CSE 489/589
Programming Assignment 1

Li Sun, Swetank Kumar Saha
{lsun3, swetankk}@buffalo.edu

How to run your program

- Server

  $ ./assignment1 s 4322

- Client

  $ ./assignment1 c 4322

- SHELL

  [PA1]$ IP

  …..

  [PA1]$ LOGIN 128.56.78.31 4322

  [PA1]$ …

Commands

- All commands should be in UPPER CASE
- How to recognize the command input?
  - Tokenize input

```
/* Parse cmd and args from the input */
argc = 0;
arg = strtok(cmd, " ");
while(arg){
    strcpy(argv[argc], arg);
    argc += 1;

    arg = strtok(NULL, " ");
}
```

  - Compare with command strings

```
//Process commands
if( !(strcmp(argv[0], "HELP")) ){
```

Commands

- [PA1]$ IP

....

IP:128.205.36.8

....

- Display the external/public IP address
- 127.0.0.1 is NOT the correct address
- Create a UDP socket to any valid destination IP address

Commands

- [PA1]$ PORT

....

PORT:4322

....

- Display the port number your host is listening on

Commands

- [PA1]$ LOGIN 128.205.36.8 4322
- Clients login to the server
  - Identify themselves to the server
  - Get list of other logged-in clients
  - Get buffered messages

Commands

- List the currently logged-in clients
- [PA1]$ LIST

1	stones.cse.buffalo.edu	128.205.36.32	4545
2	embankment.cse.buffalo.edu	128.205.36.35	5454
3	highgate.cse.buffalo.edu	128.205.36.33	5000
4	euston.cse.buffalo.edu	128.205.36.34	5100

```
for(host=0; host<MAX_HOSTS; host+=1){

    .......
    .......
    .......

    printf("%-5d%-35s%-20s%-8d\n",  host_id, hostname, ip_addr, port_num);
}
```

Commands

- [PA1]$ SEND 128.45.12.1 Hello
- Send message to client with IP address: 128.45.12.1, *through the server*
- Max. length of message: 256 bytes
- Only valid ASCII characters

