



# CS4514 B06

## HELP Session 1

Presented by Mingzhe Li  
lmz@cs.wpi.edu

CS4514 – TCP/IP Socket Programming





# Outline

- **Project 1 Overview**
- **Unix Network Programming**
  - TCP Client
  - TCP Server
- **Processing commands**
- **How to find help and other tips.**



# CS4514 Project1

- Your programs should compile and work on [ccc.wpi.edu](http://ccc.wpi.edu) computers, which are running Linux.
- Programs should be written in **C** or **C++**.
- If your program is developed on another platform or machine, you should **test** the software on **ccc** before turning in the assignment.
- Make sure you have the correct **#include** in your program.



# Project 1 missions (in handout)

- **The Client:**
  1. Reading a command from a script file or from console.
  2. Sending the command to the server.
  3. Receiving and displaying the information from the server.
  4. Writing the results to the log file *LClient.log*.



# Project 1 missions (in handout)

- **Server:**

1. **Processing the command from the client and return the result to the client.**
2. **Maintaining the records to keep the location information.**
3. **Writing the complete database to the file *LDatabase.txt* when the server received the “quit EOF” command.**



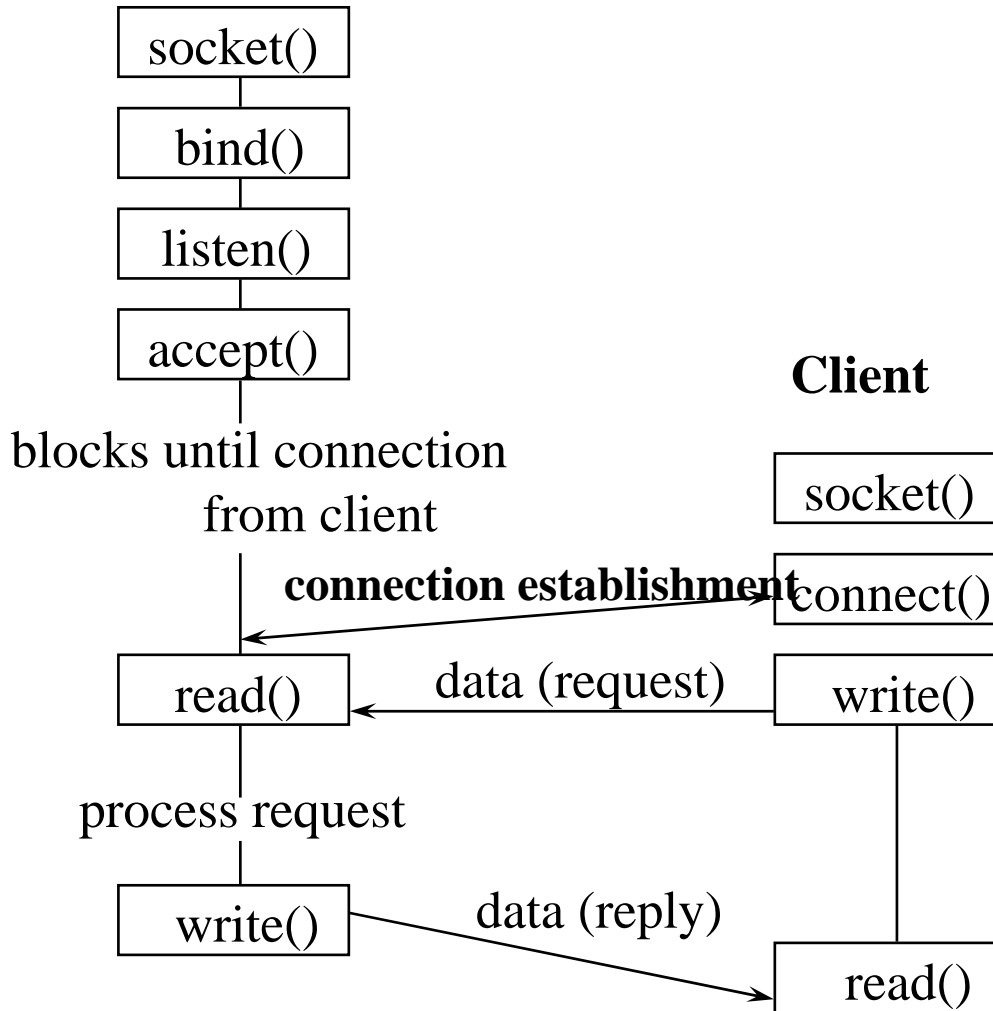
# Outline

- **Project 1 Overview**
- **Unix Network Programming**
  - **TCP Client**
  - **TCP Server**
- **Processing commands**
- **How to find help and other tips.**



