CS 422/522 Design & Implementation of Operating Systems

Lecture 21: Networking & Protocol Stacks

Zhong Shao
Dept. of Computer Science
Yale University

Acknowledgement: some slides are taken from previous versions of the CS422/522 lectures taught by Prof. Bryan Ford and Dr. David Wolinsky, and also from the official set of slides accompanying the OSPP textbook by Anderson and Dahlin.

1

Overview

- Introduction
  - History, Basic Concepts
- Networking Fundamentals
  - Layering, Internetworking, Addressing, Naming
- Internet Protocols
  - Link Layer Protocols, IP, Transport Protocols (TCP, UDP)
- Network Programming
  - BSD sockets

2

What is the internet?

- History:
  - 1960s: ARPAnet - Defense Advanced Research Projects Agency
    - * research project into packet switching networks
    - * wanted communications infrastructure capable of exploiting redundancy to route around damaged links
  - 1970s: ARPA needed:
    - * A common OS for researchers with ARPA funding
    - * Technology to keep geographically dispersed ARPA researchers in contact with each other

    ⇒ funding for BSD Unix project, Univ. of Calif. Berkeley
  - 1980s: BSD Unix
    - * Included support for Internet network protocols (**TCP/IP**)

3

What is the internet?

- The **Internet** is really a *network of networks* connecting millions of computing devices throughout the world.
- Each network is administered independently of all other networks
  - ***There is no central authority running the Internet.***

4

Internet model

- The Internet is a **packet-switched** network.
- All data transmission is broken into chunks (**packets**).
- Each packet contains:
  - the data to be transmitted (the payload)
  - identification of the packet' s source and destination
- The key points to understand about packet switching are:
  - Each packet is processed independently of all other packets.
  - There is no need to set up a "connection" or "circuit" with another node before sending it a packet.

5

The internet protocols

- The Internet works because all **host computers** connected to the Internet adhere to a set of **Internet Standards**. These standards specify the **protocols** used for communication across the network.
- The particular protocols used for the Internet are alternatively called the **Internet Protocols**, or just **TCP/IP.**
  - **TCP** stands for **Transmission Control Protocol**
  - **IP** stands for **Internet Protocol**
- TCP and IP are just two specific protocols, but they are so important that **TCP/IP** is used to refer to all Internet protocols.

6

Overview

- Introduction
  - History, Basic Concepts
- Networking Fundamentals
  - Layering, Internetworking, Addressing, Naming
- Internet Protocols
  - Link Layer Protocols, IP, Transport Protocols (TCP, UDP)
- Network Programming
  - BSD sockets

7

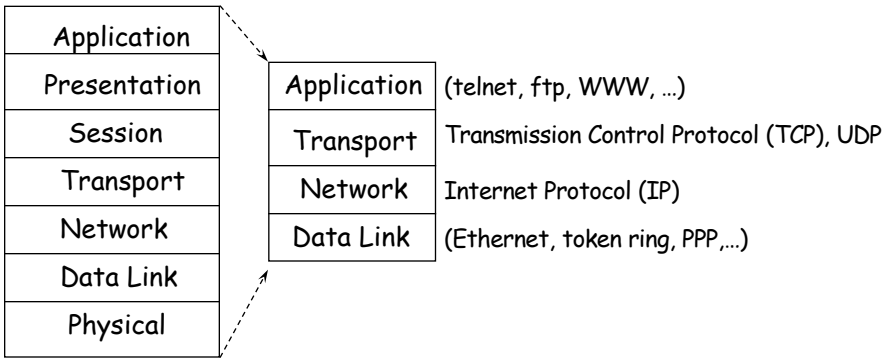
The OSI seven-layer networking model

Layer Name	Description
Application	Specific application (file transfer, remote login, etc.)
Presentation	Data formatting and conversion (e.g. byte-swapping)
Session	Long-lived "virtual connection" primitives
Transport	Reliable or unreliable end-to-end data delivery
Network	Packet routing through intermediate hosts
Data Link	Controls physical link between two endpoints
Physical	Electrical signals over physical media

8

TCP/IP and the OSI seven-layer model

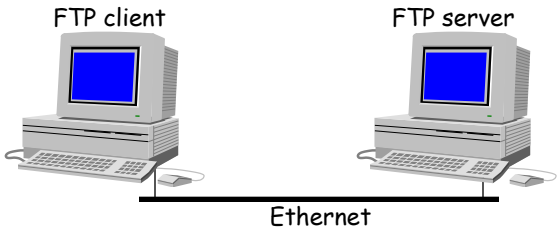
- TCP/IP based on a simplified form of OSI model:



