

# **CSE 489/589**

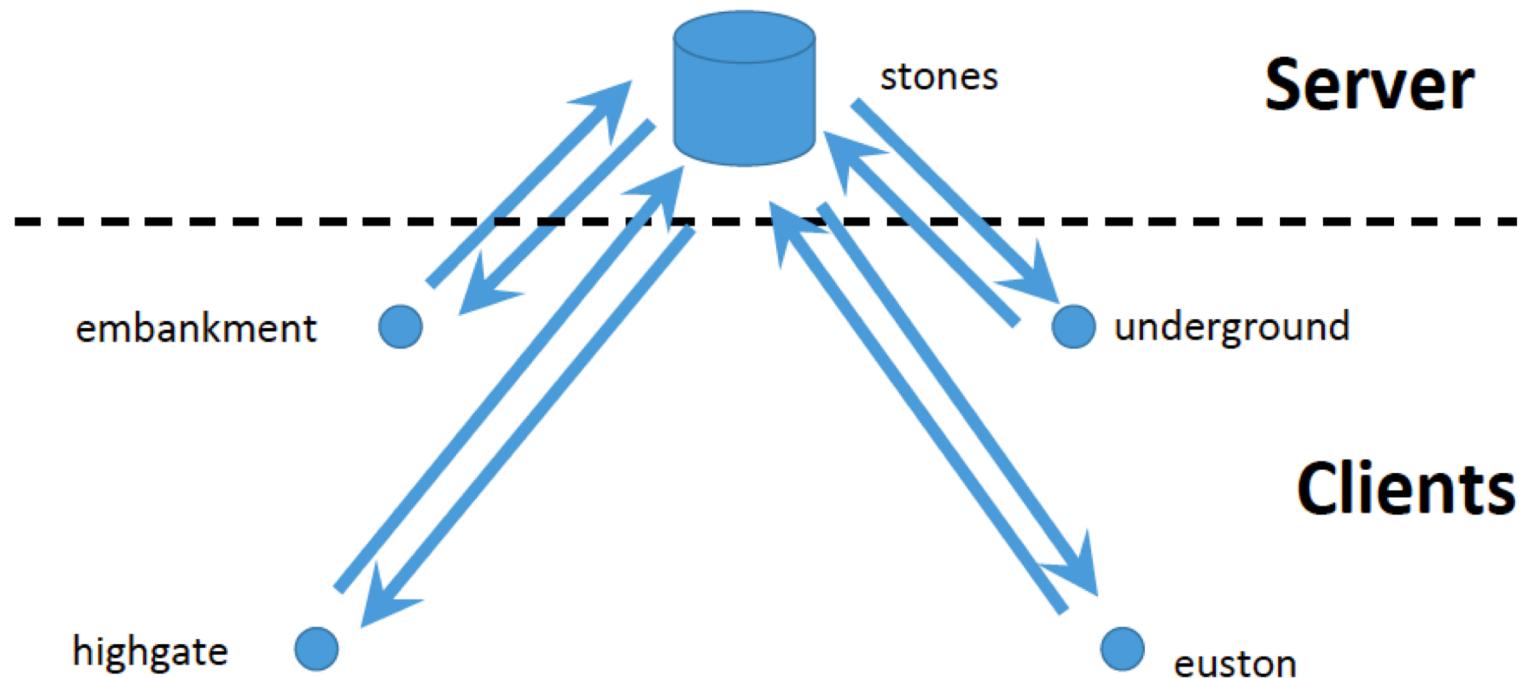
# **Programming Assignment 1**

Chenglin Miao  
[cmiao@buffalo.edu](mailto:cmiao@buffalo.edu)

# Project Objective

- Develop a text chat application for message exchange among remote hosts:
  - One Server
  - Multiple (at most 4) Clients
  - Communication using TCP sockets

