CS3516 B14

Help Session 1

Presented by Hao Wan
hale@wpi.edu

Outline

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

CS3516 Project1

- Your programs should compile and work on 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 “*LClient.txt*” 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.*

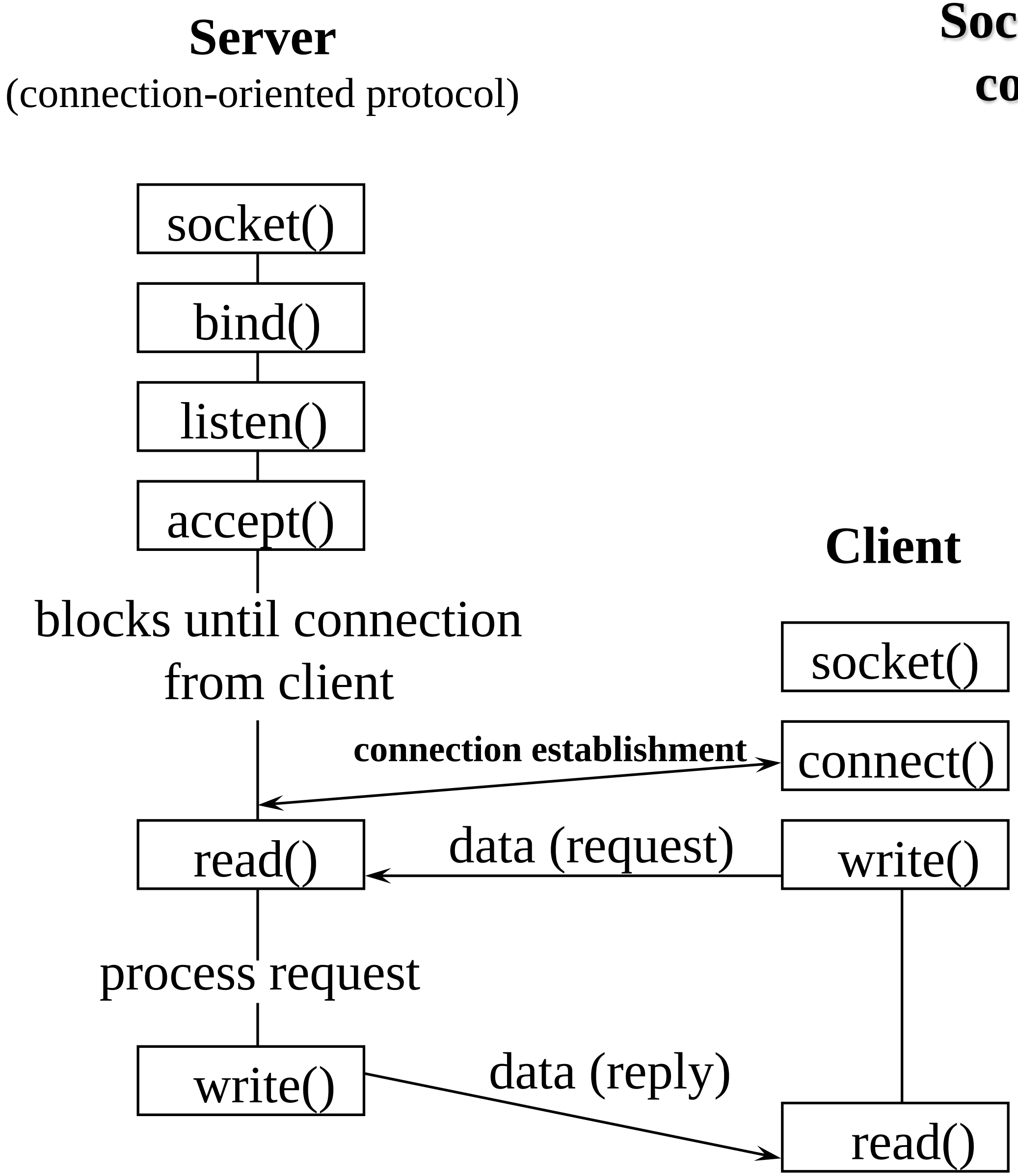
Project 1 missions (in handout)

- 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.

Socket system calls for connection-oriented protocol (TCP)



Server

(connection-oriented protocol)

socket()

bind()

listen()

accept()

blocks until connection from client

Client

socket()

connect()

connection establishment

read()

data (request)

write()

process request

write()

data (reply)

read()

What Do We Need?

- Data communication between two hosts on the Internet require the five components :

  {*protocol, local-addr, local-process, remote-addr, remote-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.

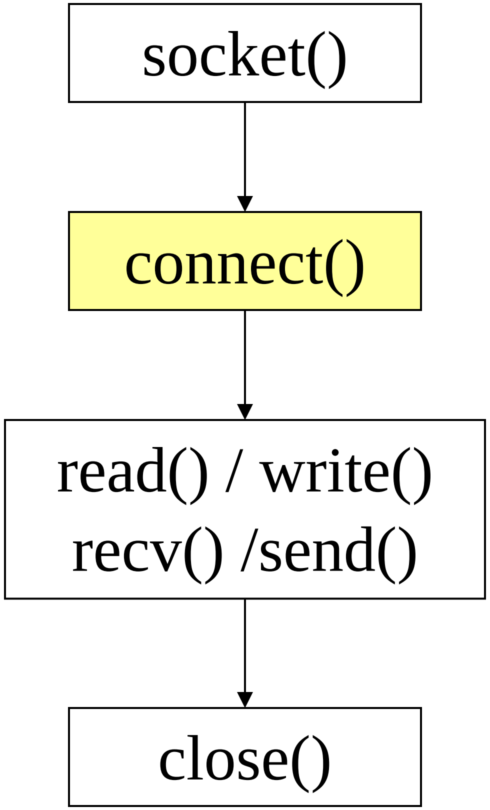
	protocol	*local_addr,* *local_process*	*remote_addr,* *remote_process*
Connection-oriented Server (TCP)	socket()	bind()	accept()
Connection-oriented Client (TCP)	socket()	connect()	
Connectionless Server (UDP)	socket()	bind()	recvfrom()
Connectionless Client (UDP)	socket()	*bind()*	sendto()

