CS458/CS558: Introduction to Security

- This class
  - Socket Programming (C, Cont)
  - Assignment 1

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_INET1: 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
bzero(&servaddr, 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)

Low-order byte	High-order byte
Address A	Address A+1

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

High-order byte	Low-order byte
Address A	Address A+1

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
bzero(&servaddr, 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

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  128.226.6.39
Tue Sep  06 17:00:00 2011
```


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 */
    bzero(&servaddr, 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 */
        bzero(&servaddr, 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));
```

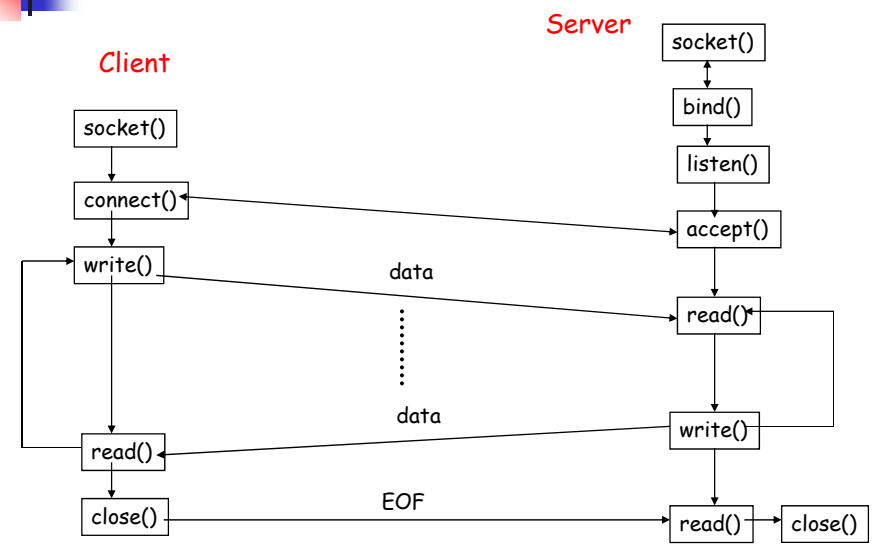


```
        /* Convert socket to a listening socket – max 100 pending clients*/
    listen(listenfd, 100);

    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 -lresolv -lsocket -lnsl
- gcc ser.c -o ser -lresolv -lsocket -lnsl

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.

Reference

- http://beej.us/guide/bgnet/output/html/multipage/index.html
- google: "socket programming" tutorial C
- Java socket programming: http://java.sun.com/docs/books/tutorial/networking/sockets/

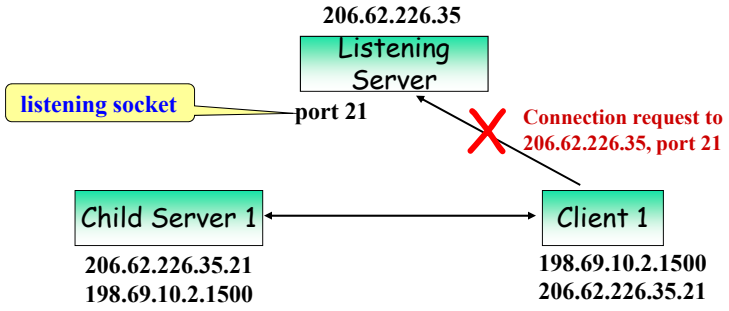
Concurrent Servers

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

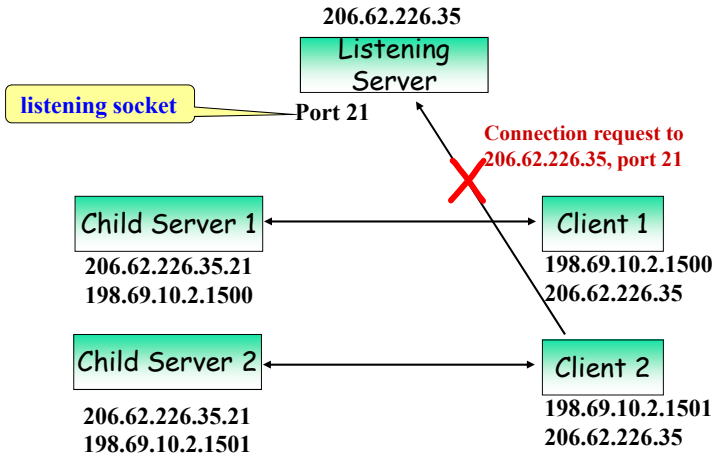
Concurrent Servers

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

Forking Concurrent Servers



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("bingsuns.binghamton.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();
        }}}
```


