Computer Networks and Internets with Internet Applications, 4e

By Douglas E. Comer

Lecture PowerPoints

By Lami Kaya, LKaya@ieee.org

Chapter 3

Network Programming and Applications

Topics Covered

- 3.1 Introduction
- 3.2 Network Communication
- 3.3 Client-Server Computing
- 3.4 Communication Paradigm
- 3.5 An Example Application Program Interface
- 3.6 An Intuitive Look At The API
- 3.7 Definition Of The API
  - 3.7.1 The Await_Contact Function
  - 3.7.2 The Make_Contact Function
  - 3.7.3 The Appname_To_Appnum Function
  - 3.7.4 The Cname_To_Comp Function
  - 3.7.5 The Send Function
  - 3.7.6 The Recv And Recvln Functions
  - 3.7.7 The Send_Eof Function
  - 3.7.8 Summary Of API Types

Topics Covered (cont)

- 3.8 Code For An Echo Application
  - 3.8.1 Example Echo Server Code
  - 3.8.2 Example Echo Client Code
- 3.9 Code For A Chat Application
  - 3.9.1 Example Chat Server Code
  - 3.9.2 Example Chat Client Code
- 3.10 Code For A Web Application
  - 3.10.1 Example Web Client Code
  - 3.10.2 Example Web Server Code
- 3.11 Managing Multiple Connections With The Select Function

3.1 Introduction

This chapter

- describes NW from a programmer's point of view
- outlines the NW facilities available to a programmer
- examines example applications that use a NW
- introduces a small set of library functions
- shows how the library functions can be used

This chapter will demonstrate an important idea:

- A programmer can create Internet applications
  - without understanding the underlying NW technology or communication protocols

3.2 Network Communication

- When applications use a NW, they do so in pairs
  - The pair uses the NW merely to exchange messages
- Ex: imagine a distributed database service that allows remote users to access a central database
- Such a service requires two applications,
  - one running on the computer that has the database
  - and the other running on a remote computer
- The application on the remote computer sends a request to the application running on the database computer
  - When the request arrives, the application running on the database computer consults the database and returns a response
- Only the two applications understand the message format and meaning

3.3 Client-Server Computing (1)

- One application starts first and waits for the other application to contact it
- The second application must know the location where the first application is waiting
- The arrangement in which a NW application waits for contact from another application is known as the
  - client-server paradigm or client-server computing
- The program that waits for contact is called a server
- The program that initiates contact is known as a client
  - To initiate contact, a client must know where the server is running, and must specify the location

3.3 Client-Server Computing (2)

How does a client specify the location of a server?

