PART XIV

Internet Applications

(Client-Server Concept, Use of Protocol Ports, Socket API, DNS, E-mail, TELNET, FTP)

Functionality

- Transport layer and layers below
  - Basic communication
  - Reliability
- Application layer
  - Abstractions
    - Files
    - Services
    - Databases
  - Names

Dichotomy Of Duties

- Network
  - Transfers bits
  - Operates at application's request
- Applications determine
  - What to send
  - When to send
  - Where to send
  - Meaning of bits

Important Point

Although an internet system provides a basic communication service, the protocol software cannot initiate contact with, or accept contact from, a remote computer. Instead, two application programs must participate in any communication: one application initiates communication and the other accepts it.

How Two Application Programs Make Contact

- One application
  - Begins execution first
  - Waits passively at prearranged location
- Another application
  - Begins execution later
  - Actively contacts first program
- Called *client-server interaction*

Client-Server Paradigm

- Used by all network applications
- Passive program called a *server*
- Active program called a *client*

Internet Communication

All network applications use a form of communication known as the client-server paradigm. *A server application waits passively for contact, while a client application initiates communication actively.*

Characteristics Of A Client

- Arbitrary application program
- Becomes client temporarily
- Can also perform other computations
- Invoked directly by user
- Runs locally on user's computer
- Actively initiates contact with a server
- Contacts one server at a time

Characteristics Of A Server

- Special-purpose, privileged program
- Dedicated to providing one service
- Can handle multiple remote clients simultaneously
- Invoked automatically when system boots
- Executes forever
- Needs powerful computer and operating system
- Waits passively for client contact
- Accepts requests from arbitrary clients

Terminology

- *Server*
  - An executing program that accepts contact over the network
- *Server-class computer*
  - Hardware sufficient to execute a server
- Informally
  - Term *server* often applied to computer

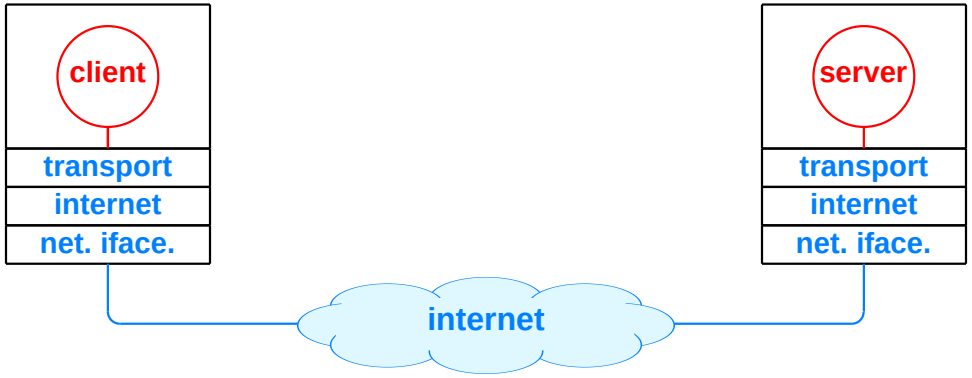
Direction Of Data Flow

- Data can flow
  - From client to server only
  - From server to client only
  - In both directions
- Application protocol determines flow
- Typical scenario
  - Client sends request(s)
  - Server sends response(s)

Key Idea

Although the client initiates contact, information can flow in either or both directions between a client and server. Many services arrange for the client to send one or more requests and the server to return one response for each request.

Clients, Servers, And Other Protocols



- Clients and servers are application programs

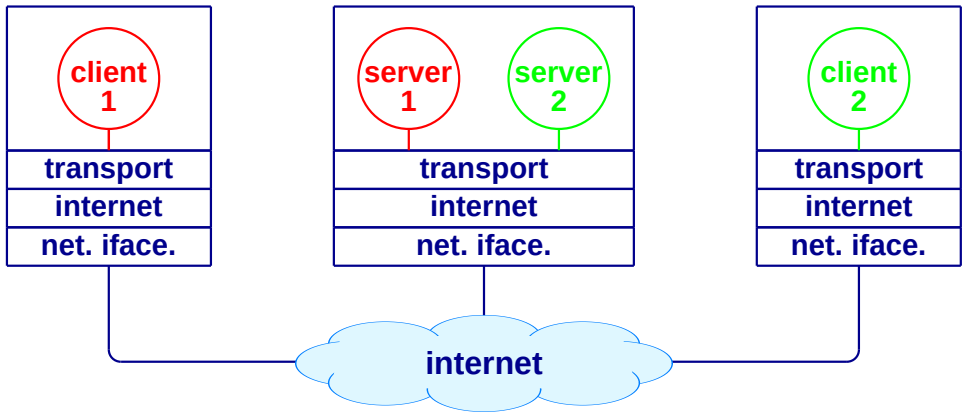
Server CPU Use

- Facts
  - Server operates like other applications
    - Uses CPU to execute instructions
    - Performs I/O operations
  - Waiting for data to arrive over a network does not require CPU time
- Consequence
  - Server program only uses CPU when servicing a request

Multiple Services

- Can have multiple servers on single computer
- Servers only use processor when handling a request
- Powerful hardware required to handle many services simultaneously

Illustration Of Multiple Servers



- Each server offers one service
- One server can handle multiple clients

Identifying A Service

- Protocol port number used
- Each service given unique port number, *P*
- Server
  - Informs OS it is using port *P*
  - Waits for requests to arrive
- Client
  - Forms request
  - Sends request to port *P* on server computer

The Point About Ports

Transport protocols assign each service a unique port identifier. A server must specify the identifier when it begins execution. A client must specify the identifier when it requests transport protocol software to contact a server. Protocol software on the server computer uses the identifier to direct an incoming request to the correct server.

