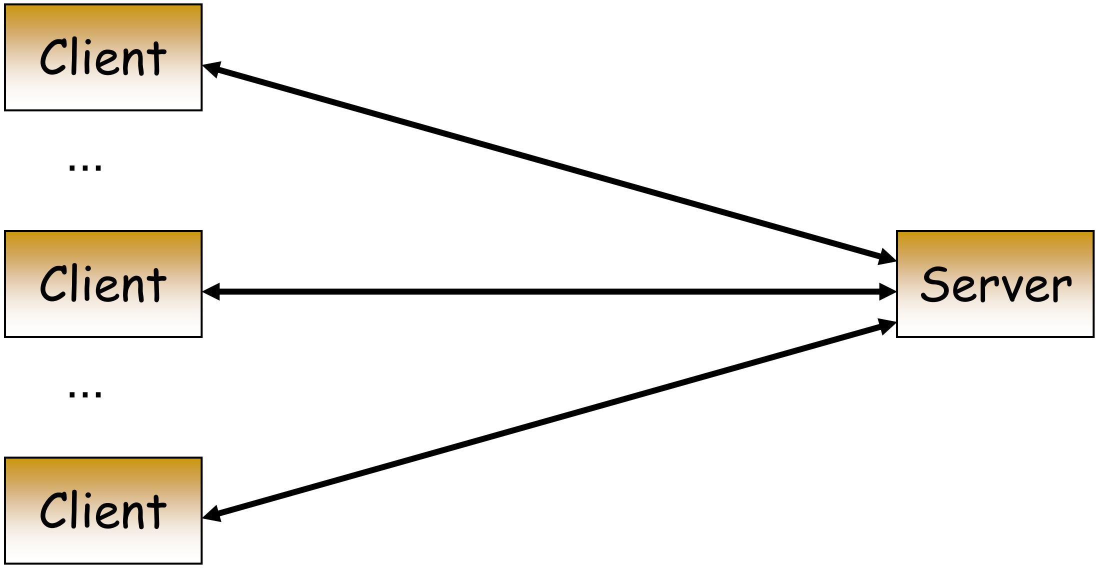
Application Layer III (CS-471)

Week 4

Network Application Development

Client-Server Model

- Most network applications use the client-server model.



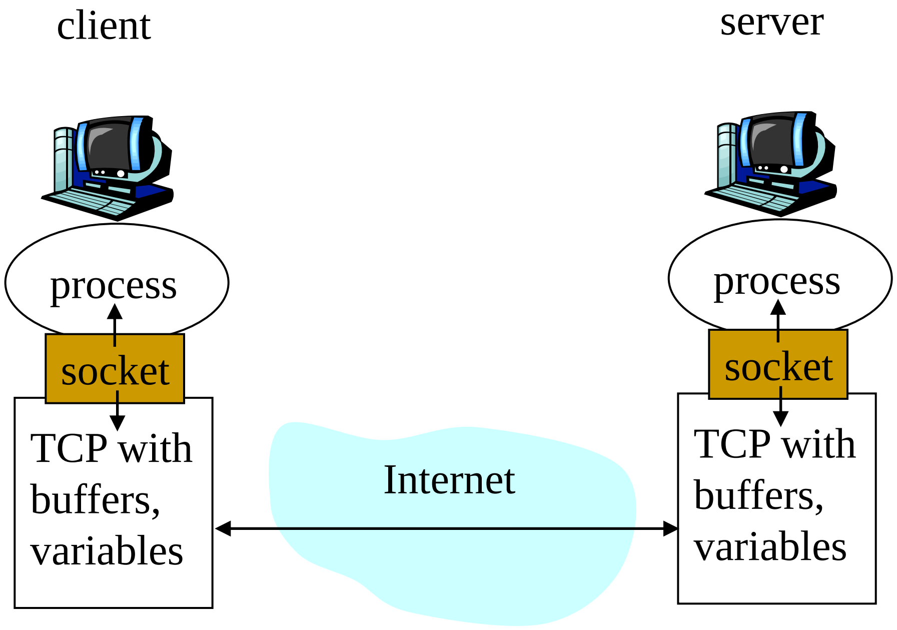
  - Clients usually communicate with one server a time
  - It is not unusual for a server to be communicating with multiple clients

Socket

- The system calls for establishing a connection are different for the client and the server
- But both involve the basic construct of a socket.