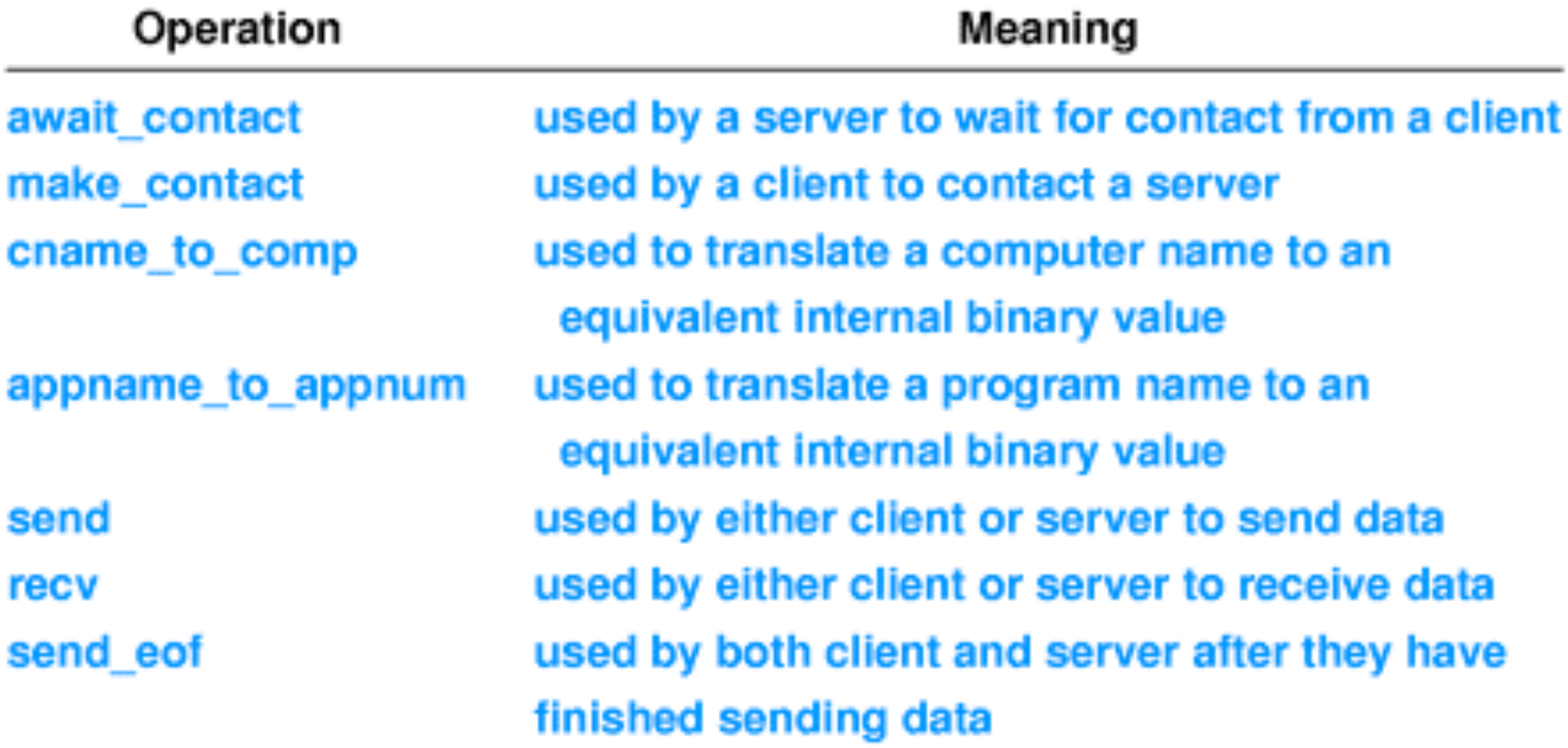
- In the Internet, a location is given by a pair of identifiers
  - (computer, application)
    - computer identifies the computer on which the server is running
    - application identifies a particular application program on that computer
- Application SW represents the two values as binary numbers
- Humans never need to deal with the binary representation directly
  - Instead, the values are also given alphabetic names
- Humans enter the names, and software translates each name to a corresponding binary value automatically

3.4 Communication Paradigm

- Two applications
  - establish communication,
  - exchange messages back and forth,
  - and then terminate
- The steps (in details) are:
  - The server starts first, and waits for contact from a client
  - The client specifies the server's location and requests a connection be established
  - Once a connection is in place, the client and server use the connection to exchange messages
  - After they finish sending data, the client and server each send an end-of-file (EOF) and the connection is terminated

3.5 An Example Application Program Interface

- Application Program Interface (API) is used to describe the set of operations available to a programmer
- The API specifies the arguments for each operation as well as the semantics
- Figure 3.1 lists the seven functions that an application can call
  - Functions **send** and **recv** are supplied directly by the OS
  - Other functions in the API consist of routines that are written
  - These seven functions are sufficient for most NW applications