Java on Bingsuns

- Different versions of Java on bingsuns: 1.4, 1.5, 1.6.
- We will use Java version 1.6 (/usr/bin).
- Add alias java /usr/bin/java and alias javac /usr/bin/javac in .cshrc

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

Assignment 1

Due: 09/25/2011 (Sunday)

1. Learn how to write and use makefile

http://www.delorie.com/djgpp/doc/ug/larger/makefiles.html (C)
http://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html (Java)

Assignment 1

2. Implement a telnet client and an iterative telnet server using TCP sockets.
   - Support commands: ls, cd, pwd, mkdir, and exit.

telnet > ls //lists contents of the directory on the server side

telnet > cd <directory-absolute-path> //change the current working
// directory to absolute path named <directory-absolute-path>.
// E.g. cd /home/path

telnet > cd <directory-relative-path> //change the current working
//directory to a path that is relative to the current directory.
// E.g. cd path

telnet > cd .. //move up one folder

telnet > pwd //print working directory

telnet > mkdir <directory-relative path> //create a directory

telnet > exit

Assignment 1

- The server is invoked as: telnet-serv <port_number>
  - <port_number>: the port at which the server accepts connection requests.
  - The server is an iterative server.

Assignment 1

- The client is invoked as: telnet-cli <server_domain> <server_port>
  - <server_domain>: the domain name of the machine hosting the server. If you use C/C++, you will need to convert the domain to the 32-bit IP address using gethostbyname(…) etc.
    http://retran.com/beej/gethostbynameman.html
  - <server_port>: the port number on which server is listening.
- Upon connecting to the server, the client prints out telnet >, which allows the user to type commands.

Assignment 1

- CS558: Error handling
  - Invalid commands, i.e., commands other than ls, cd, pwd, mkdir, and exit.
  - mkdir <dir name>: directory name <dir name> already exists.
  - cd <dir name>: directory name <dir name> does not exist.

Submission Guideline

- Hand in your source code and a Makefile electronically. You must make sure that this code compiles and runs correctly on bingsuns.binghamton.edu.
- If you use C/C++, the Makefile must give the executable server code the name telnet-serv and the executable client code the name telnet-cli.
- If you use Java, generate Serv.class and Cli.class.

Submission Guideline

- Write a README file (text file, do not submit a .doc file):
  - You name and email address.
  - Whether your code has been tested on bingsuns.
  - The language you are using (C/C++/Java)
  - How to execute your program.
  - Anything special about your submission that the TA should take note of.

Submission Guideline (Cont.)

- Place all your files under one directory with a unique name (such as p1-<userid> for assignment 1, e.g. p1-pyang).
- Tar the contents of this directory:
  tar –cvf <directory_name>.tar <directory_name>
  E.g. tar -cvf p1-pyang.tar p1-pyang/
- Use the blackboard to upload the tared file you created above.

Grading Guideline (CS458)

- ls, mkdir, exit: 45'
- Cd, pwd: 45
- Readme, correct execution format: 4'
- Correct makefile: 6'

Grading Guideline (CS558)

- ls, mkdir, exit: 40'
- pwd, cd: 40'
- Error handling: 10'
- Readme, correct execution format: 4'
- Correct makefile: 6'