Sockets

- Process sends/receives messages to/from its socket
- Socket analogous to door
  - Sending process shoves message out door
  - Transport infrastructure brings message to the door at receiving process



Addressing Processes

- For a process to receive messages, it must have an identifier.

Addressing Processes

- For a process to receive messages, it must have an identifier.
- Identifier includes both the IP address and port number associated with the process on the host.
  - A host has an IP address
  - Does the IP address of the host on which the process runs suffice for identifying the process?
    - Answer: no, many processes can be running on same host
  - Port: A 16-bit number to identify the application process that is a network endpoint.

IP Address (IPv4)

- An identifier for each machine connected to an IP network.
  - 32 bit binary number
  - Represented as dotted decimal notation:
    - 4 decimal values, each representing 8 bits (octet), in the range 0 to 255.
- Example:
  - Dotted Decimal: 140.179.220.200
  - Binary: 10001100.10110011.11011100.11001000

Ports

- A 16-bit number to identify the application process that is a network endpoint.
- Reserved ports or well-known ports (0 to 1023)
- Standard TCP ports for well-known applications: Telnet (23), ftp(21), http (80).
- Ephemeral ports (1024-65535) : for ordinary user-developed programs.

Establish A TCP Socket on the Client Side

- Create a socket with the socket() system call
- Specify server's IP address and port
- Establish connection with server using the connect() system call
- Send and receive data, e.g., use the read() and write() system calls.

Socket()

- Create a socket with the socket() system call

  //Contains data definitions and socket structures.

  ```
  #include <sys/socket.h>
  int socket(int family, int type, int protocol)
  ```

  Returns: non-negative descriptor if OK, -1 on error

  - Integer descriptor: identify the socket in all future function calls
  - Protocol family constants
    - e.g. AF_INET: IPv4 protocol, AF_INET6: IPv6 protocol.
  - Type of socket
    - SOCK_STREAM: stream socket, SOCK_DGRAM: datagram socket
  - Protocol: normally 0 except for raw socket

Specify Server's IP Address and Port

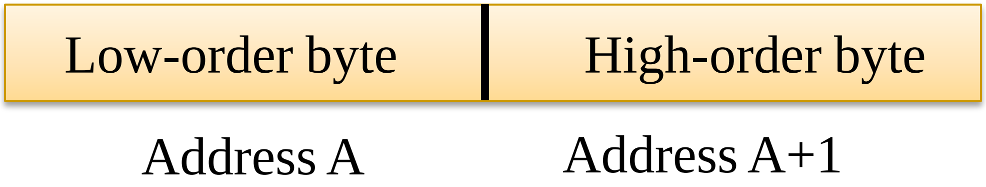
- Specify server's IP address and port
- E.g. for TCP connection:

```
 struct sockaddr_in servaddr;
 //set the socket address structure 0
 memset(&servaddr, 0, sizeof(servaddr));
 //set the address family to AF_INET
 servaddr.sin_family = AF_INET;
 //set the port number.
 servaddr.sin_port = htons(<port number>);
 //set the ip address.
if (inet_pton(AF_INET, <ip addr>, &servaddr.sin_addr) <= 0)
```

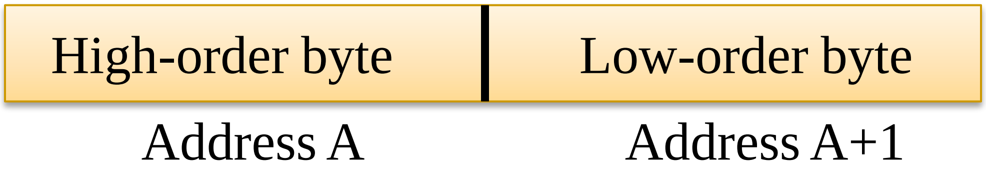
Network-Byte Ordering

Two ways to store 16-bit/32-bit integers

- Little-endian byte order (e.g. Intel)



- Big-endian byte order (E.g. Sparc)



Network-Byte Ordering (cont.)