9

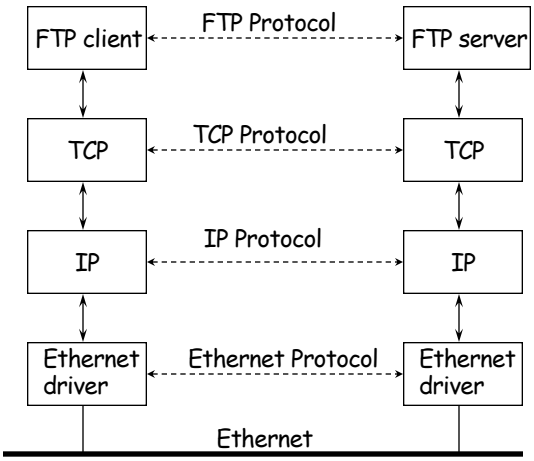
An illustrative example: file transfer

- We want to transfer a file between two hosts directly connected by a Local Area Network (LAN):



10

Logical view of connectivity



11

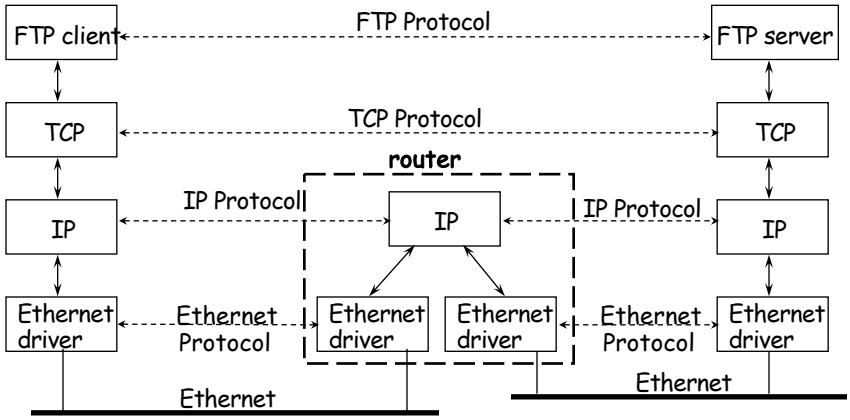
Internetworking

- Earlier, we noted that *the Internet* is not One HUGE network, but instead is really a *network of networks.*
- This raises the questions:
  - How are multiple networks interconnected?
  - What is the logical view of cross-network connectivity?

12

Routing

- Multiple networks are linked together by **routers** - special hosts with multiple network connections:



13

Routing, cont.

- Items to note from this diagram:
  - The **router** has multiple **network interfaces** (physical network connections). Hosts with multiple network interfaces are said to be **multi-homed.**
  - A router has a distinguished role as a host because it will **forward** IP packets across network boundaries.
  - All internetworking is done below the level of IP; In fact, TCP and FTP don't even know that multiple networks are involved!

14

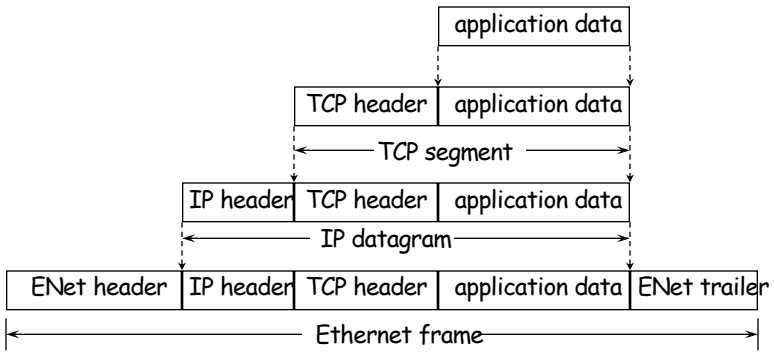
Network layering revisited

- In the previous example, FTP is "above" TCP which is "above" IP, which is "above" the Ethernet driver.
- **Q:** What does this mean from an implementation point of view?
- **A:** Each **network layer** may only interact with layers located directly above and below it in the protocol stack.
  - Each network layer provides a well-defined set of services to the layers above and below it through an **Application Programmer Interface (API).**
  - The only network API directly accessible to user programs is the **sockets API** for transport-layer access. Lower-level APIs are in the OS kernel.

