Chat

Issues and Ideas for Service Design

Refs: RFC 1459 (IRC)

Service Design Issues

- Pretend we are about to design a chat system.
- We will look at a number of questions that would need to be answered during the design process.
- We will look at some possible system architectures.

Multi-user Chat Systems

Functional Issues

- Message types.
- Message destinations (one vs. many groups)
- Scalability (how many users can be supported)
- Reliability?
- Security
  - authentication
  - authorization
  - privacy

Message Types

- Some options:
  - text only
  - audio
  - images
  - anything (MIME)?

Scalability

- How large a group do we want to support?
- How many groups?
- What kind of service architecture will provide efficient message delivery?
- What kind of service architecture will allow the system to support *many* users/groups?

Message Destinations

- Each message goes to a group (multi-user chat).
  - Can we also send to individuals?
  - Should we support more than one group?
    - Are groups dynamic or static?
    - What happens when there is nobody in a group?
    - Can groups communicate?
    - Can groups merge or split?

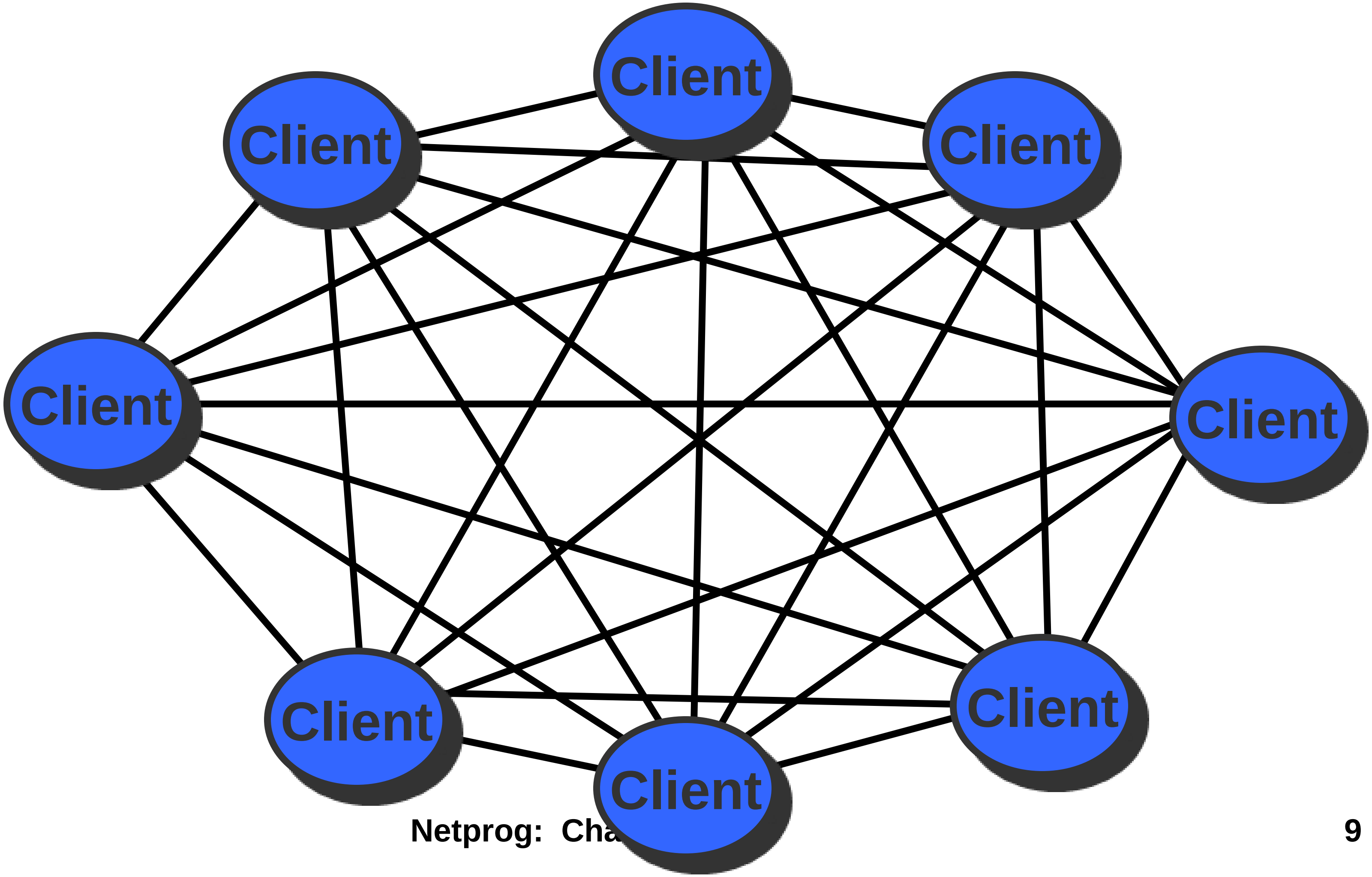
Reliability

- Does a user need to know (reliably) all the other users that receive a message?
- What happens if a message is lost?
  - resend? application level or at user level?
- What happens when a user quits?
  - Does everyone else need to know?

Security

- Authentication: do we need to know who each user is?
- Authorization: do some users have more privileges than others?
- Privacy:
  - Do messages need to be secure?
  - Do we need to make sure messages cannot be forged?

Peer-to-Peer Service Architecture

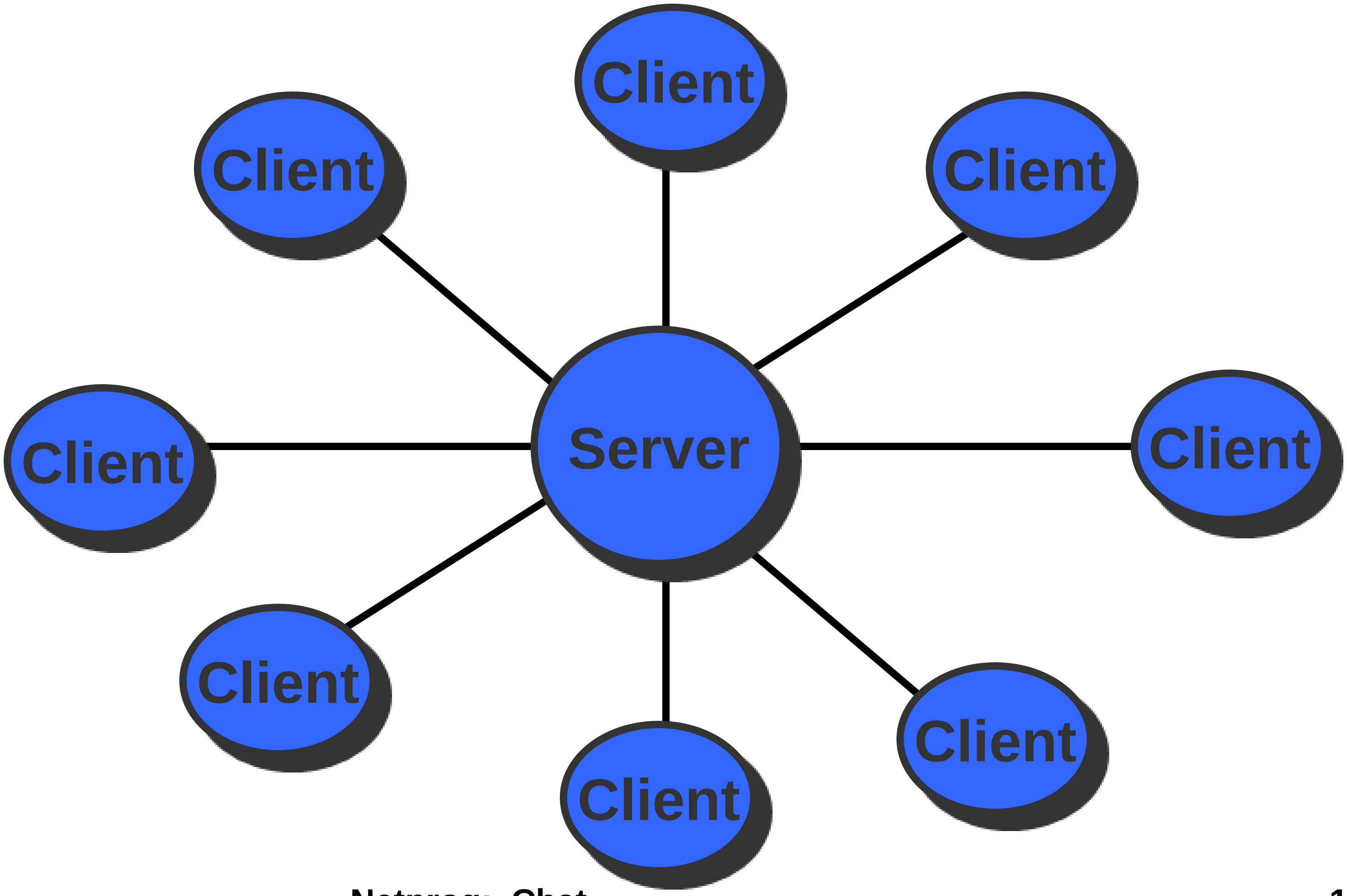


Peer-to-Peer Service Architecture (cont.)

Each client talks to many other clients.

- Who’s on first? Is there a well known address for the service?
- How many peers can we keep track of?
- If 2 peers (clients) are on the same machine, do we need to send a message to the machine twice?

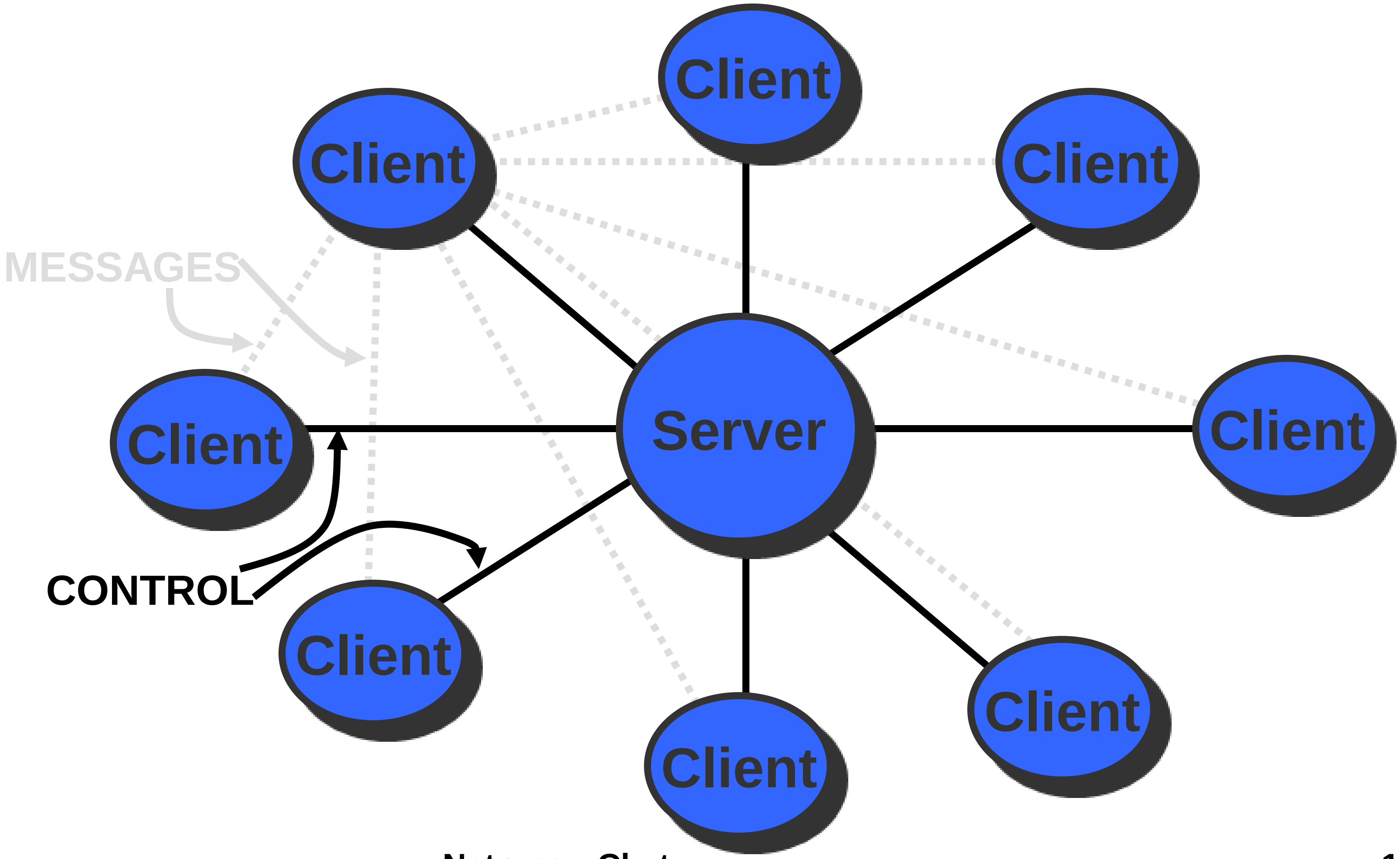
Client/Server



Client/Server

- Server is well known.
- Life is easier for clients - don’t need to know about all other clients.
- Limited number of clients?
- Security is centralized.
- Server might get overloaded?

Hybrid Possibility



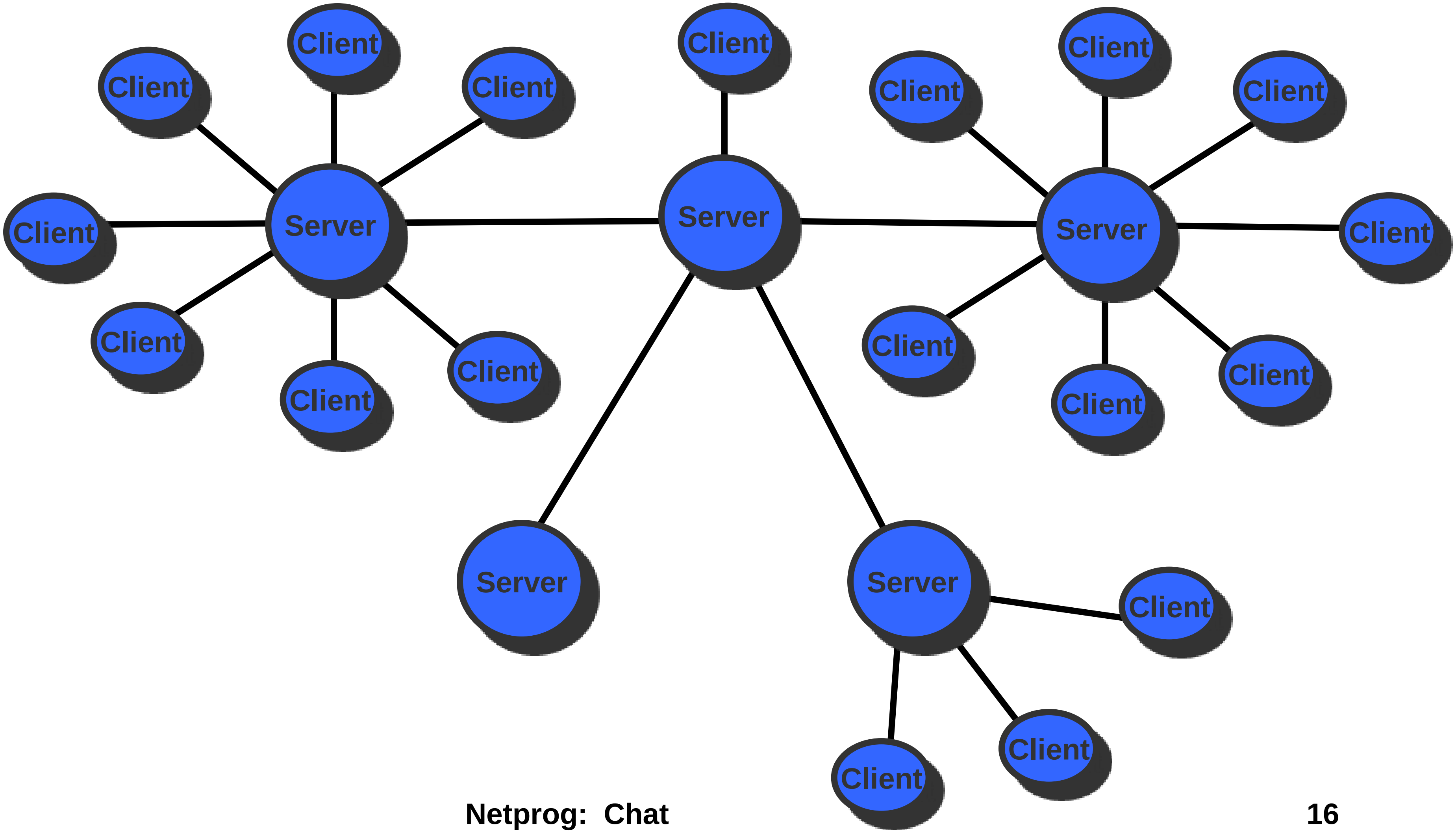
Hybrid

- Clients connect to server and gather control information:
  - List of other clients.
  - List of chat groups.
- Messages are sent directly (not through server).
  - Could use connectionless protocol (UDP or transaction based TCP).

Internet Relay Chat

- IRC is a widely used multi-user chat system.
  - Supports many chat groups (channels).
  - Extensive administrative controls.
  - Distributed service architecture.
  - Still in use today, although WWW based chat is now more common.

IRC Architecture



Server Topology

- Servers are connected in a spanning tree
  - Single path between any 2 servers.
  - New servers can be added dynamically
    - support for preventing cycles in the server graph.
- A collection of servers operates as a unified system, users can view the system as a simple client/server system.

Server Databases

- Each server keeps track of
  - all other servers
  - all users (*yes, really all users!*)
  - all channels (chat groups)
- Each time this information changes, the change is propagated to all participating servers.

Clients

- A client connects to the system by establishing a TCP connection to any server.
- The client registers by sending:
  - (optional) password command
  - a nickname command
  - a username command.

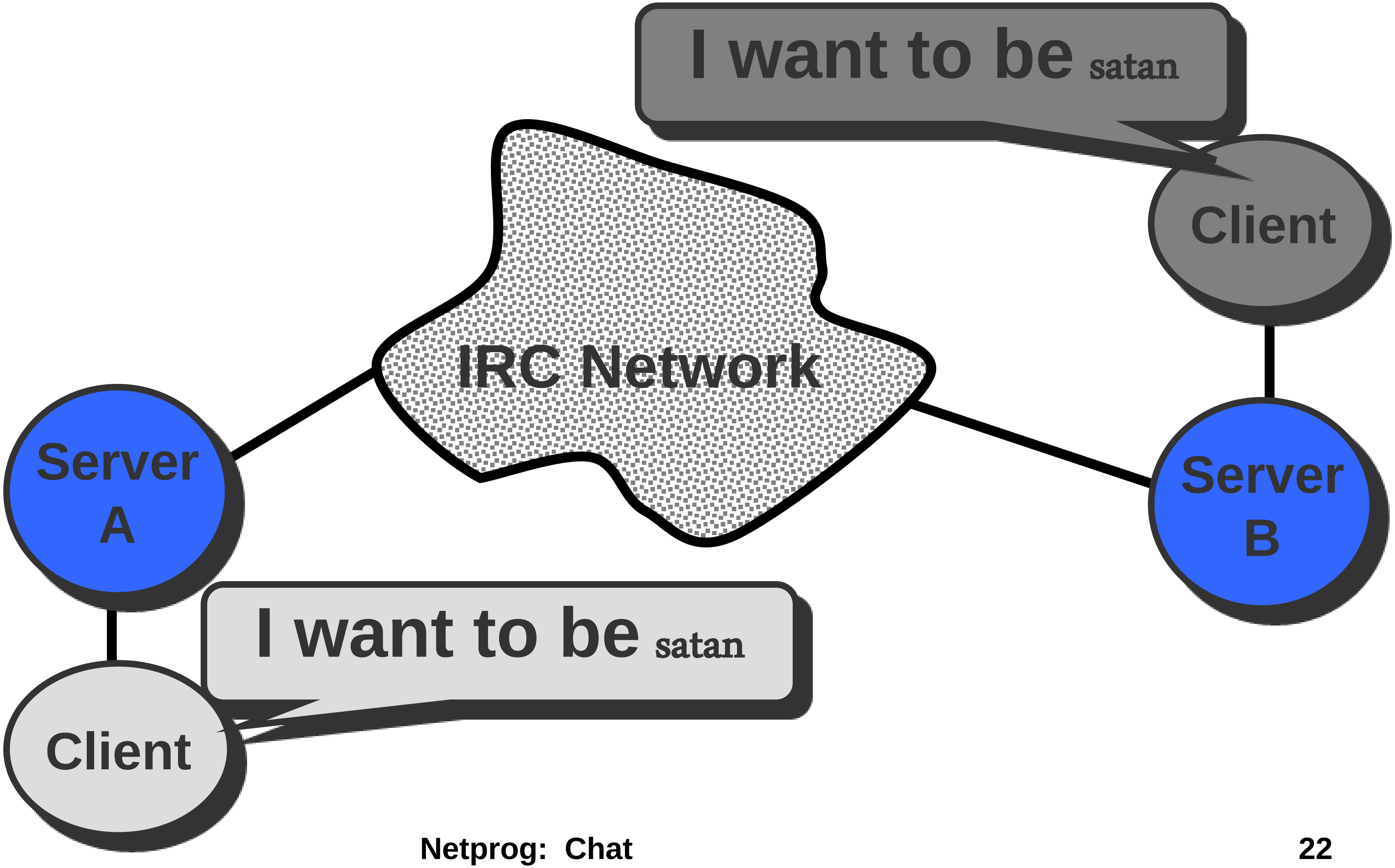
Nicknames and user names

- A nickname is a user supplied identifier that will accompany any messages sent.
  - Wizard, kilroy, gargoyle, death_star, gumby
- The username could be faked, some implementations use RFC931 lookup to check it.
- Users can find out the username associated with a nickname.

