it returns -1 and sets the errno variable to indicate the error. In the example, we use the perror() function to print an error message if the bind operation fails.

How to listen and accept incoming traffic

To listen for incoming connections on a socket in C, you can use the listen() function. Once a conne
received, you can accept it using the accept() function. Here's an example of how to do it:



```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main() {
    int sockfd, connfd;
    struct sockaddr_in addr;
    socklen_t addrlen;

    // Create a socket using the Internet address family (IPv4)
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        return 1;
    }

    // Bind the socket to a specific address and port
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY; // Bind to any available address
    addr.sin_port = htons(12345); // Bind to port 12345
    if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("bind failed");
        return 1;
    }

    // Listen for incoming connections
    if (listen(sockfd, 5) < 0) { // backlog is 5
        perror("listen failed");
        return 1;
    }

    // Accept incoming connections
    addrlen = sizeof(addr);
    connfd = accept(sockfd, (struct sockaddr *)&addr, &addrlen);
    if (connfd < 0) {
        perror("accept failed");
        return 1;
    }

    // The connection is now established
    printf("Connection established successfully\n");

    return 0;
}
```

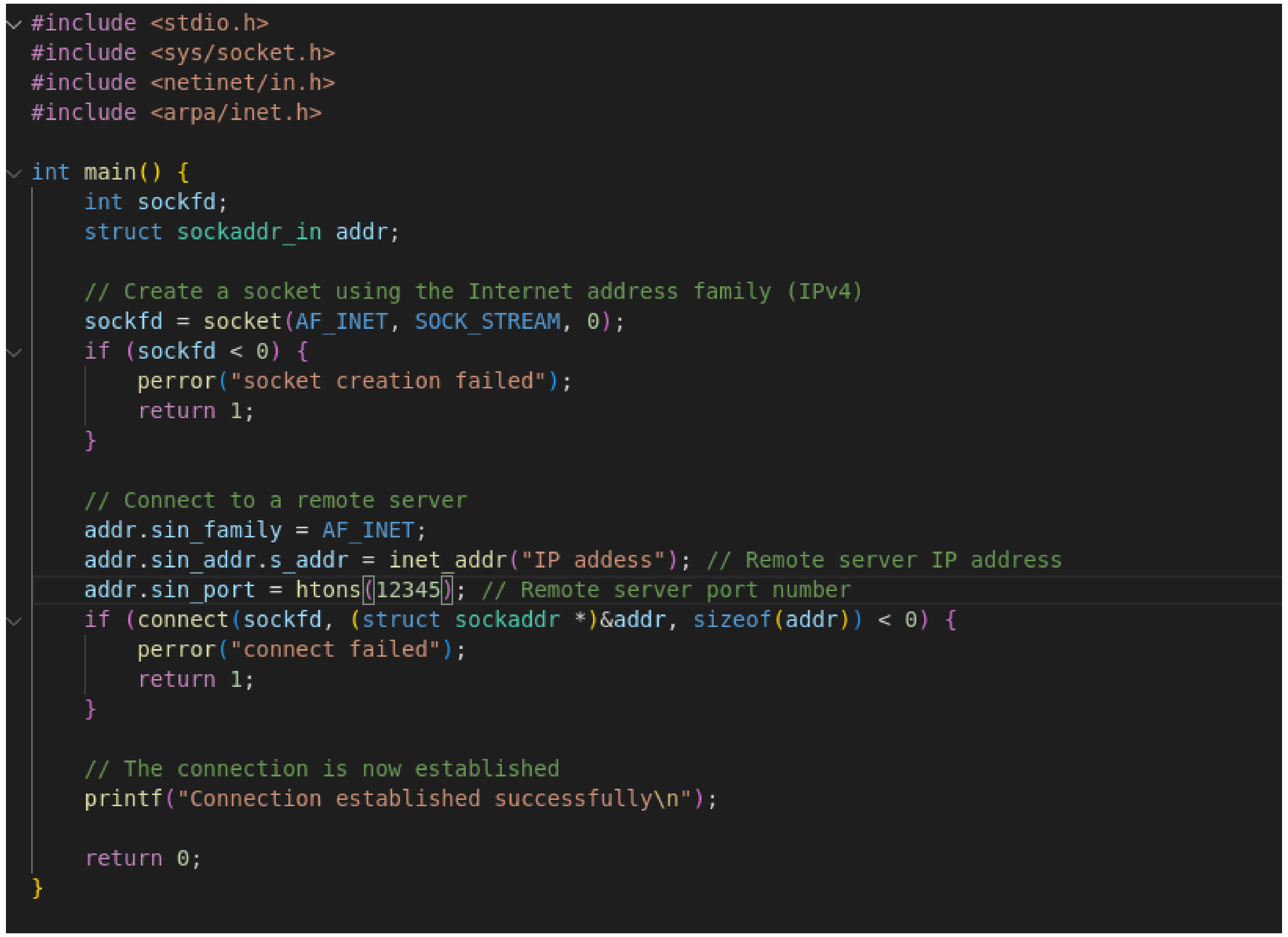
- In this example, we first create a TCP stream socket and bind it to a specific address and port as described in the previous answer. We then call the listen() function to listen for incoming connections on the socket. The second argument to listen() specifies the maximum length of the queue of pending connections.