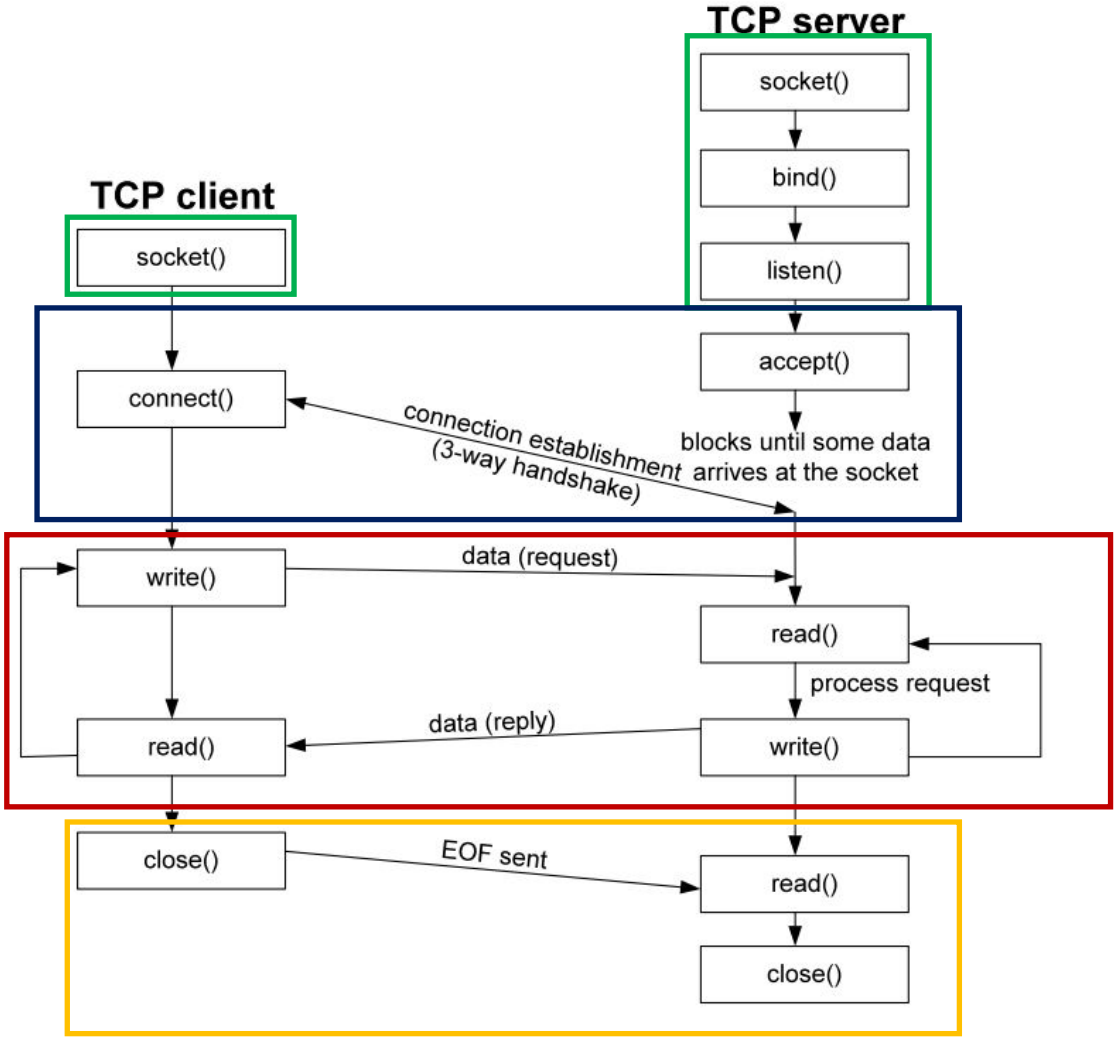
Commands

- [PA1]$ LOGOUT
- Logout from the server
- DO NOT terminate/exit the application
- Server should buffer messages for logged-out clients

Commands

- [PA1]$ EXIT
- LOGOUT and exit the application with status code 0
- No buffering of messages for exited clients

TCP Socket Flow



Server Socket Setup

```
server_socket = socket(AF_INET, SOCK_STREAM, 0);
if(server_socket < 0)
    return err_msg_ERR("Cannot create socket");

bzero(&server_addr, sizeof(server_addr));

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(port);

printf("Port: %d:", ntohs(server_addr.sin_port));

if(bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0 )
    return err_msg_ERR("Bind failed");

if(listen(server_socket, BACKLOG) < 0){
    fprintf(stderr, "Unable to listen on port %d", port);
    return -1;
}
```

Server Socket Setup

```
caddr_len = sizeof(client_addr);
fdaccept = accept(server_socket, (struct sockaddr *)&client_addr, &caddr_len);
if(fdaccept < 0)
    return err_msg_ERR("Accept failed.");
```

Using connect()

```
fdsocket = socket(AF_INET, SOCK_STREAM, 0);
if(fdsocket < 0)
   return err_msg_ERR("Failed to create socket");

bzero(&server_addr, sizeof(server_addr));
server_addr.sin_family = AF_INET;
inet_pton(AF_INET, server_ip, &server_addr.sin_addr);
server_addr.sin_port = htons(server_port);

if(connect(fdsocket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
    return err_msg_ERR("Connect failed.");
```

Select based I/O

- select()
  - For I/O Multiplexing

    Command input on shell, TCP port listening, data incoming …
  - File descriptors

    System initialized OR User created/defines (e.g., files, socket())

Integer value	Name
0	Standard Input (stdin)
1	Standard Output (stdout)
2	Standard Error (stderr)

Select based I/O

- select() API function calls
  - Add fd to the set

    FD_SET(int fd, fd_set *set);
  - Remove fd from the set

    FD_CLR(int fd, fd_set *set);
  - Return true if fd is in set

    FD_ISSET(int fd, fd_set *set);
  - Clear all entries from the set

    FD_ZERO(fd_set *set);