Collisions

- If a client requests a nickname that is already in use, the server will reject it.
- If 2 clients ask for the same nickname on 2 different servers, it is possible that neither server initially knows about the other.
- In this case both requests for the nickname are rejected.

Nickname Collision



Nickname Propagation

- The command used to specify a nickname is forwarded from the server to all other servers (using the spanning tree topology).
- The command is the same, but extra information is added by the original server:
  - server name connected to client with nickname.
  - Hop count* from the server connected to the client.

  *hop count is IRC server count (not IP!)

Channels

- 2 kinds of channels
  – local to a server - start with ‘&’ character
  – global, span the entire IRC network -start with the ‘#’ character.
- Users can JOIN or PART from a channel.
- A channel is created when the first user JOINS, and destroyed when the last user PARTS.

Channel Operators

- The user that creates a channel becomes the channel operator and can set various channel properties (modes):
  - invite-only
  - moderated
  - private
  - secret

Channel Op commands

- A Channel Op can:
  - give away channel op privileges
  - set channel topic (just a string)
  - kick users out of the channel.
  - Invite a client to a channel
  - change channel mode

Messages

- All messages are text.
- A message can be sent to nicknames, channels, hosts or servers.
- There are two commands for sending messages:
  - PRIVMSG: response provided.
  - NOTICE: no response (reply) generated. Avoids loops when clients are automatons

Other Stuff

- Special class of users known as Operators.
  - Operators can remove users!
- Servers can be told to connect to another server (operators create the spanning tree).
- The tree can be split if a node or network fails - there are commands for dealing with this.

Problems

- Scalability: works well with quite a large IRC network, but needs to be changed to get much bigger.
  - Currently every server needs to know about every other server, every channel and every user.
  - Path length is determined by operators, an optimal tree could be generated automatically.

Problems

- Supporting a cyclic network (instead of a tree) could minimize disruptions.
- Need a better scheme for nicknames, too many collisions (everyone wants to be satan!)
- Current protocol means that each server must assume neighbor server is correct. *Bad guys* could screw things up.

CS4254

Computer Network Architecture and Programming

Dr. Ayman A. Abdel-Hamid

Computer Science Department

Virginia Tech

Sockets Programming Introduction

Outline

- •Sockets API and abstraction
- •Simple Daytime client
- •Wrapper functions
- •Simple Daytime Server

Sockets API

API is Application Programming Interface

•Sockets API defines interface between application and

•transport layer

➢two processes communicate by sending data into socket, reading data out of socket

•Socket interface gives a file system like abstraction to the capabilities of the network

•Each transport protocol offers a set of services

➢The socket API provides the abstraction to access these services

•The API defines function calls to create, close, read and write to/from a socket

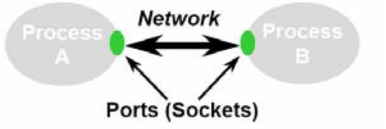
Sockets Abstraction

The *socket* is the basic abstraction for network communication in the socket API

➢Defines an endpoint of communication for a process

➢Operating system maintains information about the socket and its connection

➢Application references the socket for sends, receives, etc



Simple Daytime Client 1/5

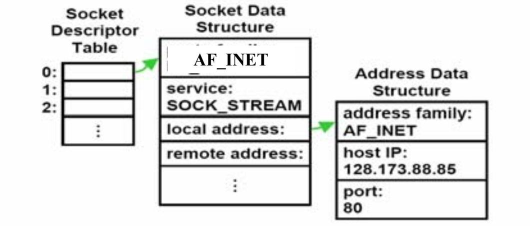
- Source code available from http://www.unpbook.com
- Read README file first!
- Source file is daytimetcpcli.c

- Include "unp.h"
  - Textbook's header file
  - Includes system headers needed by most network programs
  - Defines various constants such as MAXLINE
- Create TCP Socket
  - **sockfd = socket (AF_INET, SOCK_STREAM, 0)**
  - Returns a small integer descriptor used to identify socket
  - If returned value < 0 then error

Simple Daytime Client 2/5

Socket Descriptors

- Operating system maintains a set of socket descriptors for each process → Note that socket descriptors are shared by threads
- Three data structures
  - Socket descriptor table → Socket data structure →Address data structure



Simple Daytime Client 3/5

- Specify Server IP Address and Port
  - Fill an *Internet socket address structure* with server's IP address and port
  - Set entire structure to zero first using **bzero**
  - Set address family to AF_INET
  - Set port number to 13 (well-known port for daytime server on host supporting this service)
  - Set IP address to value specified as command line argument (argv[1])
  - IP address and port number must be in specific format
  - **htons** → host to network short
  - **inet_pton** → *presentation to numeric*, converts ASCII dotted-decimal command line argument (128.82.4.66) to proper format

Simple Daytime Client 4/5

- Establish connection with server
  - **Connect (sockfd, (SA *) &servaddr, sizeof(servaddr))**
  - Establish a TCP connection with server specified by socket address structure pointed to by second argument
  - Specify length of socket address structure as third argument
  - **SA** is #defined to be **struct sockaddr** in **unp.h**
- Read and Display server reply
  - Server reply normally a 26-byte string of the form

    Mon May 26 20:58:40 2003\r\n
  - TCP a *byte-stream* protocol, always code the **read** in a loop and terminate loop when **read** returns 0 (other end closed connection) or value less than 0 (error)

Simple Daytime Client 5/5

- Terminate program
  - ➢Exit terminates the program **exit (0)**
  - ➢Unix closes all open descriptors when a process terminates
  - ➢TCP socket closed
- Program protocol dependent on IPv4, will see later how to change to IPv6 and even make it protocol independent

Error Handling: Wrapper Functions

- Check every function call for error return
- In previous example, check for errors from **socket**, **inet_pton**, **connect**, **read**, and **fputs**
- When error occurs, call textbook functions **err_quit** and **err_sys** to print an error message and terminate the program
- Define wrapper functions in lib/wrapsock.c
- Unix **errno** value
  - ➢When an error occurs in a Unix function, global variable **errno** is set to a positive value indicating the type of error and the function normally returns -1
  - ➢**err_sys** function looks at **errno** and prints corresponding error message (e.g., connection timed out)

Simple Daytime Server 1/2

- Source code in daytimetcpsrv.c
- Create a TCP Socket
  - ➢Identical to client code
- Bind server well-known port to socket
  - ➢Fill an Internet socket address structure
  - ➢Call **Bind** (wrapper function) → local protocol address bound to socket
  - ➢Specify IP address as **INADDR_ANY**: accept client connection on any interface (if server has multiple interfaces)
- Convert socket to listening socket
  - ➢Socket becomes a listening socket on which incoming connections from clients will be accepted by the kernel
  - ➢**LISTENQ** (defined in unp.h) specifies the maximum number of client connections the kernel will queue for this listening descriptor

Simple Daytime Server 2/2

- Accept client connection, send reply
  - ➢Server is put to sleep (blocks) in the call to **accept**
  - ➢After connection accepted, the call returns and the return value is a new descriptor called the *connected descriptor*
  - ➢New descriptor used for communication with the new client
- Terminate connection
  - ➢Initiate a TCP connection termination sequence

➢Some Comments

- ➢Server handles one client at a time
- ➢If multiple client connections arrive at about the same time, kernel queues them up, up to some limit, and returns them to accept one at a time (An example of an iterative server, other options?)

IPv4 Socket Address Structure

```
struct  in_addr {
  in_addr_t   s_addr ;   // 32-bit, IPv4 network byte order (unsigned)
}

struct sockaddr_in {
    uint8_t             sin_len;   /*unsigned 8 bit integer*/
    sa_family_t         sin_family; /*AF_INET*/
    in_port_t           sin_ port ; /* 16 bit TCP or UDP port number */
    struct   in_addr   sin_addr; /* 32 bit IPv4 address */
    char                sin _zero[8];  /*unused*/
}
struct  sockaddr_in   servaddr;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

Generic Socket Address Structure

•*A socket address structure always passed by reference when passed as an argument to any socket function*

•How to declare the pointer that is passed?

•Define a generic socket address structure

```
struct sockaddr {
   uint8_t         sa_len;   /*unsigned 8 bit integer*/
   sa_family_t   sa_family; /*AF_INET*/
   char             sa_data[14] ; /* protocol specific address*/
}
Prototype for bind
int bind (int, struct sockaddr * socklen_t)

struct sockaddr_in serv;
bind (sockfd, (struct sockaddr *) &serv,sizeof(serv));
Or #define SA struct sockaddr → bind (sockfd, (SA *) &serv, sizeof(serv));
```

Value-Result Arguments

•Length of socket passed as an argument
•Method by which length is passed depends on which direction the structure is being passed (from process to kernel, or vice versa)

•Value-only: ***bind***, ***connect***, ***sendto*** (from process to kernel)
•Value-Result: ***accept***, ***recvfrom***, ***getsockname***, ***getpeername*** (from kernel to process, pass a pointer to an integer containing size)

➢*Tells process how much information kernel actually stored*

```
struct sockaddr_in     clientaddr ;
socklen_t                 len;
int                          listenfd, connectfd;

len = sizeof (clientaddr);
connectfd = accept (listenfd,  (SA *) &clientaddr,  &len);
```

Byte Ordering Functions 1/4

•Two ways to store 2 bytes (16-bit integer) in memory

➢Low-order byte at starting address → little-endian byte order

➢High-order byte at starting address → big-endian byte order

•*in a big-endian computer* → store 4F52

➢Stored as 4F52 → 4F is stored at storage address 1000, 52 will be at address 1001, for example

•*In a little-endian system* → store 4F52

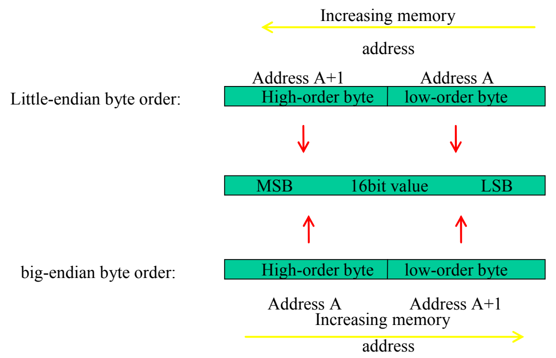
➢it would be stored as 524F (52 at address 1000, 4F at 1001)

•Byte order used by a given system known as *host byte order*

•Network programmers use *network byte order*

•Internet protocol uses big-endian byte ordering for integers (port number and IP address)

Byte Ordering Functions 2/4



Byte Ordering Functions 3/4

```
#include      "unp.h"
int main(int argc, char **argv)
{
              union {
                short  s;
                char   c[sizeof(short)];
            } un;

              un.s = 0x0102;
              printf("%s: ", CPU_VENDOR_OS);
              if (sizeof(short) == 2) {
                            if (un.c[0] == 1 && un.c[1] == 2)
                                          printf("big-endian\n");
                            else if (un.c[0] == 2 && un.c[1] == 1)
                                          printf("little-endian\n");
                            else
                                          printf("unknown\n");
              } else
                            printf("sizeof(short) = %d\n", sizeof(short));

              exit(0);
}
```

•Sample program to figure out little-endian or big-endian machine

•Source code in byteorder.c

Byte Ordering Functions 4/4

•To convert between byte orders
- ➢Return value in network byte order
  - ✓htons (s for short word 2 bytes)
  - ✓htonl (l for long word 4 bytes)
- ➢Return value in host byte order
  - ✓ntohs
  - ✓ntohl

•Must call appropriate function to convert between host and network byte order

•On systems that have the same ordering as the Internet protocols, four functions usually defined as null macros

```
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port           = htons(13);
```

Byte Manipulation Functions

#include <strings.h>

void bzero (void *dest, size_t nbytes);

// sets specified number of bytes to 0 in the destination

void bcopy (const void *src,void * dest, size_t nbytes);

// moves specified number of bytes from source to destination

void bcmp (const void *ptr1, const void *ptr2,size_t nbytes)

//compares two arbitrary byte strings, return value is zero if two byte strings are identical, otherwise, nonzero

Address Conversion Functions 1/2

Convert an IPv4 address from a dotted-decimal string “206.168.112.96” to a 32-bit network byte order binary value

#include <arpa/inet.h>
int inet_aton (const char* strptr, struct in_addr *addrptr);
// return 1 if string was valid, 0 on error. Address stored in *addrptr

in_addr_t inet_addr (const char * strptr);
// returns 32 bit binary network byte order IPv4 address, currently deprecated

char * inet_nota (struct in_addr inaddr);
//returns pointer to dotted-decimal string

Address Conversion Functions 2/2

To handle both IPv4 and IPv6 addresses
#include <arpa/inet.h>
int inet_pton (int family, const char* strptr, void *addrptr);
// return 1 if OK, 0 on error. 0 if not a valid presentation, -1 on error, Address stored in *addrptr

Const char * inet_ntop (int family, const void* addrptr, char *strptr, size_t len);
// return pointer to result if OK, NULL on error

```
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
                err_quit("inet_pton error for %s", argv[1]);
```

```
ptr = inet_ntop (AF_INET,&addr.sin_addr,str,sizeof(str));
```

Reading and Writing Functions 1/2

- int send (int socket, char *message, int msg_len, int flags) (TCP)
- int sendto (int socket, void *msg, int len, int flags, struct sockaddr * *to*, int *tolen*); (UDP)
- int write(int socket, void *msg, int len); /* TCP */
- int recv (int socket, char *buffer, int buf_len, int flags) (TCP)
- int recvfrom(int socket, void *msg, int len, int flags, struct sockaddr *from, int *fromlen); (UDP)
- int read(int socket, void *msg, int len); (TCP)

Reading and Writing Functions 2/2

•Stream sockets (TCP sockets) exhibit a behavior with read and write that differs from normal file I/O
•A read or write on a stream socket might input or output fewer bytes than requested (not an error)

- **readn** function
- **writen** function
- **readline** function

Unix Domain Protocols

when client and server are on the same host

- Unix domain socket address structure
- Socket functions
- Stream client-server
- Datagram client-server
- Passing descriptors
- Receiving sender credentials

Unix Domain Socket Address Structure

```
#include <sys/un.h>
struct sockaddr_un {
   uint8_t     sun_len;
   sa_family_t   sun_family;   /* AF_LOCAL */;
   char     sun_path[104];   /* null-terminated pathname */
};
```

Socket Functions

#include <sys/socket.h>

int socketpair (int family; int type, int protocol, int sockfd[2]);

returns: nonzero if OK, -1 on error

creates two sockets that are connected together

family: AF_LOCAL, protocol: 0, type: SOCK_STREAM or SOCK_DGRAM

- All socket functions for TCP and UDP sockets can be used, but several restrictions apply.

Passing Descriptors between Related/Unrelated Processes

- Create a Unix domain socket, either stream or datagram
- One process opens a descriptor
- The sending process builds a msghdr structure containing the descriptor to be passed, calls sendmsg
- The receiving process calls recvmsg

Receiving Sender Credentials through a Unix domain socket

```
Include <sys/ucred.h>
Struct fcred{
        uid_t    fc_ruid;                    /* real user ID */
        gid_t    fc_rgid;                    /* real group ID */
        char     fc_login[MAXLOGNAME];       /* setlogin() name */
        uid_t    fc_uid;                     /* effective user ID */
        fc_ngroups;                          /* number of group */
        gid_t    fc_groups[NGROUPS];         /* supplementary group IDs */
};
#define  fc_gid    fc_groups[0]    /* effective group ID */
```

Lecture 8:Advanced Sockets

References for Lecture 8:

1) Unix Network Programming, W.R. Stevens, 1990,Prentice-Hall, Chapter 6.
2) Unix Network Programming, W.R. Stevens, 1998,Prentice-Hall, Volume 1, Chapter 3-4.

It is also possible to obtain the well-known address of a service or the name of a service on a specialized port.

```
#include <netdb.h>
struct servent *getservbyname(const char *servname, const char *portname);
-- Returns NULL on error. servname = “ftp” for example.
struct servent *getservbyport(int port, const char *portname);
-- returns NULL on error.
stuct servent{
        char    *s_name;      /* official server name*/
        char **s_aliases;    /* list of aliases */
        int        s_port;        /*port number – network byte order */
        char      s_proto;      /* protocol to use */
};
```

Socket Options

Like fcntl( ) for controlling file options, and msgctl/semctl/shmctl( ) for controlling message queue/semaphore/ shared memeory options, the following two functions are for controlling socket options.

```
#include <sys/socket.h>
int getsocketopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen);
int setsocketopt(int sockfd, int level, int optname, const void *optval, socklen_t    optlen);
-- returns 0 if OK, -1 on error.
```

sockfd – an open socket descriptor;
level – who gets/sets the option: socket code, TCP/IP or XNS.
optname – predefined option name.
optval – pointer to the value to set or get. Most option values are integer type.
optlen – length of the option (size of the value), value-result for getsockopt( ); only useful for IP_OPTIONS.

An option can be either a flag (on/off) or a value that can be set or retrieved. Some options can find their places in TCP header or IP header such as TCP_MAXSEG and IP_TOS; some cannot such as TCP_NODELAY and SO_MTU. Flag options use 0 for off and a nonzero value for on. If *optval* has a value of zero after a call to getsockopt( ), that option is currently off. See Figure 6.14 [Stevens ed1:p314].

For TCP/IP, possible levels are:

SOL_SOCKET	– for socket option,
IPPROTO_IP	– for Ipv 4 option,
IPPROTO_Ipv6	– for Ipv6 option,
IPPROTO_ICMPv6	– for ICMP version6 option,
IPPROTO_TCP	– for TCP option,

Socket level optons include:

SO_BROADCAST –f– enable/disable broadcasting. Datagrams only.

SO_DEBUG –f– used for TCP connection to return detailed information on packets

SO_ERROR –f– returns the "so_errno" (defined in <sys/socketvar.h>) value for a socket error. Same value is also stored in Unix errono variable.

SO_KEEPALIVE –f– when no data has been transmitted over a socket for 2 hours, a keepalive probe is sent. If no response is received after several probes are sent, the connection is closed. Used to detect abnomal termination.

SO_LINGER –v– determines whether any unsent data should be sent or discarded when a socket is closed. Close may block until data is sent. Most value options are integer type, but this one use

```
struct    <sys/socket.h>
struct linger { int l_onoff;      /* zero=off, nonzero=on */
                Int l_linger;     /* linger time in seconds */   }
```

SO_OOBINLINE –f– specifies that OOB data also be placed int eh normal input queue.

Ipv4 level options include:

IP_OPTIONS –v–set or fetch options in the IP header.

IP_TOS –v– specifies the type-of-service field in the IP header.

IP_TTL –v– set or fetch the TTL(time-to-live) field – maximum number of hopes.

TCP level options includes:

TCP_MAXSEG –v– returns the maximum segment size. The value is set when the connection is established.

TCP_KEEPALIVE –v– changes the keepalive interval for this connection.

TCP_NODELAY –f– prevents TCP for buffering data to create larger packets. Used for interactive application such as telnet.

#include < fcntl.h>

int fcntl(int *fd*, int *cmd*, int *arg*); /* See[Stvens ed 1: 41-43], here we only discuss socket-related *cmds**/

-- returns 0 if OK, -1 on error.

fd – an open socket descriptor;

cmd – operation to be performed on *fd.*

val – the value to set or get.

Cmd:

- **fcntl(fd, F_GETOWN / F_SETOWN, arg):** get or set the associated process number (*arg* > 0) or the associated process group number (*arg* <0) in order to receive SIGIO or SIGURG.. Only available for terminals and sockets.
- **fcntl(fd, F_GETFL / F_SETFL, FNDELAY / FASYNC):** set or get file flag bits FNDELAY or FASYNC. FNDELAY affects accept, connect, read, write, recv, send, sendto and recvfrom. FASYNC enables the receipt of SIGIO.

Question: How many ways to set a nonblocking socket?

Asynchronous I/O

Process can wait for the kernel to send signal SIGIO when a specified descriptor is ready for I/O. 3 things to do:

1) Establish a handler for SIGIO by calling signal(SIGIO, ???);
2) Set PID or PGID for the descriptor to receive SIGIO by calling fcntl(fd, F_SETOWN, getpid());
3) Enable asynchronous I/O by calling fcntl(fd, F_SETFL,FASYNC).