In Theory

- Port numbers are merely integers
- Any server could use any port number

In Practice

- Protocol port numbers used as service identifiers
- Need uniform numbering
  - To allow arbitrary client to contact server on arbitrary machine
  - To avoid inventing ‘‘directory assistance’’ mechanism
- Port numbers
  - Uniform throughout Internet
  - Set by standards bodies

Terminology

- *Sequential program*
  - Typical of most programs
  - Single thread of control
- *Concurrent program*
  - Multiple threads of control
  - Execution proceeds ‘‘in parallel’’
  - More difficult to create

Servers And Concurrency

- Sequential server
  - Also called *iterative*
  - Handles one request at a time
- Concurrent server
  - Can handle multiple requests at a time
  - No waiting

Delay In Servers

- Concurrent server
  - Server creates new thread of control to handle each request
  - Client only waits for its request to be processed
- Sequential server
  - Client waits for all previous requests to be processed as well as for its request to be processed
  - Unacceptable to user if long request blocks short request

Concurrency In Servers

Concurrent execution is fundamental to servers because concurrency permits multiple clients to obtain a given service without having to wait for the server to finish previous requests. In a concurrent server, the main server thread creates a new service thread to handle each client.

Protocol Ports And Concurrent Servers

- Apparent problem
  - One port number assigned to each service
  - Concurrent server has multiple copies (threads) running
  - Client and server may interact
  - Messages sent to server's port must be delivered to correct copy

Protocol Ports And Concurrent Servers (continued)

- Solution to problem: use information about client as well as server to deliver incoming packets
- TCP uses four items to identify connection
  - Server's IP address
  - Server's protocol port number
  - Client's IP address
  - Client's protocol port number

Demultiplexing In A Concurrent Server

Transport protocols assign an identifier to each client as well as to each service. Protocol software on the server's machine uses the combination of client and server identifiers to choose the correct copy of a concurrent server.

Variations On A Theme

- A server can use
  - Connectionless transport (UDP)
  - Connection-oriented transport (TCP)
  - Both for a single service
- A single server can offer multiple services
  - Often used for ‘‘trivial’’ services
  - Server uses multiple port numbers simultaneously

Variations On A Theme
(continued)

- A server can
  - Maintain interaction with a client for days or hours
  - Send a short response and terminate interaction
  - Perform I/O on the local computer
  - Become a client for another service (potential cycle problem)

Example Of Circularity

- Time server
  - Returns time of day
- File server
  - Allows client to read or write a file
  - Calls time server when generating time stamp for file
- Suppose programmer modifies time server to log requests to a file

Interacting With Protocol Software

- Client or server uses transport protocols
- Protocol software inside OS
- Applications outside OS
- Mechanism needed to bridge the two
  - Called *Application Program Interface (API)*

Application Program Interface

- Part of operating system
- Permits application to use protocols
- Defines
  - Operations allowed
  - Arguments for each operation

Socket API

- Originally designed
  - For BSD UNIX
  - To use with TCP/IP protocols
- Now
  - Industry standard
  - Available on many operating systems

Socket

- OS Abstraction (not hardware)
- Created dynamically
- Persists only while application runs
- Referenced by a descriptor

Descriptor

- Small integer
- One per active socket
- Used in all operations on socket
- Generated by OS when socket created
- Only meaningful to application that owns socket
- In UNIX, integrated with file descriptors

Creating A Socket

- Application calls *socket* function

```
sdesc = socket(protofamily, type, proto)
```

- OS returns descriptor for socket
- Descriptor valid until application closes socket or exits

Socket Functionality

- Socket completely general
- Can be used
  - By client
  - By server
  - With a CO transport protocol
  - With a CL transport protocol
  - To send data, receive data, or both
- Large set of operations

Socket Operations

- *Close*
  - Terminate use of socket
  - Permanent
- *Bind*
  - Specify protocol port for a socket
  - Specify local IP address for a socket
  - Can use *INADDR_ANY* for any IP address

Socket Operations
(continued)

- *Listen*
  - Used by server
  - Prepares socket to accept incoming connections
- *Accept*
  - Used by server
  - Waits for next connection and returns new socket

Socket Operations
(continued)

- *Connect*
  - Used by client
  - Either
    * Forms a TCP connection
    * Fully specifies addresses for UDP

Two Purposes Of The Connect Function

The connect *function, which is called by clients, has two uses. With connection-oriented transport,* connect *establishes a transport connection to a specified server. With connectionless transport,* connect *records the server's address in the socket, allowing the client to send many messages to the same server without specifying the destination address with each message.*

Socket Operations
(continued)

- *Send*, *sendto*, and *sndmsg*
  - Transfer outgoing data from application
- *Recv*, *recvfrom*, and *recvmsg*
  - Transfer incoming data to application
- Many additional functions
  - Supply support and utility services
  - Some implemented as library calls

Examples Of Socket Support Functions

- *Gethostbyname*
  - Maps domain name to IP address
  - Example of argument

    `www.netbook.cs.purdue.edu`

- *Getprotobyname*
  - Maps name of protocol to internal number
  - Argument usually `"tcp"` or `"udp"`