# Server (connection-oriented protocol)

# Socket system calls for connection-oriented protocol ( TCP )





# What Do We Need?

- **Data communication between two hosts on the Internet require the five components :**  
*{protocol, local-addr, local-process, foreign-addr, foreign-process}*
- **The different system calls for sockets provides values for one or more of these components.**



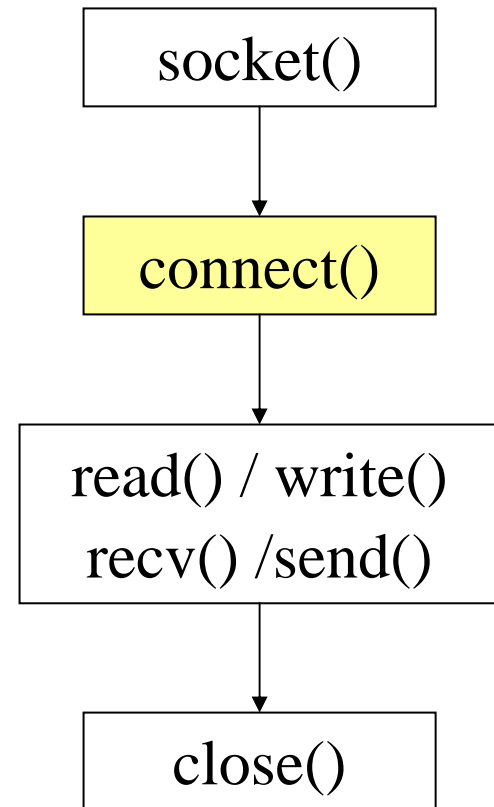
# What Do We Need ?

- The socket system call just fills in one element of the five-tuple we've looked at - the protocol. The remaining are filled in by the other calls as shown in the figure.

	<i>protocol</i>	<i>local_addr, local_process</i>	<i>foreign_addr, foreign_process</i>
Connection-Oriented Server (TCP)	socket()	bind()	accept()
Connection-oriented Client (TCP)	socket()	connect()	
Connectionless Server (UDP)	socket()	bind()	recvfrom()
Connectionless Client (UDP)	socket()	<i>bind()</i>	sendto()

# TCP Connection (Client)

- **Connection Oriented**
  - Specify transport address once at connection
- **Use File Operations**
  - `read() / write()`
  - or
  - `recv() / send()`
- **Reliable Protocol**



# Example: TCP Client

```
int sd;
struct hostent *hp;          /* /usr/include/netdb.h */
struct sockaddr_in server;  /* /usr/include/netinet/in.h */

/* prepare a socket */
if ( (sd = socket( AF_INET, SOCK_STREAM, 0 )) < 0 ) {
    perror( strerror(errno) );
    exit(-1);
}
```

# Example: TCP Client (Continued)

```
/* prepare server address */
```

```
bzero( (char*)&server, sizeof(server) );
```

```
server.sin_family = AF_INET;
```

```
server.sin_port = htons( SERVER_PORT );
```

```
if ( (hp = gethostbyname(SERVER_NAME)) == NULL) {
```

```
    perror( strerror(errno) );
```

```
    exit(-1);
```

```
}
```

```
bcopy( hp->h_addr, (char*)&server.sin_addr, hp->h_length);
```

# Example: TCP Client (Continued)

```
/* connect to the server */
```

```
if (connect( sd, (struct sockaddr*) &server, sizeof(server) ) < 0 ) {  
    perror( strerror(errno) );  
    exit(-1);  
}
```

```
/* send/receive data */
```