```
/*      Copy standard input to standard output.      */
#define  BUFFSIZE   4096
main()
{     int         n;
      char        buff[BUFFSIZE];

      while ( (n = read(0, buff, BUFFSIZE)) > 0) write(1, buff, n);
}
```

```
/*    Copy standard input to standard output, using asynchronous I/O.    */
#include<signal.h>
#include<fcntl.h>
#define  BUFFSIZE   4096
int   sigflag;
main()
{     int         n;
      char        buff[BUFFSIZE];
      int         sigio_func();
      signal(SIGIO, sigio_func);                  /*    Step 1: set up signal handler*/
      fcntl(0, F_SETOWN, getpid();                /*    Step 2: set descriptor's process ID*/
      fcntl(0, F_SETFL, FASYNC) ;                 /*    Step 3: Enable Asynchronous I/O*/
      for ( ; ; ) {
            sigblock(sigmask(SIGIO));             /*    block signal SIGIO to avoid race condition */
            while (sigflag == 0)    sigpause(0);   /* release signals when waiting for a signal.
                                                     Note the difference between pause() and sigpause(0)*/
            /* We're here if (sigflag != 0).    Also, we know that the SIGIO signal is currently blocked.*/
            if ( (n = read(0, buff, BUFFSIZE)) > 0) write(1, buff, n) ;      /* not a loop structure */
           else if (n == 0)      exit(0);         /* EOF */
            sigflag = 0;                           /* turn off our flag */
            sigsetmask(0);                        /* and reenable signals */
      }
}

int sigio_func( )
{     sigflag = 1;          /* just set flag and return */
      /* the 4.3BSD signal facilities leave this handler enabled for any further SIGIO signals. */
}
```

Select( )

When a server (or client) has multiple connections, it can be difficult to guess which clients(or servers) have written data on a socket. One approach, called **polling**, is to use nonblocking recv( ) and loop through all the connections. This is inefficient. Another approach, using **fork( ),** is to fork a child process for each connections. This is also inefficient. A better option is to wait on all the connections simultaneously. This can be done using select( ) function.

#include <sys/select.h>
#include <sys/time.h>
int select (int *maxfdp1*, fd_set **readset*, fd_set **writeset*, fd_set **exceptset*, const strut timeval **timeout*);
-- returns # of ready descriptors, 0 if timeout occurs, -1 on error.

maxfdp1 – the maximum descriptor to test +1, the possible number of descriptors to test, ≤256.
readset – used to check which connections have data read.
writeset – used to check which connections have space for more output.
exceptset – used to check which connections have exceptions, such as OOB data.
timeout – specifies how long to block waiting for ready connction
There are three options;
 = 0 means the call is nonblocking. Used for polling connections.
 > 0 means the call times out after this amount of time if there are no ready connection during this time.
NULL means the call blocks until a connection is ready for I/O.

The format of the timeval structure is:

```
struct timeval {
   long tv_sec;     /*seconds*/
   long tv_usec;    /*microseconds*/
};
```

select( ) is used to determine which socket are ready for reading, writing, or exception handling. Use NULL for any fd_set that doesn't need to be checked.

The fd_set detatype typically uses one bit per socket fd. The appropriate method for using fd_set is to zero out all the bits and then set each one that is to be tested. The select( ) call modifies the *readset*, *writeset*, and *exceptset* variables by clearing the bits that are not ready for I/O. The user then tests each bit to see which are set and processes the corresponding sockets.

Operations on fd_sets should be performed using the following macros:
void FD_ZERO(fd_set *fdset); **/*** clear all bits in fdset**/
void FD_SET(int *fd*, fd_set **dset*); **/*** turn on the bit for fd in fdset */
void FD_CLR(int *fd*, fd_set **fdset*); **/*** clear off the bits in fdset*/
int FD_ISSET(int *fd*, fd_set **fdset*); **/*** test the bit for fd in fdset */

See <sys/types.h> for definitions of sd_set and FD_XXX macros.

Example1:

```
int i, n;
fd_set    fdvar;

FD_ZERO(&fdvar);    /* initilize the Set --- all bits off */
FD_SET(1, &fdvar);    /* turn on bit for fd 1 */
FD_SET(4, &fdvar);    /* turn on bit for fd 4 */
FD_SET(5, &fdvar);    /* turn on bit for fd 5 */

If ((n=select(6, &fdvar, NULL, NULL, NULL))<0) printf("Something wrong!\n");
   /* only want to check the readset.*/

for (i=0, i<6, i++) if (FD_ISSET(i, &fdvar)>0) handle(i); /* fd i had data for read, call handle(i) */
```

Example2:

```
#include"unp.h"
void str_cli(FILE *fp, int sockfd)
{     int               maxfdp1;
      fd_set            rset;
      char      sendline[MAXLINE], recvline[MAXLINE];

      FD_ZERO(&rset);
      for ( ; ; ) {
            FD_SET(fileno(fp), &rset);
            FD_SET(sockfd, &rset);
            maxfdp1 = max(fileno(fp), sockfd) + 1;
            Select(maxfdp1, &rset, NULL, NULL, NULL);

            if (FD_ISSET(sockfd, &rset)) {      /* socket is readable */
                  if (Readline(sockfd, recvline, MAXLINE) == 0)
                        err_quit("str_cli: server terminated prematurely");
                  fputs(recvline, stdout); }

            if (FD_ISSET(fileno(fp), &rset)) {    /* input is readable */
                  if (Fgets(sendline, MAXLINE, fp) == NULL)
                        return;           /* all done */
                  writen(sockfd, sendline, strlen(sendline));   }
      }
}
```

Notes: select( ) can be used for a more accurate timer than sleep( ).
select() can be used for waiting for a connection request.

Socket-related Signals:

1) SIGIO :

- sindicates that a socket is ready for asynchronous I/O as we have discussed.
- need to specify process ID or process group ID to receive the signal.
- Need to enable asynchronous I/O.

2) SIGURG:

- indicates urgent data is coming due to 1)OOB data or 2) control status information.
- need to specify process group ID to receive the signal,e.g., fcntl(sd,F_SETOWN, -getpgid( )).
- Use flag=MSG_URG to send and receive the OOB data.
- If O_OOBINLINE is set, we must use STOCATMARK ioctl to read OOB data.