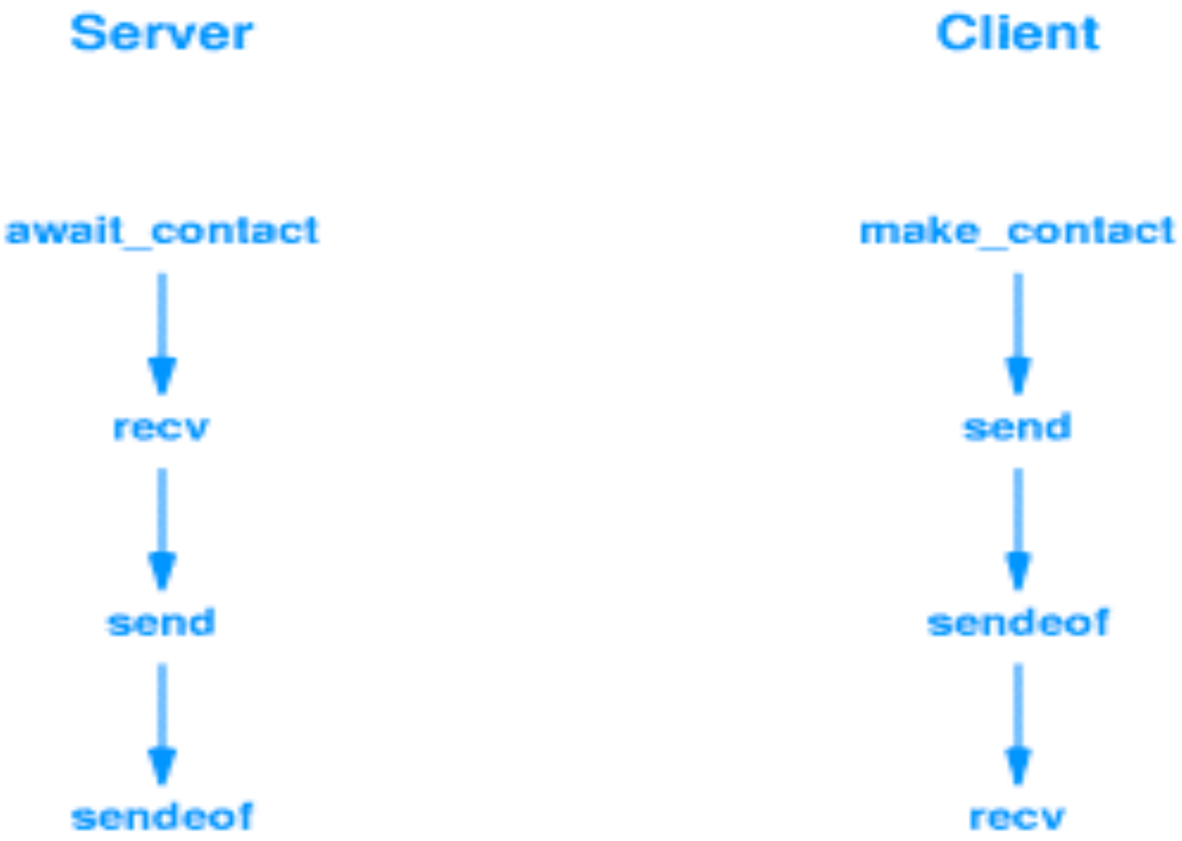


Operation	Meaning
await_contact	used by a server to wait for contact from a client
make_contact	used by a client to contact a server
cname_to_comp	used to translate a computer name to an equivalent internal binary value
appname_to_appnum	used to translate a program name to an equivalent internal binary value
send	used by either client or server to send data
recv	used by either client or server to receive data
send_eof	used by both client and server after they have finished sending data

Figure 3.1 An example API consisting of seven operations. These seven functions are sufficient for most network applications†.

3.6 An Intuitive Look At The API

- A server begins by calling **await_contact** to wait for contact from a client
- The client begins by calling **make_contact** to establish contact
- Once the client has contacted the server
  - the two can exchange messages with **send** and **recv**
- The two applications must be programmed
  - to know whether to send or receive
  - if both sides try to receive without sending, they will block forever
- After it finishes sending data, an application calls **send_eof** to send the EOF
- On the other side, **recv** returns a value of zero to indicate that the EOF has been reached
- Figure 3.2 illustrates the sequence of API calls that the client and server make for such an interaction