- How do two machines with different byte-orders communicate?
  - Using network byte-order
  - Network byte-order = big-endian order
- Converting between the host byte order and the network byte order (<netinet/in.h>)
  - h: host: s: short, l: long
    - uint16_t htons(uint16_t n)
    - uint32_t htonl(uint32_t n)
    - uint16_t ntohs(uint16_t n)
    - uint32_t ntohl(uint32_t n)

Specify Server's IP Address and Port

- Specify server's IP address and port
- E.g. for TCP connection:

```
struct sockaddr_in servaddr;
//set the socket address structure 0
memset(&servaddr, 0, sizeof(servaddr));
//set the address family to AF_INET
servaddr.sin_family = AF_INET;
//set the port number.
servaddr.sin_port = htons(<port number>);
//set the ip address.
if (inet_pton(AF_INET, <ip addr>, &servaddr.sin_addr) <= 0)
```

Inet_pton, inet_ntop

```
<arpa/inet.h>
//Returns 1 if OK, 0 if input is not a valid format, -1 on error
int inet_pton(int family, const char *strptr, void *addrptr);
//Returns the pointer to result if OK, NULL on errors
const char *inet_ntop(int family, const void *addrptr, size_t len);
```

- p: presentation
  - Usually an ASCII string
- n: network
  - Binary value that goes into a socket address structure

Connect()

- Establish a connection with the TCP server using the connect() system call

```
#include <sys/socket.h>
int connect(int sockfd,  const struct sockaddr *servaddr,
socklen_t addrlen);
```

Return 0 if OK, -1 on error

read(), write()

- Send and receive data, e.g., use the write() and read() system calls.

```
//Read up to count bytes from the socket into the buffer
// Return the number of bytes read
int read(int sockfd, void *buf, int count);

// Write data to a TCP connection
int write(int sockfd, void *buf, int count)
```

Establish A Socket on the Server Side

1. Create a socket with the socket() system call
2. Bind the socket to an address using the bind() system call.
3. Listen for connections with the listen() system call
4. Accept a connection with the accept() system call.
5. Send and receive data

bind(), listen()

- The server specifies the IP address and port number associated with a socket using bind().

  int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen)

- Listen for connections with the listen() system call.

  int listen(int sockfd, int backlog)

  backlog: the number of maximum pending clients

Accept()

- Accept a connection with the accept() system call.

  int accept(int sockfd, struct sockaddr *client_addr, socklen_t *addrlen)

- accept() returns a new descriptor that is automatically created by the kernel. This descriptor refers to the TCP connection with the client.

Example of Client-Server Operation

A Simple Daytime
Client and Server

Daytime client

- Connects to a daytime server
- Retrieves the current date and time

```
% cli  128.226.6.4
Thu 09  02 17:30:00 2010
```

Daytime client

```
int main(int argc, char **argv) {
     int  sockfd, n;
     char recvline[MAX + 1];
     struct sockaddr_in servaddr;
    if( argc != 2 ) {
         printf("Usage: cli <IP address>");
         exit(1); }
     /* Create a TCP socket */
      if((sockfd=socket(AF_INET,SOCK_STREAM, 0)) < 0){
                 perror("socket"); exit(2);}
     /* Specify server's IP address and port */
     memset(&servaddr, 0, sizeof(servaddr));
     servaddr.sin_family = AF_INET;
     servaddr.sin_port = htons(10000); /* daytime server port */
     if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
                  perror("inet_pton"); exit(3);}
    /* Connect to the server */
    if (connect(sockfd,  (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
            perror("connect"); exit(4); }

     /* Read  from socket */
     while ( (n = read(sockfd, recvline, MAX)) > 0) {
           recvline[n] = '\0';        /* null terminate */
           printf("%s", recvline);
     }

      if (n < 0) { perror("read"); exit(5); }
      close(sockfd);
}
```

Daytime Server

1. Waits for requests from Client
2. Accepts client connections
3. Sends the current time
4. Terminates connection and goes back waiting for more connections.