Example Service

- Purpose
  - Count times invoked
  - Return printable ASCII message
- Connection-oriented protocol
- Sequential execution (not concurrent)

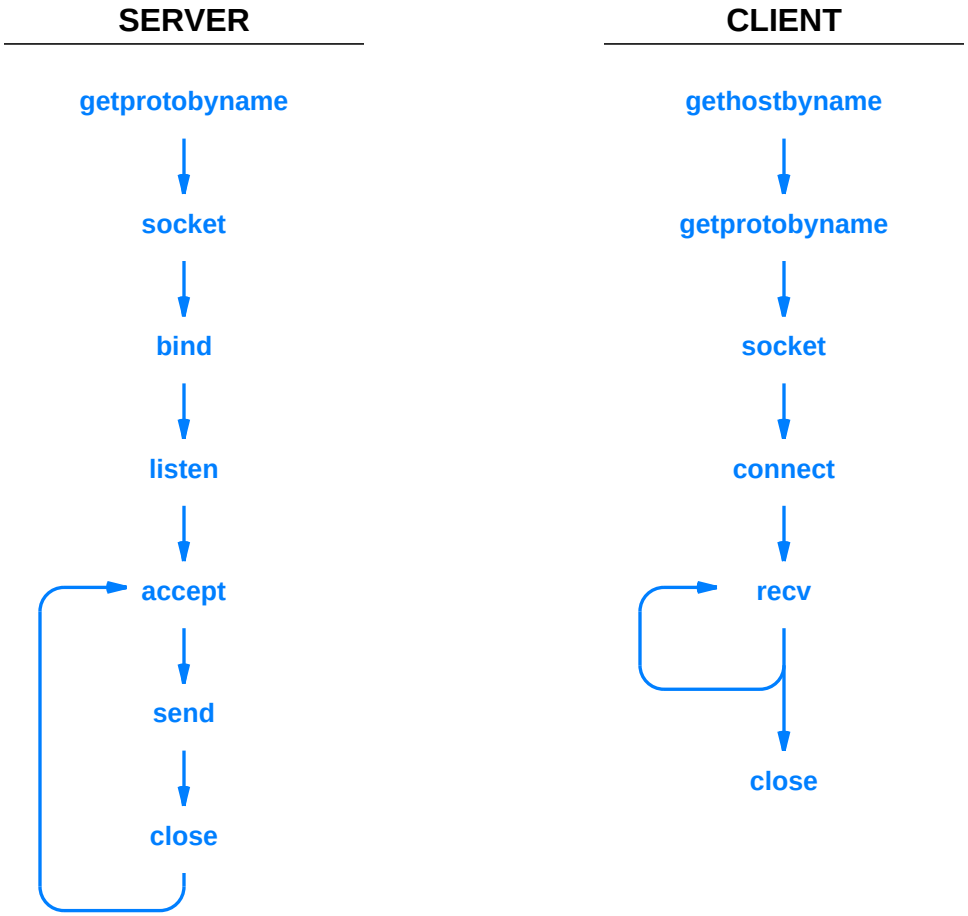
Example Client

- Open TCP connection to server
- Iterate until end-of-file
  - Receive text
  - Print characters received
- Close connection
- Exit

Example Server

- Create socket and put in passive mode
- Iterate forever
  - Accept next connection, get new socket
  - Increment count and send text message
  - Close socket for connection
- Notes
  - Main socket remains open
  - Server never exits

Socket Calls In Client And Server



- Client closes socket after use
- Server never closes original socket

Code For Client

- Arguments to program
  - Host
  - Protocol port
  - Both optional
- Many details
- Minor incompatibilities among socket implementations
  - Unix
  - Microsoft
  - Use C #ifdef

Example Client Code (1)

```
/* client.c - code for example client program that uses TCP */

#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>

#define PROTOPORT       5193            /* default protocol port number */

extern  int             errno;
char    localhost[] =   "localhost";    /* default host name            */
/*------------------------------------------------------------------------
 * Program:   client
 *
 * Purpose:   allocate a socket, connect to a server, and print all output
```

Example Client Code (2)

```
 * Syntax:    client [ host [port] ]
 *
 *               host  - name of a computer on which server is executing
 *               port  - protocol port number server is using
 *
 * Note:      Both arguments are optional.  If no host name is specified,
 *            the client uses "localhost"; if no protocol port is
 *            specified, the client uses the default given by PROTOPORT.
 *
 *---------------------------------------------------------------------
 */
main(argc, argv)
int     argc;
char    *argv[];
{
        struct  hostent  *ptrh;  /* pointer to a host table entry       */
        struct  protoent *ptrp;  /* pointer to a protocol table entry   */
        struct  sockaddr_in sad; /* structure to hold an IP address     */
        int     sd;              /* socket descriptor                   */
        int     port;            /* protocol port number                */
        char    *host;           /* pointer to host name                */
        int     n;               /* number of characters read           */
        char    buf[1000];       /* buffer for data from the server     */
#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET;         /* set family to Internet     */
```

Example Client Code (3)

```
/* Check command-line argument for protocol port and extract    */
/* port number if one is specified.  Otherwise, use the default */
/* port value given by constant PROTOPORT                       */

if (argc > 2) {                 /* if protocol port specified   */
        port = atoi(argv[2]);   /* convert to binary            */
} else {
        port = PROTOPORT;       /* use default port number      */
}
if (port > 0)                   /* test for legal value         */
        sad.sin_port = htons((u_short)port);
else {                          /* print error message and exit */
        fprintf(stderr,"bad port number %s0,argv[2]);
        exit(1);
}

/* Check host argument and assign host name. */

if (argc > 1) {
        host = argv[1];         /* if host argument specified   */
} else {
        host = localhost;
}
```