- Once the socket is listening for connections, we can use the accept() function to accept incoming connections. The accept() function blocks until a connection is received. When a connection is received, accept() returns a new socket file descriptor that can be used to communicate with the client. The addr structure is filled with the address of the client.

- In the example, we use a socklen_t variable called addrlen to store the size of the addr structure. We pass a pointer to addrlen as the third argument to accept(). After the connection is accepted, we can use the connfd file descriptor to send and receive data with the client.

How to connect to a remote application

To connect to a remote application in C, you can use the connect() function. Here's an example of how to do it:



```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main() {
    int sockfd;
    struct sockaddr_in addr;

    // Create a socket using the Internet address family (IPv4)
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        return 1;
    }

    // Connect to a remote server
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr("IP addess"); // Remote server IP address
    addr.sin_port = htons(12345); // Remote server port number
    if (connect(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("connect failed");
        return 1;
    }

    // The connection is now established
    printf("Connection established successfully\n");

    return 0;
}
```

- In this example, we create a TCP stream socket as described in the previous answers. We then initialize a sockaddr_in structure called addr with the address and port of the remote application we want to connect to.
- The sin_family field of the addr structure is set to AF_INET to indicate the Internet address family. The sin_addr.s_addr field is set to the IP address of the remote server we want to connect to. In this example, we use the inet_addr() function to convert the IP address from a string to the appropriate binary format.
- Finally, the sin_port field is set to the port number of the remote server we want to connect to.
- Once the addr structure is initialized, we call the connect() function to connect to the remote application. If the connection is successful, connect() returns 0. If it fails, it returns -1 and sets the errno variable to indicate the error. In the example, we use the perror() function to print an error message if the connect operation fails.

What is the the HTTP protocol

- The HTTP (Hypertext Transfer Protocol) is an application-level protocol that is used to transmit data over the Internet. It is a request-response protocol that is used to transfer data between web servers and clients, such as web browsers.

- HTTP was designed to enable communication between web browsers and web servers. It defines a set of rules for sending and receiving data between these two types of software.

- HTTP uses a client-server model where the client sends a request to the server, and the server responds with a message containing the requested information. The client sends a request message to the server using a standard format that includes a method, a URL, and some headers. The method indicates the type of request being made, such as GET, POST, PUT, DELETE, and so on. The URL specifies the location of the resource being requested, and the headers provide additional information about the request, such as the type of data the client can accept.

What is the the HTTP protocol

Here's the steps to create a simple HTTP server using sockets:

- Create a socket: Create a socket using the socket() system call with the AF_INET (IPv4) or AF_INET6 (IPv6) domain, depending on your needs. Use the SOCK_STREAM type to create a TCP socket for reliable, stream-oriented communication.
- Bind the socket: Bind the socket to a specific IP address and port number using the bind() system call.
- Listen for incoming connections: Use the listen() system call to put the socket into a listening state, allowing it to accept incoming connections.
- Accept incoming connections: Use the accept() system call to accept incoming connections from clients. This will create a new socket for each client connection.

What is the the HTTP protocol

Here's the steps to create a simple HTTP server using sockets:

- Handle the request: Read the request from the client socket and generate a response based on the request.
- Send the response: Send the response back to the client using the send() or write() system call.
- Close the client socket: After the response has been sent, close the client socket using the close() system call.
- Go back to step 4: Continue accepting incoming connections and handling requests until the server is shut down.

These are the general steps, but there are many details to consider when implementing a real HTTP server. For example, you'll need to parse the HTTP requests to extract the requested resource, handle errors and timeouts, support multiple clients simultaneously, and more.

Holberton
DEFINE YOUR FUTURE

Any Quetions