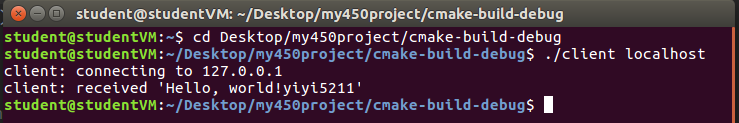
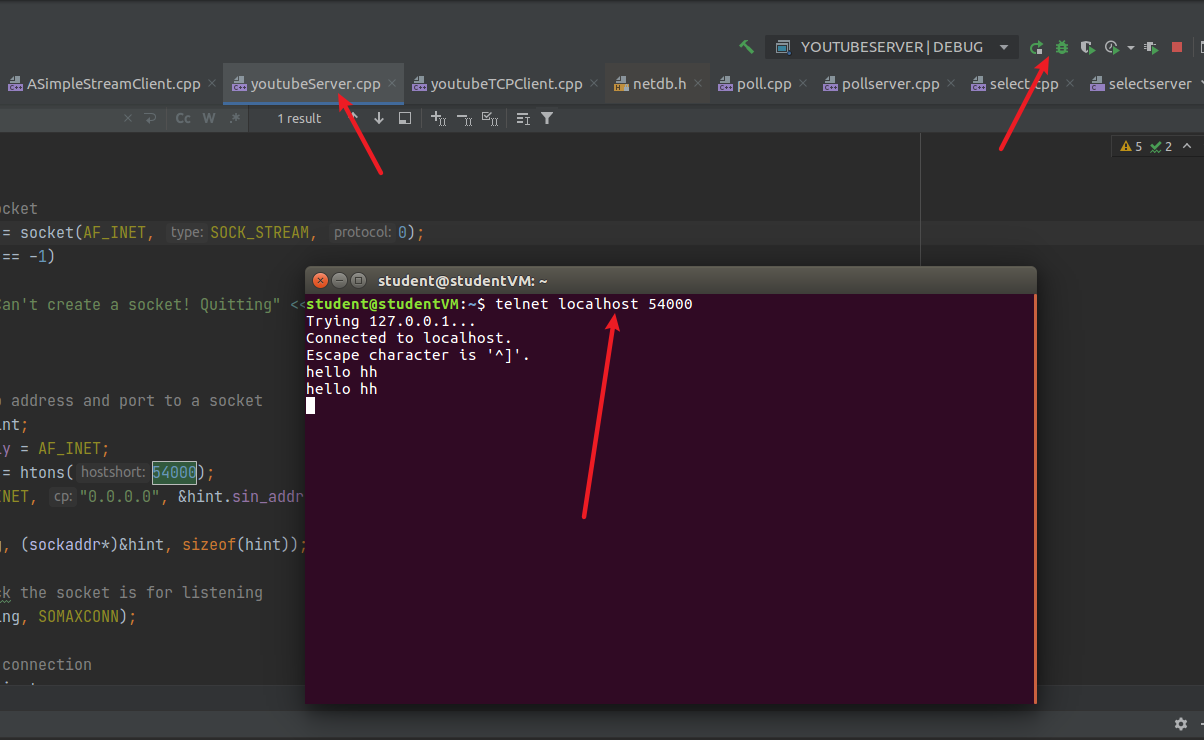
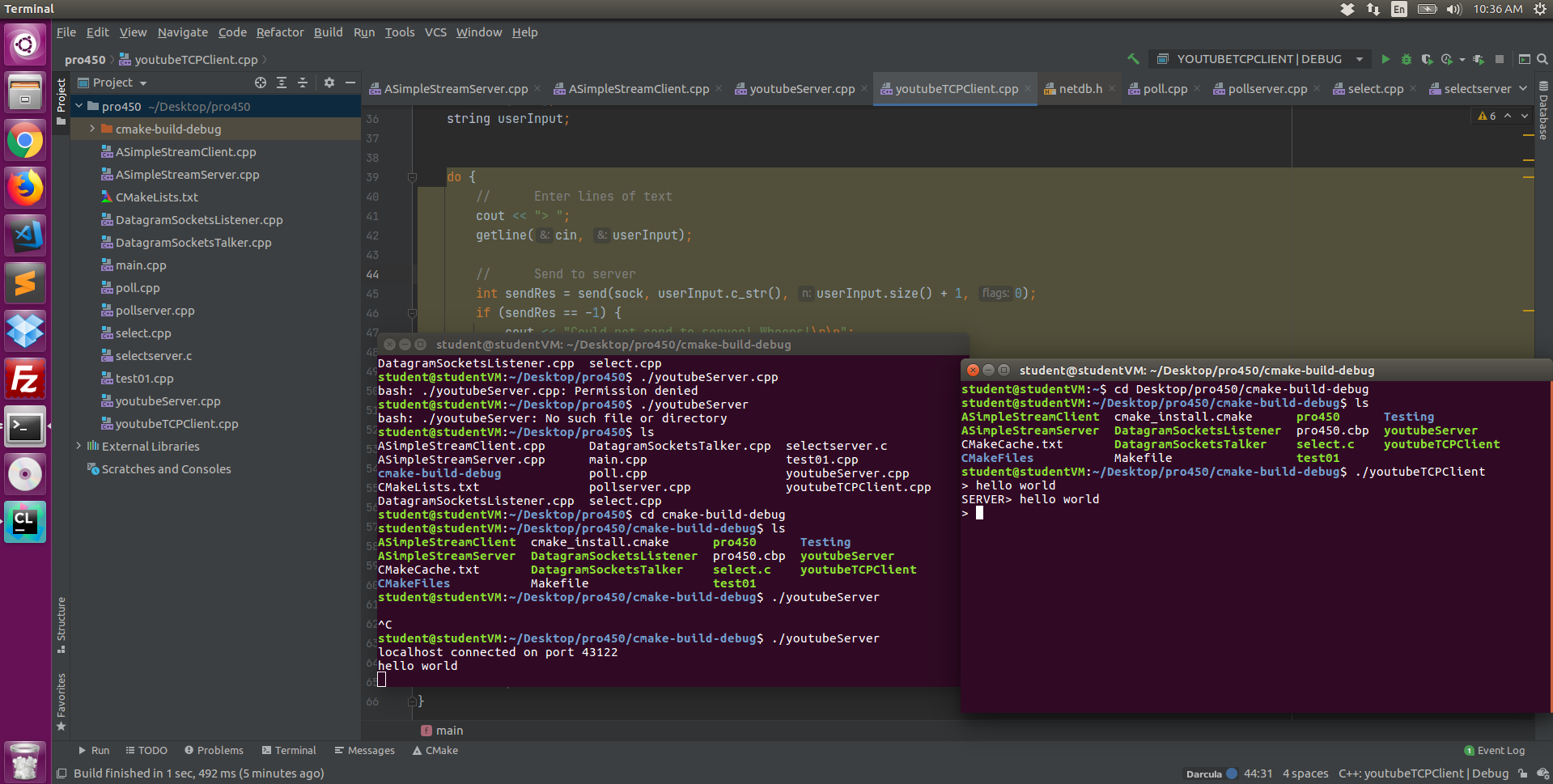
cd Desktop/my450project/cmake-build-debug