15

Encapsulation and demultiplexing

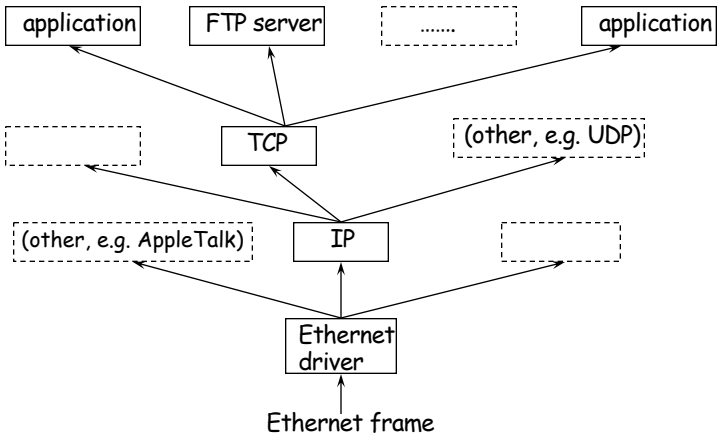
- A network layer makes use of layers beneath it through **encapsulation**:



16

Encapsulation and demultiplexing

- In the reverse direction, frames are passed from lower to higher layers by **demultiplexing:**



17

IP Addressing

- Every host machine connected to the Internet has a globally unique 32-bit **IP Address.**
- IP Addresses are usually written in **dotted-decimal** notation, e.g.:

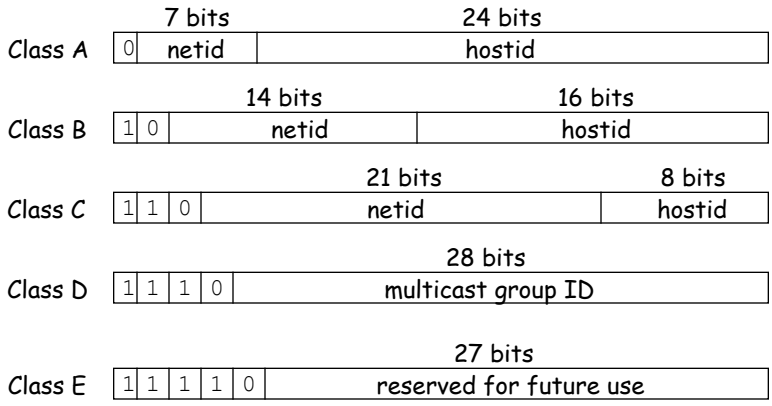
  `128.3.196.93`

- Each number is one byte of the IP address, MSB first.
- Blocks of IP addresses are given to organizations by the **InterNIC**, a central authority.

18

IP Addressing, cont.

- There are five different classes of Internet address:



19

IP Addressing, cont.

- Certain IP addresses are assigned a "special" meaning:
- **netid** = 127:
  - Always refers to *loopback* interface on local host. Try the command `telnet localhost` from any Unix system to verify this.
- **netid** = (all 1 bits), **hostid** = (all 1 bits):
  - Limited broadcast to all hosts on directly connected network.
- **netid** = (some valid network ID), **hostid** = (all 1 bits):
  - net-directed broadcast to all hosts on specified network.
- These broadcast addresses are primarily useful for locating configuration information on the local net during booting.

20

Naming of internet hosts

- IP Addresses are not particularly convenient for humans.
- Therefore, each host may be assigned a **hostname** - a friendly, string "name" identifying the host.
- For example, the hostname:

  george.lbl.gov
- maps to the IP address:

  128.3.196.93

21

The Domain Name System (DNS) - motivation

- Originally, the mapping from host names to IP addresses was administered through a single flat file: `/etc/hosts`.
  - one entry for every host on the Internet(!)
  - one "master copy" administered at a central site; periodically copied by local sys. admins.
- This approach was rife with problems:
  - **scalability** - An /etc/hosts file in 1996 would be prohibitively large.
  - **administrative autonomy** - Individual sites had to register with the NIC every time they wanted to name a machine!
- To address these problems, the **Domain Name System (DNS)** was devised.