Server

await_contact

recv

send

sendeof

Client

make_contact

send

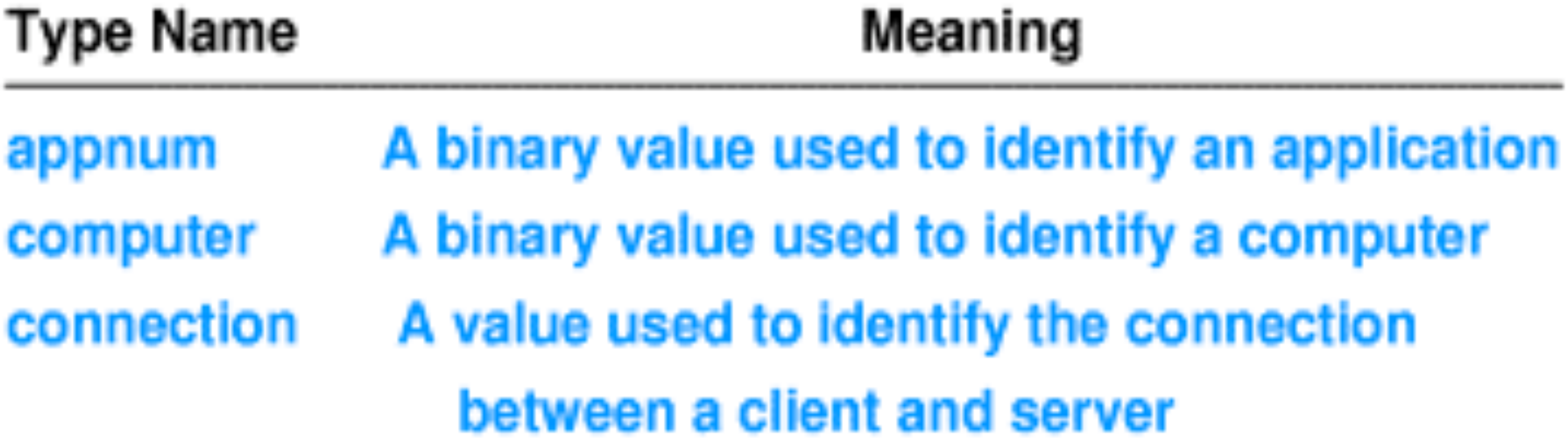
sendeof

recv

Figure 3.2 Illustration of the API calls used for a trivial interaction. The client sends one request and receives one reply.

3.7 Definition Of The API

- To keep our API independent of particular OS and NW software
  - we can define three data types and use
- Figure 3.3 lists the type names and their meanings
  - Using the three types, we can precisely define the example API



Type Name	Meaning
appnum	A binary value used to identify an application
computer	A binary value used to identify a computer
connection	A value used to identify the connection between a client and server

Figure 3.3 The three type names used in our example API. On a given computer these types are defined to be integers of a specific size.

Await_Contact Function

- A server calls function **await_contact** to wait for contact from a client

  connection await_contact(appnum a)

- The call takes one argument of type **appnum** and returns a value of type connection
  - The argument specifies a number that identifies the server application
  - a client must specify the same number when contacting the server
- The server uses the return value (type connection) to transfer data

Make_Contact Function

- A client calls function **make_contact** to establish contact with a server

  connection make_contact (computer c, appnum a)
- The call takes two arguments
  - identify a computer on which the server is running
  - and the application number that the server
- The client uses the return value
  - which is of type connection to transfer data

Appname_To_Appnum Function

- Clients and servers both use **appname_to_appnum**
  - to translate from a human-readable name for a service to an internal binary value
- The service names are standardized throughout the Internet

appnum appname_to_appnum(char *a)

- The call takes one argument and returns an equivalent binary value of type **appnum**

Cname_To_Comp Function

- Clients call **cname_to_comp**
  - to convert from a human-readable computer name to the internal binary value

computer cname_to_comp (char *c)

- The call takes one argument and returns an equivalent binary value of type computer

Send Function

- Both clients and servers use send to transfer data across the network

int send (connection con, char *buffer, int length, int flags)