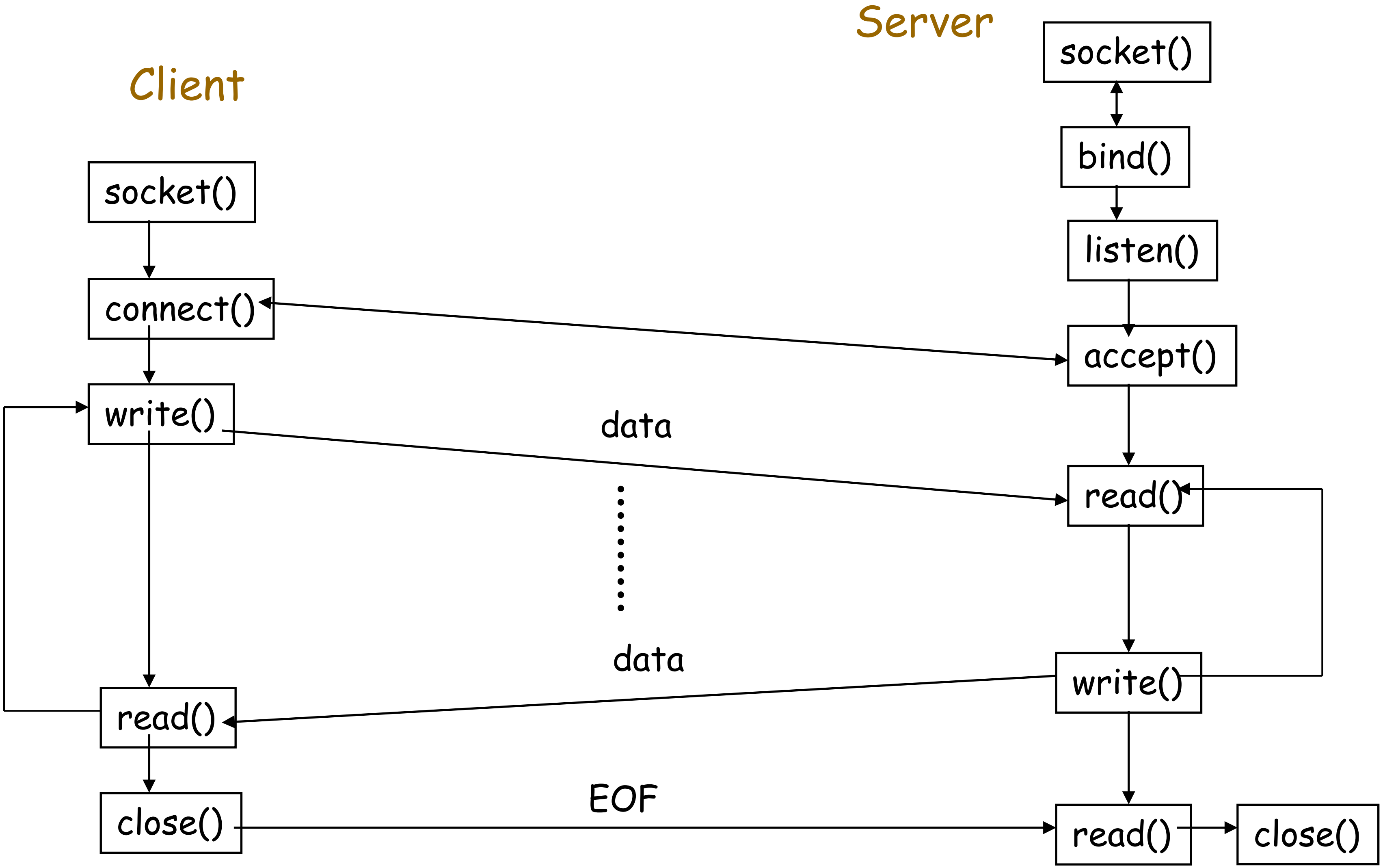
```
int main(int argc, char **argv) {
        int   listenfd, connfd;
        struct sockaddr_in servaddr, cliaddr;
        char buff[MAX];
        time_t ticks;
        /* Create a TCP socket */
        listenfd = socket(AF_INET, SOCK_STREAM, 0);
       /* Initialize server's address and well-known port */
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family  = AF_INET;
       /* allowed your program to work without knowing the IP address
      of the machine it was running on  */
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port  = htons(10000);   /* daytime server */
        /* Bind server's address and port to the socket */
        bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
```

```
    /* Convert socket to a listening socket – max 100 pending clients*/
  listen(listenfd, 100);

  for ( ; ; ) {
    /* Wait for client connections and accept them */
    clilen = sizeof(cliaddr);
     connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen);

    /* Retrieve system time */
    ticks = time(NULL);
    snprintf(buff, sizeof(buff), "%s\r\n", ctime(&ticks));

    /* Write to socket */
    write(connfd, buff, strlen(buff));

    /* Close the connection */
    close(connfd);
  }
}
```

