Network Programming

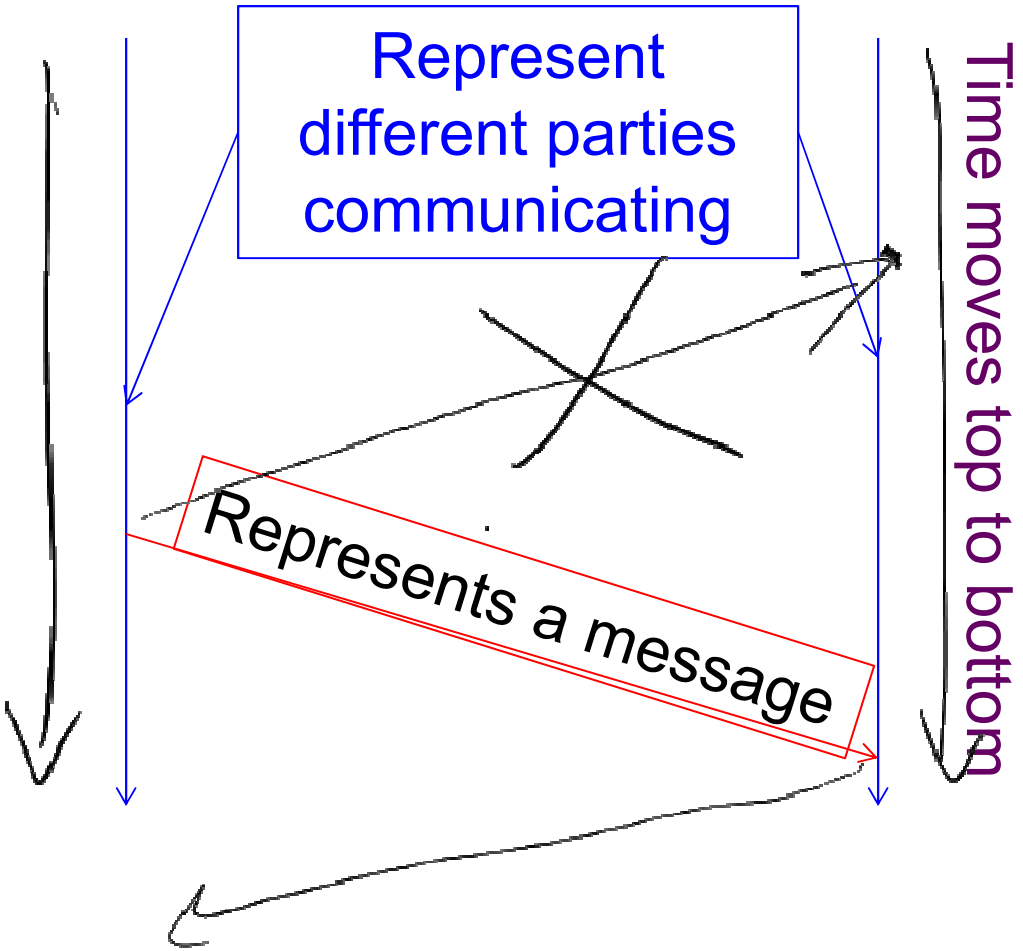
- Learning Objectives
  - Use sockets
  - Build server-side applications