- The call takes four arguments.
  - The first argument specifies a connection previously established with **await_contact** or **make_contact**
  - the second is the address of a buffer containing data to send
  - the third argument gives the length of the data in bytes (octets)
  - and the fourth argument is zero for normal transfer
- Send returns the number of bytes transferred, or a negative value if an error occurred

Recv And Recvln Functions

- Both clients and servers use recv to access data that arrives

  int recv (connection con, char *buffer, int length, int flags)
- The call takes four arguments
  - The first specifies a connection with **await_contact** or **make_contact**
  - the second is the address of a buffer into which the data to be placed
  - the third gives the size of the buffer in bytes
  - and the fourth is zero for normal transfer
- **recv** returns the number of bytes that were placed in the buffer
  - zero to indicate that EOF has been reached
  - or a negative value to indicate that an error occurred
- We can also use a library function **recvln** that repeatedly calls **recv** until an entire line of text has been received.

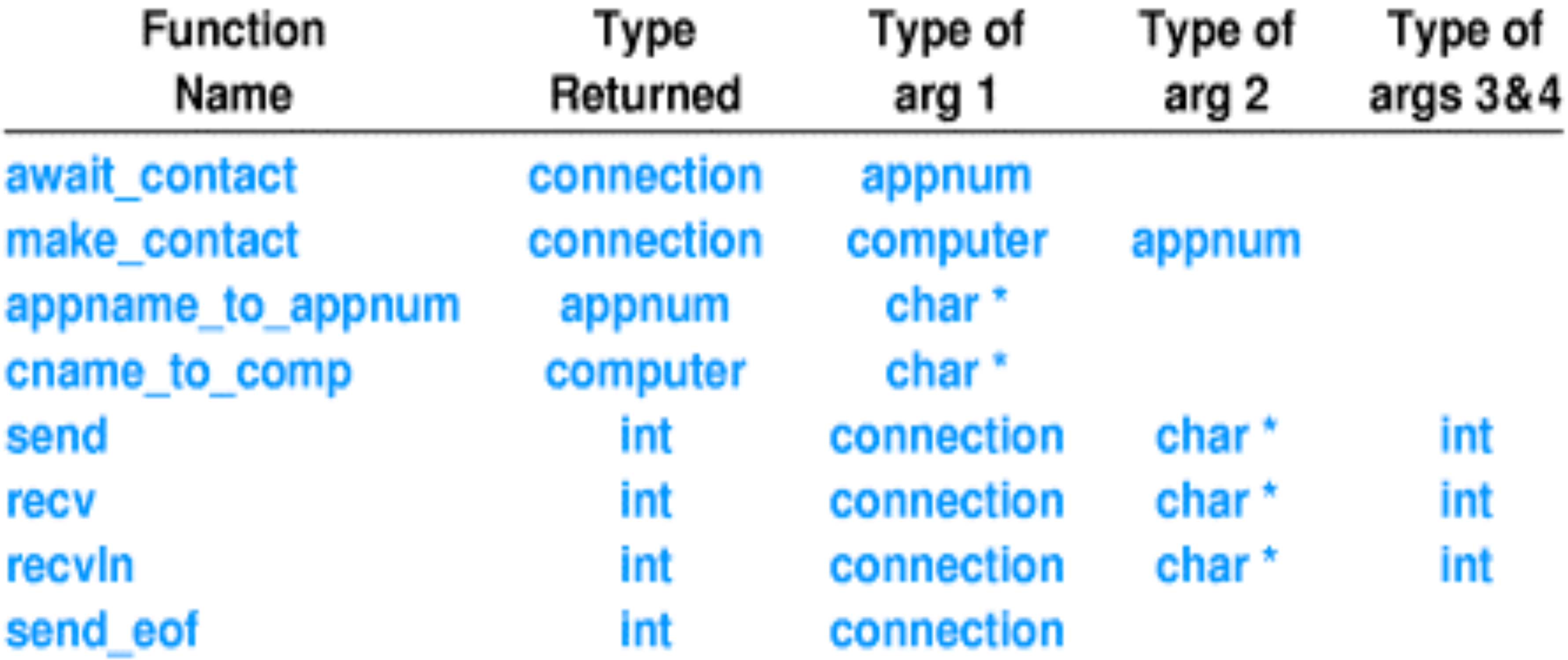
  int recvln (connection con, char *buffer, int length)