./client localhost



<https://code.visualstudio.com/docs/editor/debugging#_launch-configurations>



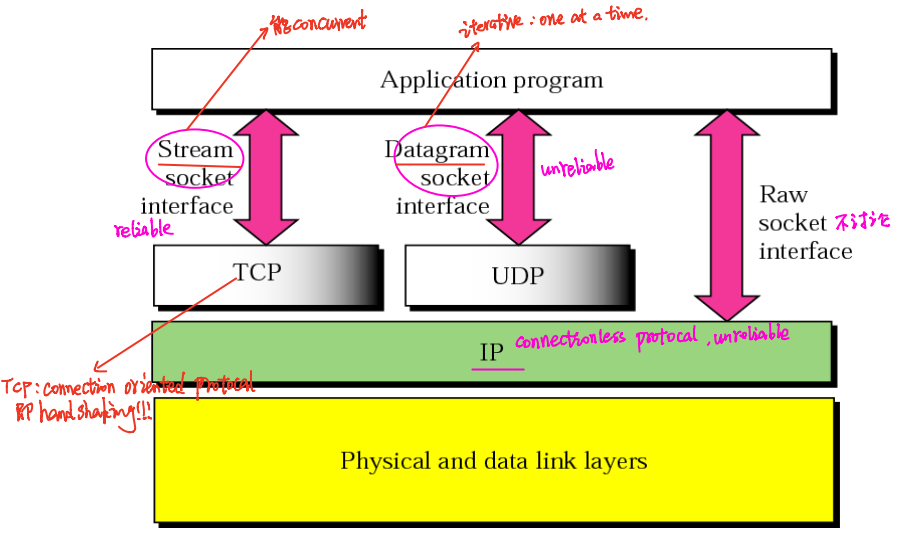


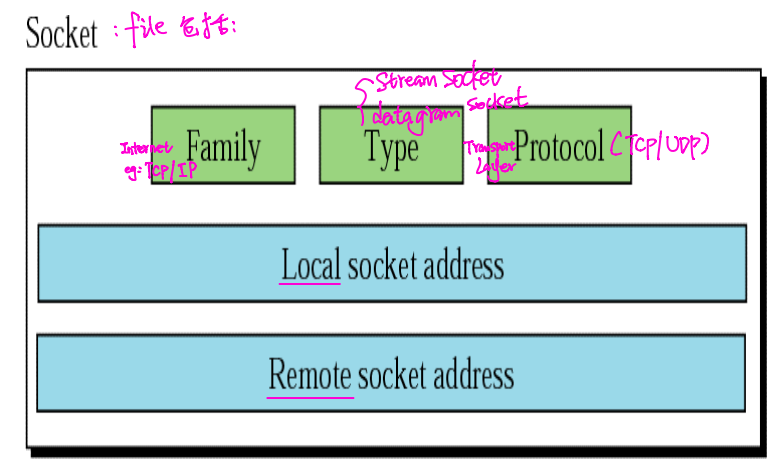
|  |
| --- |
| {  // See https://go.microsoft.com/fwlink/?LinkId=733558  // for the documentation about the tasks.json format  "version": "2.0.0",  "tasks": [  {  "label": "build hello world",  "type": "shell",  "command": "g++",  "args": [  "-g", "hello.cpp",  "-std=c++17"  ]  }  ]  } |

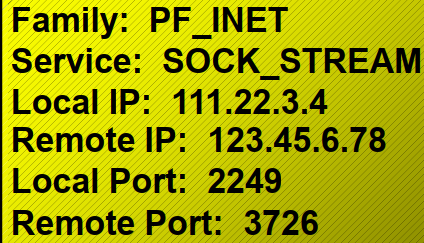
|  |
| --- |
| {  // Use IntelliSense to learn about possible attributes.  // Hover to view descriptions of existing attributes.  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387  "version": "0.2.0",  "configurations": [  {  "name": "(gdb) Launch",  "type": "cppdbg",  "request": "launch",  "program": "${workspaceFolder}/a.out",  "args": [],  "stopAtEntry": false,  "cwd": "${workspaceFolder}",  "environment": [],  "externalConsole": true,  "internalConsoleOptions": "neverOpen",  "MIMode": "gdb",  "setupCommands": [  {  "description": "Enable pretty-printing for gdb",  "text": "-enable-pretty-printing",  "ignoreFailures": true  }  ],  "preLaunchTask": "build hello world"  }  ]  } |

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  int main(int argc, char const \*argv[])  {  string str;  cin >> str;  cout << "wode: "<< str << endl;  cout << "Hello, world!" << endl;  system("pause");  return 0;  } |

A socket is uniquely identified by the IP address, Port number and the underlying transport layer protocol.