Client/Server Paradigm

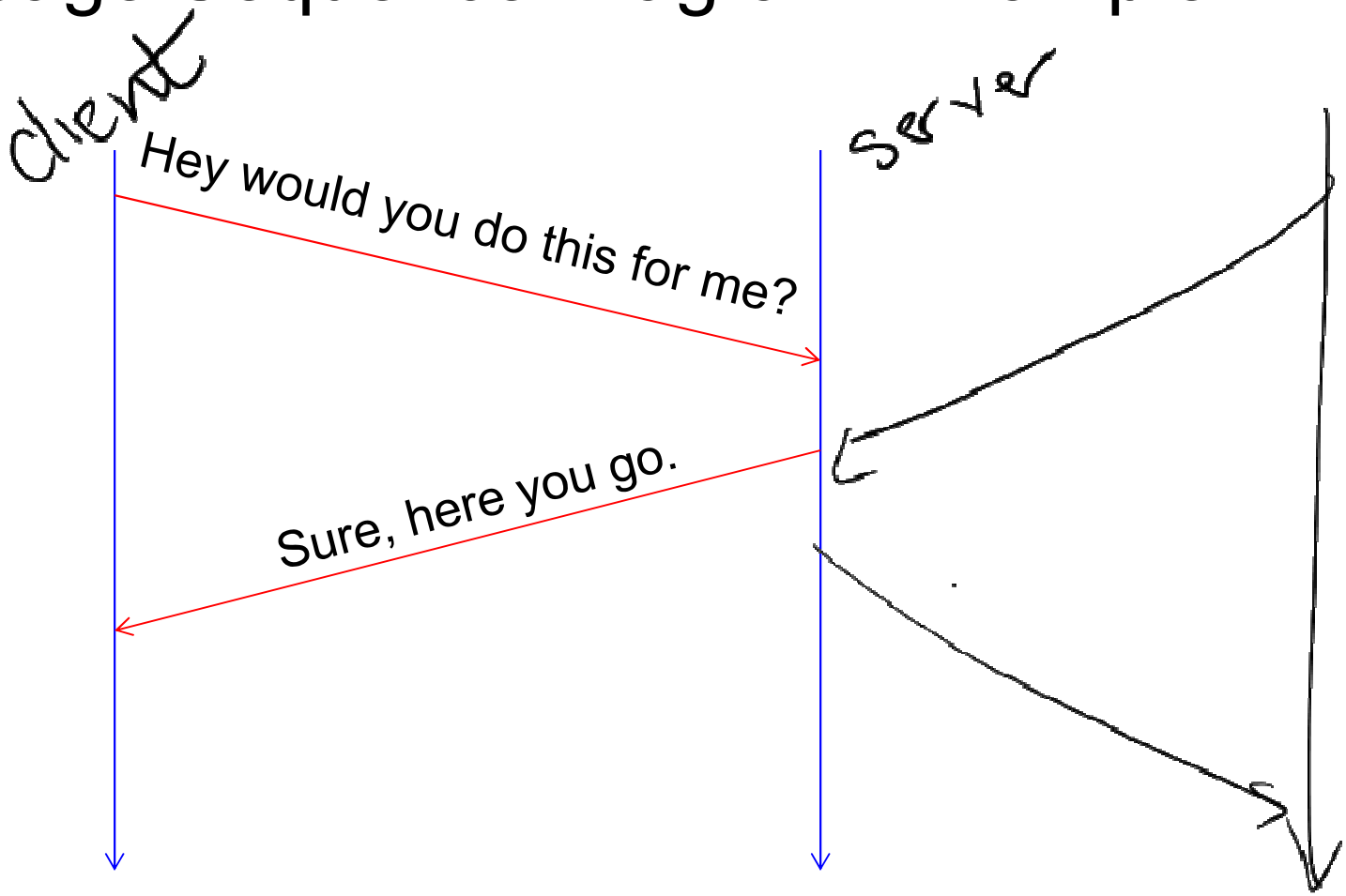
- Many networked applications use a client/server architecture.
- Server
  - A passive process that takes action only when asked to do something on behalf of a client.
- Client
  - Active party that generates requests, sends them to a server and does something with the result that it gets back.
- Protocol
  - The language or set of rules that clients and servers user to interact.


Select font size T T T
Given the definition of a server from the previous slide, which is a server most like?
Allow Single Choice Only
Allow Multiple Choices
Shuffle Answers
Allow Retry
Limit Attempts
A location in memory
A register
A thread
The operating syswtem
Preview
Terms
Privacy & cookies

Message Sequence Diagrams



Message Sequence Diagram: Example



An Abstraction for Communication

- What abstraction did we use to communicate between multiple processes?