22

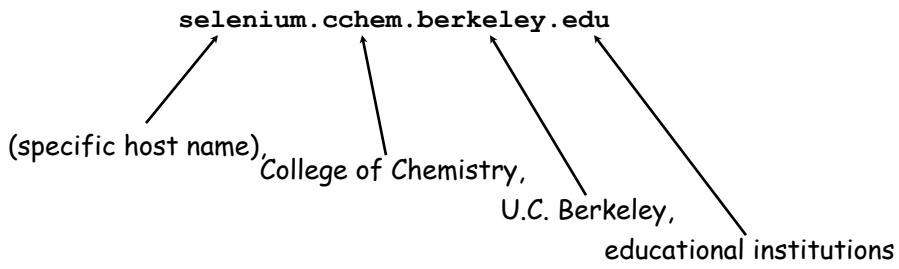
DNS - Concepts

- Each **hostname** is now called a **domain name.**
- The DNS is implemented as a *distributed database:*
  - No flat /etc/hosts file
  - Individual sites handle their own name registration
  - Individual sites provide information to other sites about domain names they are responsible for administering
- The DNS contains more than just hostname to IP address mapping:
  - MX records - information about how/where to deliver email
  - PTR records - map IP addresses back to host names

23

DNS, cont.

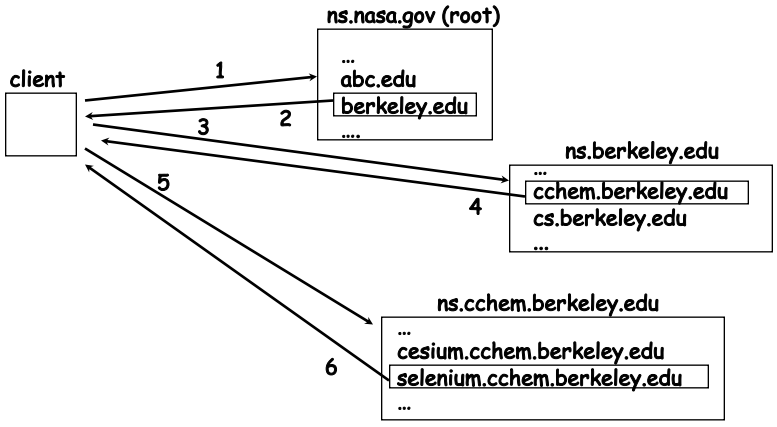
- The DNS **namespace** is organized and administered hierarchically.
- A domain name is really a sequence of components, which become increasingly general from left to right.
- Example:



24

DNS - name resolution

- The process of looking up a domain name to address mapping is called **name resolution.** Name resolution happens recursively:



25

Overview

- Introduction
  - History, Basic Concepts
- Networking Fundamentals
  - Layering, Internetworking, Addressing, Naming
- Internet Protocols
  - Link Layer Protocols, IP, Transport Protocols (TCP, UDP)
- Network Programming
  - BSD sockets

26

Internet Protocols -- The Link Layer

Application	(telnet, ftp, WWW, …)
Transport	Transmission Control Protocol (TCP), UDP
Network	Internet Protocol (IP)
Data Link	**(Ethernet, token ring, PPP,…)**

27

The link layer

- The **Link Layer** refers to the software directly responsible for a physical link.
  - This is generally found in the driver for a particular piece of hardware (e.g. the Ethernet driver).
- The link layer is highly dependent on the network hardware being used.
- There are different link layer standards for **Ethernet**, **RS-232 serial link**, and **token ring** hardware.

28

Link layer encapsulation

- There are standards which specify the exact format of link layer frames, as well how IP datagrams are placed in such frames.
- For example, the Ethernet link layer standard specifies that the `type` field in an Ethernet frame for an IP datagram shall be set to the value `0x800`.
- Link layer standards for serial links (such as **SLIP** and **PPP)** specify special framing bytes to place around IP datagrams before transmission.

29

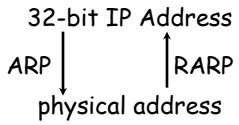
Link layer addressing

- Some data link technologies provide their own form of addressing. Such addresses are called **physical addresses.**
- For example, every Ethernet card has a unique, 6-byte **Ethernet address** pre-assigned by the manufacturer.
- We need some mechanism for translating between link-layer physical addresses and 32-bit IP addresses.
- To address this problem, the **Address Resolution Protocol (ARP)** and **Reverse Address Resolution Protocol (RARP)** were devised.

30

ARP and RARP

ARP and RARP translate between physical addresses and IP addresses:



- Key Points:
  - ARP and RARP work by broadcasting a query on the local network.
  - ARP and RARP are network-layer protocols, in parallel to IP in layering diagrams.
  - RARP is only used by a machine at boot-time to discover its own IP address.