# Project Description

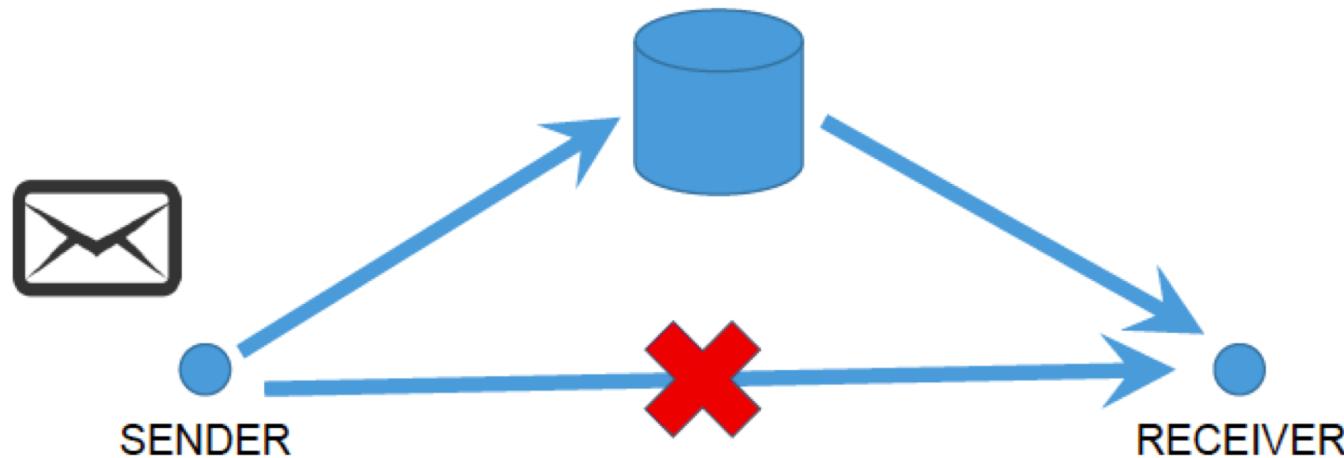


# Project Description: Clients

- When launched
  - Login to the server
  - Identify themselves to the server
  - Get list of other logged-in clients
  - Get buffered messages
- Clients send messages
  - Unicast
  - Broadcast
- Clients communicate with each other **ONLY** through the server

# Project Description: Server

- Facilitate exchange of messages between clients
  - Relays all messages
  - Maintains list of logged-in clients
  - Stores/Buffers messages for offline clients



# Commands / Events

- Assignment description lists the commands/events you need to accept/handle
- Mandatory/Required output for each command/event
- Follow command output syntax EXACTLY
  - Use the supplied format strings
- All printing using the *cse4589\_print\_and\_log(char\* format, ...)* function
- Extra output will make the automated grader fail test cases

# Dedicated CSE servers

- Ensure that your code compiles and operates correctly on the following 5 servers
  - stones.cse.buffalo.edu
  - highgate.cse.buffalo.edu
  - euston.cse.buffalo.edu
  - embankment.cse.buffalo.edu
  - undergound.cse.buffalo.edu
- For development and testing you should only use the directory we created for you on each of the 5 servers at /local/Fall\_2019/<Your-UBIT-Name>/
  - You should use these directories for development and testing (not your home directories)

# Project Requirements

- Make/Run on CSE Servers (stones, underground, embankment, ....), make sure it compiles/runs on these servers
- Only one program is running on each server, but takes different arguments
  - ./assignment1 s 4322
  - ./assignment1 c 4322
- C or C++
  - No external libraries for socket programming
  - No external binaries/utilities (e.g., ifconfig, nslookup, etc.)
- Use the PA1 template (**MANDATORY**)