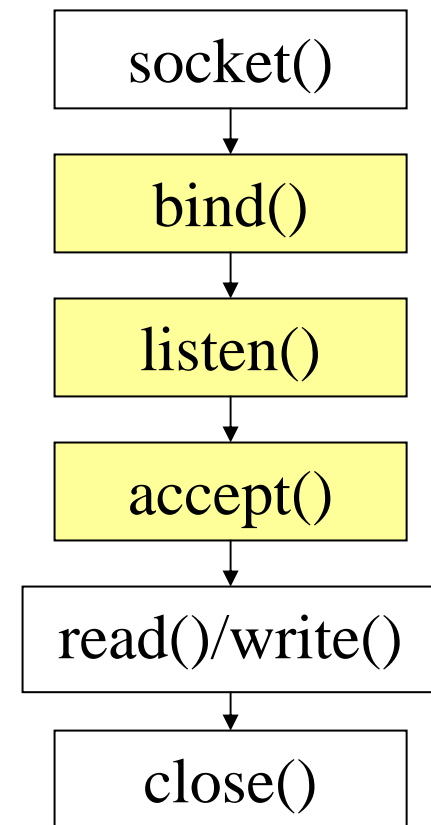
```
while (1) {  
    read/write();  
}
```

```
/* close socket */
```

```
close( sd );
```

# TCP Connection (Server)

- Bind transport address to socket
- Listen to the socket
- Accept connection on a new socket



# Example: TCP Server

```
int sd, nsd;
struct sockaddr_in server; /* /usr/include/netinet/in.h */

sd = socket( AF_INET, SOCK_STREAM, 0 );

bzero( (char*)&server, sizeof(server) );

server.sin_family = AF_INET;
server.sin_port = htons( YOUR_SERVER_PORT );
server.sin_addr.s_addr = htonl( INADDR_ANY );
```



# Example: TCP Server (Continued)

```
bind( sd, (struct sockaddr*) &server, sizeof(server) );
```

```
listen( sd, backlog );
```

```
while (1) {
```

```
    nsd = accept( sd, (struct sockaddr *) &client, sizeof(client) );
```

```
    read()/write();
```

```
    close( nsd );
```

```
}
```

```
close( sd );
```





# Outline

- **Project 1 Overview**
- **Unix Network Programming**
  - TCP Client
  - TCP Server
- **Processing commands**
- **How to find help and other tips.**



# Processing commands

- Each command triggers a communication conversion, between client and server. Then, we have
  - login
  - add
  - remove
  - quit
  - *list* (*attn: this one is different from above commands, most complex one*).



# Commands

- In the *login*, *add*, *remove*, and *quit* commands:

The server only returns one message to the client.

- In the *list* command, The server could return multiple messages to the client.

“Each entry, which meets the search condition, is sent as a separate TCP message back to the Client.”



# Login Command

- Login Command Format.  
*login*
- Login Command Handling
  - For The Client: When the Client reads a **login** command, the client establishes a TCP connection to the Server.
  - For The Server: When the Server receives a “**login**”, it replies “**Hello!**” to the client.



# Add Command

- **Add Command Format:**

***add** first\_name last\_name id\_number location*

Notes:

- **first\_name**, **last\_name**, and **location** are nonblank ascii string

For example:

Tony Smith 12\_Institute\_rd\_worcester

- **id\_number** is 9 digital number similar to SSN number.  
(example: 321654987)

- **For the Client:**

reads and sends the **add** command to the server,  
and displays the result returned from server.

# Add Command (cont'd)

- **For the Server:**

**When the server gets the **Add** command, it will**

- add the four items as an entry into the location database in the proper location, and return a successful message to client.
- If a duplicate *id\_number* is received, the server simply overwrites information associated with this *id\_number*, and returns a successful message to the server.
- If the command's parameter is not valid, the server returns an Error message to the client.

For example,

*Add Tony Smith 12033\_000 worcester MA*

→ returns “*an invalid add commands*”.



# Remove Command

- Remove command format

***remove id\_number***

*example: “ remove 123456789 “ is a valid command.*

- For the Client,

sends the **remove** command to the server,  
and displays the result returned from  
server.