```
    setsockopt(sd, SOL_SOCKET, SO_OOBINLINE, &seton, sizeof(seton)); /*let seton=1*/
  if ((n=ioctl(sd,STOCATMARK, &start)>0) read(sd, buf, n);    /*OOB data is in buf with n bytes.*/
```

3) SIGPIPE:

- indicates socket, pipe, or FIFO can never be written to.
- Sent only to the associated process,

Internet Superserver --- inetd

How many typical network servers?

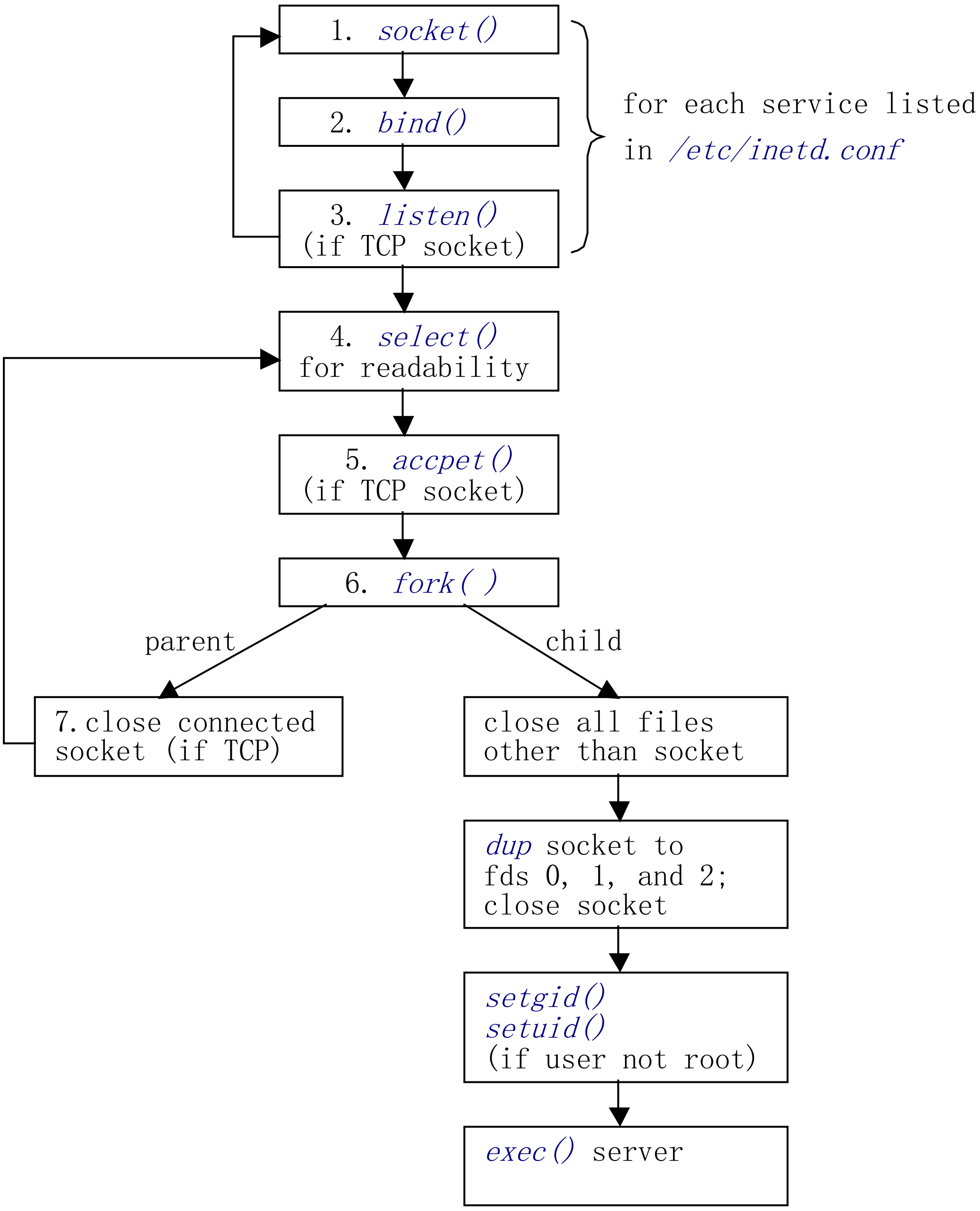
- telnet, ftp, tftp, remote login, remote shell
- started from /etc/rc
- did the same startup tasks: socket, bind, listen, accept, fork, …

How to use select( ) to combine them into one daemon?

- 4.3 BSD supersever: inetd
- reduce the number of processes
- simplify the writing of daemon processes since they have the same startup tasks and skeleton daemon tasks (see Lecture 1 for skeleton daemon).

Flow chart of inetd (version2: section 12.5 or version1:section 6.16)

1) read /etc/inetd.conf to create one socket for each service in the file.
2) read /etc/services to bind well-known port numbers to each service.
3) Listen() only for TCP.
4) Select() can be used for connect requests that arrives at the socket for reading.
5) If it is TCP request, call accept().
6) Fork a child process to handle the request
   6.1) close all files except socket
   6.2) dup2(sd,0), dup2(sd,1), and dup2(sd, 2).
   6.3) login program: a superuser can become any user. Must in the order of setgid() first and then setuid().
   6.4) exec() to execute server_program accordingly.
7) Parent goes up to accept next request without wait.



Steps performed by inetd

Writing Client/Server Programs in C Using Sockets (A Tutorial) Part I

Session 5958

Greg Granger
grgran@sas.com

SAS/C & C++ Support
SAS Institute
Cary, NC

Part I: Socket Programming Overview

- Sockets (to me)
- Networking (or what’s natural about natural logs)
- TCP/IP (and what it means to your life)
- More Sockets (we didn’t get enough the first time)

What is “Sockets”

- An Application Programming Interface (API) used for InterProcess Communications (IPC). [A well defined method of connecting two processes, locally or across a network]
- Protocol and Language Independent
- Often referred to as Berkeley Sockets or BSD Sockets

Connections and Associations

- In Socket terms a connections between two processes in called an association.
- An association can be abstractly defined as a 5-tuple which specifies the two processes and a method of communication. For example:
  - *{protocol, local-addr, local-process, foreign-addr, foreign-process}*
- A half-association is a single "side" of an association (a 3-tuple)
  - *{protocol, addr, process}*

Networking Terms

- packet - the smallest unit that can be transferred "through" the network by itself
- protocol - a set of rules and conventions between the communicating participants
- A collection of protocol layers is referred to as a "protocol suite", "protocol family" or "protocol stack". TCP/IP is one such protocol suite.

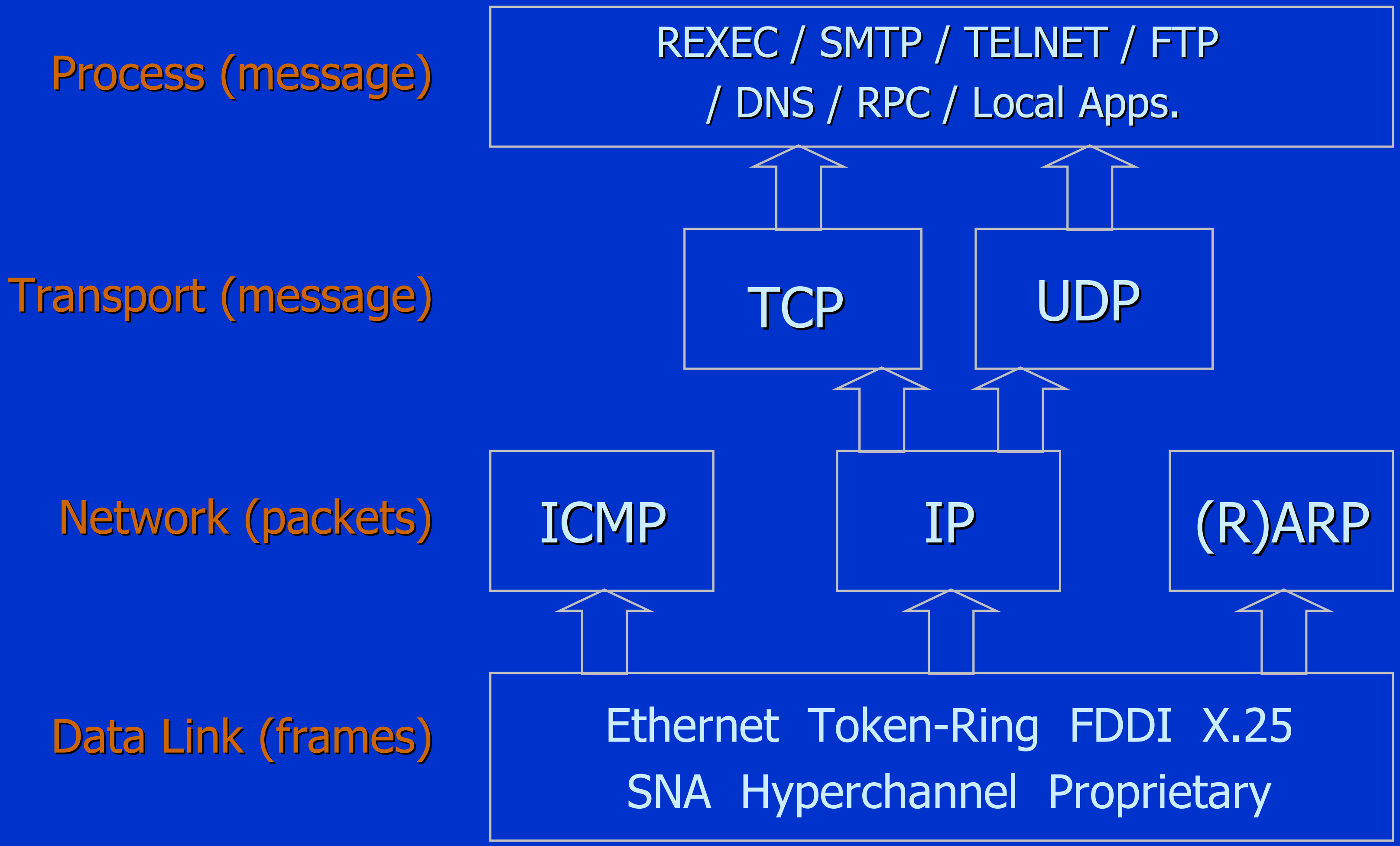
Introduction to TCP/IP

- What (the heck) is TCP/IP?
- Internet Protocol (IP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- TCP/IP Applications
- Name Resolution Processing
- TCP/IP Network Diagram

What is TCP/IP?

- Transmission Control Protocol/Internet Protocol
- A network protocol suite for interprocess communication
- The protocol of the Internet
- Open, nonproprietary
- Integrated into UNIX operating systems
- Many popular networking applications
    - telnet
    - X11 GUI
    - www
    - NFS (network file system)
    - SMTP (mail)
    - ftp (file transfer protocol)

TCP/IP Architectural Model



Process (message)

REXEC / SMTP / TELNET / FTP / DNS / RPC / Local Apps.

Transport (message)

TCP

UDP

Network (packets)

ICMP

IP

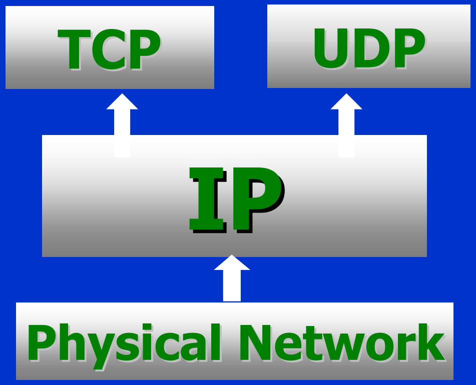
(R)ARP

Data Link (frames)

Ethernet Token-Ring FDDI X.25 SNA Hyperchannel Proprietary

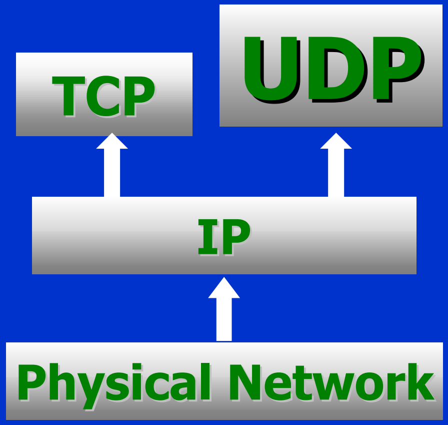
Internet Protocol (IP)

- Establishes a “virtual” network between hosts, independent of the underlying network topology
- Provides “routing” throughout the network, using IP addressing. For example: 149.173.70.9
- Features
  - Best-effort packet delivery
  - Connectionless (stateless)
  - Unreliable



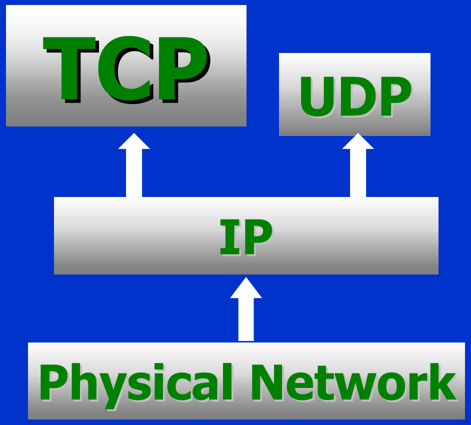
User Datagram Protocol (UDP)

- Application Interface to IP - Packet Oriented
- Establishes a "port", which allows IP to distinguish among processes running on the same host
- Features resemble IP semantics
  - Connectionless
  - Unreliable
  - Checksums (optional)



Transmission Control Protocol (TCP)

- Connection-oriented
- Stream Data Transfer
- Reliable
- Flow-Control
- Full-Duplex
- Suited for critical data transfer applications



The Importance of Ports

- Both the TCP and UDP protocols use 16 bit identifiers called ports to uniquely identify the processes involved in a socket.
- In UNIX the first 1024 ports for both protocols are called “well known ports” and are defined in the file /etc/services. Programs that bind to these ports require “root” access.
- These numbers are managed by the Internet Assigned Numbers Authority (IANA). A complete list of these assignments and more information about IANA can be found in RFC 1700

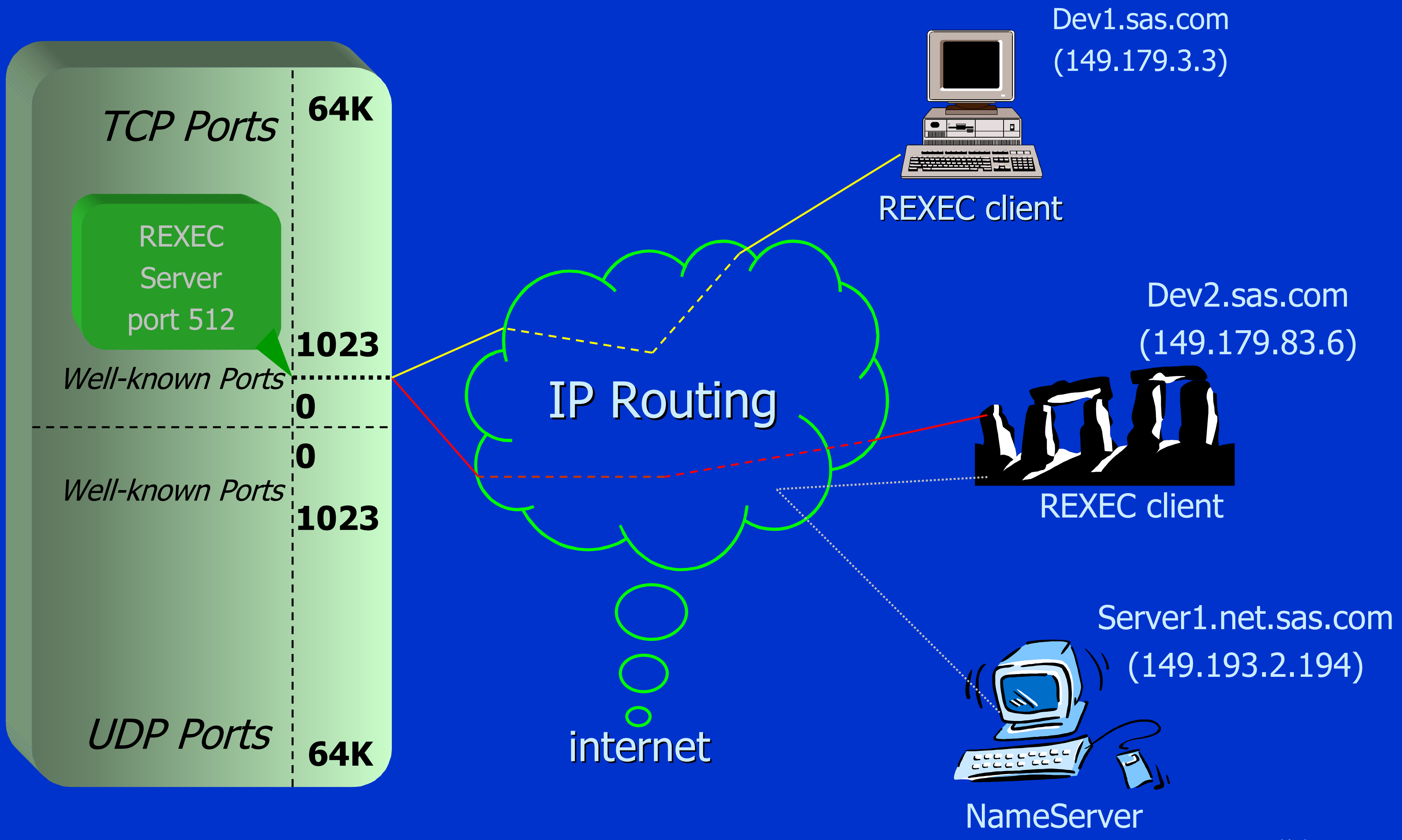
How stuff gets around (routing)

- TCP/IP packets are routed based on their destination IP address (ex: 10.24.2.123)
- Packets are passed from one network segment to another by machines called “routers” until the packet arrives at the network segment attached to the host with the destination IP address.
- Routers that act as gates to larger networks are called gateways.

Name Resolution Processing

- Associates an IP address to a “name” (hostname)
- Structured method of identifying hosts within an internet
- The Domain Name System (DNS) implements a hierarchical naming scheme which maps names like “mvs.sas.com” to an IP address
- DNS is implemented by a set of cooperating servers
- Machines that process DNS requests are called nameservers
- A set of library routines called “the resolver” provide the logic to query nameservers

TCP/UDP/IP Diagram



Back to Sockets

- Socket Definition and Components
- Socket Library Functions
- Primary Socket Header Files
- Sample Client/Server Dialog
- Ancillary Socket Topics
- Beyond Sockets

Definition and Components

- Socket - endpoint of communication
- Sockets - An application programming interface (API) for interprocess communication (IPC)
- Attributes:
  - Protocol Independent
  - Language Independent
  - Sockets implies (not requires) TCP/IP and C
- Socket and Connection Association
  - A local host can be identified by it’s protocol, IP address and port.
  - A connection adds the IP address & port of the remote host.

Socket Library Function

- System calls
  - startup / close
  - data transfer
  - options control
  - other
- Network configuration lookup
  - host address
  - ports for services
  - other
- Utility functions
  - data conversion
  - address manipulation
  - error handling

Primary Socket Calls

socket()	- create a new socket and return its descriptor
bind()	- associate a socket with a port and address
listen()	- establish queue for connection requests
accept()	- accept a connection request
connect()	- initiate a connection to a remote host
recv()	- receive data from a socket descriptor
send()	- send data to a socket descriptor
close()	- "one-way" close of a socket descriptor

Network Database Administration functions

- gethostbyname - given a hostname, returns a structure which specifies its DNS name(s) and IP address(es)
- getservbyname - given service name and protocol, returns a structure which specifies its name(s) and its port address
- gethostname - returns hostname of local host
- getservbyname, getservbyport, getservent
- getprotobyname, getprotobynumber, getprotobyent
- getnetbyname, getnetbyaddr, getnetent

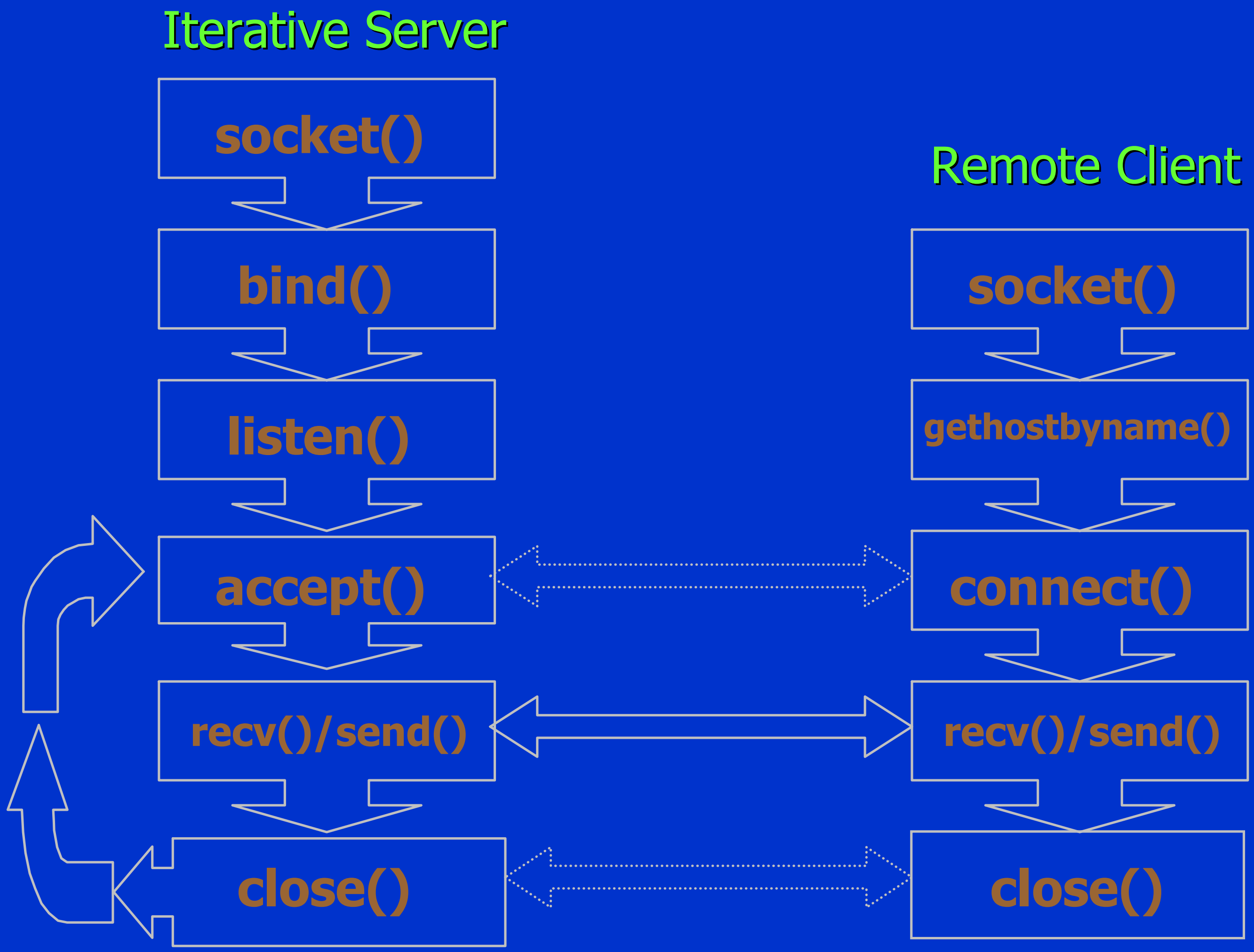
Socket Utility Functions

- ntohs/ntohl - convert short/long from network byte order (big endian) to host byte order
- htons/htonl - convert short/long from host byte order to network byte order
- inet_ntoa/inet_addr - convert 32-bit IP address (network byte order to/from a dotted decimal string)
- perror() - print error message (based on "errno") to stderr
- herror() - print error message for gethostbyname() to stderr (used with DNS)

Primary Header Files

- Include file sequence may affect processing (order is important!)
  - <sys/types.h> - prerequisite typedefs
  - <errno.h> - names for "errno" values (error numbers)
  - <sys/socket.h> - struct sockaddr; system prototypes and constants
  - <netdb.h.h> - network info lookup prototypes and structures
  - <netinet/in.h> - struct sockaddr_in; byte ordering macros
  - <arpa/inet.h> - utility function prototypes

Sample TCP Client / Server Session



Ancillary Socket Topics

- UDP versus TCP
- Controlling/managing socket characteristics
  - get/setsockopt() - keepalive, reuse, nodelay
  - fcntl() - async signals, blocking
  - ioctl() - file, socket, routing, interface options
- Blocking versus Non-blocking socket
- Signal based socket programming (SIGIO)
- Implementation specific functions

Design Considerations

- Data representation and conversion
- Server design alternatives
- Security Issues
- Portability Considerations

Data Representation

- Transport Protocols detail data exchange/movement; applications must interpret the data!
- Byte order affects data - not just addresses
- Text is often sent in ASCII, but ASCII versus EBCDIC is decided by the application-level protocol
- Structure alignment and floating point pose problems
- External Data Representation (XDR) can be used (even without RPC)

Server Design Alternatives

- Single Threaded
  - more complex code (must track multiple concurrent requests)
  - generally lower system overhead
  - crash of thread disables service
- Multi-Tasking
  - less complex code (written only for handling only one connection)
  - higher system overhead (each task requires it’s own process space)
  - highly crash resistant (one or more tasks can fail without losing service)
- [Multi-]Threaded
  - shares less complex code of Multi-Tasking model
  - system overhead between Single-Threaded and Multi-Tasking model
  - crash resistant (but one badly behaved thread ‘can’ crash service)

Security Considerations

- Socket semantics do NOT address security problems, such as:
  - IP and adapter addresses
  - Userid and passwords
  - data encryption
  - traces
- UNIX systems require "root" privilege when a program binds a "reserved" (<1024) port
- getpeername() returns the peer's port and IP-address: determine "privileged" peers and "trusted" hosts
- The Kerberos protocol provides password and data encryption, along with service authentication

Portability Considerations

- Limit applications to “standard” socket routines, BSD 4.x
- Implement a portable transport module
- Mainframe Environment - Distribute existing applications
  - API Programmer’s Reference - Details
  - SAS/C, C/370, Interlink, Open Connect, NSC
- OS/2 - REXX Sockets, Programmer’s Toolkit
- MS Windows Sockets 1.1 - 2 WINSOCK.DLL (http://www.stardust.com ftp.stardust.com:/pub/winsock)

Summary

- Basic networking and features of TCP/IP protocols
- Socket library organization
- Socket library coding techniques
- Awareness of more advanced topics

What's Next

- Session 5959 - Part II - Client/Server Application

Bibliography

- Internetworking with TCP/IP: Volumes I, II & III, Douglas Comer, Prentice Hall, 1991 (ISBN Vol I: 0134685059, Vol III: 0138487146)
- The Whole Internet User’s Guide & Catalog by Ed Kroll; O’Reilly & Associates
- UNIX Network Programming by W. Richard Stevens; Prentice Hall, 1990 (ISBN 0139498761)
- Socket API Programmer’s Reference
- UNIX “man” pages
- TCP/IP Illustrated: Volumes 1 & 2, W. Richard Stevens (v2 with Gary R. Wright); Addison-Wesley Publishing Company, 1994

core

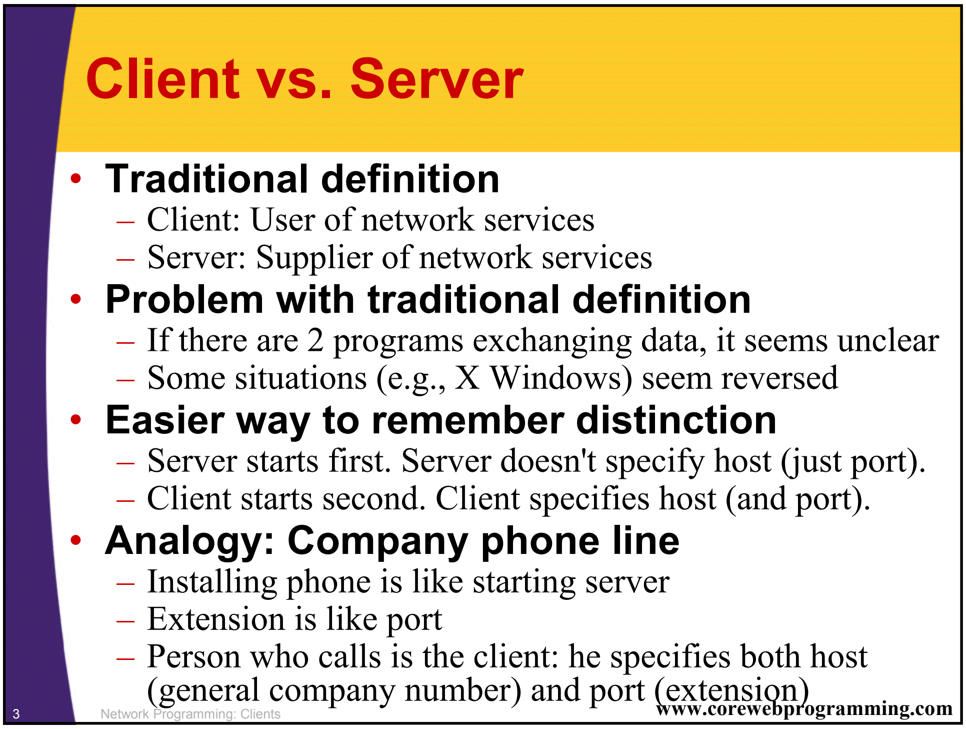
WEB

programming

Network Programming: Clients

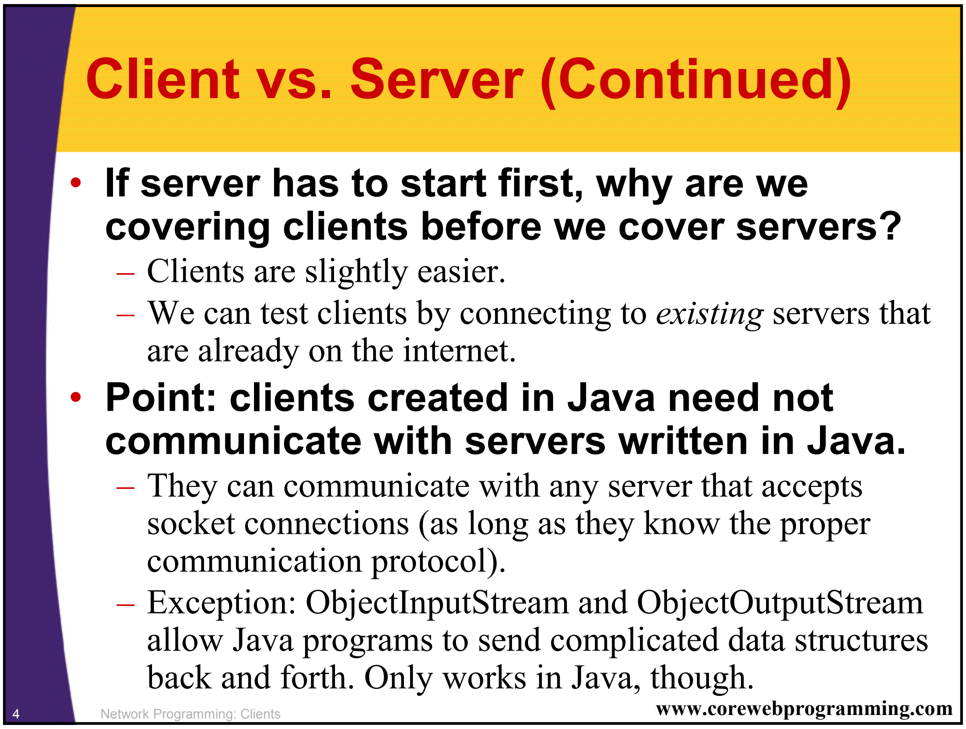
Agenda

- Creating sockets
- Implementing a generic network client
- Parsing data using StringTokenizer
- Retrieving files from an HTTP server
- Retrieving Web documents by using the URL class



Client vs. Server

- **Traditional definition**
  - Client: User of network services
  - Server: Supplier of network services
- **Problem with traditional definition**
  - If there are 2 programs exchanging data, it seems unclear
  - Some situations (e.g., X Windows) seem reversed
- **Easier way to remember distinction**
  - Server starts first. Server doesn't specify host (just port).
  - Client starts second. Client specifies host (and port).
- **Analogy: Company phone line**
  - Installing phone is like starting server
  - Extension is like port
  - Person who calls is the client: he specifies both host (general company number) and port (extension)



Client vs. Server (Continued)

- **If server has to start first, why are we covering clients before we cover servers?**
  - Clients are slightly easier.
  - We can test clients by connecting to *existing* servers that are already on the internet.
- **Point: clients created in Java need not communicate with servers written in Java.**
  - They can communicate with any server that accepts socket connections (as long as they know the proper communication protocol).
  - Exception: ObjectInputStream and ObjectOutputStream allow Java programs to send complicated data structures back and forth. Only works in Java, though.

Steps for Implementing a Client

1. **Create a Socket object**

```
Socket client = new Socket("hostname", portNumber);
```

2. **Create an output stream that can be used to send info to the Socket**

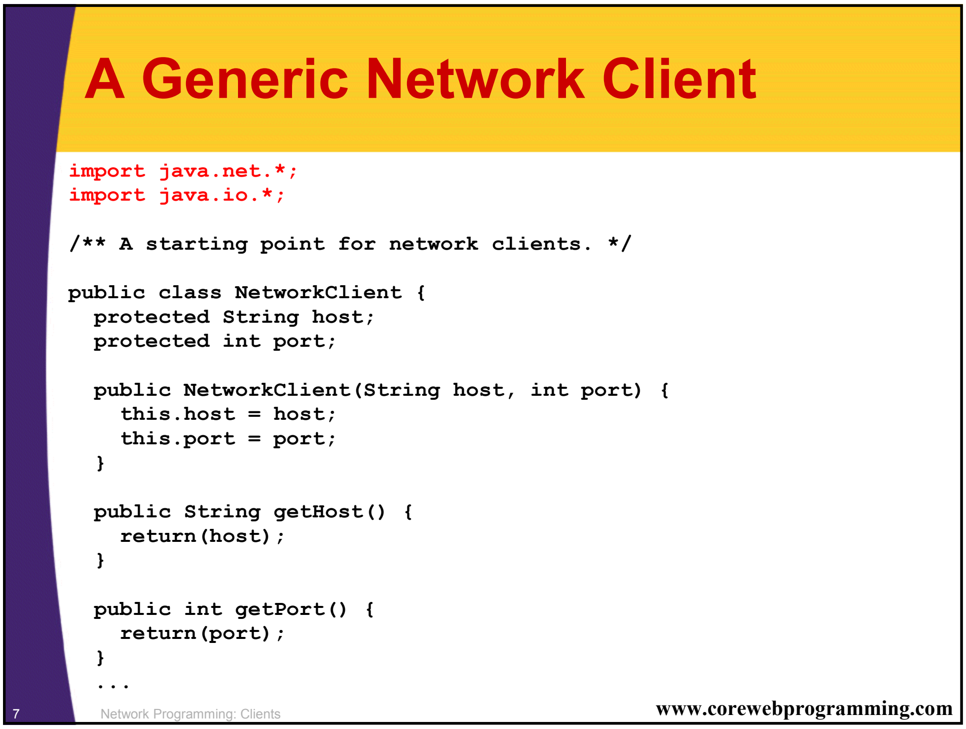
```
// Last arg of true means autoflush -- flush stream
// when println is called
PrintWriter out =
  new PrintWriter(client.getOutputStream(), true);
```

3. **Create an input stream to read the response from the server**

```
BufferedReader in =
  new BufferedReader
    (new InputStreamReader(client.getInputStream()));
```

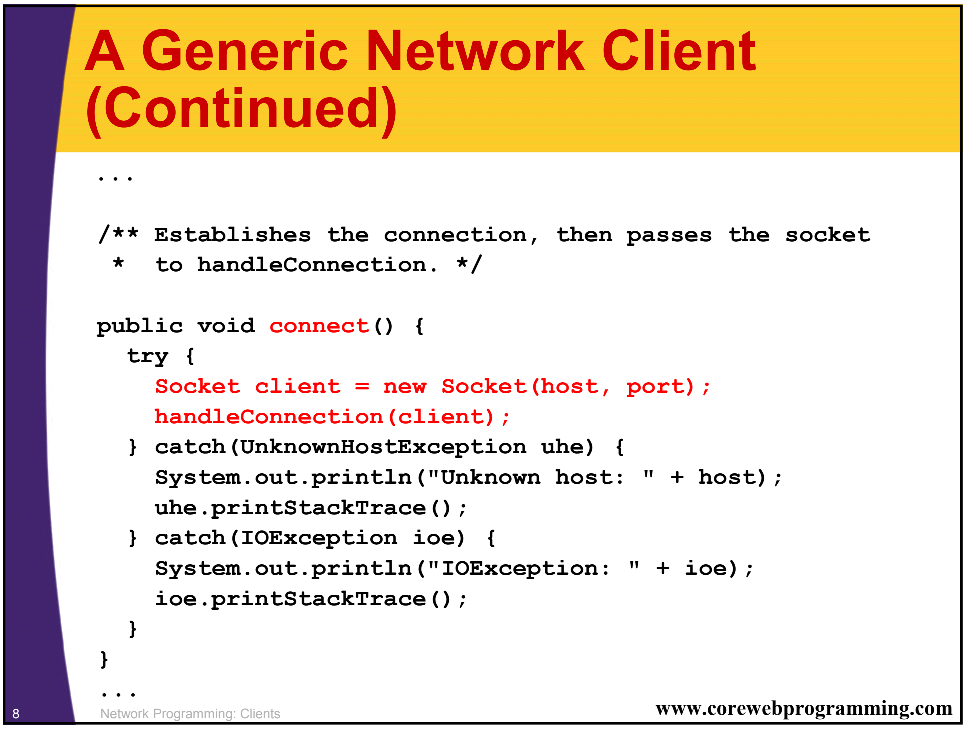
Steps for Implementing a Client (Continued)

4. **Do I/O with the input and output Streams**
   - For the output stream, PrintWriter, use print and println, similar to System.out.println
     - The main difference is that you can create PrintWriters for different Unicode characters sets, and you can't with PrintStream (the class of System.out).
   - For the input stream, BufferedReader, you can call read to get a single character or an array of characters, or call readLine to get a whole line
     - Note that readLine returns null if the connection was terminated (i.e. on EOF), but waits otherwise
5. **Close the socket when done**



A Generic Network Client

```
import java.net.*;
import java.io.*;

/** A starting point for network clients. */

public class NetworkClient {
  protected String host;
  protected int port;

  public NetworkClient(String host, int port) {
    this.host = host;
    this.port = port;
  }

  public String getHost() {
    return(host);
  }

  public int getPort() {
    return(port);
  }
  ...
```



A Generic Network Client (Continued)

```
  ...