An Abstraction for Communication

- What abstraction did we use to communicate between multiple processes?
  - A Pipe!
- Would a pipe work across machines?

An Abstraction for Communication

- What abstraction did we use to communicate between multiple processes?
  - A Pipe!
- Would a pipe work across machines?
  - No?
- Why not?

An Abstraction for Communication

- What abstraction did we use to communicate between multiple processes?
  - A Pipe!
- Would a pipe work across machines?
  - No?
- Why not?
  - No parent process to connect them.
- What if we could create a channel on which we could read/write as we did with a pipe, but that didn't require a parent to set it up?

Socket: An Endpoint for Communication

A special type of file descriptor (fd).

Client Setup

1. Look up the address of a server.
2. Create a socket.
3. Connect to the server.

Server Setup

1. Create a socket.
2. Set socket options.
3. Bind socket to a port.
4. Listen for new connections.
5. Accept new connections

Client Code

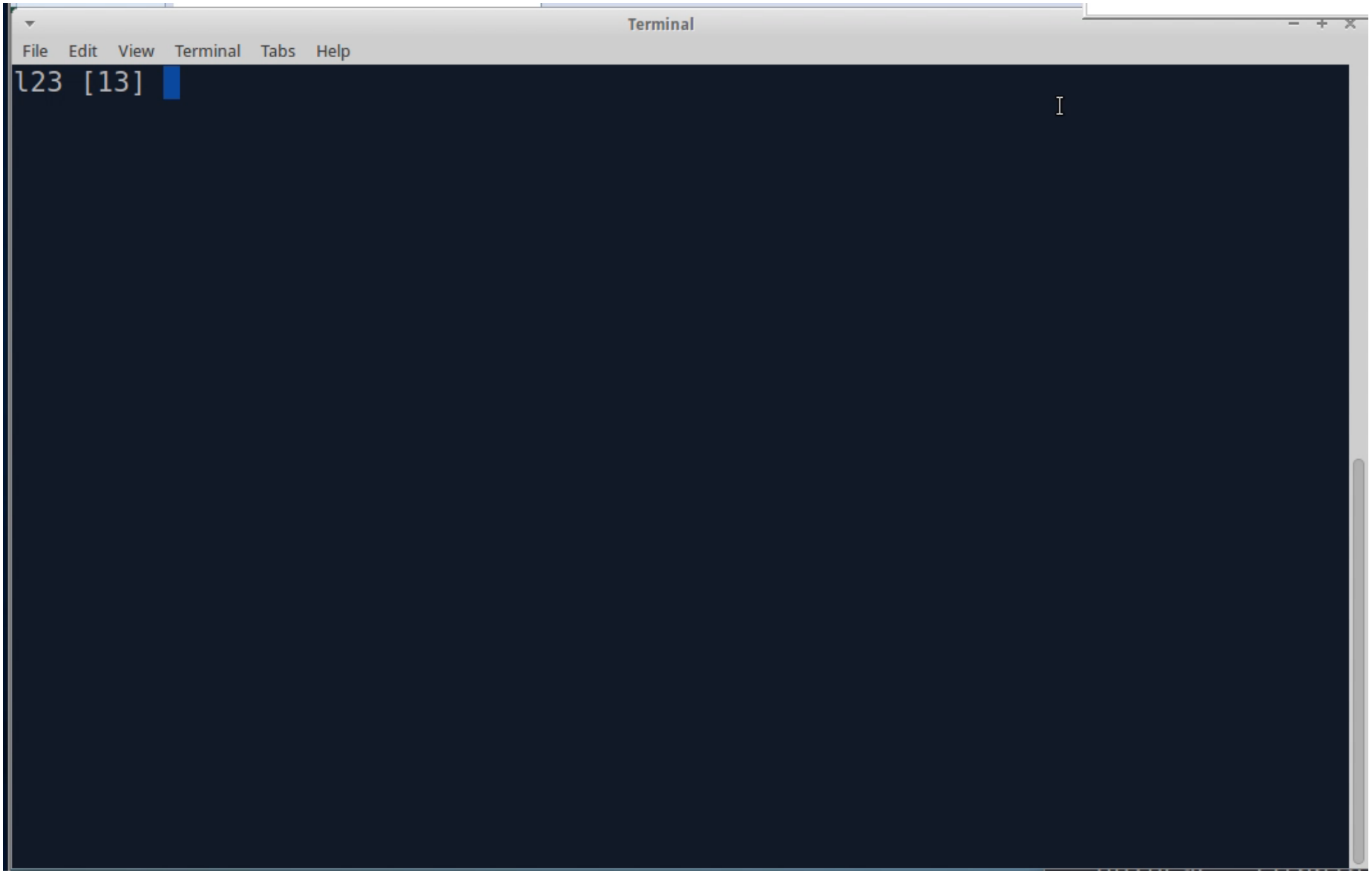
1. Look up the address of a server.