Run Daytime Client-Server

- gcc cli.c -o cli
- gcc ser.c -o ser

TCP Connection Sequence



Summary: Socket API

- int socket(int family, int type, int protocol)
  - Creates a socket
- int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen)
  - Enables a client to connect to a server.
- int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen)
  - Allows a server to specify the IP address/port_number associated with a socket
- int listen(int sockfd, int backlog)
  - Allows the server to specify a socket that can be used to accept connections.
- int accept(int sockfd, struct sockaddr *client_addr, socklen_t *addrlen)
  - Allows a server to wait till a new connection request arrives.
- int close(int sockfd)
  - Terminates any connection associated with a socket and releases the socket descriptor.

UDP Sockets: Sending and Receiving

- **ssize_t sendto(int** *sockfd*, **const void ****buf*, **size_t** *len*, **int** *flags*, **const struct sockaddr ****dest_addr*, **socklen_t** *addrlen***);**
  - “Send the information in buffer buf of size len, to the socket sockfd and corresponding address stored in dest_addr structure of size addr_len”

- **ssize_t recvfrom(int** *sockfd*, **void ****buf*, **size_t** *len*, **int** *flags*, **struct sockaddr ****src_addr*, **socklen_t ****addrlen***)**
  - Receive len bytes from socket sockfd and corresponding address stored in src_addr, and store the bytes in buffer buf.

UDP Sockets: Example

- Example: see files udpserver.cpp and udpclient.cpp on Titanium.

Establish A UDP Socket on the Client Side

- Create a socket with the socket() system call
- Specify server's IP address and port
- Send and receive data, e.g., use the sendto() and recvfrom() system calls.

Establish A UDP Socket on the Server Side

1. Create a socket with the socket() system call
2. Bind the socket to an address using the bind() system call.
3. Send and receive data, e.g., use the sendto() and recvfrom() system calls.