  /** Establishes the connection, then passes the socket
   *  to handleConnection. */

  public void connect() {
    try {
      Socket client = new Socket(host, port);
      handleConnection(client);
    } catch(UnknownHostException uhe) {
      System.out.println("Unknown host: " + host);
      uhe.printStackTrace();
    } catch(IOException ioe) {
      System.out.println("IOException: " + ioe);
      ioe.printStackTrace();
    }
  }
  ...
```

A Generic Network Client (Continued)

```
  /** This is the method you will override when
   *  making a network client for your task.
   *  This default version sends a single line
   *  ("Generic Network Client") to the server,
   *  reads one line of response, prints it, then exits.
   */

  protected void handleConnection(Socket client)
      throws IOException {
    PrintWriter out =
      SocketUtil.getPrintWriter(client);
    BufferedReader in =
      SocketUtil.getBufferedReader(client);
    out.println("Generic Network Client");
    System.out.println
      ("Generic Network Client:\n" +
       "Made connection to " + host +
       " and got '" + in.readLine() + "' in response");
    client.close();
  }
}
```

SocketUtil – Simplifying Creation of Reader and Writer

```
import java.net.*;
import java.io.*;

public class SocketUtil {
  /** Make a BufferedReader to get incoming data.  */
  public static BufferedReader getBufferedReader
                        (Socket s) throws IOException {
    return(new BufferedReader(
      new InputStreamReader(s.getInputStream())));
  }

  /** Make a PrintWriter to send outgoing data.
   *  This PrintWriter will automatically flush stream
   *  when println is called.
   */
  public static PrintWriter getPrintWriter(Socket s)
      throws IOException {
    // 2nd argument of true means autoflush
    return(new PrintWriter(s.getOutputStream(), true));
  }
}
```

Example Client

```
public class NetworkClientTest {
  public static void main(String[] args) {
    String host = "localhost";
    if (args.length > 0)
      host = args[0];
    int port = 8088;
    if (args.length > 1)
      port = Integer.parseInt(args[1]);
    NetworkClient nwClient
      = new NetworkClient(host, port);
    nwClient.connect();
  }
}
```

Example Client, Result

```
> java NetworkClientTest ftp.netscape.com 21
Generic Network Client:
Made connection to ftp.netscape.com and got
‘220 ftp26 FTP server (UNIX(r) System V Release 4.0)
ready.’ in response
>
```

Aside: Parsing Strings Using StringTokenizer

- **Idea**
  - Build a tokenizer from an initial string
  - Retrieve tokens one at a time with `nextToken`
  - You can also see how many tokens are remaining (`countTokens`) or simply test if the number of tokens remaining is nonzero (`hasMoreTokens`)

```
StringTokenizer tok
  = new StringTokenizer(input, delimiters);
while (tok.hasMoreTokens()) {
  doSomethingWith(tok.nextToken());
}
```

StringTokenizer

- **Constructors**
  - StringTokenizer(String input, String delimiters)
  - StringTokenizer(String input, String delimiters,
    boolean includeDelimiters)
  - StringTokenizer(String input)
    - Default delimiter set is "` \t\n\r\f`" (whitespace)
- **Methods**
  - nextToken(), nextToken(String delimiters)
  - countTokens()
  - hasMoreTokens()
- **Also see methods in String class**
  - substring, indexOf, startsWith, endsWith, compareTo, …
  - JDK 1.4 has regular expressions in java.util.regex!

Interactive Tokenizer: Example

```
import java.util.StringTokenizer;

public class TokTest {
  public static void main(String[] args) {
    if (args.length == 2) {
      String input = args[0], delimiters = args[1];
      StringTokenizer tok
        = new StringTokenizer(input, delimiters);
      while (tok.hasMoreTokens()) {
        System.out.println(tok.nextToken());
      }
    } else {
      System.out.println
        ("Usage: java TokTest string delimiters");
    }
  }
}
```

Interactive Tokenizer: Result

```
> java TokTest http://www.microsoft.com/~gates/ :/.
http
www
microsoft
com
~gates


> java TokTest "if (tok.hasMoreTokens()) {" "(){. "
if
tok
hasMoreTokens
```

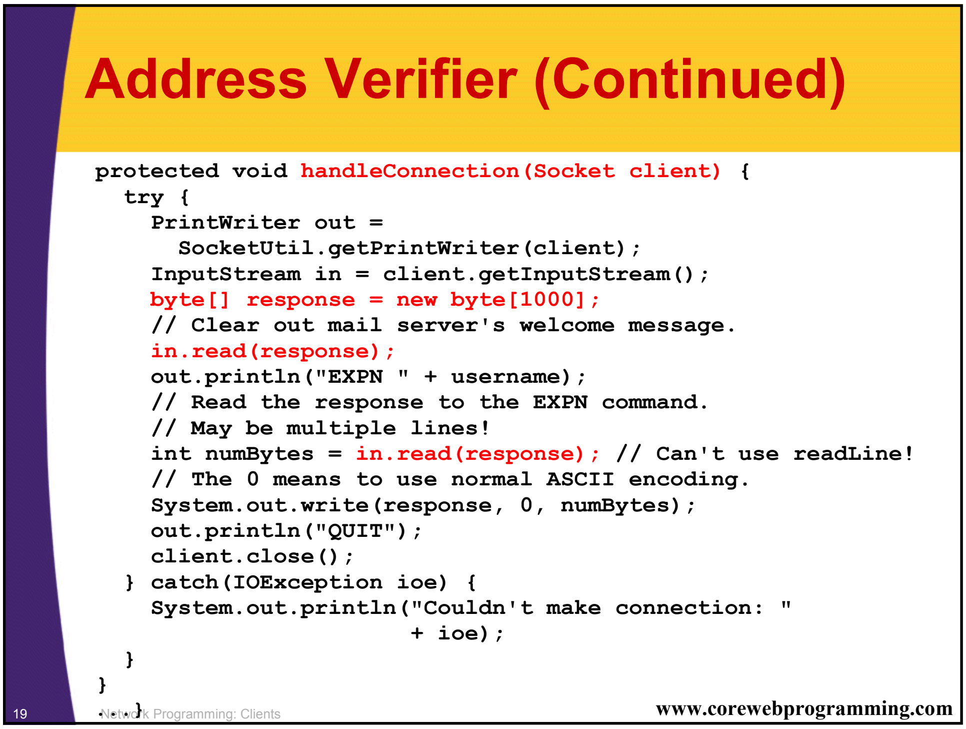
A Client to Verify Email Addresses

- **Talking to a mail server**
  - One of the best ways to get comfortable with a network protocol is to telnet to the port a server is on and try out commands interactively
- **Example talking to apl.jhu.edu's server**

```
> telnet apl.jhu.edu 25
Trying 128.220.101.100 ...Connected … Escape character …
220 aplcenmp.apl.jhu.edu Sendmail SMI-8.6/SMI-SVR4 ready …
expn hall
250 Marty Hall <hall@aplcenmp.apl.jhu.edu>
expn root
250 Gary Gafke <…>
250 Tom Vellani <…>
quit
221 aplcenmp.apl.jhu.edu closing connection
Connection closed by foreign host.
```

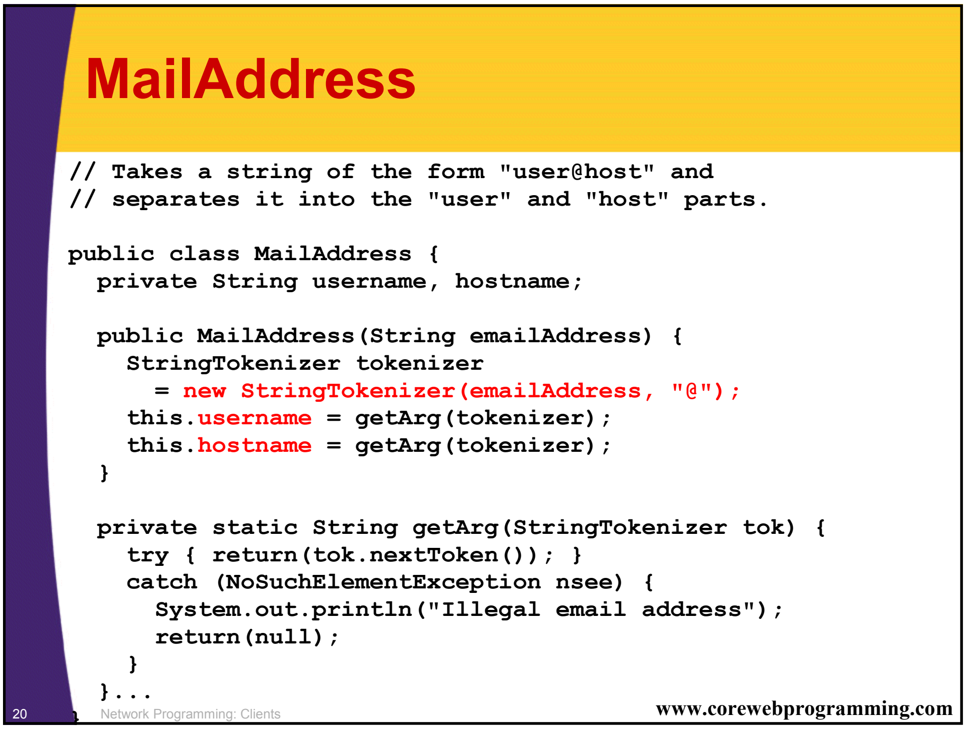
Address Verifier

```
/** Given an email address of the form user@host,
 *  connect to port 25 of the host and issue an
 *  'expn' request for the user. Print the results.
 */

public class AddressVerifier extends NetworkClient {
  private String username;

  public static void main(String[] args) {
    MailAddress address = new MailAddress(args[0]);
    AddressVerifier verifier
      = new AddressVerifier(address.getUsername(),
                            address.getHostname(),
                            25);
    verifier.connect();
  }
  ...
```



Address Verifier (Continued)

```
protected void handleConnection(Socket client) {
  try {
    PrintWriter out =
      SocketUtil.getPrintWriter(client);
    InputStream in = client.getInputStream();
    byte[] response = new byte[1000];
    // Clear out mail server's welcome message.
    in.read(response);
    out.println("EXPN " + username);
    // Read the response to the EXPN command.
    // May be multiple lines!
    int numBytes = in.read(response); // Can't use readLine!
    // The 0 means to use normal ASCII encoding.
    System.out.write(response, 0, numBytes);
    out.println("QUIT");
    client.close();
  } catch(IOException ioe) {
    System.out.println("Couldn't make connection: "
                       + ioe);
  }
}
...}
```



MailAddress

```
// Takes a string of the form "user@host" and
// separates it into the "user" and "host" parts.

public class MailAddress {
  private String username, hostname;

  public MailAddress(String emailAddress) {
    StringTokenizer tokenizer
      = new StringTokenizer(emailAddress, "@");
    this.username = getArg(tokenizer);
    this.hostname = getArg(tokenizer);
  }

  private static String getArg(StringTokenizer tok) {
    try { return(tok.nextToken()); }
    catch (NoSuchElementException nsee) {
      System.out.println("Illegal email address");
      return(null);
    }
  }...
}
```

Address Verifier: Result

```
> java AddressVerifier tbl@w3.org
250 <timbl@hq.lcs.mit.edu>

> java AddressVerifier timbl@hq.lcs.mit.edu
250 Tim Berners-Lee <timbl>

> java AddressVerifier gosling@mail.javasoft.com
550 gosling... User unknown
```

Brief Aside: Using the HTTP GET Command

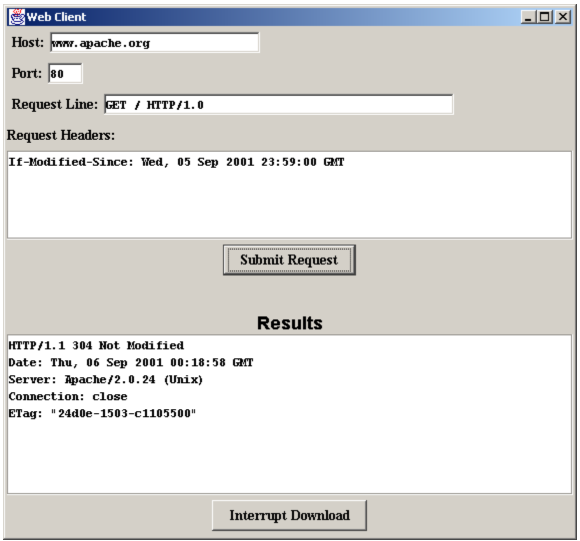
- For the URL http://www.apl.jhu.edu/~lmb/

```
Unix> telnet www.apl.jhu.edu 80
Trying 128.220.101.100 ...
Connected to aplcenmp.apl.jhu.edu.
Escape character is '^]'.
GET /~lmb/ HTTP/1.0

HTTP/1.0 200 Document follows
Date: Sat, 30 Jun 2001 14:34:58 GMT
Server: NCSA/1.5.2
Last-modified: Tue, 11 Jul 2001 15:13:56 GMT
Content-type: text/html
Content-length: 50479

<!DOCTYPE HTML PUBLIC
          "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
...
</HTML>Connection closed by foreign host.
Unix>
```

Talking to Web Servers Interactively

- **WebClient**
  - Simple graphical user interface to communicate with HTTP servers
  - User can interactively specify:
    - Host
    - Port
    - HTTP request line
    - HTTP request headers
  - HTTP request is performed in a separate thread
  - Response document is placed in a scrollable text area
  - Download all source files for WebClient from http://archive.corewebprogramming.com/Chapter17.html

WebClient: Example



Host: www.apache.org

Port: 80

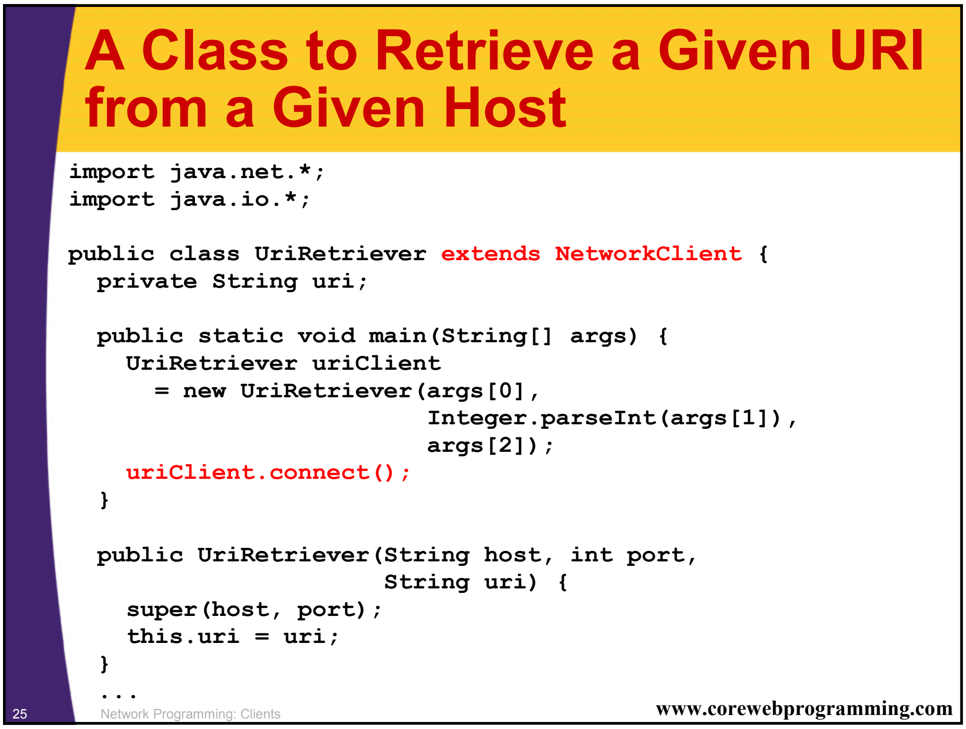
Request Line: GET / HTTP/1.0

Request Headers:

```
If-Modified-Since: Wed, 05 Sep 2001 23:59:00 GMT
```

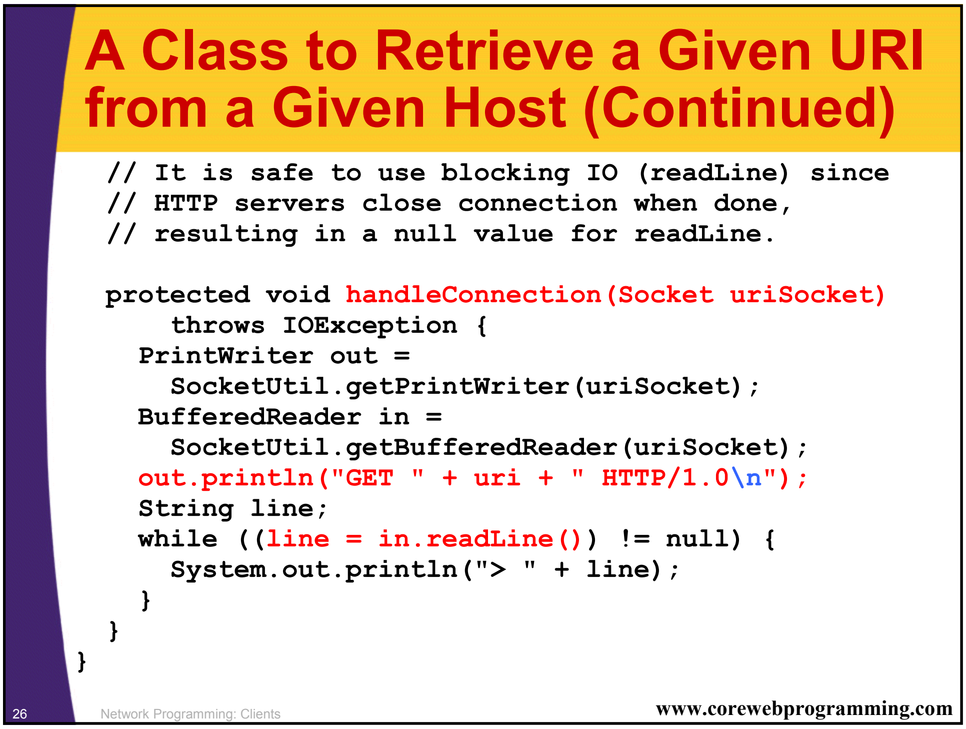
Results

```
HTTP/1.1 304 Not Modified
Date: Thu, 06 Sep 2001 00:18:58 GMT
Server: Apache/2.0.24 (Unix)
Connection: close
ETag: "24d0e-1503-c1105500"
```



A Class to Retrieve a Given URI from a Given Host