Example Client Code (4)

```
/* Convert host name to equivalent IP address and copy to sad. */

ptrh = gethostbyname(host);
if ( ((char *)ptrh) == NULL ) {
        fprintf(stderr,"invalid host: %s0, host);
        exit(1);
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

/* Map TCP transport protocol name to protocol number. */

if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map
        exit(1);
}

/* Create a socket. */

sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
        fprintf(stderr, "socket creation failed0);
        exit(1);
}

/* Connect the socket to the specified server. */

if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
        fprintf(stderr,"connect failed0);
        exit(1);
}
```

Example Client Code (5)

```
        /* Repeatedly read data from socket and write to user's screen. */

        n = recv(sd, buf, sizeof(buf), 0);
        while (n > 0) {
                write(1,buf,n);
                n = recv(sd, buf, sizeof(buf), 0);
        }

        /* Close the socket. */

        closesocket(sd);

        /* Terminate the client program gracefully. */

        exit(0);
}
```

Code For Server

- Arguments to program
  - Protocol port
- C language ifdefs for socket variants

Example Server Code (1)

```
/* server.c - code for example server program that uses TCP */
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>

#define PROTOPORT       5193            /* default protocol port number */
#define QLEN            6               /* size of request queue        */

int     visits      =   0;              /* counts client connections    */
/*------------------------------------------------------------------------
 * Program:   server
 *
 * Purpose:   allocate a socket and then repeatedly execute the following:
 *              (1) wait for the next connection from a client
 *              (2) send a short message to the client
 *              (3) close the connection
 *              (4) go back to step (1)
```

Example Server Code (2)

```
 * Syntax:    server [ port ]
 *
 *               port  - protocol port number to use
 *
 * Note:      The port argument is optional.  If no port is specified,
 *            the server uses the default given by PROTOPORT.
 *
 *------------------------------------------------------------------------
 */
main(argc, argv)
int     argc;
char    *argv[];
{
        struct  hostent  *ptrh;  /* pointer to a host table entry       */
        struct  protoent *ptrp;  /* pointer to a protocol table entry   */
        struct  sockaddr_in sad; /* structure to hold server's address  */
        struct  sockaddr_in cad; /* structure to hold client's address  */
        int     sd, sd2;         /* socket descriptors                  */
        int     port;            /* protocol port number                */
        int     alen;            /* length of address                   */
        char    buf[1000];       /* buffer for string the server sends  */

#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET;         /* set family to Internet     */
        sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address   */
```

Example Server Code (3)

```
/* Check command-line argument for protocol port and extract    */
/* port number if one is specified.  Otherwise, use the default */
/* port value given by constant PROTOPORT                       */

if (argc > 1) {                 /* if argument specified        */
        port = atoi(argv[1]);   /* convert argument to binary   */
} else {
        port = PROTOPORT;       /* use default port number      */
}
if (port > 0)                   /* test for illegal value       */
        sad.sin_port = htons((u_short)port);
else {                          /* print error message and exit */
        fprintf(stderr,"bad port number %s0,argv[1]);
        exit(1);
}

/* Map TCP transport protocol name to protocol number */

if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map
        exit(1);
}

/* Create a socket */

sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
        fprintf(stderr, "socket creation failed0);
        exit(1);
}
```

Example Server Code (4)

```
        /* Bind a local address to the socket */

        if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
                fprintf(stderr,"bind failed0);
                exit(1);
        }

        /* Specify size of request queue */

        if (listen(sd, QLEN) < 0) {
                fprintf(stderr,"listen failed0);
                exit(1);
        }

        /* Main server loop - accept and handle requests */

        while (1) {
                alen = sizeof(cad);
                if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
                        fprintf(stderr, "accept failed0);
                        exit(1);
                }
                visits++;
                sprintf(buf,"This server has been contacted %d time%s0,
                        visits,visits==1?".":"s.");
                send(sd2,buf,strlen(buf),0);
                closesocket(sd2);
        }
}
```

Stream Interface

- Sender
  - Calls *send* repeatedly
  - Specifies number of octets per call
- TCP
  - Divides stream into segments
- Receiver
  - Calls *recv* repeatedly
  - Receives one or more octets per call
  - Count of zero means ‘‘end of file’’
  - Size received unrelated to size sent

Internet Applications

- Domain Name System
- Electronic mail
- Remote login
- File transfer
- World Wide Web
- All use client-server model

Names

- Internet communication requires IP addresses
- Humans prefer to use computer names
- Automated system available to translate names to addresses
- Known as *Domain Name System (DNS)*

DNS Functionality

- Given
  - Name of a computer
- Returns
  - Computer's internet address
- Method
  - Distributed lookup
  - Client contacts server(s) as necessary

Domain Name Syntax

- Alphanumeric *segments* separated by dots
- Examples

`www.netbook.cs.purdue.edu`

`www.eg.bucknell.edu`

- Most significant part on right

Obtaining A Domain Name

- Organization
  - Chooses desired name
  - Must be unique
  - Registers with central authority
  - Placed under one *top-level domain*
- Names subject to international law for
  - Trademarks
  - Copyright