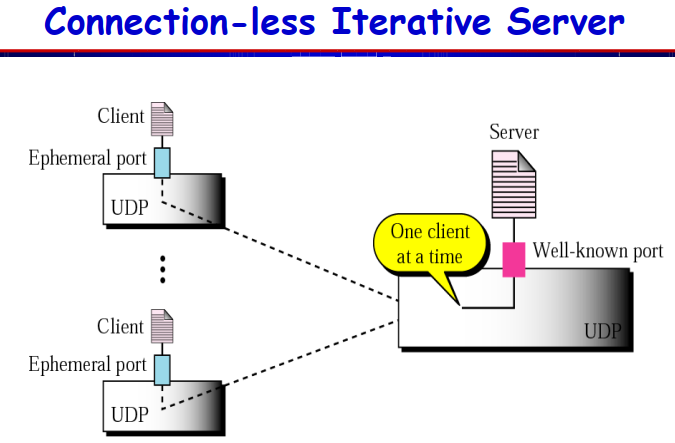
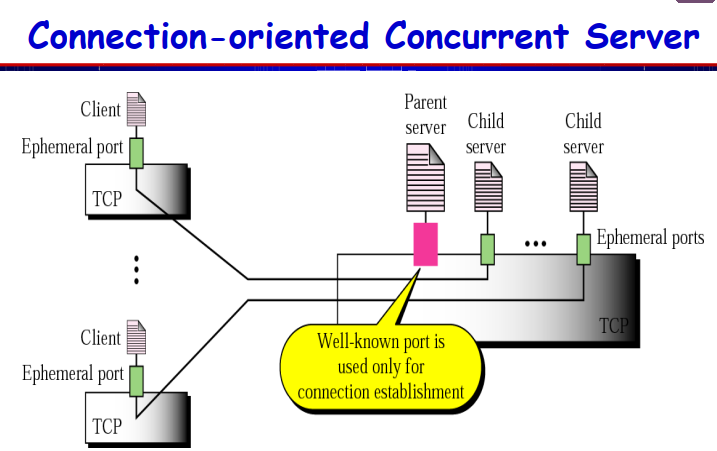


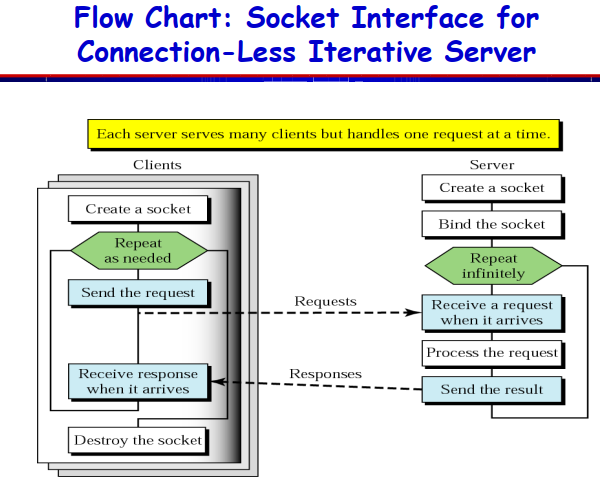
UDP socket connectionless no handshake no parent/child socket. One client at a time.

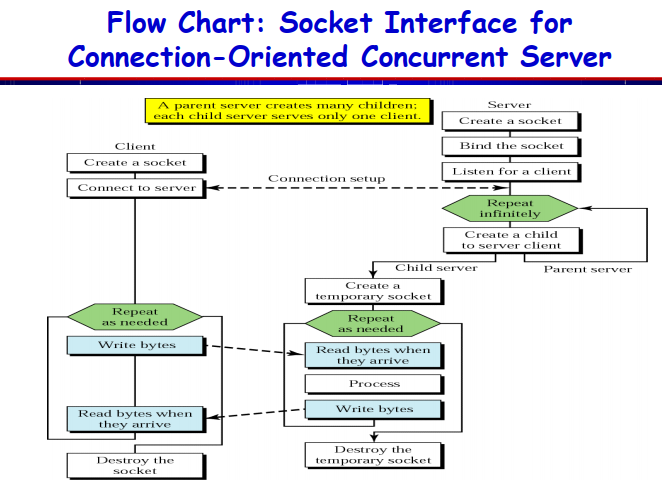
# Lecture

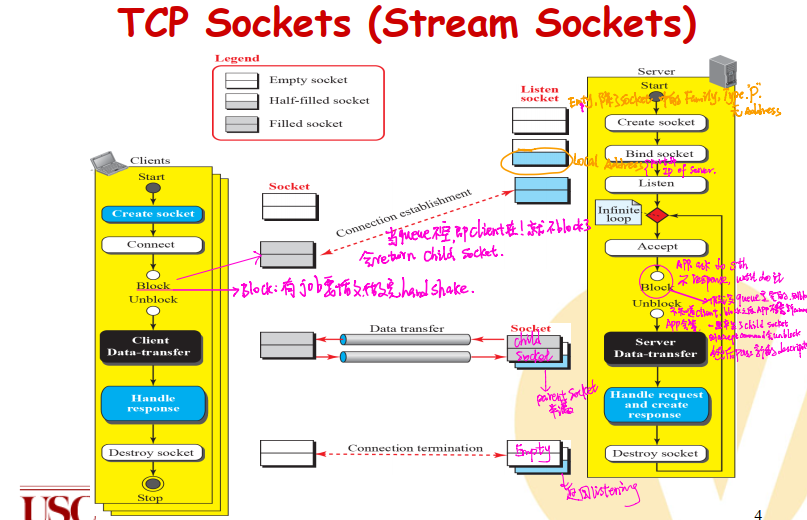
|  |  |
| --- | --- |
| Client side 🡪 actively open   1. int socket() 联系完就关了 2. int connect (int socket, foreign address, address length)   server’s port# & IP(well know)   1. int send (int socket, message, message length) 2. int recv (int socket, recv buffer, buffer length) 3. int close (int socket) | Server side🡪 passively open   1. int socket() 一直开着 2. int bind(): socket 建成之后绑定port# 3. int listen (int socket, int queue limit) 4. int accept (int socket, client address, address length), Server gets a socket(孩子) for an incoming client connection by calling accept().   对于每一个client,建立了child socket, serve multiple clients concurrently. 然后就可发送接收   1. int send (int socket, message, message length) 2. int recv (int socket, recv buffer, buffer length) 3. int close (int socket) |

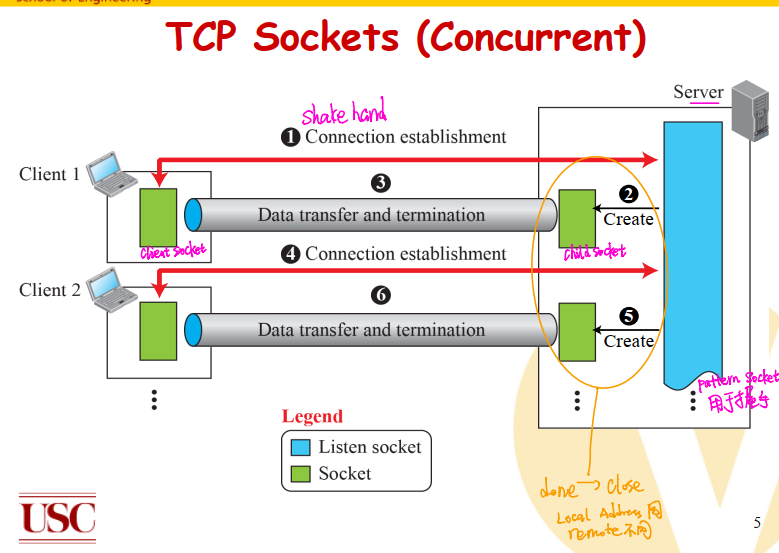
The socket that has been bounded to a port and marked for listening is never actually used for sending and receiving. The socket (known as the welcoming socket or the “parent” socket. “Child” sockets are created for each client. It is this socket that is actually used for sending and receiving. parent socket用于握手，握手后server产生child socket.