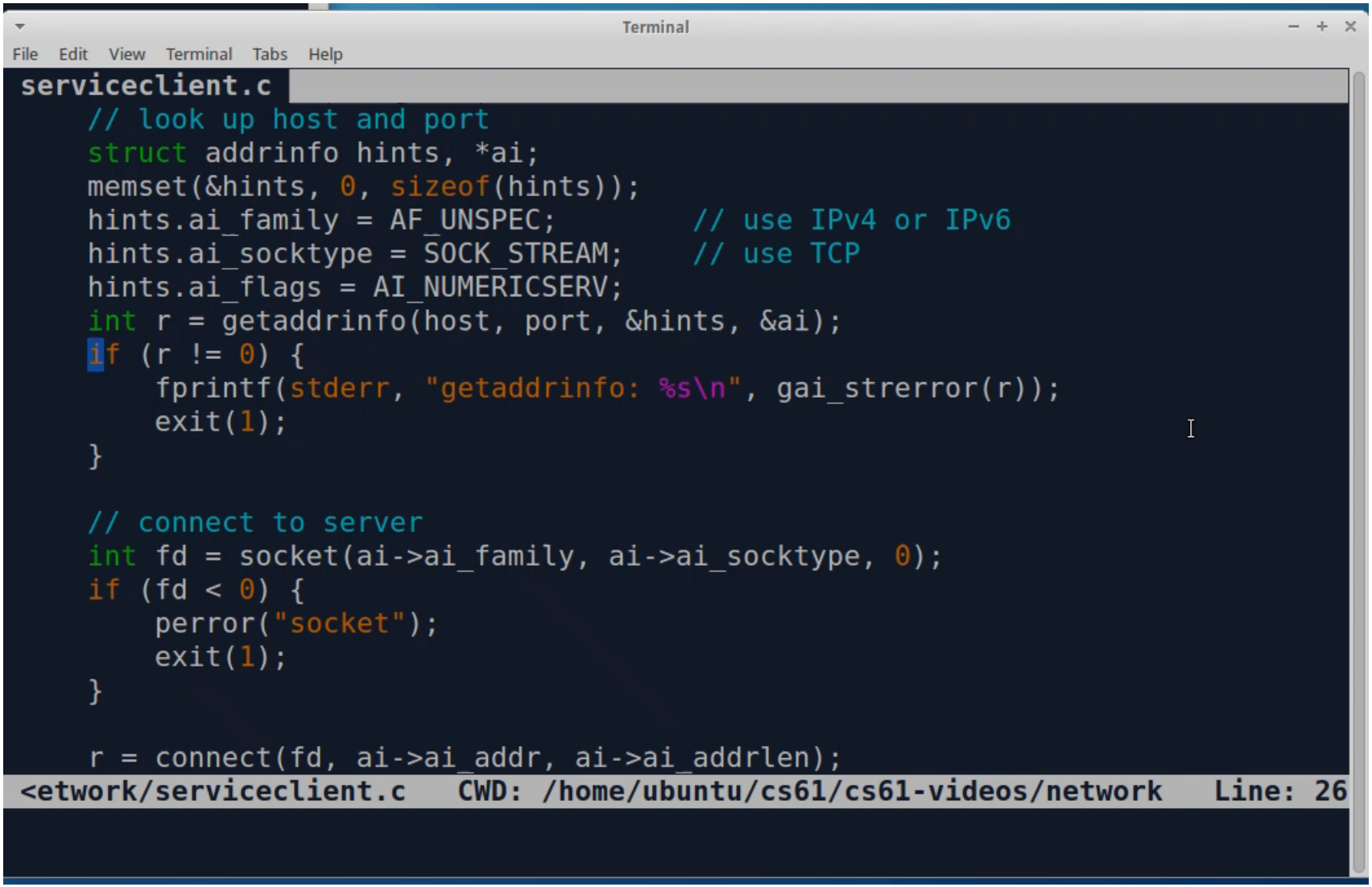
   `getaddrinfo(node, service, hints, res)`