31

Internet Protocols - The Network Layer

Application	(telnet, ftp, WWW, …)
Transport	Transmission Control Protocol (TCP), UDP
Network	**Internet Protocol (IP)**
Data Link	(Ethernet, token ring, PPP,…)

32

Internet protocol - basic concepts

- IP is a
  stateless,
  connectionless,
  unordered,
  unreliable,

protocol. More sophisticated facilities such as logical connections and reliability are provided by higher layers.

- Internet hosts communicate by exchanging IP **datagrams.**

33

IP – basic concepts

- IP is **stateless** - hosts do not retain any information at the level of IP about previous transmissions.
- IP is **connectionless** - a datagram may be sent from one node to another without first "opening a connection" to the node.
- IP is **unordered** - packets may arrive at their destination in a different order than they were sent.
- IP is **unreliable** - packets may be dropped or corrupted.

34

IP - reliability (or lack thereof):

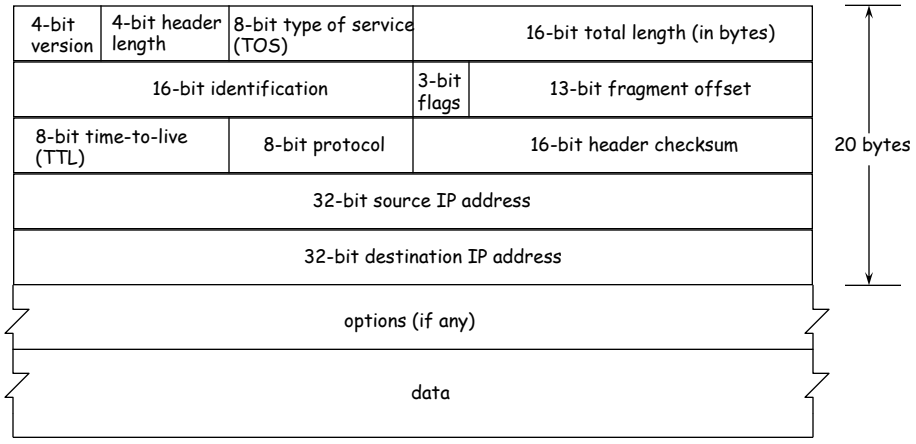
- IP is a *best-effort* delivery service. Packets may be:
  corrupted,
  duplicated,
  reordered, or
  dropped

en route from their source to their destination.

- Higher-level protocols (e.g. TCP) must be prepared to deal with all of these possibilities.
  - Techniques: acknowledgement/retransmission, data checksum, sequence numbers

35

Format of an IP datagram



36

IP datagram format - notes

- 16-bit *header size* limits max IP datagram size to 64KB
- *TTL* limits number of routers a datagram may traverse
  - decremented by 1 every time packet forwarded by some router
- *header checksum* calculated over IP header only
- *identification* field uniquely identifies each datagram sent by a host
- *options* are rarely used, but provided for things like:
  - recording routes (with severe size limitations)
  - loose/strict source routing

37

The Internet Protocols - Transport Layer

Application	(telnet, ftp, WWW, …)
Transport	**Transmission Control Protocol (TCP),** UDP
Network	Internet Protocol (IP)
Data Link	(Ethernet, token ring, PPP,…)

38

TCP – basic concepts

- TCP is a transport-layer protocol layered on top of IP. TCP provides a
  connection-oriented,
  two-way,
  ordered,
  reliable,
  byte-stream

  model of communication.

- IP provides *none* of the above services, so all of this functionality is found in the TCP protocol.

39

TCP – basic concepts

- TCP is **connection-oriented**. A logical connection must be established before communication begins.
- TCP is **ordered** - data is delivered to a receiving application in the order it was transmitted by the sender.
- TCP is **reliable** - Retransmissions and acknowledgements are used to ensure that all data arrives at the destination. Checksums are used to ensure that data is not corrupted in transit.
- TCP presents a **byte-stream** model - data may be delivered in different-sized chunks than it was transmitted.

40

TCP addressing – port numbers

- Every host has an IP address which identifies that host.
- We would like to support more than one simultaneous transport connection per host. TCP uses a 16-bit **Port Number** to distinguish different connections.
- Certain port numbers are reserved for specific applications:

21	**ftp (file transfer protocol)**
23	**telnet (remote login service)**
25	**SMTP (electronic mail)**
80	**http (World Wide Web)**

- We use the pair (IP Address,port number) to identify a particular endpoint for communication.