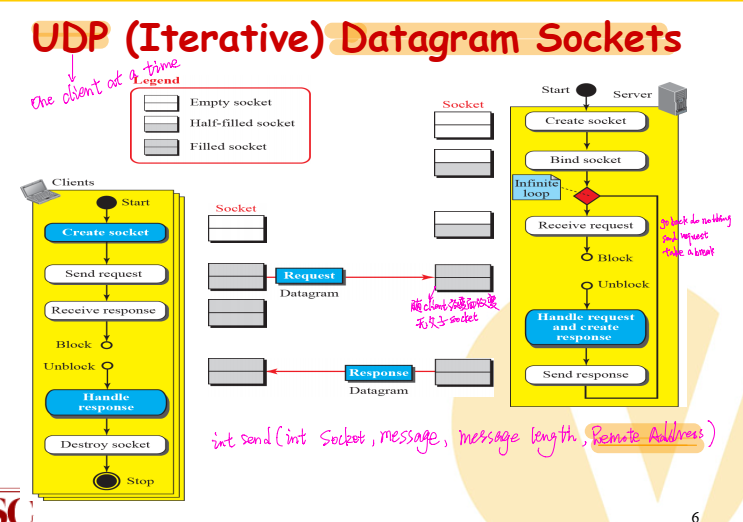
 

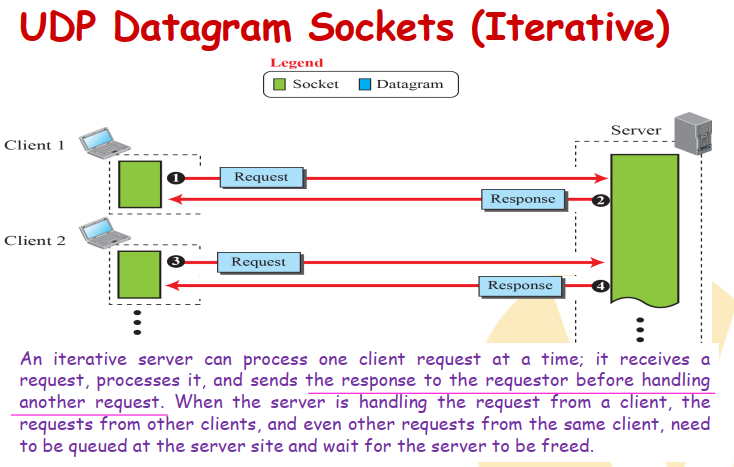












# Structs

为了访问结构的成员，我们使用**成员访问运算符（.）**

book.title

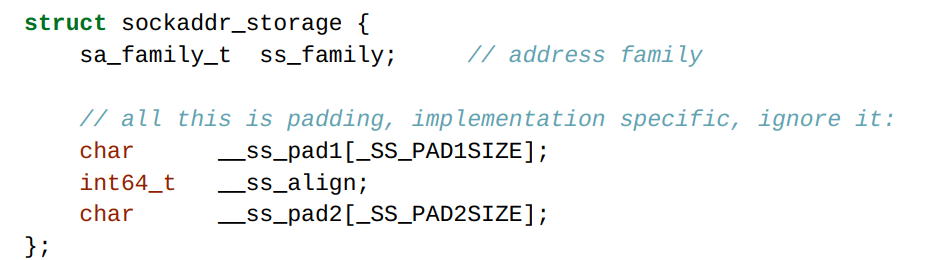
**指向结构的指针**

struct Books \*struct\_pointer;

struct\_pointer = &Book1;

为了使用指向该结构的指针访问结构的成员，您必须使用 -> 运算符struct\_pointer->title;

|  |
| --- |
| 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; // use 0 for "any"  size\_t ai\_addrlen; // size of ai\_addr in bytes  struct sockaddr \*ai\_addr; // struct sockaddr\_in or \_in6  char \*ai\_canonname; // full canonical hostname  struct addrinfo \*ai\_next; // linked list, next node  };  call getaddrinfo(). It’ll return a pointer to a new linked list of these structures filled out with all the goodies you need    **sockaddr** holds socket address information for many types of sockets  struct **sockaddr** {  unsigned short sa\_family; // address family, AF\_xxx  char sa\_data[14]; // 14 bytes of protocol address  };  struct **sockaddr\_in** {  short int sin\_family; // Address family, AF\_INET  unsigned short int sin\_port; // Port number  struct in\_addr sin\_addr; // Internet address  unsigned char sin\_zero[8]; // Same size as struct sockaddr  };  AF\_INET or AF\_INET6 (for IPv4 or IPv6) |