   - Given `node` and `service`, return `addrinfo` structures that can be used in `connect`.
2. Create a socket.

   `socket(domain, type, protocol)`
3. Connect to the server.

   `connect(socket, address, address_length)`


Terminal
File Edit View Terminal Tabs Help
l23 [13]


Terminal
File Edit View Terminal Tabs Help
serviceclient.c
// look up host and port
struct addrinfo hints, *ai;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC; // use IPv4 or IPv6
hints.ai_socktype = SOCK_STREAM; // use TCP
hints.ai_flags = AI_NUMERICSERV;
int r = getaddrinfo(host, port, &hints, &ai);
if (r != 0) {
fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(r));
exit(1);
}
// connect to server
int fd = socket(ai->ai_family, ai->ai_socktype, 0);
if (fd < 0) {
perror("socket");
exit(1);
}
r = connect(fd, ai->ai_addr, ai->ai_addrlen);
<etwork/serviceclient.c CWD: /home/ubuntu/cs61/cs61-videos/network Line: 26

Server Code

1. Create a socket.

   You know how to do this already:

   ```
   socket(domain, type, protocol)
   ```

2. Set socket options.

   ```
   setsockopt(sockfd, level, optname, optval, optlen);
   ```

3. Bind socket to a port.

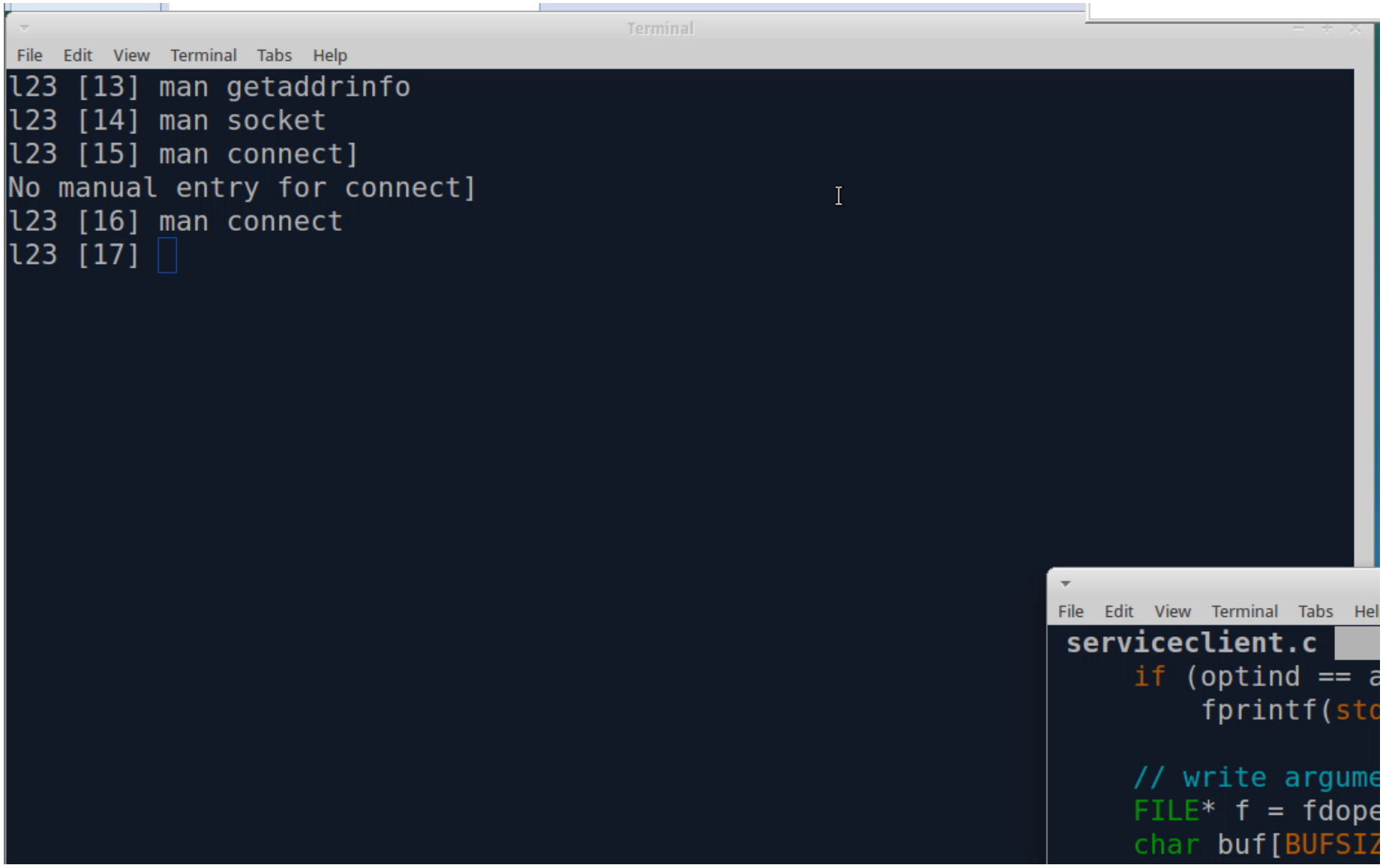
   ```
   bind(sockfd, addr, addrlen);
   ```