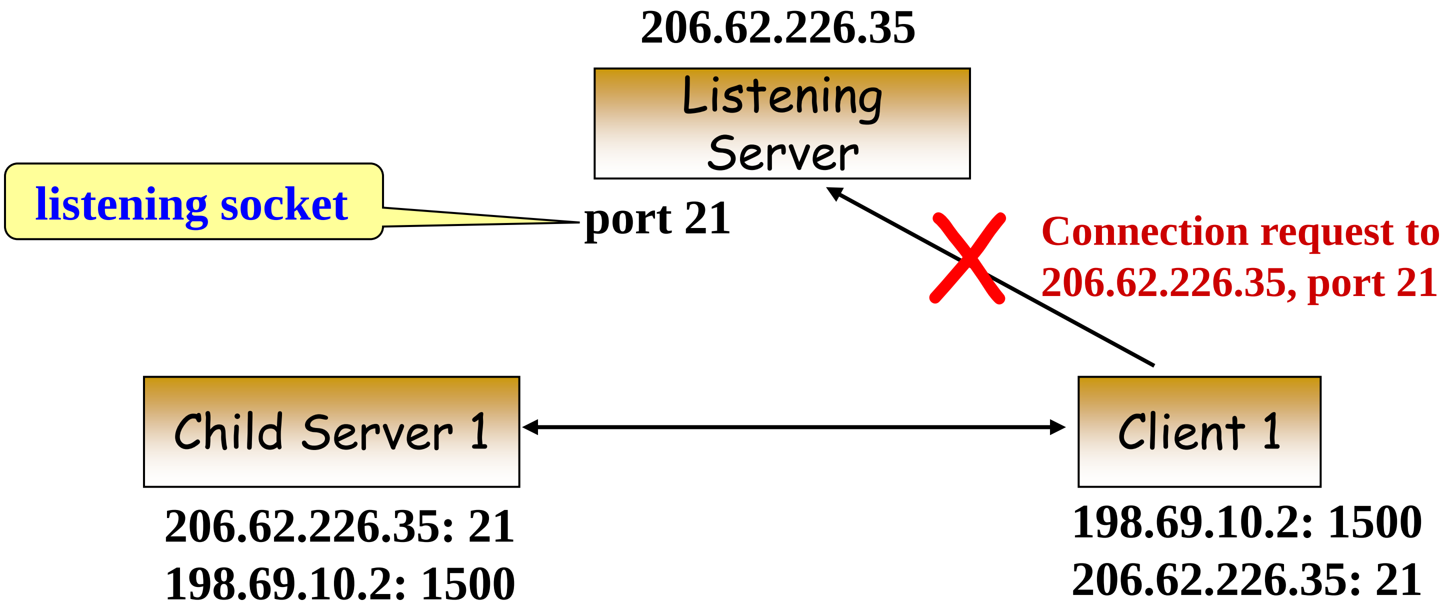
Concurrent Servers

- Daytime client-server: iterative servers
- Concurrent Servers: handle multiple clients simultaneously
  - Fork
  - Threads

Concurrent Servers

- Daytime client-server: iterative servers
- Concurrent Servers: handle multiple clients simultaneously
  - Fork
  - Threads