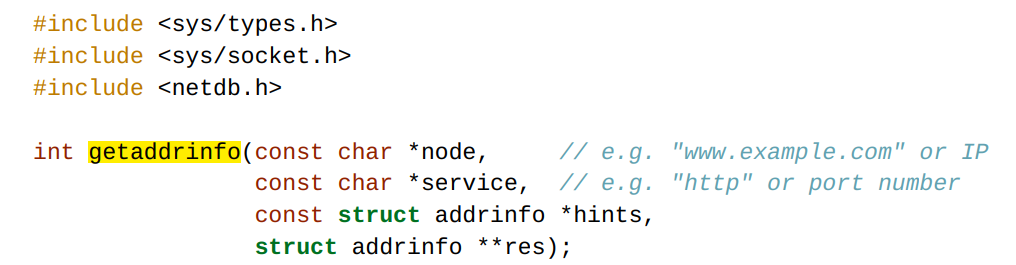


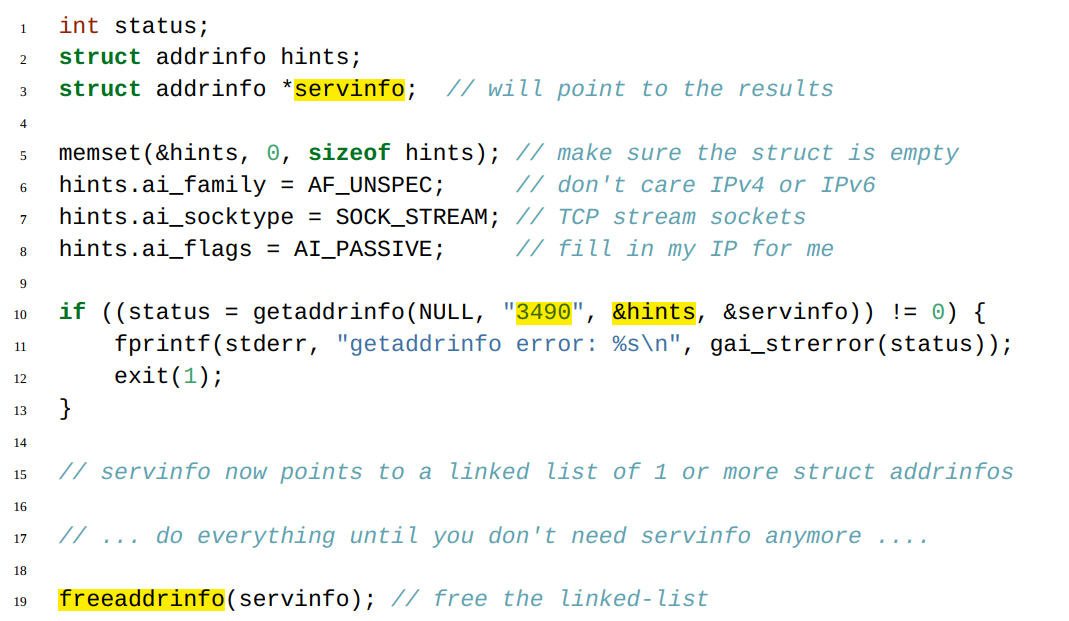
## IP Addresses

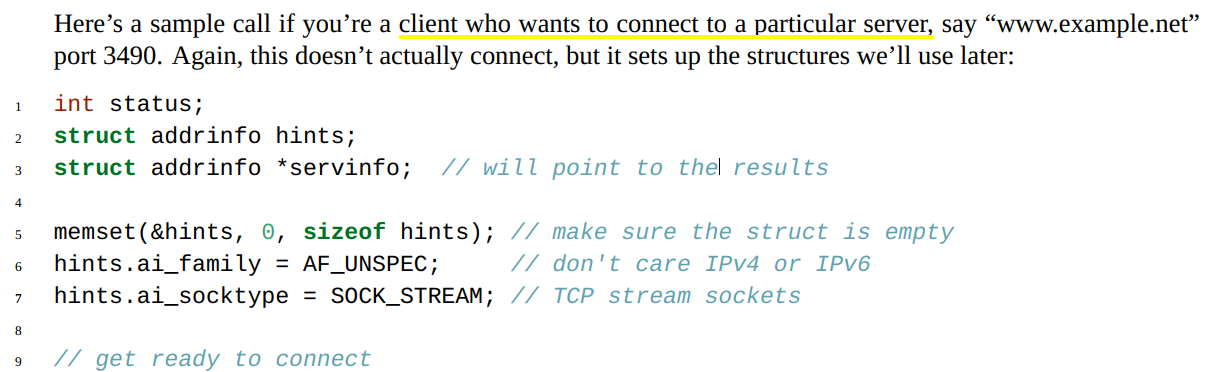
**struct** sockaddr\_in sa; *// IPv4*

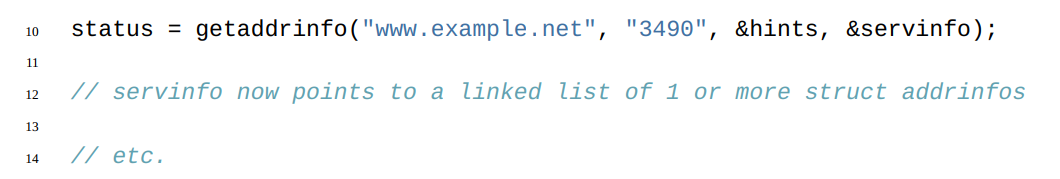
inet\_pton(AF\_INET, "10.12.110.57", &(sa.sin\_addr)); *// IPv4*