```
import java.net.*;
import java.io.*;

public class UriRetriever extends NetworkClient {
  private String uri;

  public static void main(String[] args) {
    UriRetriever uriClient
      = new UriRetriever(args[0],
                         Integer.parseInt(args[1]),
                         args[2]);
    uriClient.connect();
  }

  public UriRetriever(String host, int port,
                      String uri) {
    super(host, port);
    this.uri = uri;
  }
  ...
```



A Class to Retrieve a Given URI from a Given Host (Continued)

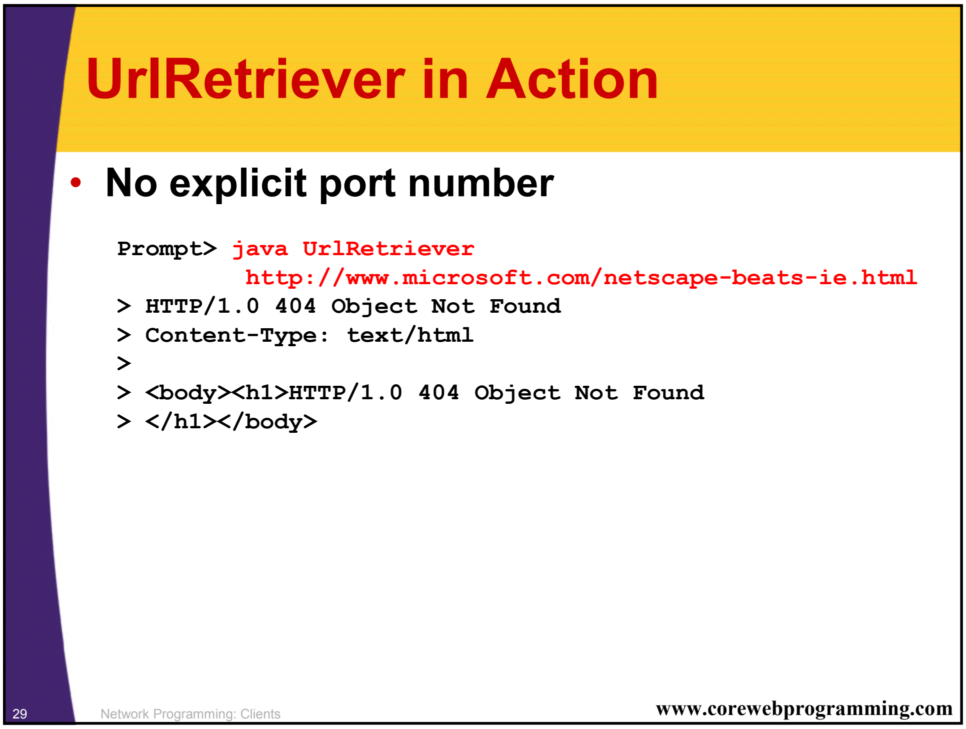
```
  // It is safe to use blocking IO (readLine) since
  // HTTP servers close connection when done,
  // resulting in a null value for readLine.

  protected void handleConnection(Socket uriSocket)
      throws IOException {
    PrintWriter out =
      SocketUtil.getPrintWriter(uriSocket);
    BufferedReader in =
      SocketUtil.getBufferedReader(uriSocket);
    out.println("GET " + uri + " HTTP/1.0\n");
    String line;
    while ((line = in.readLine()) != null) {
      System.out.println("> " + line);
    }
  }
}
```

A Class to Retrieve a Given URL

```
public class UrlRetriever {
  public static void main(String[] args) {
    checkUsage(args);
    StringTokenizer tok = new StringTokenizer(args[0]);
    String protocol = tok.nextToken(":");
    checkProtocol(protocol);
    String host = tok.nextToken(":/");
    String uri;
    int port = 80;
    try {
      uri = tok.nextToken("");
      if (uri.charAt(0) == ':') {
        tok = new StringTokenizer(uri);
        port = Integer.parseInt(tok.nextToken(":/"));
        uri = tok.nextToken("");
      }
    } catch(NoSuchElementException nsee) {
      uri = "/";
    }
```

A Class to Retrieve a Given URL (Continued)

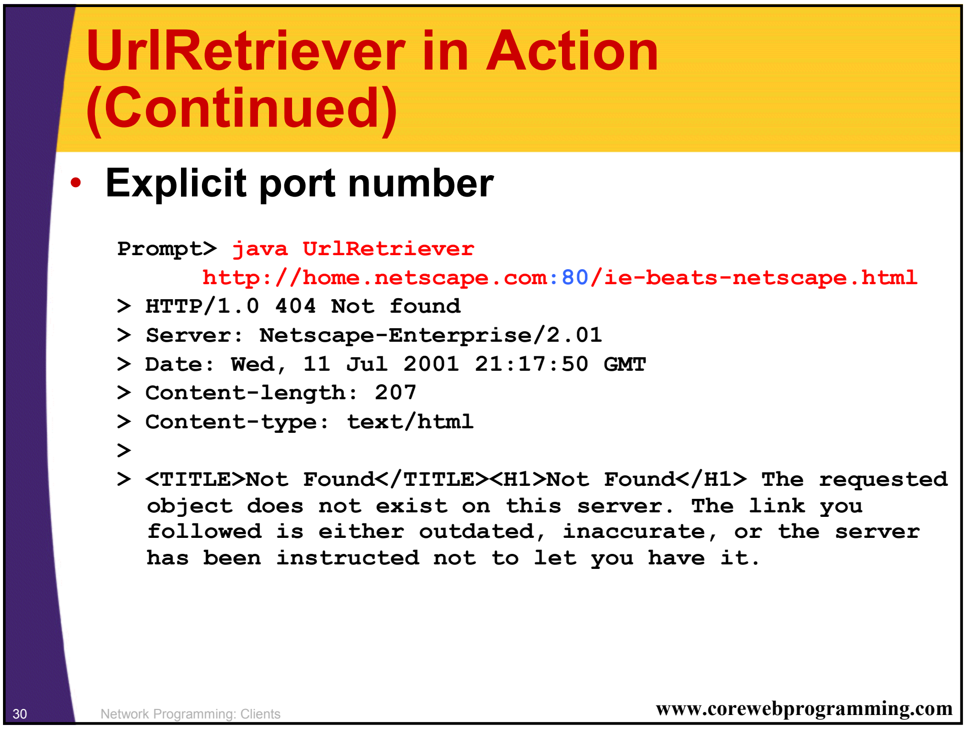
```
    UriRetriever uriClient =
      new UriRetriever(host, port, uri);
    uriClient.connect();
  }

  /** Warn user if they forgot the URL. */
  private static void checkUsage(String[] args) {
    if (args.length != 1) {
      System.out.println("Usage: UrlRetriever <URL>");
      System.exit(-1);
    }
  }

  /** Tell user that this can only handle HTTP. */
  private static void checkProtocol(String protocol) {
    if (!protocol.equals("http")) {
      System.out.println("Don't understand protocol "
                         + protocol);
      System.exit(-1);
    }
  }}
```



UrlRetriever in Action

- No explicit port number

```
Prompt> java UrlRetriever
         http://www.microsoft.com/netscape-beats-ie.html
> HTTP/1.0 404 Object Not Found
> Content-Type: text/html
>
> <body><h1>HTTP/1.0 404 Object Not Found
> </h1></body>
```

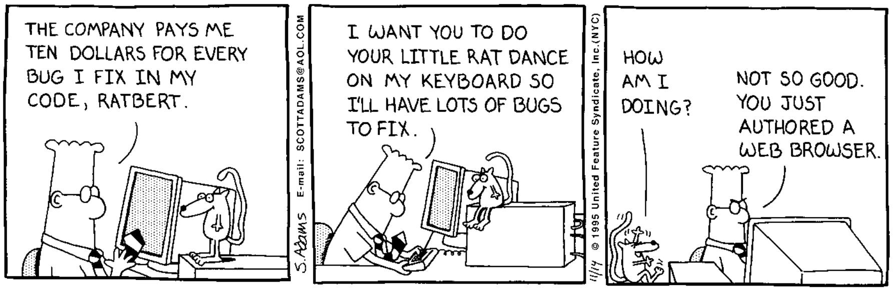


UrlRetriever in Action (Continued)

- Explicit port number

```
Prompt> java UrlRetriever
      http://home.netscape.com:80/ie-beats-netscape.html
> HTTP/1.0 404 Not found
> Server: Netscape-Enterprise/2.01
> Date: Wed, 11 Jul 2001 21:17:50 GMT
> Content-length: 207
> Content-type: text/html
>
> <TITLE>Not Found</TITLE><H1>Not Found</H1> The requested
  object does not exist on this server. The link you
  followed is either outdated, inaccurate, or the server
  has been instructed not to let you have it.
```

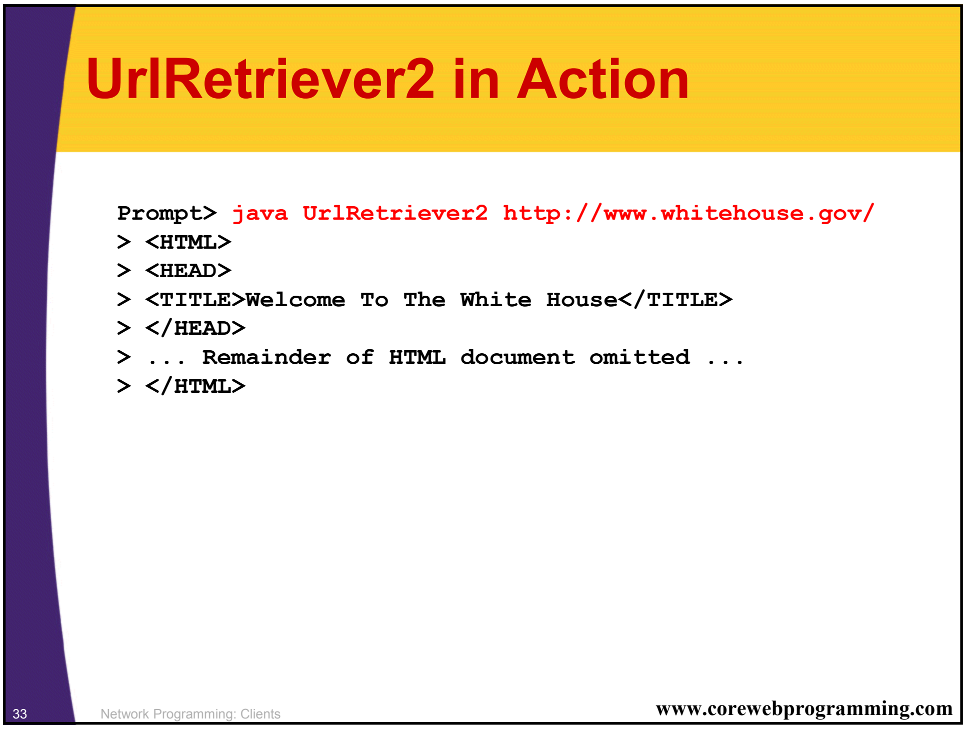
Writing a Web Browser


THE COMPANY PAYS ME TEN DOLLARS FOR EVERY BUG I FIX IN MY CODE, RATBERT.
I WANT YOU TO DO YOUR LITTLE RAT DANCE ON MY KEYBOARD SO I'LL HAVE LOTS OF BUGS TO FIX.
HOW AM I DOING?
NOT SO GOOD. YOU JUST AUTHORED A WEB BROWSER.
S. Adams
E-mail: SCOTTADAMS@AOL.COM
© 1995 United Feature Syndicate, Inc.(NYC)

- **Wow! We just wrote a Web browser in 3 pages of code.**
  - Didn't format the HTML, but still not bad for 3 pages
  - But we can do even better…

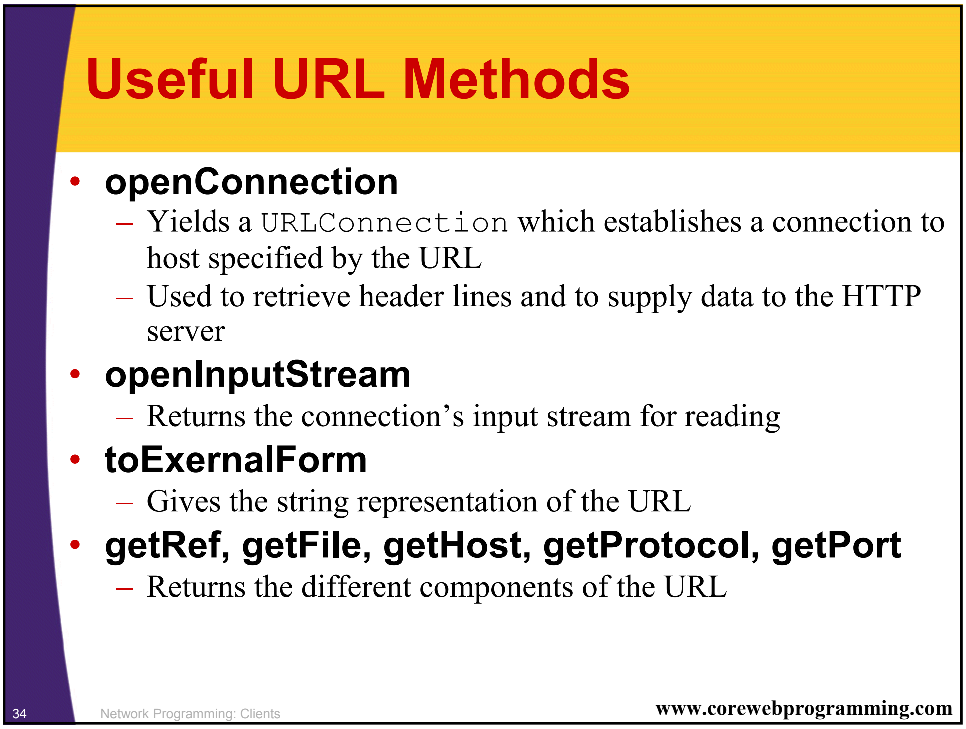
Browser in 1 Page: Using URL

```
public class UrlRetriever2 {
  public static void main(String[] args) {
    try {
      URL url = new URL(args[0]);
      BufferedReader in = new BufferedReader(
                            new InputStreamReader(
                              url.openStream()));
      String line;
      while ((line = in.readLine()) != null) {
        System.out.println("> " + line);
      }
      in.close();
    } catch(MalformedURLException mue) { // URL c'tor
      System.out.println(args[0] + "is an invalid URL: "
                         + mue);
    } catch(IOException ioe) { // Stream constructors
      System.out.println("IOException: " + ioe);
    }
  }
}
```



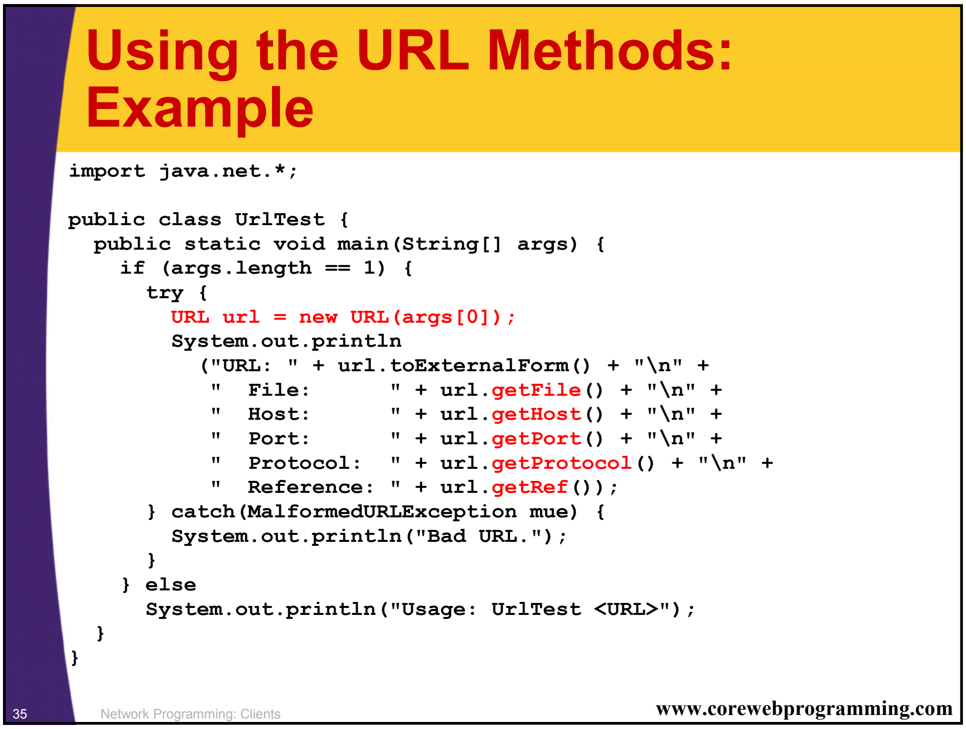
UrlRetriever2 in Action

```
Prompt> java UrlRetriever2 http://www.whitehouse.gov/
> <HTML>
> <HEAD>
> <TITLE>Welcome To The White House</TITLE>
> </HEAD>
> ... Remainder of HTML document omitted ...
> </HTML>
```



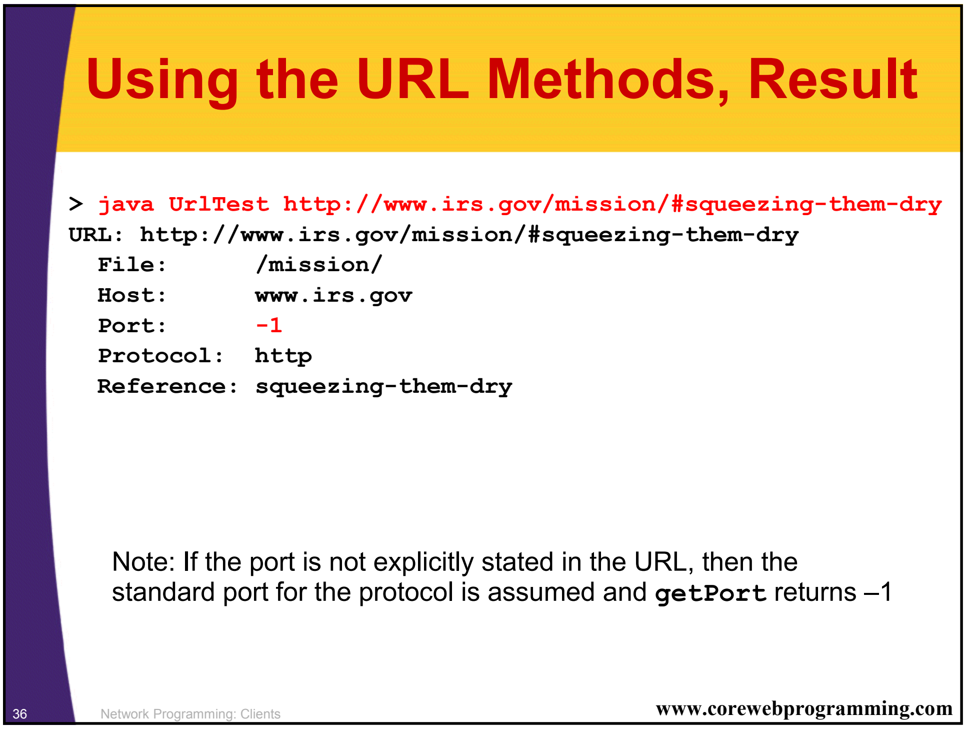
Useful URL Methods

- **openConnection**
  - Yields a `URLConnection` which establishes a connection to host specified by the URL
  - Used to retrieve header lines and to supply data to the HTTP server
- **openInputStream**
  - Returns the connection’s input stream for reading
- **toExernalForm**
  - Gives the string representation of the URL
- **getRef, getFile, getHost, getProtocol, getPort**
  - Returns the different components of the URL



Using the URL Methods: Example

```
import java.net.*;

public class UrlTest {
  public static void main(String[] args) {
    if (args.length == 1) {
      try {
        URL url = new URL(args[0]);
        System.out.println
          ("URL: " + url.toExternalForm() + "\n" +
           "  File:      " + url.getFile() + "\n" +
           "  Host:      " + url.getHost() + "\n" +
           "  Port:      " + url.getPort() + "\n" +
           "  Protocol:  " + url.getProtocol() + "\n" +
           "  Reference: " + url.getRef());
      } catch(MalformedURLException mue) {
        System.out.println("Bad URL.");
      }
    } else
      System.out.println("Usage: UrlTest <URL>");
  }
}
```



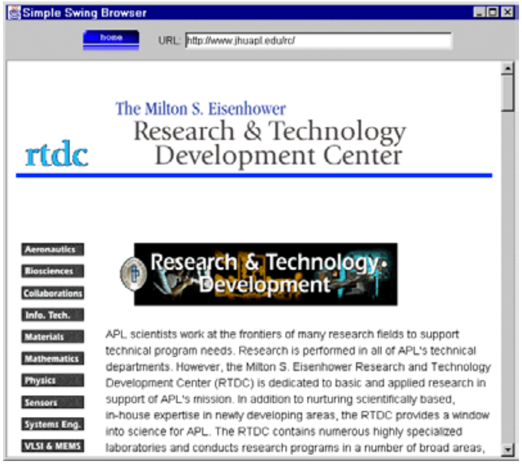
Using the URL Methods, Result