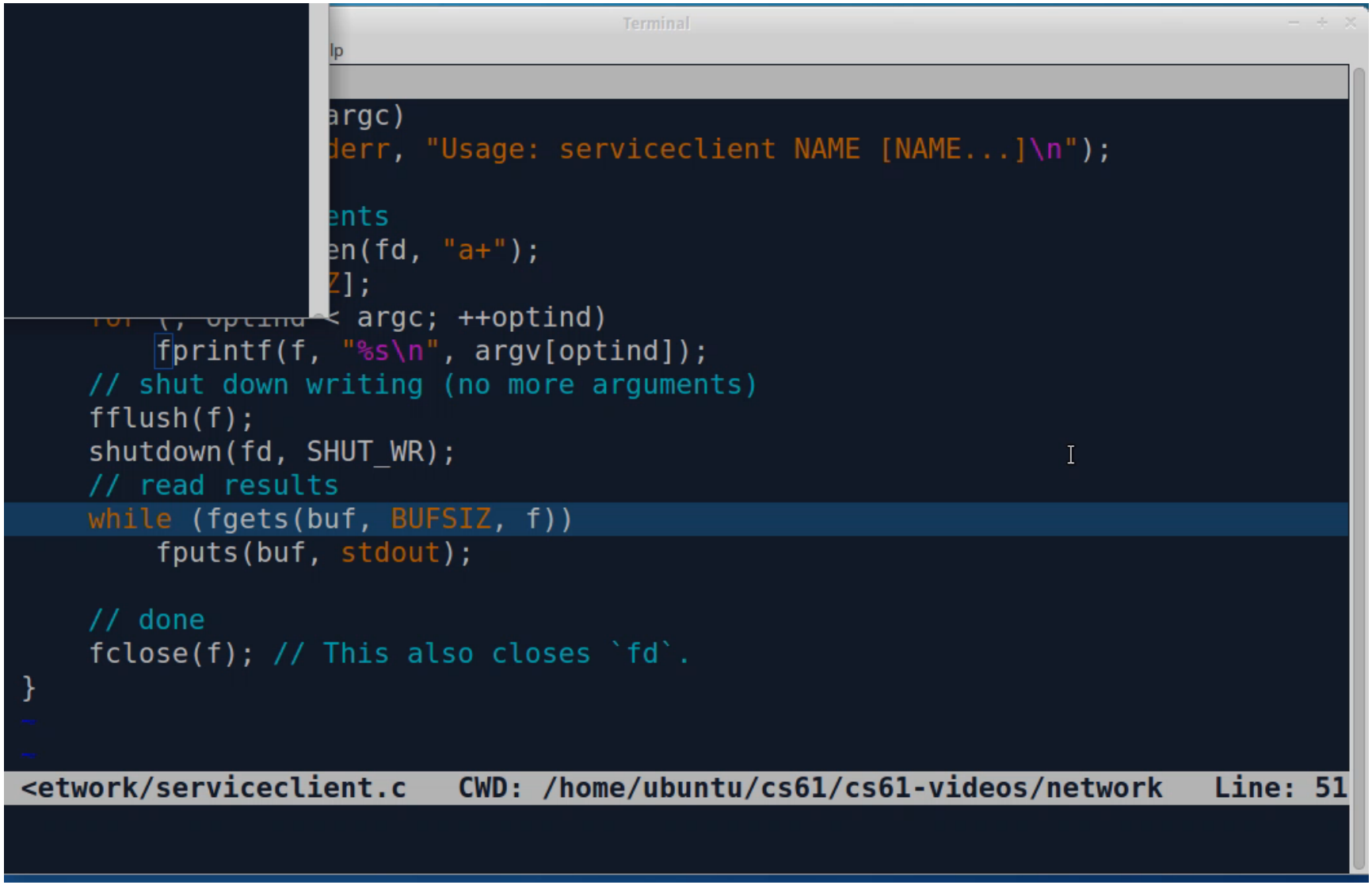
4. Listen for new connections.