Top-Level Domains

Domain Name	*Assigned To*
com	*Commercial organization*
edu	*Educational institution*
gov	*Government organization*
mil	*Military group*
net	*Major network support center*
org	*Organization other than those above*
arpa	*Temporary ARPA domain (still used)*
int	*International organization*
country code	*A country*

- Meaning assigned to each

Within Organization

- Subdivision possible
- Arbitrary levels allowed
- Not standardized
- Controlled locally by organization

Example Name Structure

- First level is *.com*
- Second level is company name
- Third level is division within company
- Fourth level either
  - Company subdivision
  - Individual computer

An Example

- Assume
  - Company is *Foobar*
  - Has two divisions
    - Soap division
    - Candy division
- Candy division has subdivisions
- Soap division has no subdivisions

An Example
(continued)

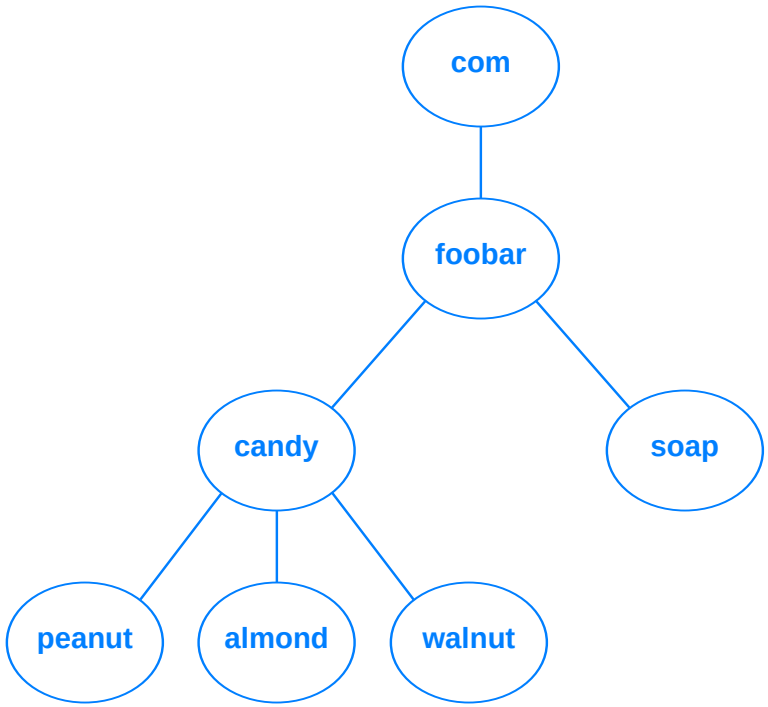
- Names in soap division have form

computer . soap . foobar . com

- Names in candy division have form

computer . *subdivision* . candy . foobar . com

Illustration Of Foobar Naming Hierarchy



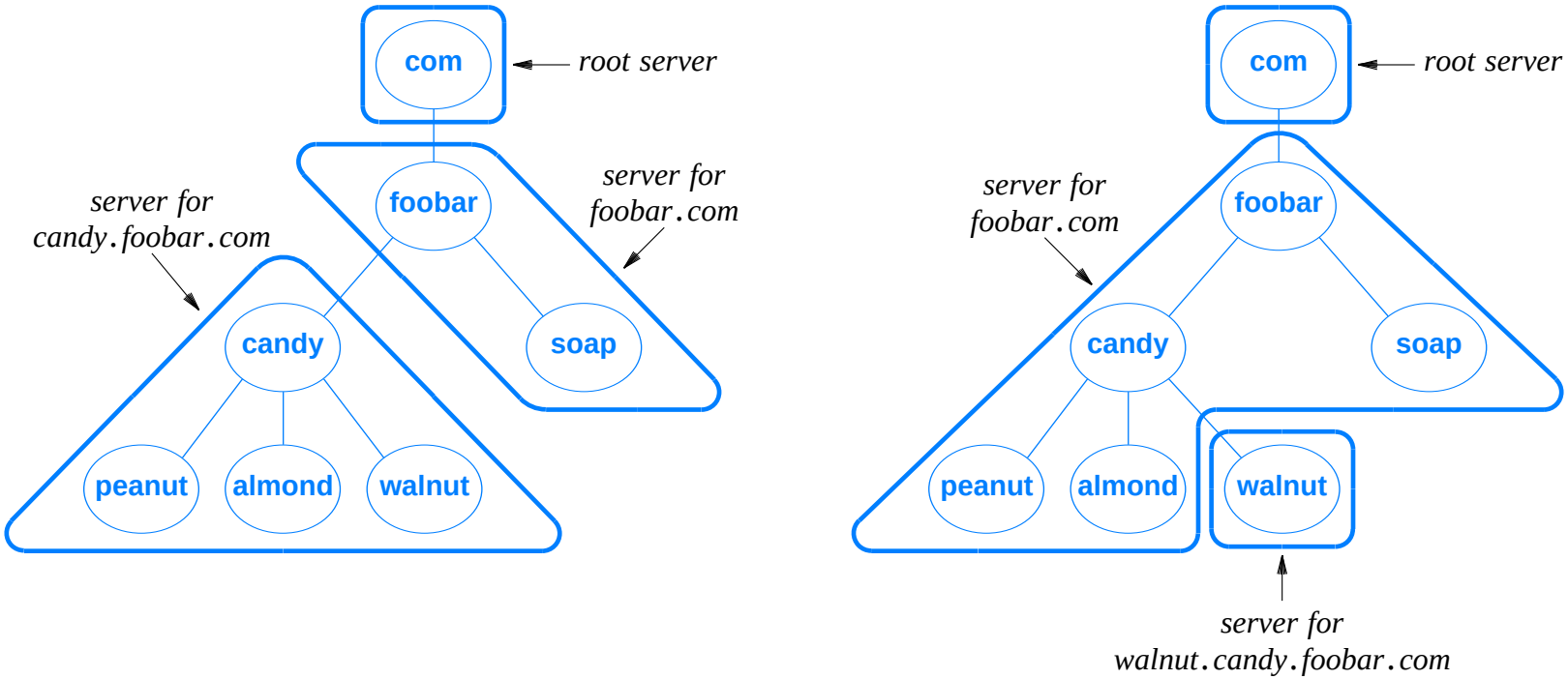
The Point About Names

The number of segments in a domain name corresponds to the naming hierarchy. There is no universal standard; each organization can choose how to structure names in its hierarchy. Furthermore, names within an organization do not need to follow a uniform pattern; individual groups within the organization can choose a hierarchical structure that is appropriate for the group.

DNS Client-Server Interaction

- Client known as *resolver*
- Multiple DNS servers used
- Arranged in hierarchy
- Each server corresponds to contiguous part of naming hierarchy

Two Possible DNS Hierarchies



- Choice made by organization

Inter-Server Links

All domain name servers are linked together to form a unified system. Each server knows how to reach a root server and how to reach servers that are authorities for names further down the hierarchy.

In Practice

- DNS uses backup server(s)
- ISPs and others
  - Offer DNS service to subscribers
- Small organizations and individuals
  - Only need domain names for computers running servers
  - Contract with an ISP for domain service

DNS Lookup

- Application
  - Becomes DNS client
  - Sends request to local DNS server
- Local server
  - If answer known, returns response
  - If answer unknown
    - Starts at top-level server
    - Follows links
    - Returns response
- Called *name resolution*

Caching In DNS

- Server always caches answers
- Host can cache answers
- Caching
  - Improves efficiency
  - Eliminates unnecessary search
  - Works well because high locality of reference

DNS Types

- Each entry in server consists of
  - Domain name
  - DNS type for name
  - Value to which name corresponds
- During lookup, client must supply
  - Name
  - Type
- Server
  - Matches both name and type

The Point About Types