```
> java UrlTest http://www.irs.gov/mission/#squeezing-them-dry
URL: http://www.irs.gov/mission/#squeezing-them-dry
  File:      /mission/
  Host:      www.irs.gov
  Port:      -1
  Protocol:  http
  Reference: squeezing-them-dry
```

Note: If the port is not explicitly stated in the URL, then the standard port for the protocol is assumed and **getPort** returns –1

A Real Browser Using Swing

- The `JEditorPane` class has builtin support for HTTP and HTML

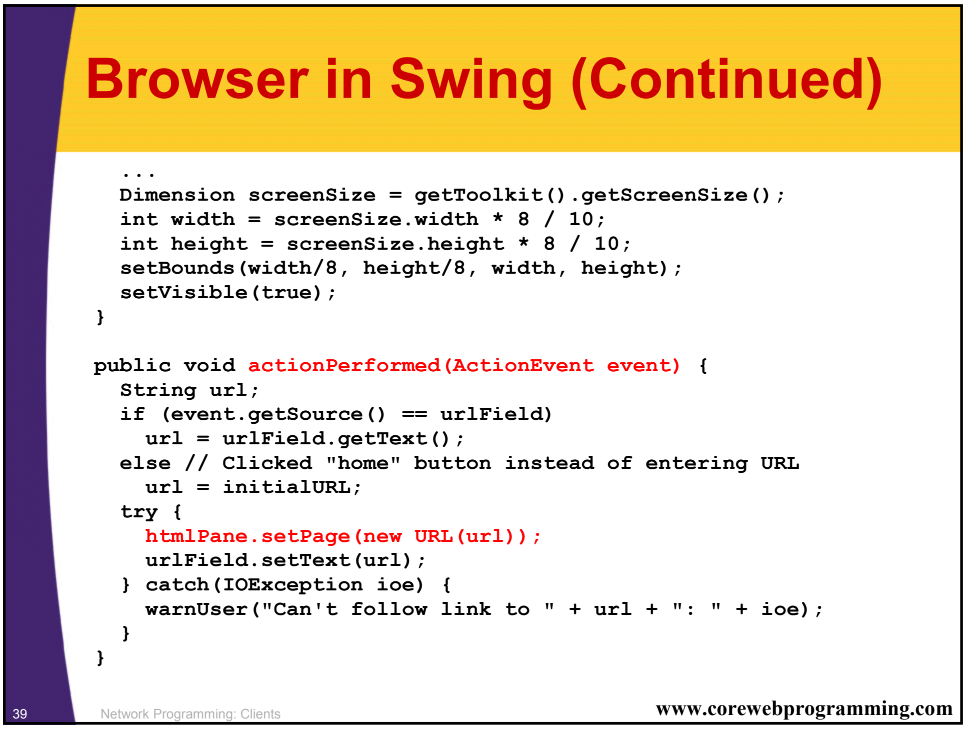


The Milton S. Eisenhower
Research & Technology
Development Center

APL scientists work at the frontiers of many research fields to support technical program needs. Research is performed in all of APL's technical departments. However, the Milton S. Eisenhower Research and Technology Development Center (RTDC) is dedicated to basic and applied research in support of APL's mission. In addition to nurturing scientifically based, in-house expertise in newly developing areas, the RTDC provides a window into science for APL. The RTDC contains numerous highly specialized laboratories and conducts research programs in a number of broad areas,

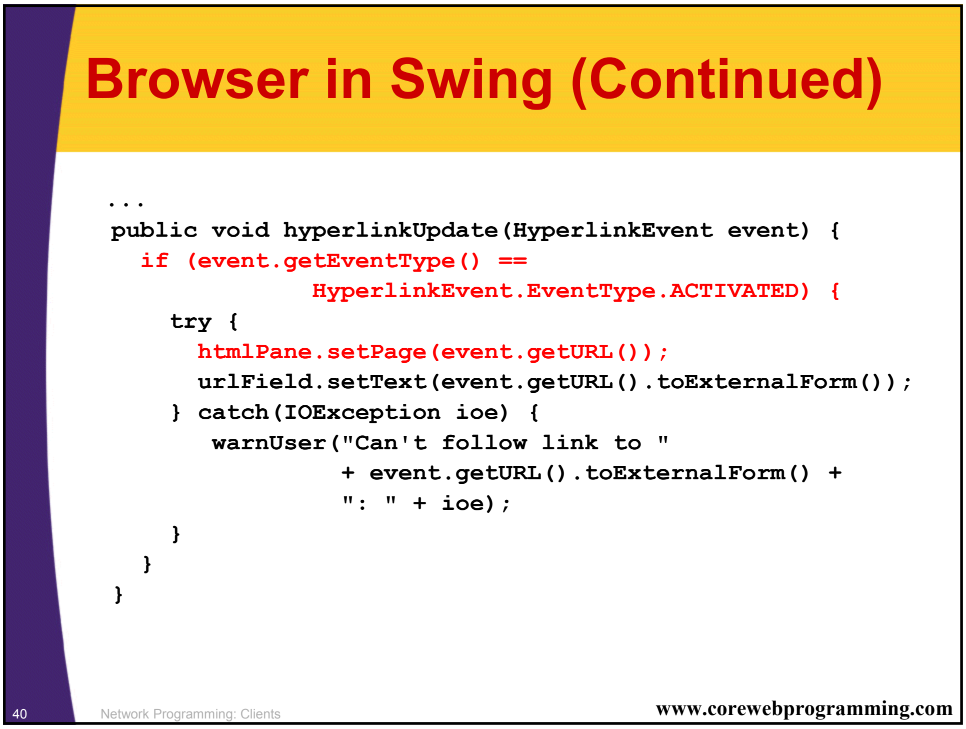
Browser in Swing: Code

```
import javax.swing.*;
import javax.swing.event.*;
...

public class Browser extends JFrame implements HyperlinkListener,
                                               ActionListener {
  private JEditorPane htmlPane;
  ...

  public Browser(String initialURL) {
    ...
    try {
        htmlPane = new JEditorPane(initialURL);
        htmlPane.setEditable(false);
        htmlPane.addHyperlinkListener(this);
        JScrollPane scrollPane = new JScrollPane(htmlPane);
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    } catch(IOException ioe) {
       warnUser("Can't build HTML pane for " + initialURL
                + ": " + ioe);
    }
```



Browser in Swing (Continued)

```
  ...
  Dimension screenSize = getToolkit().getScreenSize();
  int width = screenSize.width * 8 / 10;
  int height = screenSize.height * 8 / 10;
  setBounds(width/8, height/8, width, height);
  setVisible(true);
}

public void actionPerformed(ActionEvent event) {
  String url;
  if (event.getSource() == urlField)
    url = urlField.getText();
  else // Clicked "home" button instead of entering URL
    url = initialURL;
  try {
    htmlPane.setPage(new URL(url));
    urlField.setText(url);
  } catch(IOException ioe) {
    warnUser("Can't follow link to " + url + ": " + ioe);
  }
}
```



Browser in Swing (Continued)

```
...
public void hyperlinkUpdate(HyperlinkEvent event) {
  if (event.getEventType() ==
              HyperlinkEvent.EventType.ACTIVATED) {
    try {
      htmlPane.setPage(event.getURL());
      urlField.setText(event.getURL().toExternalForm());
    } catch(IOException ioe) {
       warnUser("Can't follow link to "
                + event.getURL().toExternalForm() +
                ": " + ioe);
    }
  }
}
```

Summary

- **Opening a socket requires a hostname (or IP address) and port number**
- **A PrintWriter lets you send string data**
  - Use autoflush to send the full line after each println
- **A BufferedReader allows you to read the input one line at a time (readLine)**
  - The `readLine` method blocks until a response is sent
  - For a typical GET request, after the HTTP server sends the response the connection is closed and `readLine` returns `null`
- **StringTokenizer provides simple parsing**
- **The URL and URLConnection classes simplify communication with Web servers**

core WEB programming

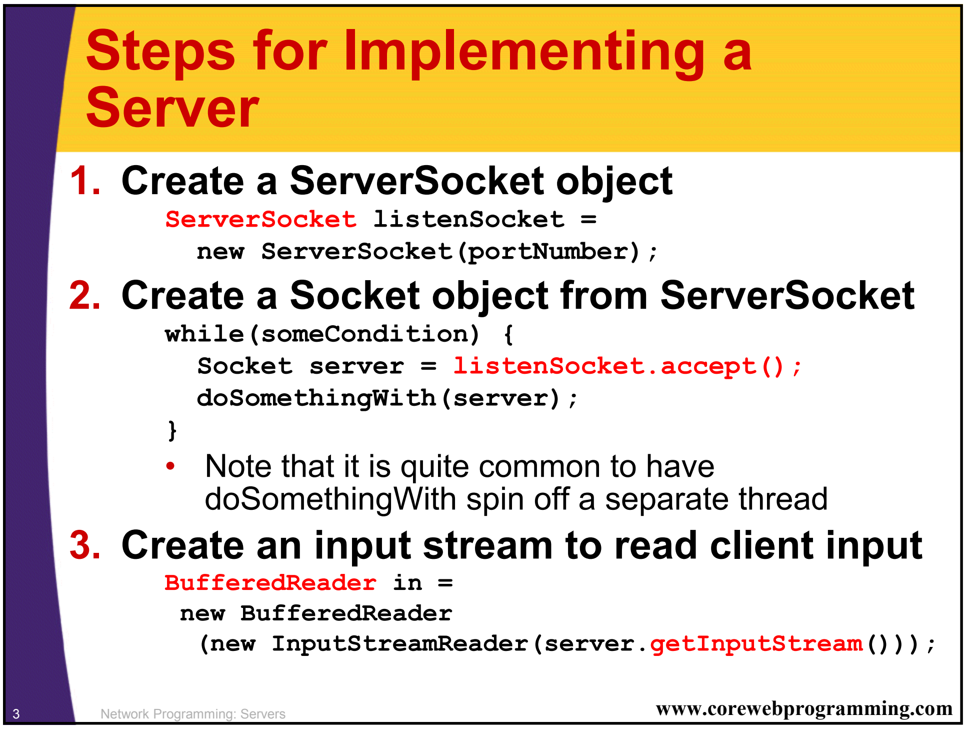
Questions?

core WEB programming

Network Programming: Servers

Agenda

- **Steps for creating a server**
  1. Create a ServerSocket object
  2. Create a Socket object from ServerSocket
  3. Create an input stream
  4. Create an output stream
  5. Do I/O with input and output streams
  6. Close the socket
- **A generic network server**
- **Accepting connections from browsers**
- **Creating an HTTP server**
- **Adding multithreading to an HTTP server**



Steps for Implementing a Server

1. **Create a ServerSocket object**

```
ServerSocket listenSocket =
  new ServerSocket(portNumber);
```

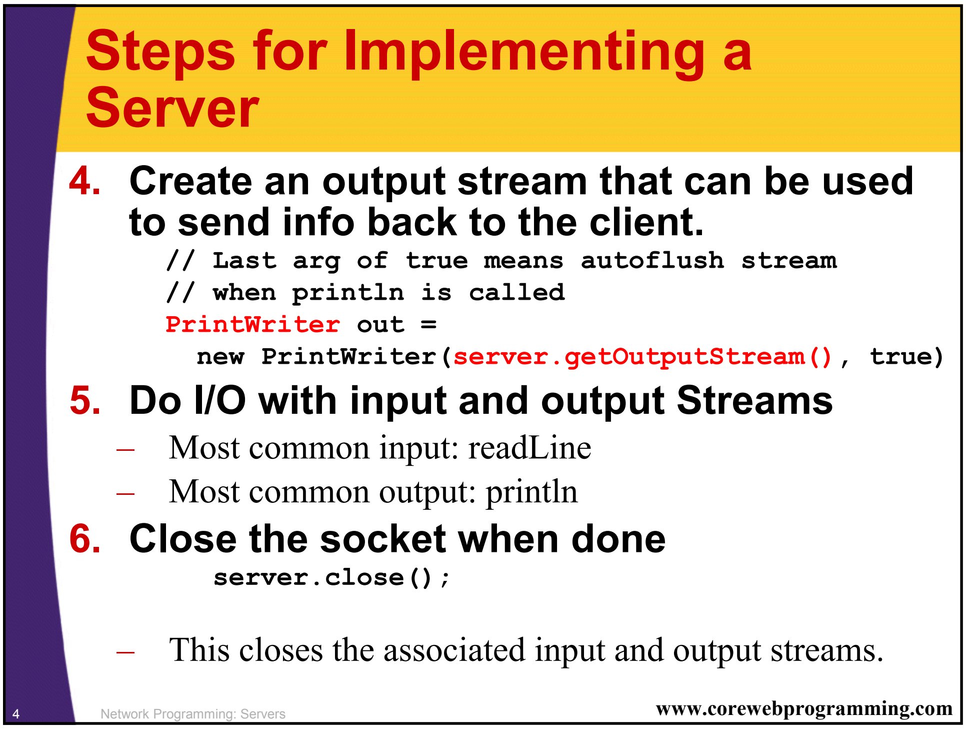
2. **Create a Socket object from ServerSocket**

```
while(someCondition) {
  Socket server = listenSocket.accept();
  doSomethingWith(server);
}
```

- Note that it is quite common to have doSomethingWith spin off a separate thread

3. **Create an input stream to read client input**

```
BufferedReader in =
 new BufferedReader
  (new InputStreamReader(server.getInputStream()));
```



Steps for Implementing a Server

4. **Create an output stream that can be used to send info back to the client.**

```
// Last arg of true means autoflush stream
// when println is called
PrintWriter out =
  new PrintWriter(server.getOutputStream(), true)
```

5. **Do I/O with input and output Streams**

   – Most common input: readLine

   – Most common output: println

6. **Close the socket when done**

```
server.close();
```

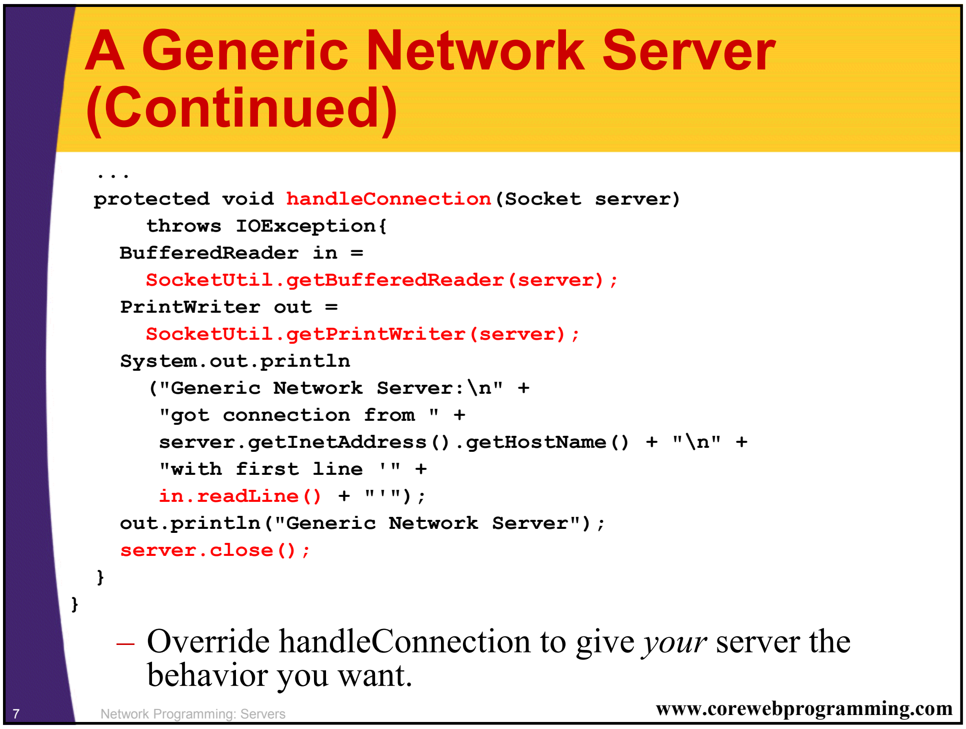
   – This closes the associated input and output streams.

A Generic Network Server

```
import java.net.*;
import java.io.*;

/** A starting point for network servers. */

public class NetworkServer {
  protected int port, maxConnections;

  /** Build a server on specified port. It will continue
   *  to accept connections (passing each to
   *  handleConnection) until an explicit exit
   *  command is sent (e.g. System.exit) or the
   *  maximum number of connections is reached. Specify
   *  0 for maxConnections if you want the server
   *  to run indefinitely.
   */

  public NetworkServer(int port, int maxConnections) {
    this.port = port;
    this.maxConnections = maxConnections;
  }
  ...
```

A Generic Network Server (Continued)

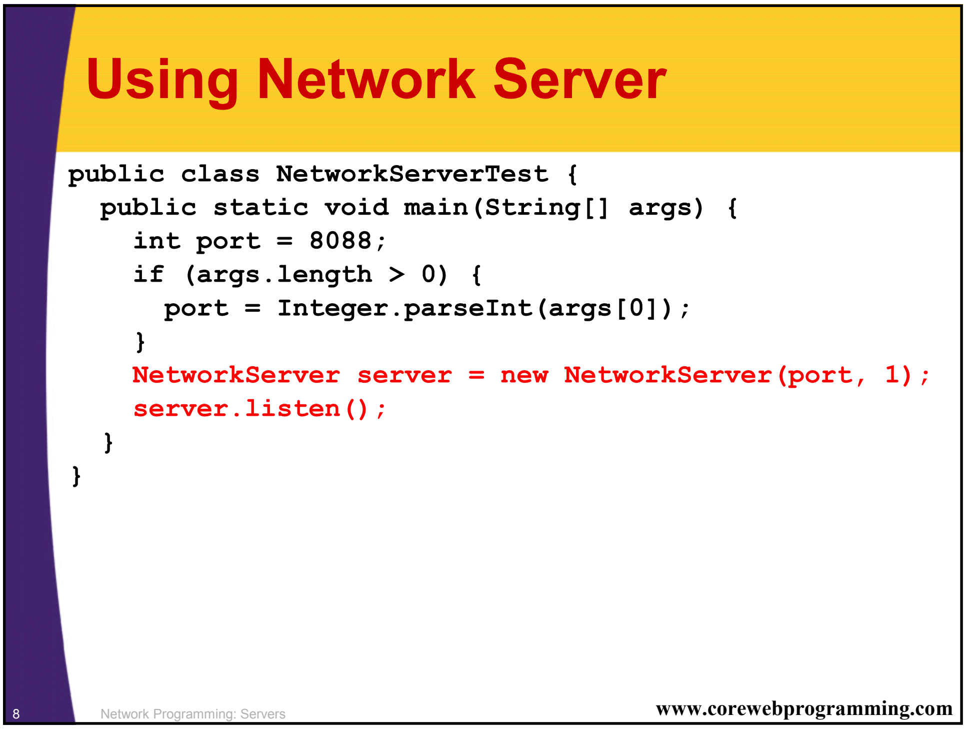
```
  /** Monitor a port for connections. Each time one
   *  is established, pass resulting Socket to
   *  handleConnection.
   */

  public void listen() {
    int i=0;
    try {
      ServerSocket listener = new ServerSocket(port);
      Socket server;
      while((i++ < maxConnections) ||
            (maxConnections == 0)) {
        server = listener.accept();
        handleConnection(server);
      }
    } catch (IOException ioe) {
      System.out.println("IOException: " + ioe);
      ioe.printStackTrace();
    }
  }
```



A Generic Network Server (Continued)

```
  ...
  protected void handleConnection(Socket server)
      throws IOException{
    BufferedReader in =
      SocketUtil.getBufferedReader(server);
    PrintWriter out =
      SocketUtil.getPrintWriter(server);
    System.out.println
      ("Generic Network Server:\n" +
       "got connection from " +
       server.getInetAddress().getHostName() + "\n" +
       "with first line '" +
       in.readLine() + "'");
    out.println("Generic Network Server");
    server.close();
  }
}
```

– Override handleConnection to give *your* server the behavior you want.



Using Network Server

```
public class NetworkServerTest {
  public static void main(String[] args) {
    int port = 8088;
    if (args.length > 0) {
      port = Integer.parseInt(args[0]);
    }
    NetworkServer server = new NetworkServer(port, 1);
    server.listen();
  }
}
```

Network Server: Results

- **Accepting a Connection from a WWW Browser**
  - Suppose the above test program is started up on port 8088 of `server.com`:

```
server> java NetworkServerTest
```

  - Then, a standard Web browser on `client.com` requests `http://server.com:8088/foo/`, yielding the following back on `server.com`:

```
Generic Network Server:
got connection from client.com
with first line 'GET /foo/ HTTP/1.0'
```

HTTP Requests and Responses

- **Request**

```
GET /~gates/ HTTP/1.0
Header1: …
Header2: …
…
HeaderN: …
Blank Line
```

  - All request headers are optional except for `Host` (required only for HTTP/1.1 requests)
  - If you send `HEAD` instead of `GET`, the server returns the same HTTP headers, but no document

- **Response**