## getaddrinfo()—Prepare to launch

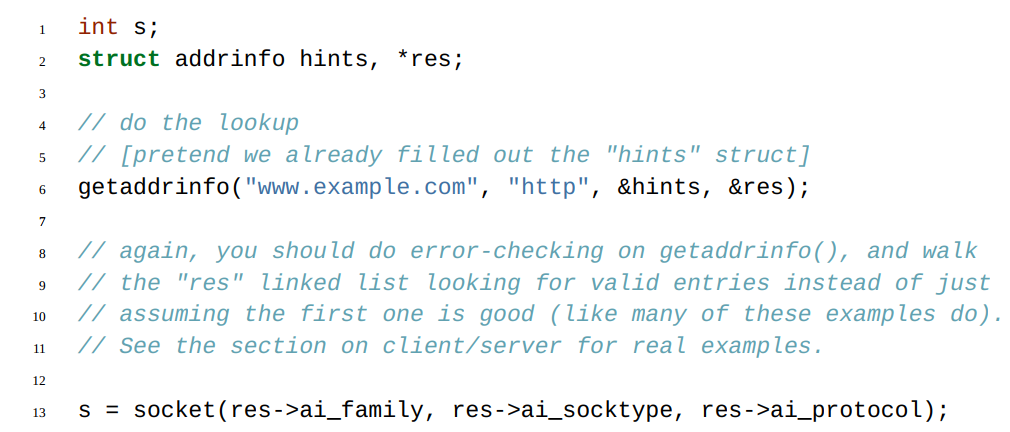




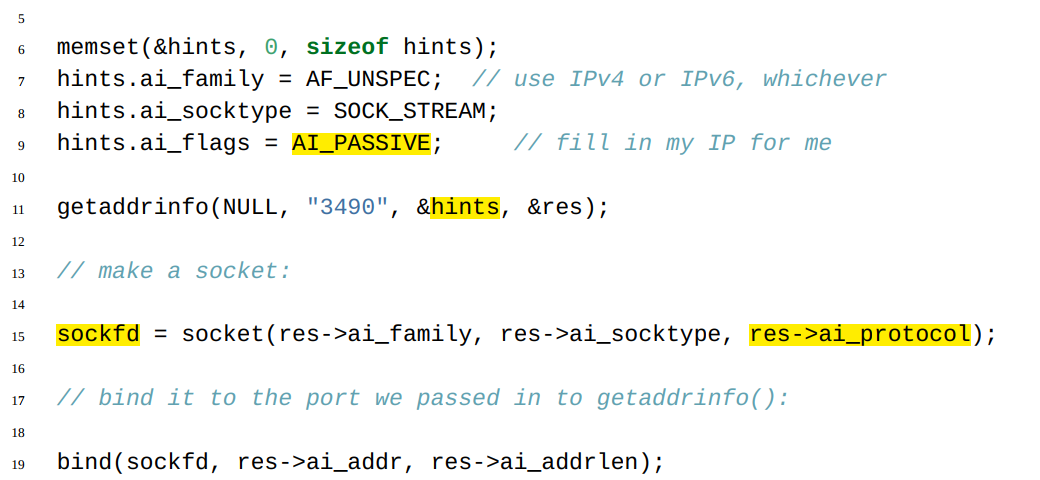
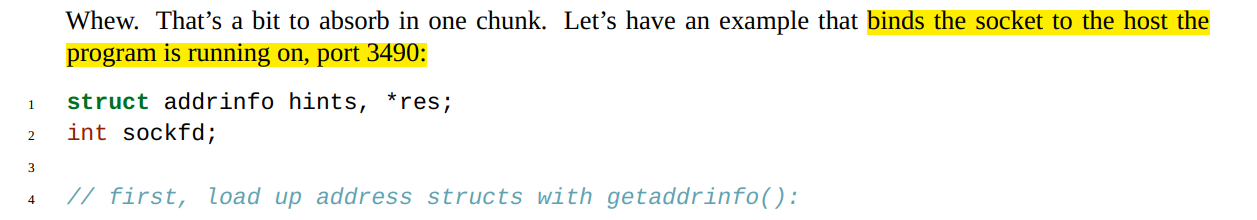




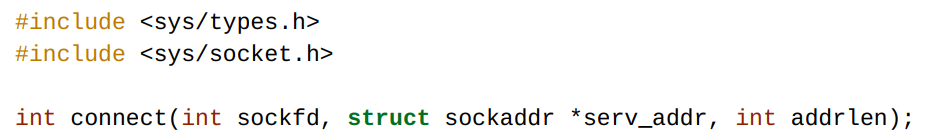
## socket()—Get the File Descriptor!