The domain name system stores a type with each entry. When a resolver looks up a name, the resolver must specify the type that is desired; a DNS server returns only entries that match the specified type.

Example DNS Types

- Type *A (Address)*
  - Value is IP address for named computer
- Type *MX (Mail eXchanger)*
  - Value is IP address of computer with mail server for name
- Type *CNAME (Computer NAME)*
  - Value is another domain name
  - Used to establish alias (*www*)

Domain Name Abbreviation

- DNS lookup uses full names
- Users desire abbreviations
- Technique
  - Configure resolver with list of suffixes
  - Try suffixes one at a time

Example Of DNS Abbreviation

- Suffixes are
  - *cs . purdue . edu*
  - *purdue . edu*
  - *ecn . purdue . edu*
- User enters name *www*
- Resolver tries
  - *www*
  - *www . cs . purdue . edu*
  - *www . purdue . edu*
  - *www . ecn . purdue . edu*

Other Internet Applications

- Invoked directly by user
  - E-mail
  - Remote login
  - File Transfer
  - Web browsing

Electronic Mail

- Originally
  - Memo sent from one user to another
- Now
  - Memo sent to one or more *mailboxes*
- Mailbox
  - Destination point for messages
  - Can be storage or program
  - Given unique address

E-mail Address

- Text string
- Specifies mail destination
- General form

mailbox@computer

- *computer*
  - Domain name of computer
  - Actually type MX
- *mailbox*
  - Destination on the computer

Use Of E-mail Address

Each electronic mailbox has a unique address, which is divided into two parts: the first identifies a user's mailbox, and the second identifies a computer on which the mailbox resides. E-mail software on the sender's computer uses the second part to select a destination; e-mail software on the recipient's computer uses the first part to select a particular mailbox.

Mail Message Format

- Header
  - Identifies sender, recipient(s), memo contents
  - Lines of form

keyword : information

- Blank line
- Body
  - Contains text of message

Example E-mail Header Fields

Keyword	Meaning
From	Sender's address
To	Recipients' addresses
Cc	Addresses for carbon copies
Date	Date on which message was sent
Subject	Topic of the message
Reply-To	Address to which reply should go
X-Charset	Character set used (usually ASCII)
X-Mailer	Mail software used to send the message
X-Sender	Duplicate of sender's address
X-Face	Encoded image of the sender's face

- Most header lines optional

Extending E-mail

- Original e-mail
  - Message restricted to ASCII text
- Users desire to send
  - Image files
  - Audio clips
  - Compiled (binary) programs
- Solution
  - *Multi-purpose Internet Mail Extensions (MIME)*

MIME

- Allows transmission of
  - Binary data
  - Multimedia files (video/audio clips)
  - Multiple types in single message
  - Mixed formats
- Backward compatible

MIME Encoding

- Sender
  - Inserts additional header lines
  - Encodes binary data in (printable) ASCII
- Sent like standard message
- Receiver
  - Interprets header lines
  - Extracts and decodes parts
- Separate standards for content and encoding