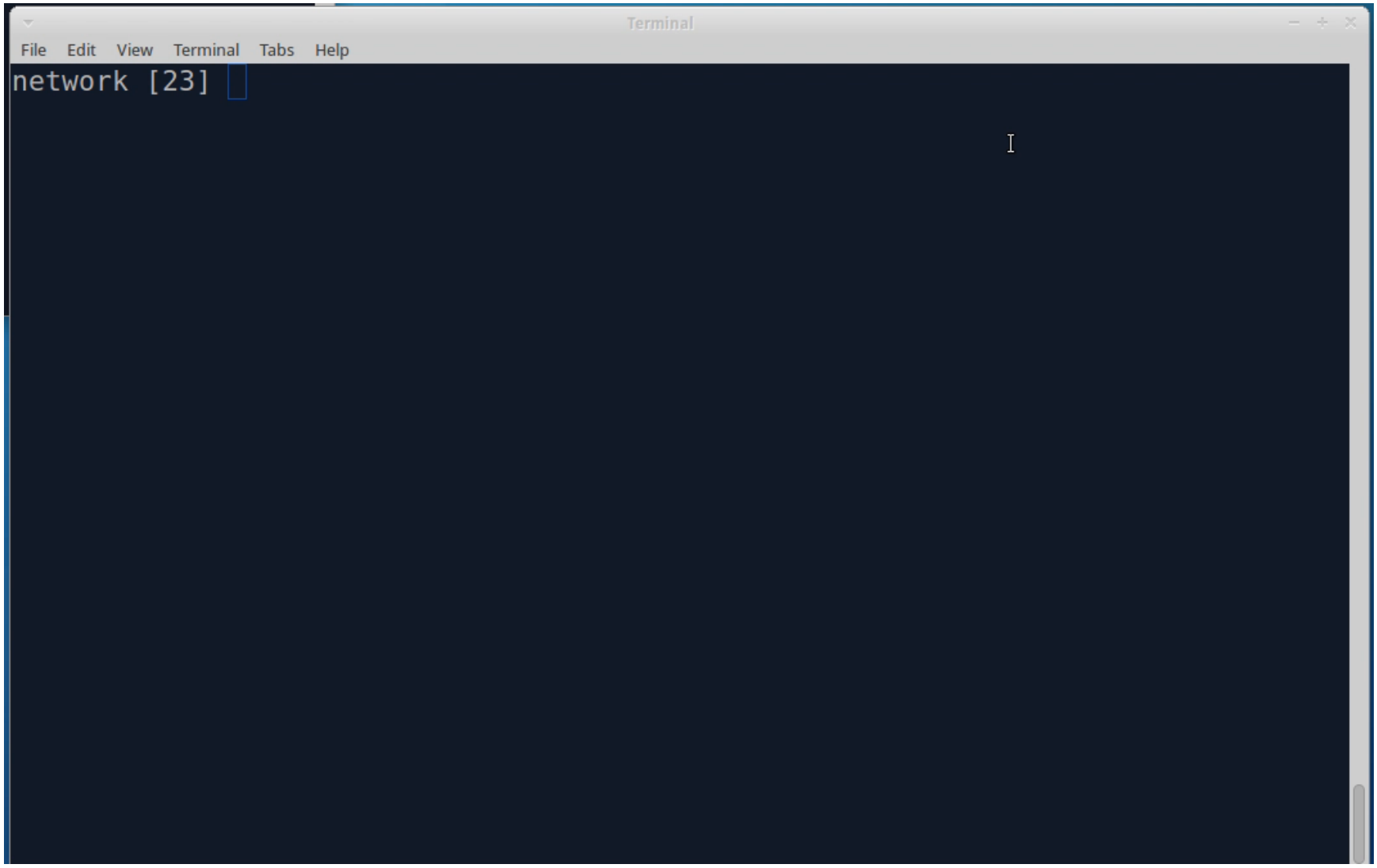
   ```
   listen(sockfd, backlog);
   ```

5. Accept an incoming message

   ```
   accept(sockfd, addr, addrlen);
   ```


Terminal
File Edit View Terminal Tabs Help
l23 [13] man getaddrinfo
l23 [14] man socket
l23 [15] man connect]
No manual entry for connect]
l23 [16] man connect
l23 [17]
File Edit View Terminal Tabs Hel
serviceclient.c
if (optind == a
fprintf(std
// write argume
FILE* f = fdope
char buf[BUFSIZ


Terminal
argc)
derr, "Usage: serviceclient NAME [NAME...]\n");
ents
en(fd, "a+");
Z];
< argc; ++optind)
fprintf(f, "%s\n", argv[optind]);
// shut down writing (no more arguments)
fflush(f);
shutdown(fd, SHUT_WR);
// read results
while (fgets(buf, BUFSIZ, f))
fputs(buf, stdout);
// done
fclose(f); // This also closes `fd`.
}
<etwork/serviceclient.c CWD: /home/ubuntu/cs61/cs61-videos/network Line: 51


Terminal
File Edit View Terminal Tabs Help
network [23]

Wrapping Up

- In a client/server paradigm:
  - The server is a passive entity that responds to requests from clients.
- While clients and servers communicate using an `fd` as two processes would using a pipe, network programs require a bit more setup.