```
HTTP/1.0 200 OK
Content-Type: text/html
Header2: …
…
HeaderN: …
Blank Line
<!DOCTYPE …>
<HTML>
…
</HTML>
```

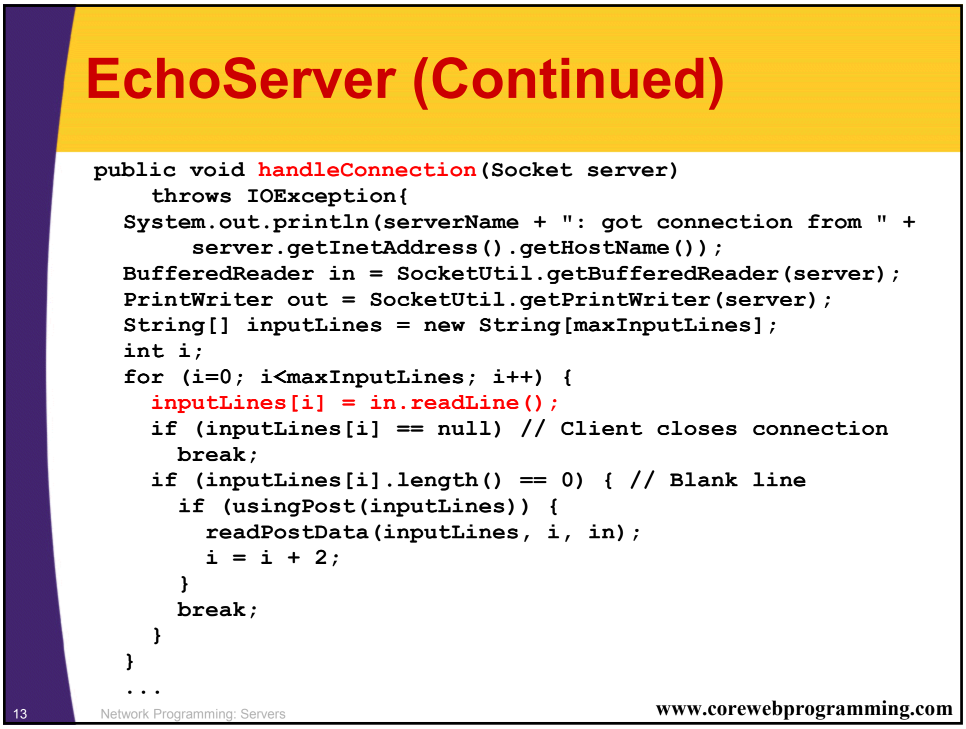
  - All response headers are optional except for `Content-Type`

A Simple HTTP Server

- **Idea**
  1. Read all the lines sent by the browser, storing them in an array
     - Use readLine a line at a time until an empty line
       - Exception: with POST requests you have to read some extra data
  2. Send an HTTP response line (e.g. "HTTP/1.0 200 OK")
  3. Send a Content-Type line then a blank line
     - This indicates the file type being returned (HTML in this case)
  4. Send an HTML file showing the lines that were sent
  5. Close the connection

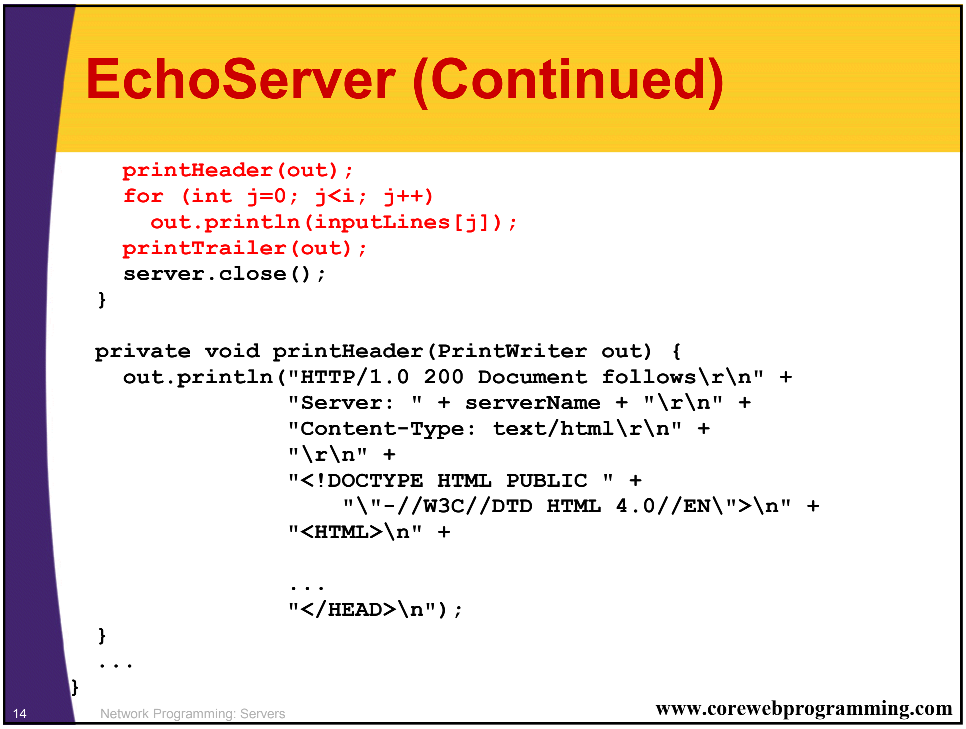
EchoServer

```
import java.net.*;
import java.io.*;
import java.util.StringTokenizer;

 /** A simple HTTP server that generates a Web page
  *  showing all of the data that it received from
  *  the Web client (usually a browser). */

public class EchoServer extends NetworkServer {
  protected int maxInputLines = 25;
  protected String serverName = "EchoServer 1.0";

  public static void main(String[] args) {
    int port = 8088;
    if (args.length > 0)
      port = Integer.parseInt(args[0]);
    EchoServer echoServer = new EchoServer(port, 0);
    echoServer.listen();
  }

  public EchoServer(int port, int maxConnections) {
    super(port, maxConnections);
  }
```



EchoServer (Continued)

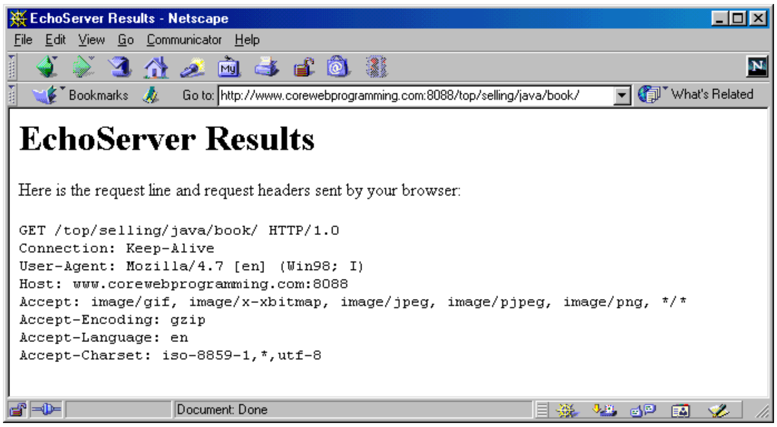
```
public void handleConnection(Socket server)
    throws IOException{
  System.out.println(serverName + ": got connection from " +
       server.getInetAddress().getHostName());
  BufferedReader in = SocketUtil.getBufferedReader(server);
  PrintWriter out = SocketUtil.getPrintWriter(server);
  String[] inputLines = new String[maxInputLines];
  int i;
  for (i=0; i<maxInputLines; i++) {
    inputLines[i] = in.readLine();
    if (inputLines[i] == null) // Client closes connection
      break;
    if (inputLines[i].length() == 0) { // Blank line
      if (usingPost(inputLines)) {
        readPostData(inputLines, i, in);
        i = i + 2;
      }
      break;
    }
  }
  ...
```



EchoServer (Continued)

```
    printHeader(out);
    for (int j=0; j<i; j++)
      out.println(inputLines[j]);
    printTrailer(out);
    server.close();
  }

  private void printHeader(PrintWriter out) {
    out.println("HTTP/1.0 200 Document follows\r\n" +
                "Server: " + serverName + "\r\n" +
                "Content-Type: text/html\r\n" +
                "\r\n" +
                "<!DOCTYPE HTML PUBLIC " +
                    "\"-//W3C//DTD HTML 4.0//EN\">\n" +
                "<HTML>\n" +

                ...
                "</HEAD>\n");
  }
  ...
}
```

EchoServer in Action


EchoServer Results - Netscape
File Edit View Go Communicator Help
Bookmarks
Go to: http://www.corewebprogramming.com:8088/top/selling/java/book/
What's Related
Document: Done

EchoServer Results

Here is the request line and request headers sent by your browser:

```
GET /top/selling/java/book/ HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.7 [en] (Win98; I)
Host: www.corewebprogramming.com:8088
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

EchoServer shows data sent by the browser

Adding Multithreading

```
import java.net.*;
import java.io.*;

/** A multithreaded variation of EchoServer. */

public class ThreadedEchoServer extends EchoServer
                                implements Runnable {
  public static void main(String[] args) {
    int port = 8088;
    if (args.length > 0)
      port = Integer.parseInt(args[0]);
    ThreadedEchoServer echoServer =
      new ThreadedEchoServer(port, 0);
    echoServer.serverName = "Threaded Echo Server 1.0";
    echoServer.listen();
  }

  public ThreadedEchoServer(int port, int connections) {
    super(port, connections);
  }
```

Adding Multithreading (Continued)

```
  public void handleConnection(Socket server) {
    Connection connectionThread =
      new Connection(this, server);
    connectionThread.start();
  }

  public void run() {
    Connection currentThread =
      (Connection)Thread.currentThread();
    try {
      super.handleConnection(currentThread.serverSocket);
    } catch(IOException ioe) {
      System.out.println("IOException: " + ioe);
      ioe.printStackTrace();
    }
  }
}
```

Adding Multithreading (Continued)

```
/** This is just a Thread with a field to store a
 *  Socket object. Used as a thread-safe means to pass
 *  the Socket from handleConnection to run.
 */

class Connection extends Thread {
  protected Socket serverSocket;

  public Connection(Runnable serverObject,
                    Socket serverSocket) {
    super(serverObject);
    this.serverSocket = serverSocket;
  }
}
```

Summary

- Create a ServerSocket; specify port number
- Call `accept` to wait for a client connection
  - Once a connection is established, a Socket object is created to communicate with client
- Browser requests consist of a GET, POST, or HEAD line followed by a set of request headers and a blank line
- For the HTTP server response, send the status line (HTTP/1.0 200 OK), Content-Type, blank line, and document
- For improved performance, process each request in a separate thread

core WEB programming

Questions?

Chapter 2

Accessing Web Resources using URL Connections

Advanced Topics in Java

Khalid Azim Mughal
khalid@ii.uib.no
http://www.ii.uib.no/~khalid/atij/

Version date: 2003-10-01

Overview

- HTTP-support for Client-Side Applications through the following classes:
  - `URL`
  - `URLConnection`
  - `HttpURLConnection`
- Usefulness of `URLEncoder`/ `URLDecoder` classes.

URL Connections

- *Client-side* support for accessing and retrieving web resources.
- Encapsulate much of the low-level (TCP/IP stack) complexity involved in accessing web resources.
- Support for URL connections is provided in the `java.net` package by the following important classes:
  - `URL`
  - `URLConnection`
  - `HttpURLConnection`

Universal Resource Identifier: URI

- A URI is a superset of URL and URN. It is an identifier that identifies a resource. The resource may or may not exist. Neither does it imply how we can retrieve the resource.
  `ATIJ/lecture-notes-kam/atij-application-protocols`

Universal Resource Locator: URL

- A URL specifies a unique address/location for a resource on the Web.
- Common form:

  <protocol>`://`*<hostname>*`[:`*<TCP port number>*`]/`*<pathname>*`[?`*<query>*`][#`*<reference>*`]`

  `http://www.ii.uib.no:80/~khalid/pgjc2e/`

  `mailto:khalid@ii.uib.no?Subject=Urgent%20Message`

  `http://www.w3.org/TR/REC-html32#intro` <--- Tag to indicate particular part of a document.

Universal Resource Name: URN

- A URN is a unique identifier that identifies a resource, irrespective of its location and mode of retrieval.
  `ISBN: 0-201-72828-1`

The URL class

- Represents a URL (Uniform Resource Locator), i.e. a unique address/location to access a web resource.

  <protocol>`://`*<hostname>[*`:`*<port>]*`/`*<pathname>[*`?`*<query>][*`#`*<reference>]*

- A web resource can be:
  - a file
  - a directory
  - a query to a database or to a search engine

Note that an `URL` *instance need not represent a valid resource, but it must contain the following components: protocol, hostname and pathname.*

URL Constructors

- All constructors throw a `java.net.MalformedURLException` if the *protocol* is missing or unknown.
- If the port is not specified, the default port for the protocol is assumed.
- When constructing a `URL`, an appropriate stream protocol handler (`URLStreamHandler`) is automatically loaded.

Constructor	Example

`URL(String urlStr)`
`throws MalformedURLException`
	`URL url3 = new URL( "http://www.bond.edu.au" +`
	`"/it/subjects/subs-pg.htm#inft718" );`

`URL(String protocol,`
`String hostname,`
`String filename)`
`throws MalformedURLException`
	`URL url4 = new URL( "ftp",`
	`"www.javaworld.com",`
	`"javaforums/ubbthreads.txt" );`

Constructor	Example

`URL(String protocol,`
`    String hostname,`
`    int portNumber,`
`    String filename)`
`throws MalformedURLException`
	`URL url9 = new URL( "http",`
	`                    "java.sun.com",`
	`                     80,`
	`                    "/j2se/1.4.2/docs/api/index.html" );`

`URL(URL context,`
`    String spec)`
`throws MalformedURLException`
	`URL url5 = new URL( "http://www.ii.uib.no" );`
	`URL url6 = new URL( url5, "undervisning" );`
	`//Final URL: "http://www.ii.uib.no/undervisning"`
	
	`URL url10 = new URL( null, // Same as first constructor.`
	`                    "http://java.sun.com" +`
	`                    "/j2se/1.4.2/docs/api/index.html" );`

Misc. URL Methods

- Get the different components of the URL instance (*See* URLParser.java).

```
String getProtocol()
String getHost()
String getPort()
String getFile()
String getPath()
String getQuery()
String getRef()
```

If no port is present, -1 is returned by the getPort() method.

If no file name or path is present, empty string is returned.

The string returned by the getFile() method has the query, if any, but the getPath() method excludes the query.

If no query or reference is present, null is returned.

- Compare URL instances.

```
boolean equals(Object obj)

boolean sameFile(URL url)
```

The equal() method can block as it requires name resolution.

The sameFile() method excludes the reference component.

- Convert a URL to a string.

```
String toString()
String toExternal()
```

Both methods give identical results.

Retrieving a Resource via an URL

- Open an input stream to retrieve the resource identified by the `URL` instance.

`InputStream openStream()`	Establishes a connection with the server and returns an input stream to retrieve the source. *See* `FetchResourceViaURL.java`.

- Retrieve the contents of resource identified by the `URL` instance.

`Object getContent()`
`       throws IOException`
	The method is equivalent to `openConnection().getContent()`.
	See `FetchResourceViaMethodgetContent.java`.
	See also class `URLConnection`.

- Return an `URLConnection` instance which can be used to retrieve the contents of resource identified by the `URL` instance.

`URLConnection openConnection()`	The method does *not* establish any connection to retrieve the resource. *See* class `URLConnection`.

- The URL class *only* provides an input stream to retrieve the contents of the resource.
  - Other information about the request sent or the response received is not accessible.

The `URLConnection` Class

- A `URLConnection` represents a communications link between the application and a URL.
- A `URLConnection` allows access to all pertinent information about the requests it sends and the responses it receives.
  - Allows interaction with the resource and makes querying of requests and responses possible.
- The class is abstract, and a concrete `URLConnection` is obtained via an `URL` instance.

```
URL url = new URL( urlStr );
URLConnection connection = url.openConnection();
// No connection established so far.
```

Misc. `URLConnection` Methods

- Customizing setup parameters for the connection.

`void setIfModifiedSince(long time)` Only fetches data that has been modified since the specified time (in seconds, from midnight, GMT, 1970-01-01).

`void setUseCaches(boolean permit)` If `permit` is `true` (default), the connection can cache documents.

`void setDoInput(boolean status)` If `status` is `true` (default), then the connection can be used to receive a response.

`void setDoOutput(boolean status)` If `status` is `true`, then the connection can be used to send a request. The default status is `false`.

```
void setAllowUserInteraction(
        boolean allow)
```

If `allow` is `true`, then the user can be password authenticated.

- Customizing general request header fields

```
void setRequestProperty(
        String key, String value)
```

The key/value pair must be permissible according to the protocol.

The `set`-methods above have corresponding `get`-methods.

- Establishing a connection to the remote resource.

```
void connect()
     throws IOException
```

Establishes connection and retrieves response header fields.

The call is ignored if the connection is already established.

- Querying response header information.

`String getHeaderFieldKey(int n)` Returns header field key at index `n` (n>=0), or `null` for invalid n.

`String getHeaderField(int n)` Returns header field value at index `n` (n>=0), or `null` for invalid n.

```
String getHeaderField(
          String field)
```

Returns the value of the `field`.

`Map getHeaderFields()` Returns an unmodifiable `Map` of header field name - value entries.

```
String getContentLength()
String getContentType()
String getContentEncoding()
String getDate()
String getExpiration()
String getLastModified()
```

Return the value of a specific response header field.

- Obtaining the input and output streams of the connection.

```
InputStream getInputStream()
          throws IOException
OutputStream getOutputStream()
          throws IOException
```

- Obtaining the contents of the requested resource.

```
Object getContent()
          throws IOException
```

A suitable content handler is chosen depending on the content type.

Retrieving a Resource via an `URLConnection`

- *See* `FetchResourceViaURLConnection.java`.

1. Create an `URL` instance with the address of the resource.

```
      url = new URL( urlStr );
```

2. Obtain an `URLConnection` from the `URL` instance.

```
      URLConnection connection = url.openConnection();
```

3. Customize any request fields.

```
      connection.setRequestProperty("User-Agent",
                        "Mozilla/4.0 (compatible; JavaApp)");
      connection.setRequestProperty("Referer",
                        "http://www.ii.uib.no/");
      connection.setUseCaches(false);
```

4. Establish a connection to the remote resource, which also sends the request.
   – A response will be issued by the server.

```
      connection.connect();
```

5. Query the response header information.

```
System.out.println("Content-Type:     "
                   + connection.getContentType());
System.out.println("Content-Length:   "
                   + connection.getContentLength());
System.out.println("Content-Encoding: "
                   + connection.getContentEncoding());
System.out.println("Date:             "
                   + connection.getDate());
System.out.println("Expiration-Date:  "
                   + connection.getExpiration());
System.out.println("Last-modified:    "
                   + connection.getLastModified());
```

   - Alternatively, header fields can also be looked up using a map. Following code prints all the header fields:

```
Map allFields = connection.getHeaderFields();
System.out.println("No. of field headers: " + allFields.size());
System.out.println(allFields);
```

6. Obtain an input stream to access the resource content.

```
InputStream input = connection.getInputStream();
reader = new BufferedReader(
            new InputStreamReader(input));
System.out.println("Reading the contents ...");
for(;;) {
  String line = reader.readLine();
  if (line == null) break;
  System.out.println(line);
}
```

   - Alternatively, we use the `getContent()` method. *See* `FetchResourceViaMethodgetContent.java`.

The `HttpURLConnection` Class

- The HttpURLConnection class is a subclass of the URLConnection class.
- It provides *HTTP-specific* functionality for dealing with HTTP requests and responses.
- The class defines constants for the HTTP response codes that can occur is a response status line.

```
HttpURLConnection.HTTP_OK              // HTTP Status-Code 200: OK
HttpURLConnection.HTTP_NOT_FOUND       // HTTP Status-Code 404: Not Found
HttpURLConnection.HTTP_NOT_IMPLEMENTED // HTTP Status-Code 501: Not Implemented
```

- As the class does not have a public constructor, a `HttpURLConnection` is often obtained as follows:

```
URL url = new URL( urlStr );                          // Create a URL.
URLConnection connection = url.openConnection();      // Get an URLConnection.
if (connection instanceof HttpURLConnection) {       // Is it a HttpURLConnection?
   HttpURLConnection httpConnection = (HttpURLConnection) connection;
   // Can access http-functionality of the connection.
}
```

  – If the *protocol* of the `URL` is HTTP then the `URLConnection` returned is a `HttpURLConnection`.

Misc. `HttpURLConnection` Methods

- In addition to inheriting methods from the `URLConnection` class, the `HttpURLConnection` overrides some methods from the superclass and also defines some HTTP-specific methods of its own.

`void setRequestMethod(`
`String method)`
`throws ProtocolException`
	Sets the request method to use for the connection. The request method is be subject to the protocol restrictions. Default method is GET.

`String getRequestMethod()`	Returns the request method that will be used.
`void connect()`	Inherited from the superclass `URLConnection`. It establishes a connection and sends the request, with the server subsequently issuing the response.

`int getResponseCode()`
`throws IOException`
	Returns the response code in the status line.

`String getResponseMessage()`
`throws IOException`
	Returns the status message from the status line.

`void disconnect()`	Future requests are unlikely on this connection.

- The procedure for retrieving a resource using a `HttpURLConnection` is very similar to that of using a `URLConnection`, with the added functionality of accessing HTTP features.

  See `FetchResourceViaHttpURLConnection.java`.