Select() Setup

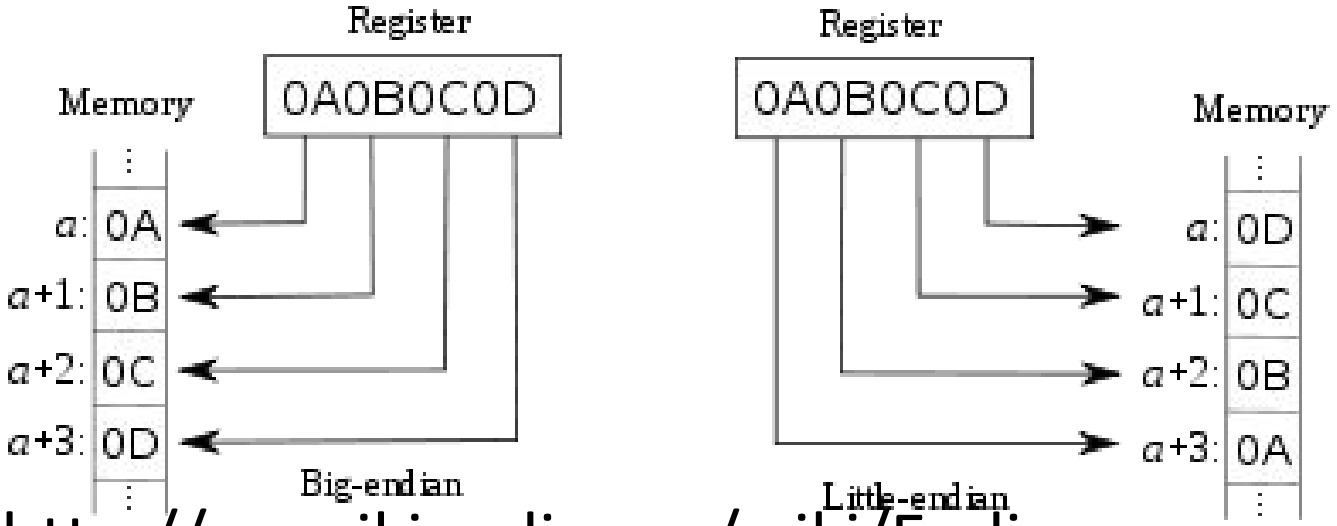
```
FD_ZERO(&master_list);
FD_ZERO(&watch_list);
head_socket = server_socket;
/* Register the listening socket */
FD_SET(server_socket, &master_list);
/* Register STDIN */
FD_SET(STDIN, &master_list);
```

Select() Control flow

```
while(TRUE){
    .....
    .....
    selret = select(head_socket + 1, &watch_list, NULL, NULL, NULL);
    if(selret<0)
        return printf("select failed.");
    /* Check if we have sockets/STDIN to process */
    if(selret > 0){
        /* Loop through socket descriptors to check which ones are ready */
        for(sock_index=0; sock_index<=head_socket; sock_index+=1){
            if(FD_ISSET(sock_index, &watch_list)){
                /* Check if new command on STDIN */
                if (sock_index == STDIN){
                    ....
                }
                /* Check if new client is requesting connection */
                else if(sock_index == server_socket){
                    .....
                    /* Add to watched socket list */
                }
                /* Read from existing clients */
                else{
                    ....
                }
            }
        }
    }
}
```

Endianess

- Different architectures use different byte orderings internally for their multi-byte datatypes



- http://en.wikipedia.org/wiki/Endianness

Endianess

- Network byte-order is big-endian
- Need to convert all data to network byte-order before sending over a link
- htons(), htonl(), ntohs(), ntohl()
  - htons() host to network short
  - htonl() host to network long
  - ntohs() network to host short
  - ntohl() network to host long multi-byte integers

BONUS:

P2P File Transfer

P2P File Transfer

- Additional functionality to transfer files between clients
  - Direct transfer between clients, NO server involvement
  - Binary and text files
  - Open a TCP connection between the given two clients
- NO broadcast file transfers