41

TCP - implementation

- The model TCP supports is very different from the model IP supports.
- TCP must therefore do all the work required to **provide a reliable service over an unreliable network**:
  - TCP breaks data to be sent into optimally sized **segments**. Each segment is assigned a monotonically increasing **sequence number**.
  - When TCP sends a segment, it sets a **retransmit timer**.
  - TCP periodically sends **acknowledgements** indicating the highest sequence number it has received.
  - If the retransmit timer expires before an acknowledgement is received, TCP sends the segment again.
  - The sequence numbers can also be used for **duplicate supression** and **ordering**.

42

Aside: User Datagram Protocol (UDP)

- In addition to TCP, there is another transport-layer protocol called **UDP (User Datagram Protocol).**
- UDP provides an unreliable, unordered datagram delivery service, much like IP. In fact, each UDP datagram results in the transmission of exactly one IP datagram.
- UDP adds only two things to IP:

  Port Numbers - like TCP, UDP supports port numbers in order to allow multiple applications to use UDP simultaneously.
  * N.B. Unlike TCP, however, UDP is *connectionless*.

  Checksums - An end-to-end checksum offers a minimal guarantee that data received wasn't corrupted in transit.

43

UDP, Cont.

- **Q:** Given the availability of TCP, why would anyone want to use UDP?
- **A:** The facilities provided by TCP impose a certain overhead on communications. Certain applications either
  - don't need the extra facilities TCP provides, or
  - don't find the overhead acceptable.
- Example: soft real-time traffic, such as audio or video.
  - Minimizing delay is much more important than getting every bit perfect.
  - If loss is rare, user probably won't notice the loss.

44

Overview

- Introduction
  - History, Basic Concepts
- Networking Fundamentals
  - Layering, Internetworking, Addressing, Naming
- Internet Protocols
  - Link Layer Protocols, IP, Transport Protocols (TCP, UDP)
- Network Programming
  - BSD sockets

45

Network programming

- Application programmers can use a number of different techniques to write applications for the Internet:
  - **BSD sockets** - a transport-layer API which originated in Berkeley UNIX, now found on nearly every major platform.
  - **TLI (Transport Layer Interface)** - another transport-layer API which originated in AT&T's System V UNIX.
    - more complex interface than sockets, without much benefit
  - **RPC (Remote Procedure Call)**:
    - At a "higher level" than straight sockets
    - attempts to add structure to network communication, and achieve integration with application programming language

46

BSD sockets - the bare essentials

- A **socket** is a network communication endpoint, obtained through the `socket()` system call.
- The return value from `socket()` is also a Unix *file descriptor*, and may be passed to `read()`, `write()`, `select()`, `fcntl()` and `close()` just like any other fd.
- There are also socket-specific system calls, such as:

`bind()`	binds a socket to a specific port
`connect()`	establishes a remote connection
`sendto()`	sends a datagram to a given address
`accept()`	accepts an incoming connection

47

Sockets programming example

- A very simple `echo` server:
  - accepts a connection
  - reads a line of input from the connection
  - writes the line back to the connection
  - closes the connection

48

Echo Server - code

```
#include "echosrv.h" /* standard socket includes */
#define BUFSIZE 256
#define ECHOPORT 12345
int main(int argc,char *argv[])
{
     int servfd, clifd;       /* server and client fd's */
     struct sockaddr_in serv_addr, cli_addr;
     int clilen, servlen, nbytes;
     char buf[BUFSIZE];

     /* create a server socket */
     if ((servfd=socket(AF_INET,SOCK_STREAM,0)) < 0) {
            perror("socket");
            exit(1);
     }

     /* bind our local address to the echo port */
     memset(&serv_addr,0,sizeof(serv_addr));
     serv_addr.sin_family=AF_INET;
     serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
     serv_addr.sin_port=htons(ECHOPORT);

     if (bind(servfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0) {
            perror("bind");
            exit(1);
     }
```

49

Echo Server, cont…

```
      /* allow up to 5 pending connections */
      listen(servfd,5);

      while (1) {
             /* accept a new connection in clifd */
             if ((clifd=accept(servfd,(struct sockaddr *) &cli_addr,
                                  &clilen)) < 0) {
                  perror("accept");
                  exit(1);
             }
             printf("server accepted connection...\n");

             /* read a line of input */
             if ((nbytes=read(clifd,buf,BUFSIZE)) < 0) {
                  perror("read");
                  goto done;
             }
             /* echo line back to client */
             (void) write(clifd,buf,nbytes);

      done:
             (void) close(clifd);
      }
 }
```

50