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;
struct sockaddr_in server;

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

AF_INET address family sockets can be either connection-oriented (type SOCK_STREAM) or they can be connectionless (type SOCK_DGRAM). Connection-oriented AF_INET sockets use TCP as the transport protocol. Connectionless AF_INET sockets use UDP as the transport protocol.

Example: TCP Client (Continued)

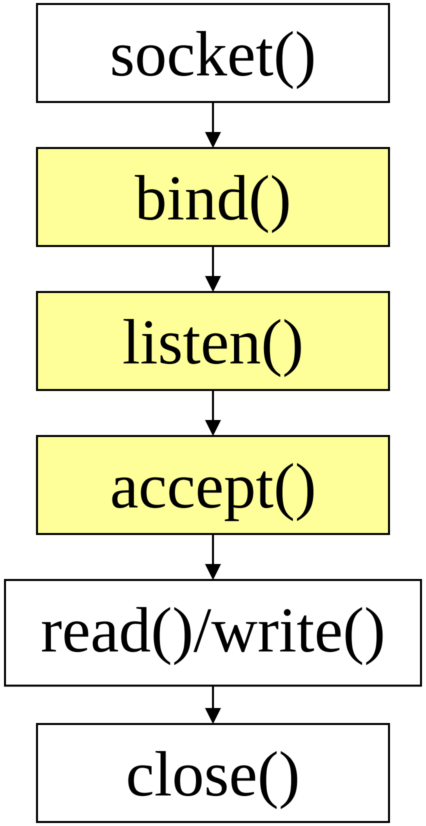
```
/* prepare server address */
bzero( (char*)&server, sizeof(server) );
server.sin_family = AF_INET;
server.sin_port = htons( SERVER_PORT );
if ( (hp = gethostbyname(SERVER_NAME)) == NULL) {
        perror( strerror(errno) );
        exit(-1);
}
bcopy( hp->h_addr, (char*)&server.sin_addr, hp->h_length);
```

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 );

unsigned int cltsize=sizeof(client);

while (1) {
   nsd = accept( sd, (struct sockaddr *) &client, &cltsize );
   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 name

- 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 name", it replies "Hello, name!" to the client.

Add Command

- **Add Command Format:**

  add id_number first_name last_name 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 sends an error message back to the client.
  - If the command's parameter is not valid, the server returns an Error message to the client.

    For example,

    Add 12033 Tony Smith worcester MA

    ➔ returns "an invalid add command".

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 to the Client.
  - 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 the server, the client knows 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 and including a count of the number of commands that are issued by *name*. 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 sends back to the Location Client a *count* of the number of clients processed, then terminates.

List Command

- **List Command format**

  list *start [finish]*

  Notes: start – one or two character
  finish – two character

  Examples:

  - **list**

    Find all the entries.

  - **list *A***

    Find the entries, whose last_name starts with *A*

  - **list *A*a Bb**

    Find the entries, whose *last_name* is greater than or equal to *A*a but smaller than or equal to Bb.

List Command (cont’d)

- **For the Client:**

  Sends the command to the server, and displays the response messages from the server.

- **For the Server:**

  When it receives 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.
    - example
    - list Aa
    - list Aa B
    - list A Bb
    - list Bb Aa

Outline

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

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 <sys/socket.h>
#include <netinet/in.h>         #include <arpa/inet.h>
#include <netdb.h>              #include <unistd.h>
#include <signal.h>             #include <stdio.h>
#include <fcntl.h>              #include <errno.h>
#include <sys/time.h>           #include <stdlib.h>
#include <memory.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/

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.

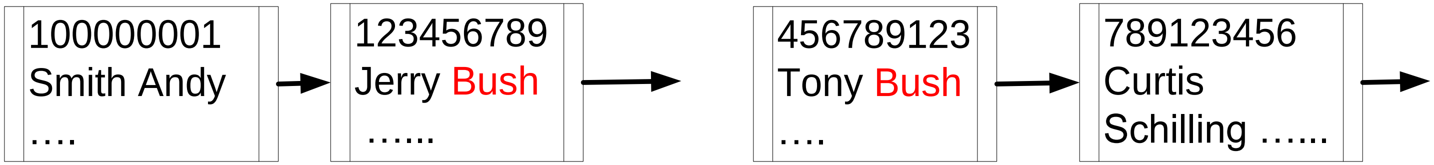
- Array:

  The simplest data structure.

Sorting in Database

- The server's database is sorted ascending by *last_name.*

For example, (based on a linked list)



Ldatabase.txt

```
last_name    first_name   id,          location
Andy         Smith        100000001    …
Bush         Jerry        123456789    …
```

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.+TAs (cs3516-ta “at” cs.wpi.edu), but do NOT expect immediate results, better to attend office hours.
  - My Office Hours: Sun, 6-9pm; Mon, 3:30-5:30pm
  - Dongqing Xiao’s Office Hours: Wed, 2-4pm; Thu, 2-4pm
- 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);
```

References

- Beej's Guide to Network Programming
- The GNU C Library
- IBM iSeries Information Center
- The Open Group Base Specifications
- Wikipedia