Example Of MIME

- Header lines added

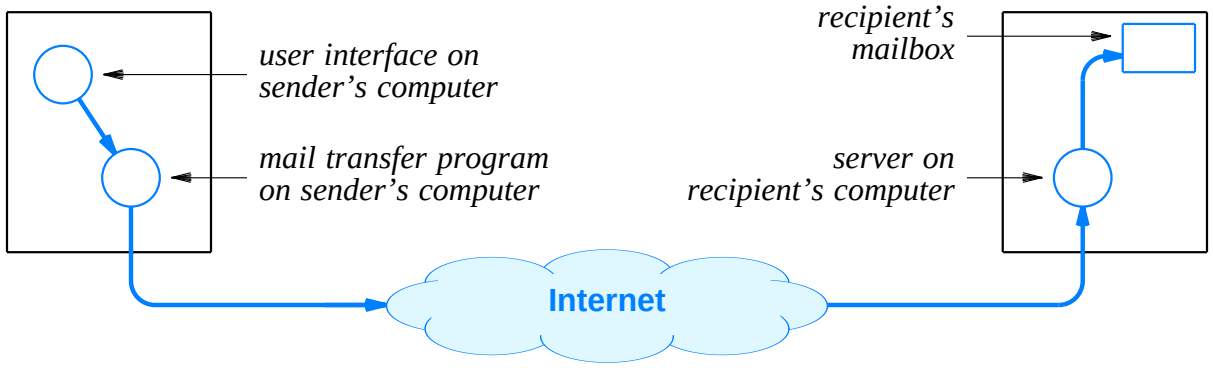
```
MIME-Version: 1.0
Content-Type: Multipart/Mixed; Boundary=xxxsep
```

- Specifies
  - Using MIME version *1.0*
  - Line *xxxsep* appears before each message part

Mail Transfer

- Protocol is *Simple Mail Transfer Protocol (SMTP)*
- Runs over TCP
- Used between
  - Mail transfer program on sender's computer
  - Mail server on recipient's computer
- Specifies how
  - Client interacts with server
  - Recipients specified
  - Message is transferred

Illustration Of Mail Transfer



- Server
  - Required to receive mail
  - Places message in user's mailbox

Terminology

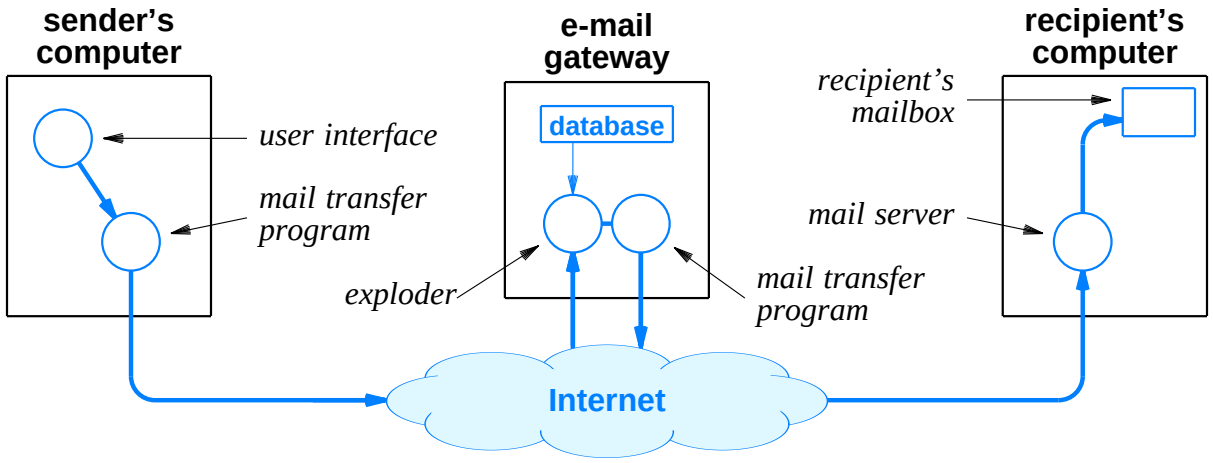
- *Mail exploder*
  - Program
  - Accepts incoming message
  - Delivers to multiple recipients
- *Mailing list*
  - Database
  - Used by exploder
- *Mail gateway*
  - Connects two mail systems

Illustration Of A Mailing List

List	Contents
friends	Joe@foo.com, Jill@bar.gov, Tim@StateU.edu Mary@acollege.edu, Hank@nonexist.com,
customers	george@xyz.com, VP_Marketing@news.com
bball-interest	hank@none.com, Linda_S_Smith@there.com, John_Q_Public@foobar.com, Connie@foo.edu

- Separate permissions for
  - Mailing to list
  - Adding/deleting members
    - *Public* – anyone can join
    - *Private* – access restricted by owner