# Remove command (cont'd)

- For the Server,

When the server receives **remove** command, the server searches the database for a match on ***id\_number***.

- If the ***id\_number*** entry **exists** in the database for a person, that entry is removed from the location database and a **success** message that contains the first and last name of the person removed is sent back.
- If there is **not a match** in the database, the server does not modify the database and sends an appropriate **error** message back to the Client.





# Quit Command

- **Quit Command format:**

*quit [EOF]*

For example, `quit` and `quit EOF` are valid commands.

- **For the Client**

- sends the quit command to the server, and when the client received the response message from server, the client know the connection will be closed.
- If **EOF** is specified, the client will close the log file, and terminate.



# Quit Command (Cont'd)

- **For the Server,**
  - When server received **quit** command, it sends a response back to the Client indicating that the connection will be closed. The server returns to wait for a new connection triggered by a subsequent login request.
  - If **quit EOF** is received, the Server additionally writes out the complete database to the file ***LDatabase.txt*** and then terminates.



# List Command

- **List Command format**

***list start finish***

Notes: start/finish are two single letter of the alphabet.

Examples:

- **list A N**

Find the entries, whose last names with **A** as the first letter up to and including all last names with **N** as the first letter. (such Anderson, Lincoln etc.)

- **list C C**

Find the entries whose last names begin with **C**.

- **list W A**

Invalid Command.



# List Command (cont'd)

- **For the Client:**

**Sends the command to the server, and displays the response message from the server.**

- **For the Server:**

**When received the list command:**

- sends All location entries satisfying the list limits.
- sends “no such records” if there are no entries satisfying the list request.
- sends “invalid command” if the list command is in illegal format. (for example, *list Z A*, or *list A* )



# Outline

- **Project 1 Overview**
- **Unix Network Programming**
  - TCP Client
  - TCP Server
- **Processing a command**
- **How to find help and other tips.**

# Some Useful System Calls

- ***Gethostbyname***: map hostname to IP addr  
struct hostent \*gethostbyname( char \*name )
- ***Getservbyname***: look up service name given  
struct servent \*getservbyname( const char \*servname,  
const char  
\*protocol )
- ***Gethostname***: get own hostname  
int gethostname( char \*name, size\_t len )

# Others Tips

- **Include files**

```
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/time.h>
#include <memory.h>
```

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
```

- **Programming tips**

- always check the return value for each function call
- consult the UNIX on-line manual pages ("**man**") for a complete description
- Internet: Beej's Guide to Network Programming  
<http://www.ecst.csuchico.edu/~beej/guide/net/>

31



# Server Database

There are many possible data structure choices for implementing the server data base. Two of them are:

- **Linked list:**

  - Easy to add/remove an entry.

- **Array:**

  - The simplest data structure.



# Sorting in Database

- The server's database is maintained alphabetically by **last name**.
  - There is no secondary **key** in the database. If the people have the same last name, you don't need to sort them based on their first name.
  - For example, (based on a linked list)  
“Andy Smith” and “Tony Smith” in the database.





# Case insensitive string comparison

- The case insensitive string compare functions in Linux.
  - `int strcasecmp(const char *s1, const char *s2);`
  - `int strncasecmp(const char *s1, const char *s2, size_t n);`
  - Their usage is similar to `strcmp()` function.
- An Alternative method.

Storing the information in upper case letters in server's database. (Smith → SMITH )



# HELP

- **Bring printouts to office hours.**
- **Email questions to Prof.+TA+SA  
([cs4514-staff@cs.wpi.edu](mailto:cs4514-staff@cs.wpi.edu))**
- **You CAN email a specific TA/SA, but do not expect immediate results, better to use the staff mailing list.**
- **We do have a class mailing list that could be used as a last resort.**



# Questions?



# More Tips ? File and Stdio

- In Linux, a device could be treated as a file.  
For example, the standard input device could be handled as a file.

```
/* fgets() will read a line from the keyboard.*/
```

```
    fp=stdin;
```

```
    fgets(buffer, buffer_len, fp);
```

```
/* next fgets() will read a line from the file named "script.txt" .*/
```

```
    fp=fopen("script.txt", "r");
```

```
    fgets(buffer, buffer_len, fp);
```