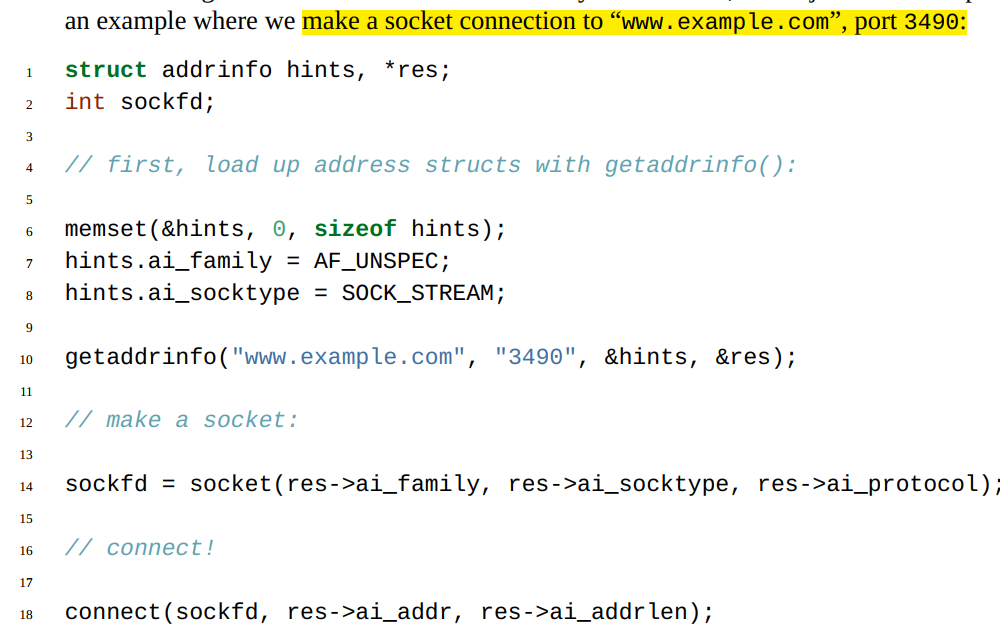


## bind()—What port am I on?



## connect()—Hey, you!

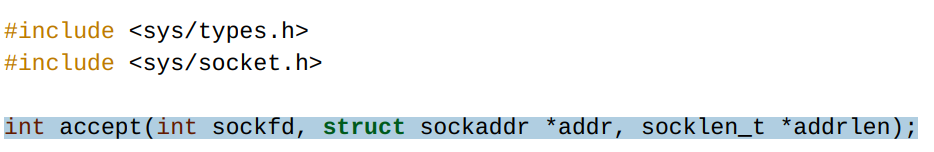


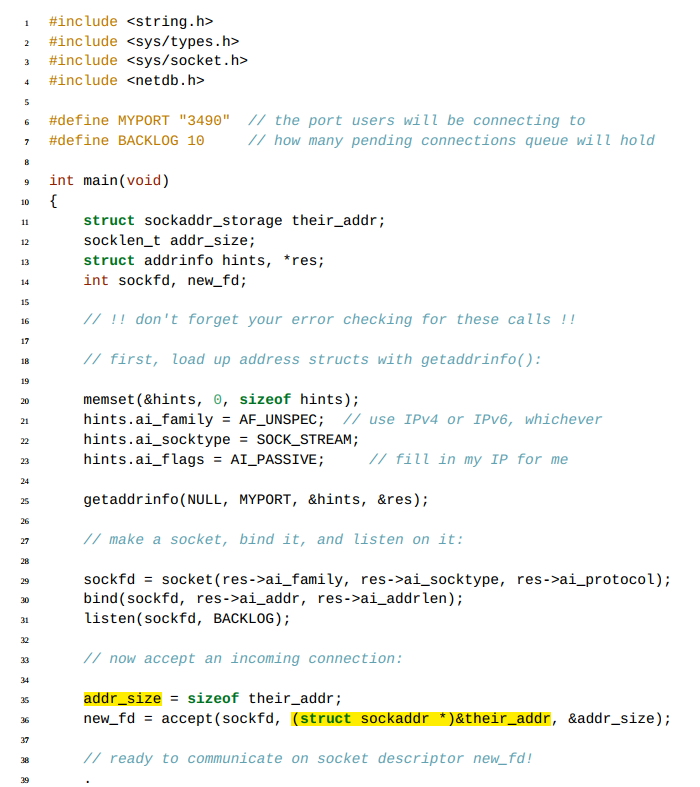


## listen()—Will somebody please call me?

int listen(int sockfd, int backlog);

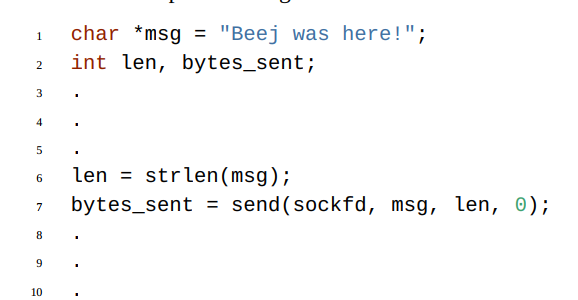
## accept()—“Thank you for calling port 3490.”

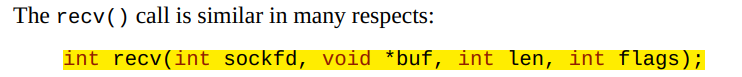




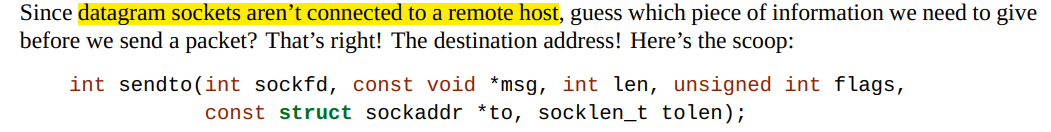
## send() and recv()—Talk to me, baby!

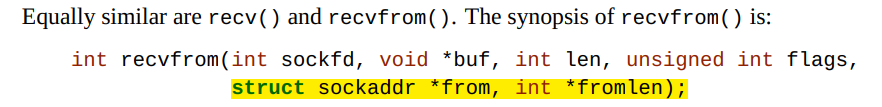






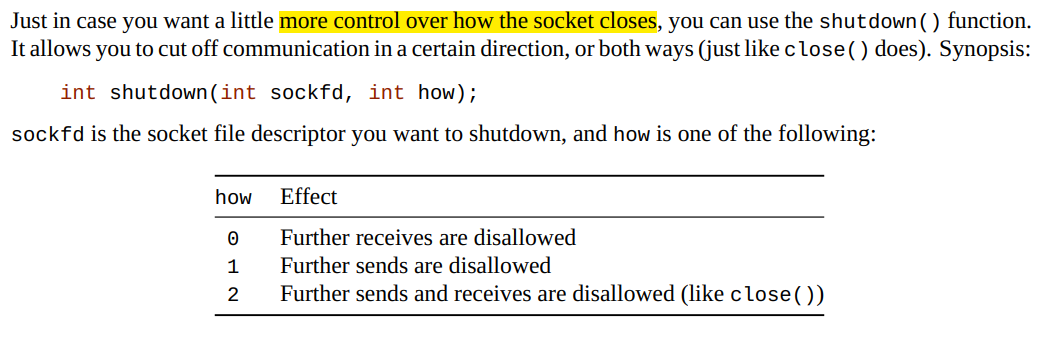
## sendto() and recvfrom()—Talk to me, DGRAM-style



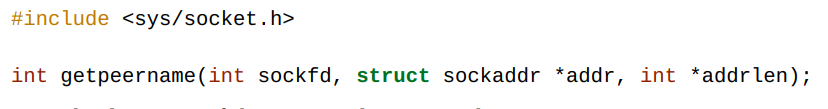


## close() and shutdown()—Get outta my face!

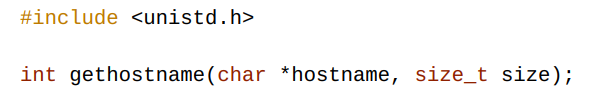
close(sockfd);



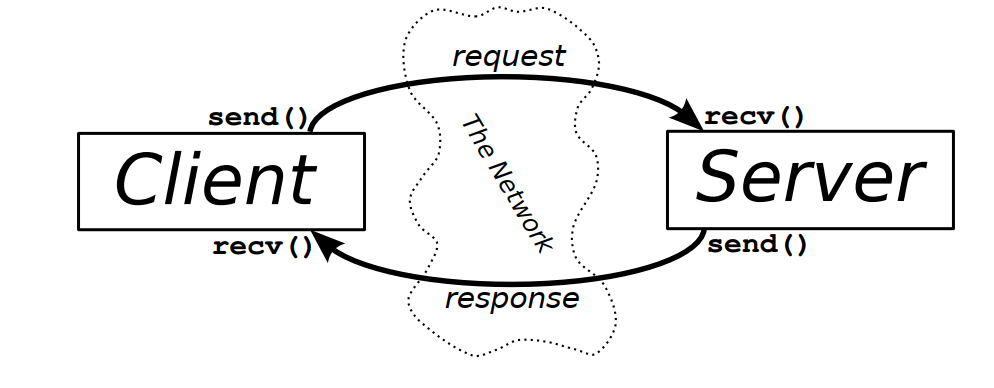
## getpeername()—Who are you?