Forking Concurrent Servers



206.62.226.35

Listening Server

listening socket

port 21

Connection request to 206.62.226.35, port 21

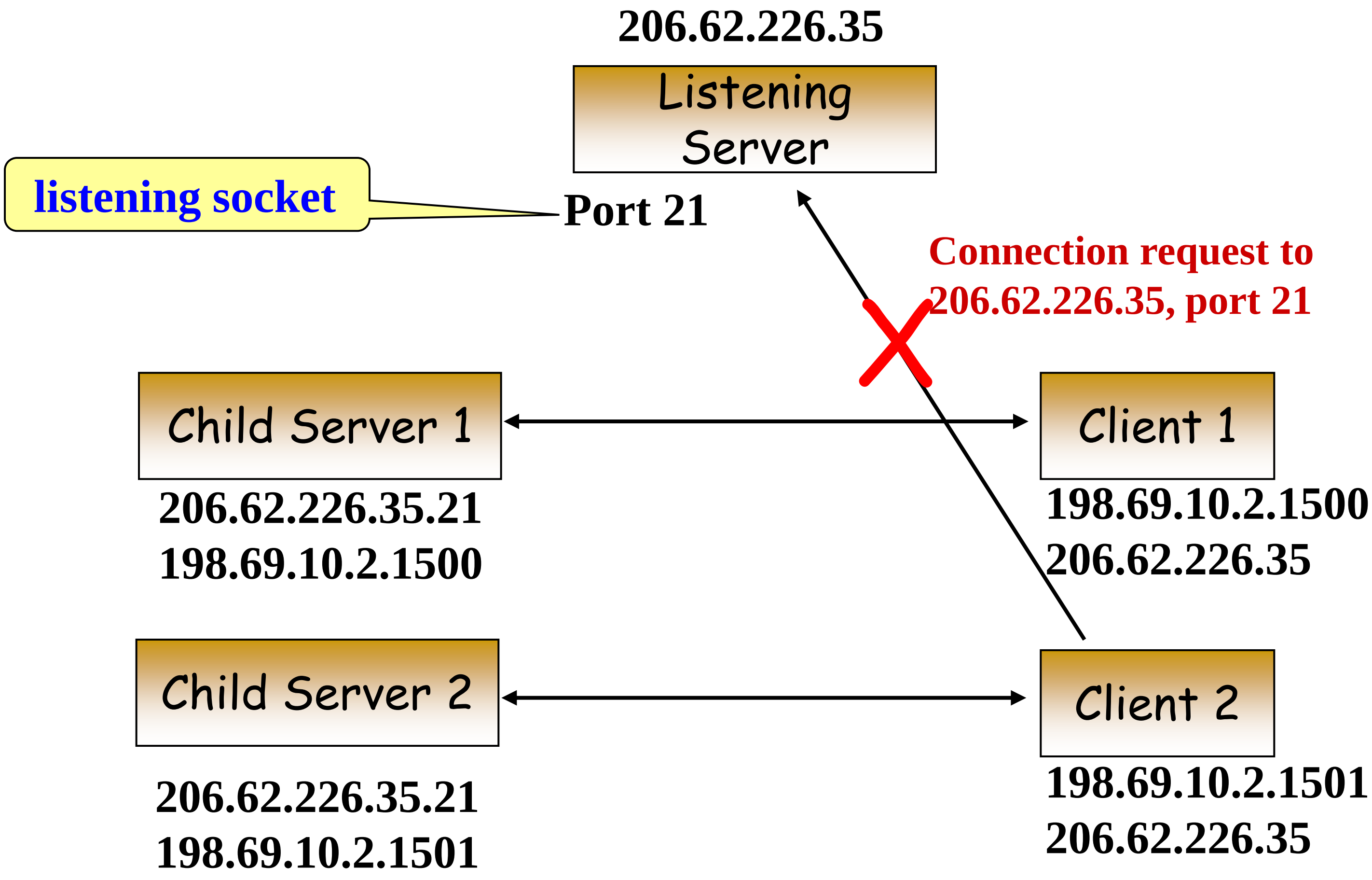
Child Server 1

206.62.226.35: 21
198.69.10.2: 1500

Client 1

198.69.10.2: 1500
206.62.226.35: 21

Forking Concurrent Servers



Forking Server Example

```
listenfd = socket( … )
bind( listenfd, … )
listen(listenfd,…);
for ( ;; ) {
    /* wait for client connection */
    connfd = accept(listenfd,…);
    if( (pid = fork() ) == 0) {
        /* Child Server */
        close(listenfd);            //child closes listening socket
        service_client(connfd);//process the request
        close(connfd);              //done with this client
        exit(0);                    //child terminates
    }
    /* Parent */
    close(connfd);}                 //parent closes connected socket
```

Java Socket Programming: An Example

Client

```
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String modifiedSentence;
        Socket sock = new Socket(“ecs.fullerton.edu", 6789);
      /*Open an input and output stream to the socket. */
       PrintWriter out =
                new PrintWriter(sock.getOutputStream(),true);
       BufferedReader in =
                new BufferedReader(
             new InputStreamReader(sock.getInputStream()));
```

Client

```
  /*Writes out the string to the underlying output stream. */
   out.println("hello");
    /*Read a line of text*/
    modifiedSentence = in.readLine();
    System.out.println("FROM SERVER: “ + 
 modifiedSentence);
    sock.close();
}}
```

Server

```
import java.io.*;
import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception{
        String clientSentence, capitalizedSentence;
        ServerSocket listen = new ServerSocket(6789);
        while(true) {
          Socket conn = listen.accept();
           BufferedReader in = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
          PrintWriter out =
               new PrintWriter(conn.getOutputStream(),true);
          clientSentence = in.readLine();
          System.out.println("FROM CLIENT:" + clientSentence);
          capitalizedSentence = clientSentence.toUpperCase();
          out.println(capitalizedSentence);
          conn.close();
        }}}
```

References

- Package java.io
  - http://java.sun.com/j2se/1.4.2/docs/api/java/io/package-summary.html
- Java socket programming:
  - http://java.sun.com/docs/books/tutorial/networking/sockets/
- Tutorials and examples
  - http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html
  - http://java.sun.com/docs/books/tutorial/networking/sockets/
  - http://www.prasannatech.net/2008/07/socket-programming-tutorial.html
  - http://zerioh.tripod.com/ressources/sockets.html
  - http://java.sun.com/docs/books/tutorial/essential/io/

References

- I/O stream (byte stream, character stream, buffered stream)
  - http://www.javapassion.com/javase/javaiostream.pdf

Acknowledgement

- Some slides are borrowed from Dr. Ping Yang