# 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")) ){
    //Do something
}
```

# 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

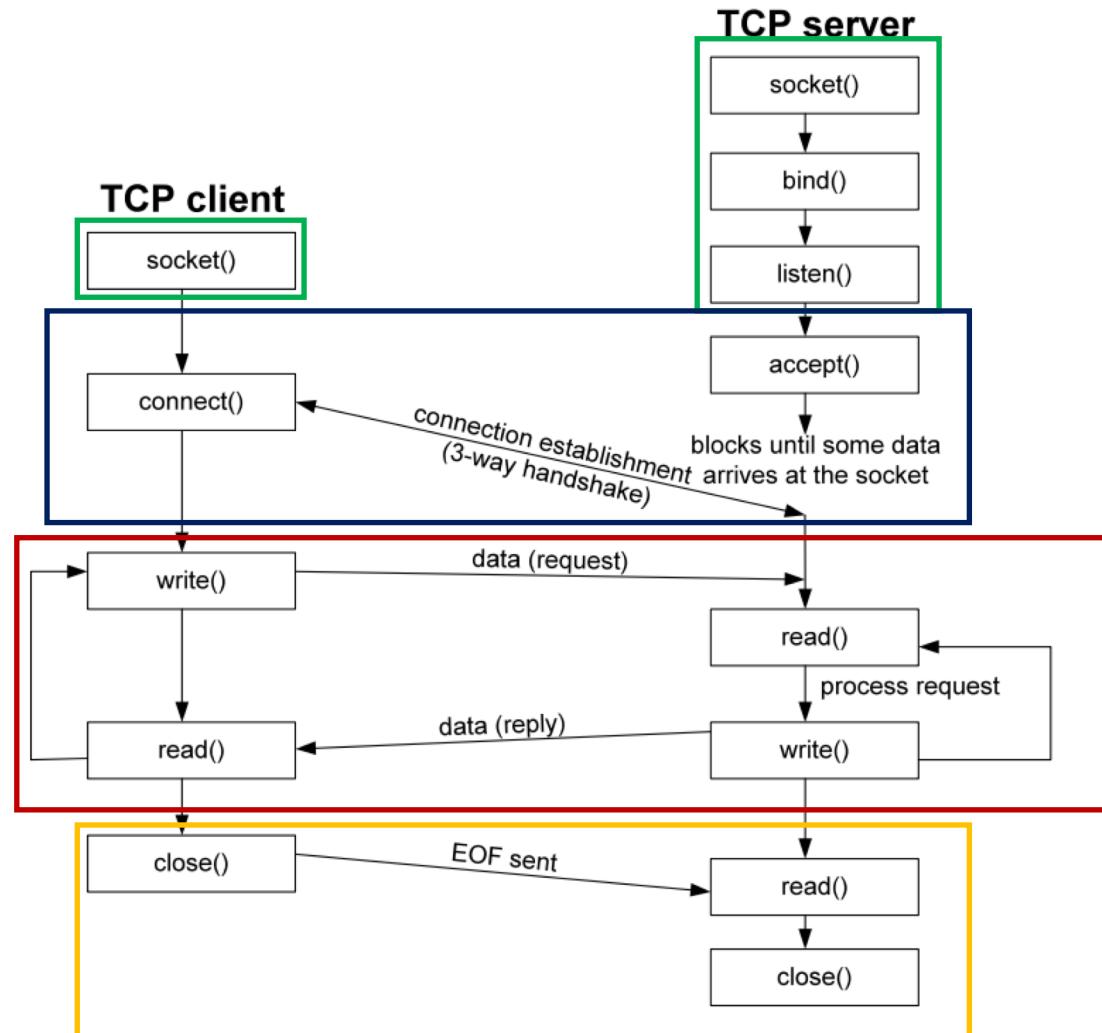
# Commands

- [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_nton(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 ( <code>stdin</code> )
1	Standard Output ( <code>stdout</code> )
2	Standard Error ( <code>stderr</code> )

# 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

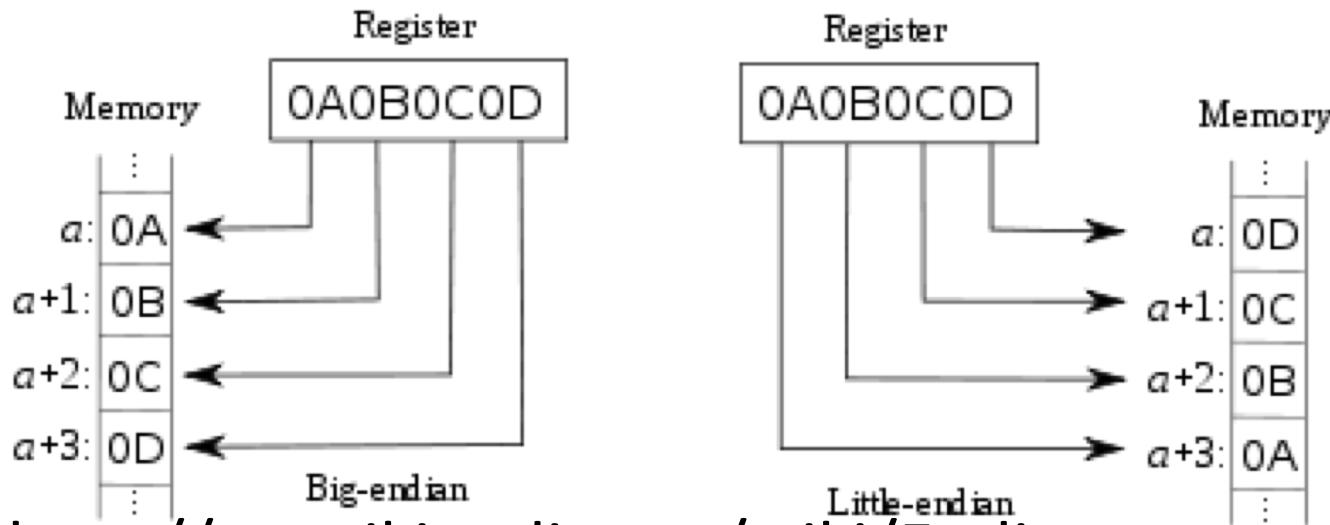
```
FD_ZERO(&master_list);
FD_ZERO(&watch_list);
head_socket = server_socket;
/* Register the listening socket */
FD_SET(server_socket, &master_list);
/* Register STDIN */
FD_SET(STDIN, &master_list);
```

# 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