Send_Eof Function

- Both the client and server must use **send_eof** after sending data
  - to inform the other side that no further transmission will occur
- On the other side, the **recv** function returns zero when it receives the EOF

int send_eof(connection con)

- Argument specifies a connection previously established with **await_contact** or **make_contact**
- The function returns a negative value
  - to indicate that an error occurred, and a non-negative value otherwise



Function Name	Type Returned	Type of arg 1	Type of arg 2	Type of args 3&4
await_contact	connection	appnum		
make_contact	connection	computer	appnum	
appname_to_appnum	appnum	char *		
cname_to_comp	computer	char *		
send	int	connection	char *	int
recv	int	connection	char *	int
recvln	int	connection	char *	int
send_eof	int	connection		

Figure 3.4 A summary of argument and return types for the example API.

3.8 Code For An Echo Application (1)

- The client repeatedly
  - prompts the user for a line of input,
  - sends the line to the server,
  - and then displays whatever the server sends back.
- Like all the applications described in this chapter,
  - the echo application operates across a NW
  - That is, the client and server programs can run on separate computers
- Figure 3.5 illustrates connection to the Internet

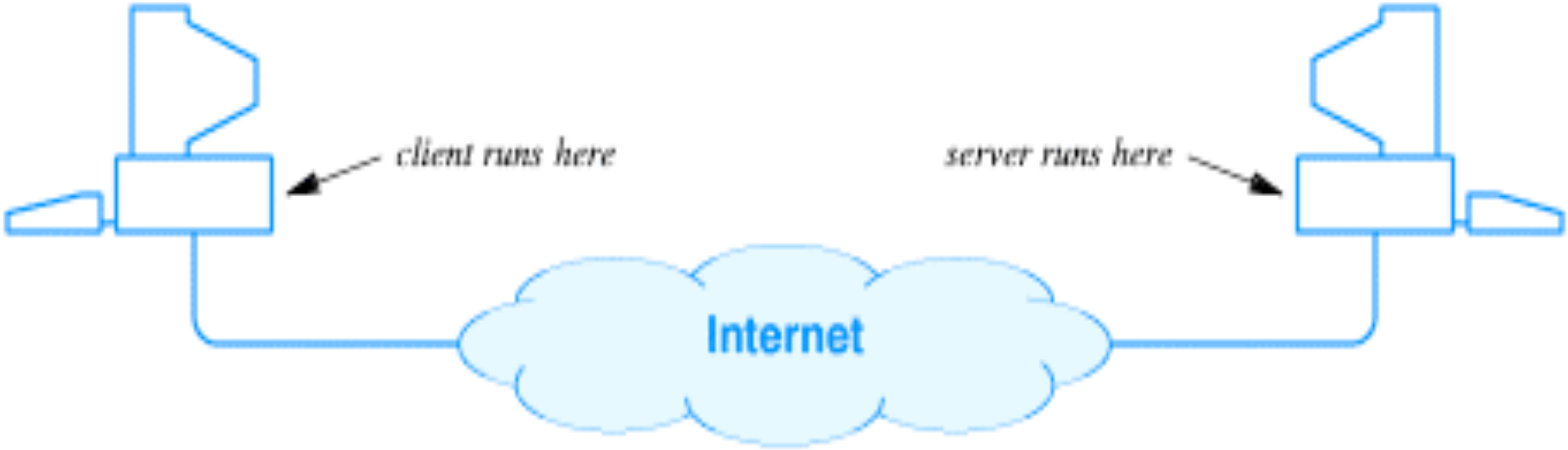


Figure 3.5 Illustration of the echo application, which can be used on any two computers connected to the Internet. The client program runs on one computer and the server program runs on another.