Illustration Of A Mail Gateway

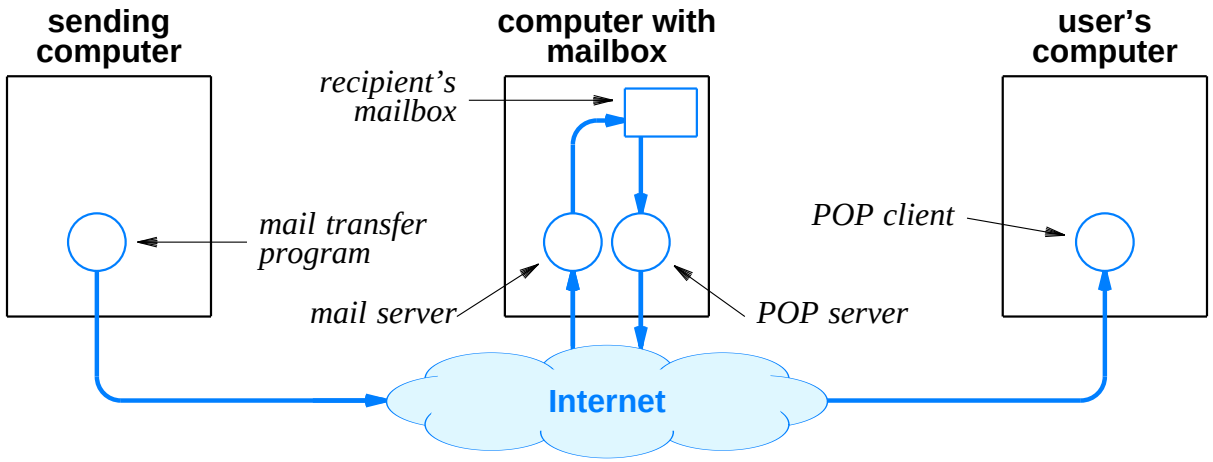


- Can connect two
  - Heterogeneous systems
  - Internet to non-Internet

Computers Without Mail Servers

- Typically
  - Small, personal computer
  - Not continuously connected to Internet
- To receive e-mail, user must
  - Establish mailbox on large computer
  - Access mailbox as necessary
- *Post Office Protocol (POP)* used

Illustration Of POP



- Current version named *POP3*

Remote Login

- Provide interactive access to computer from remote site
- Standard protocol is *TELNET*

TELNET

- Text-oriented interface
- User
  - Invokes client
  - Specifies remote computer
- Client
  - Forms TCP connection to server
  - Passes keystrokes over connection
  - Displays output on screen

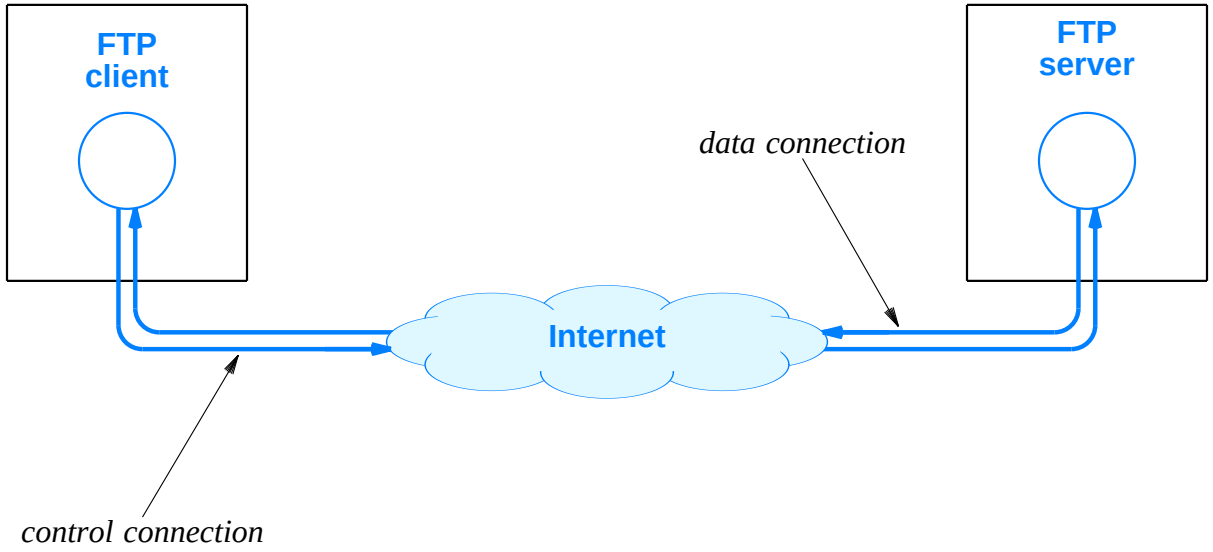
File Transfer

- Complete file copy
- Major protocol is *File Transfer Protocol (FTP)*
  - Uses TCP
  - Supports binary or text transfers
  - Large set of commands
  - Until 1995 was major source of packets in Internet

FTP Paradigm

- Command-line interface
- User
  - Forms TCP connection to server (called *control connection*)
  - Logs in
  - Enters commands to list directories, transfer files
- Server
  - Establishes new TCP connection for each transfer

Illustration Of TCP Connections During An FTP File Transfer



- Two TCP connections used

Summary

- Applications use client-server paradigm for interaction
- Client
  - Arbitrary application
  - Actively initiates communication
  - Must know server's
    * IP address
    * Protocol port number

Summary
(continued)

- Server
  - Specialized program
  - Runs forever
  - Usually offers one service
  - Passively waits for clients
  - Can handle multiple clients simultaneously

Summary
(continued)

- Socket API
  - Standardized
  - Specifies interface between applications and protocol software
- Socket
  - Operating system abstraction
  - Created dynamically
  - Used by clients and servers

Summary
(continued)

- Domain Name System
  - Maps name to IP address
  - Uses on-line servers
  - Uses caching for efficiency
- Two e-mail transfer protocols
  - SMTP
  - POP3

Summary
(continued)

- Remote login
  - Remote, interactive use
  - Protocol is *TELNET*
- File transfer
  - Copy of entire file
  - Protocol is FTP