## gethostname()—Who am I?

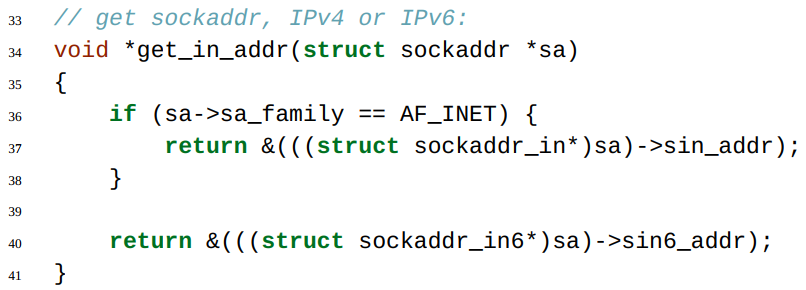
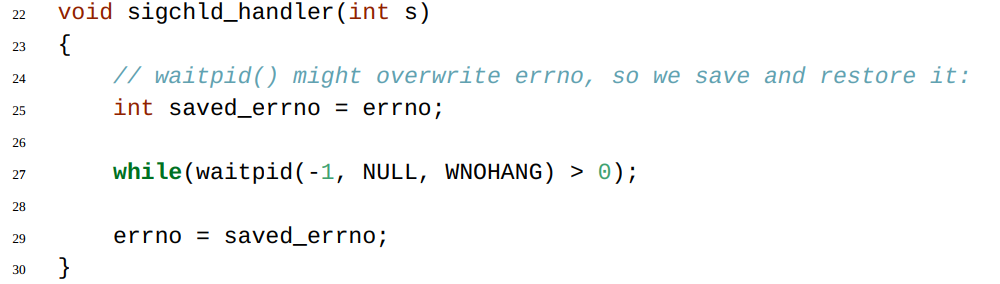


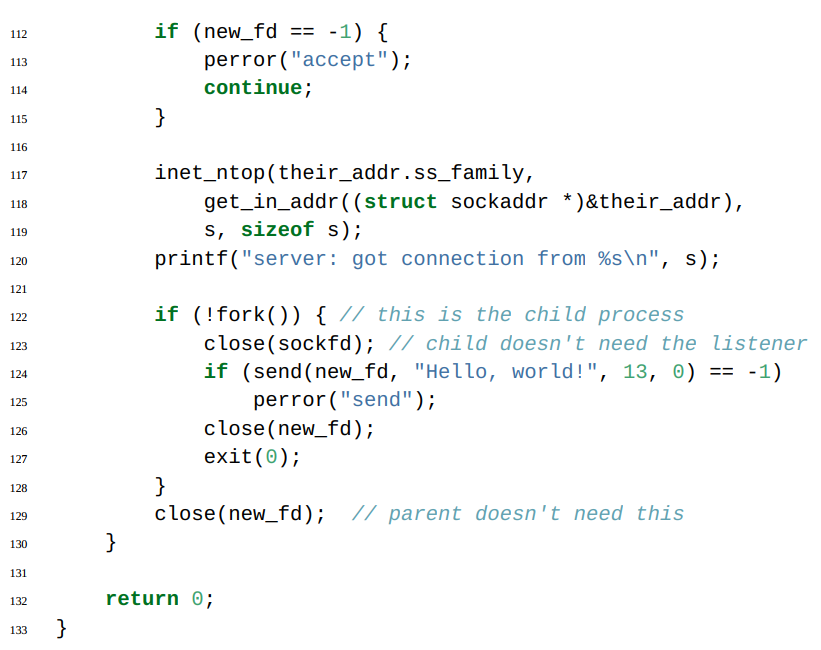
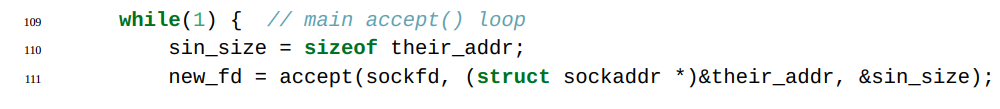
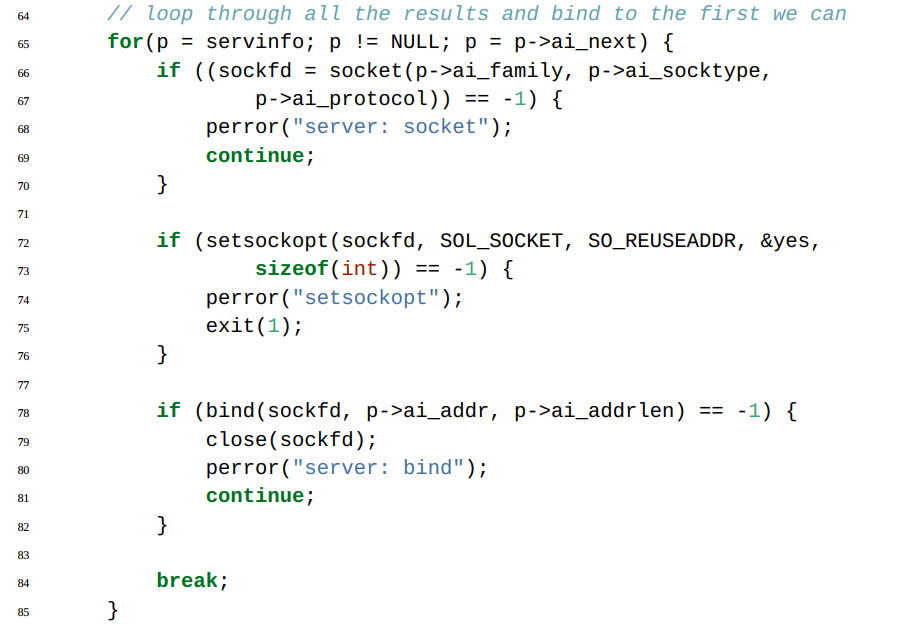
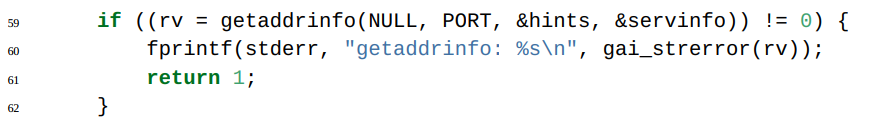
# Client-Server Background



server will wait for a connection, accept() it, and fork() a child process to handle it.

## A Simple Stream Server





# Slightly Advanced Techniques

## Blocking

If you don’t want a socket to be blocking, you have to make a call to fcntl():

fcntl(sockfd, F\_SETFL, O\_NONBLOCK);

## poll()—Synchronous I/O Multiplexing