3.8 Code For An Echo Application (2)

- Ex: suppose someone using computer **lancelot.cs.purdue.edu** chooses 20000 as the application number
- The server is invoked by the command:

```
echoserver  20000
```

- If some other application is using number 20000,
  - the server emits an appropriate error message and exits
  - the user must choose another number.
- Once the server has been invoked, the client is invoked:

```
echoclient  lancelot.cs.purdue.edu  20000
```

3.9 Code For A Chat Application

- A simplified version of chat that works between a single pair of users
- One user begins by choosing an application number and running the server
- Ex: suppose a user on excalibur.cs.purdue.edu runs the server:

  chatserver 25000
- A user on another computer can invoke the client:

  chatclient excalibur.cs.purdue.edu 25000
- To keep the code as small as possible
  - the scheme requires users to take turns entering text
  - Users alternate entering text until one of them sends an EOF

3.10 Code For A Web Application (1)

- To run the server, a user chooses an application number and invokes the server
- Ex: if a user on computer **netbook.cs.purdue.edu** chooses application number **27000**, the server can be invoked with the command:

```
webserver  27000
```

- The client specifies a computer, a path name, and an application number:

```
webclient  netbook.cs.purdue.edu  /index.html  27000
```

- It is possible to use a conventional Web browser (such as, Internet Explorer or Netscape) to access the server

```
http://netbook.cs.purdue.edu:27000/index.html
```

3.10 Code For A Web Application (2)

- The client code is extremely simple:
  - after establishing communication with the Web server, it sends a request, which must have the form :

    GET / path HTTP/1.0 CRLF CRLF

    CRLF GET
  - where path denotes the name of an item such as index.html ,
  - CRLF denotes the two characters carriage return and line feed.
- After sending the request, the client receives and prints output from the server

3.10 Code For A Web Application (3)

- Web server may seem more complex than previous examples,
  - complexity results from Web details rather than networking details
- In addition to reading and parsing a request
  - the server must send both a ``header'' and data in the response
  - The header consists of several lines of text that are terminated by CRLF
- The header lines are of the form:

```
HTTP/1.0 status  status_string   CRLF
Server: CNAI Demo Server  CRLF
Content-Length:  datasize   CRLF
Content-Type: text/html  CRLF
CRLF
```

- where **datasize** denotes the size of the data that follows

3.10 Code For A Web Application (4)

- The code is also complicated by error handling
  - error messages must be sent in a form that a browser can understand
- If a request is incorrectly formed, our server generates a

  400 error message
- If the item specified in the request cannot be found

  404 error message
- The Web server differs from the previous examples in a significant way:
  - the server program does not exit after satisfying one request
  - Instead, it remains running, ready for additional requests

3.11 Managing Multiple Connections With The Select Function (1)

- Although our example API supports 1-to-1 interaction between a client and server,
  - the API does not support 1-to-many interaction
- To see why, consider multiple connections
  - To create such connections
    - a single application must call **make_contact** multiple times
    - specifying a computer and appnum for each call
- Once the connections have been established
  - the application cannot know which of them will receive a message first

3.11 Managing Multiple Connections With The Select Function (2)

- Many OS include a function named select that solves the problem of managing multiple connections
  - The select call checks a set of connections
  - The call blocks until at least one of the specified connections has received data
  - The call then returns a value that tells which of the connections have received data

3.11 Managing Multiple Connections With The Select Function (3)

- Ex: consider an application that must receive requests and send responses over two connections
  - Such an application can have the following general form:

```
Call  make_contact  to form connection1;
Call  make_contact  to form connection2;

Repeat forever  {
    Call  select  to determine which connection is ready
    If (connection1 is ready)  {
            Call  recv  to read request from connection1;
            Compute response to request;
            Call  send  to send response over connection1;
    } if (connection2 is ready)  {
            Call  recv  to read request from connection2;
            Compute response to request;
            Call  send  to send response over connection2; }    }
```