# CSE422 Computer Networks Fall 2012
# Laboratory 1: Introduction to Socket Programming
# Due: 23:59 Monday, October 1, 2012

## 1 Goal

Gain experience with socket programming by implementing a simple game (Bulls and Cows) using both UDP and TCP sockets. In this game, one participant tries to guess a four-digit number known to the other participant, receiving feedback on each guess as described below.

## 2 Overview

In this lab, you will implement a networking version of Bulls and Cows, which will comprise two C++ programs, one client and one server. This lab will help you to gain experience with socket programming using Berkeley socket interface. In order to focus more on the details of socket programming part, the game and most of the command parsing are provided as a skeleton code. [link] (Note: using the skeleton code is not mandatory.)

This lab is worth for 50 points. This lab is due no later than 23:59 (11:59 PM) on Monday, October 1, 2012. No late submission will be accepted.

## 3 Specification

In this lab, you are required to implement a client and a server program for the game Bulls and Cows. The server program and client program will interact by exchanging messages to simulate the gameplay. The server is expected to be able to handle multiple clients. A simple message/packet format and a simple protocol are provided in the file: `packet.h`. The server and client program will handshake through TCP and the gaming interaction will base on UDP.

### 3.1 Bulls and Cows [Wikipedia Entry]

The server program holds a four-digit secret number, with no repeated digits. The client program tries to guess the secret number by sending a four-digit guess to the server. The server compares the guessed number and the secret number. If a matching digit is also on the right position, it is a **bull (A)**. If a matching digit is on different position, it is a **cow**

**(B)**. The server replies to the client with the number of bulls and cows, but the client is not told which digit is Bull and which is Cow.

For example, suppose the secret number is 7632

1. Guess: 1234, Result: 1A1B (The 3 is a bull and the 2 is a cow).
2. Guess: 5678, Result: 1A1B (The 6 is a bull and the 7 is a cow).
3. Guess: 9012, Result: 1A0B (The 2 is a bull and there are no cows).
4. and so on...

A simple implementation of the non-network aspect of Bulls and Cows game is provided in `Bulls_And_Cows.*`.

## 3.2 `packet.h`: packet format and protocol

The packet in the lab are defined as having only two fields: 1. message type: `unsigned int integer` and 2. message buffer: `char buffer[128]` The protocol consists of several message types defined in `packet.h`. The protocol is visualized in Figure. 1.

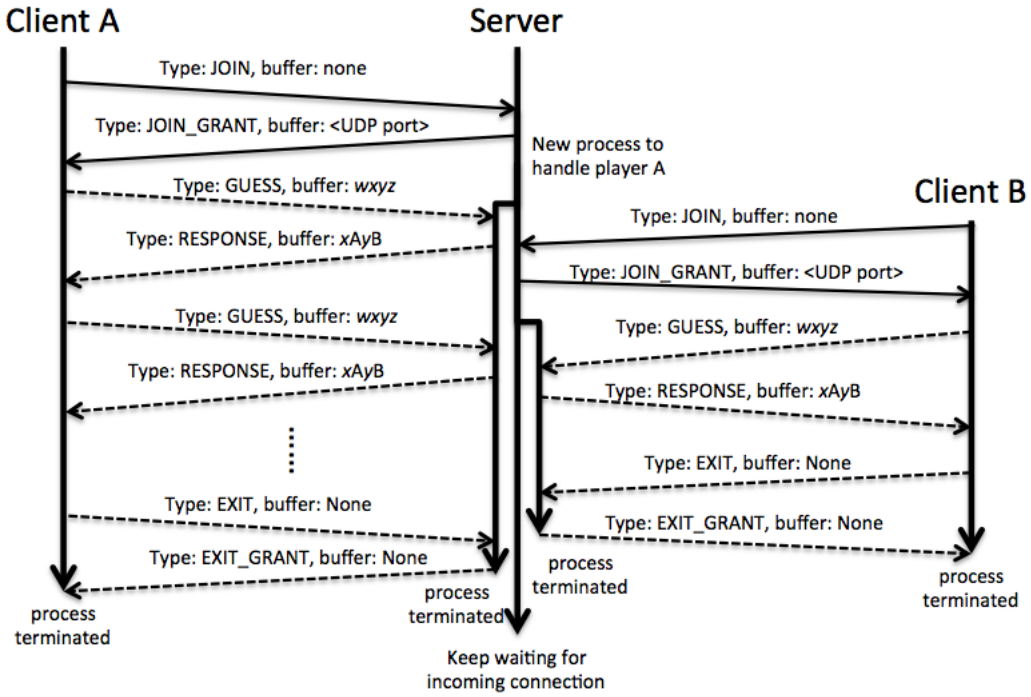


Figure 1: The protocol for lab 1. The thick vertical lines denote processes. Note that the server creates new processes to handle individual client. The solid/dashed lines denote communication through a TCP/UDP socket.

## 3.3 The server program

Example invocation: `./lab1_server`

The server program will be a multi-process program, that takes no argument. The parent process is responsible for listening for incoming client requests and the child processes are responsible for handling individual clients.

The parent process creates a TCP socket and waits for incoming TCP connections. This TCP socket's port will be assigned by the operating system and printed to the console. We assume that the clients know this TCP port number, because the clients are started after the server. For each incoming TCP connection, the server program creates a child process using the system call `fork()`. The child process for this specific TCP connection then waits a `JOIN` message from the client. The child process discards **ANY** message that is not `JOIN`, and in this case, terminates the child process. If the incoming packet's type is `JOIN`, the child process creates a UDP socket, whose port is also assigned by the OS. Via the TCP socket, the child process returns a packet of type `JOIN_GRANT` and UDP port number in the buffer.

The game (guessing, responding, and exiting) is played using the UDP socket. When a client sends a `GUESS` message, the server responds with the result in a `RESPONSE` message. If the client has all four digits correct (four bulls), the server starts a new game until the client enters the `EXIT` command. When a client sends an `EXIT` message, the server grants this exit, returns an `EXIT_GRANT` and terminates the child process for this client.

A skeleton server code is provided: `lab1_server.cc`. A `parse_args()` function is also provided in `lab1_server.h`.

## 3.4 The client program

Example invocation: `./lab1_client -p 48192 -s arctic.cse.msu.edu`

The client program is required to accept the following arguments.

- `-s` is the server address (domain name or IP address).
- `-p` is the TCP port number that the server listen for incoming connection.

The client resolves the server address using `gethostbyname()` and connects to this server over TCP. The client then sends a new game request `JOIN`, in order to obtain the UDP port number for gameplay. After the client obtains the UDP port number, the client either tries to guess the secret number by sending `GUESS` packet or exits the game by `EXIT`. As mentioned above, the server starts a new game (generate a new secret number) when the player has all

four digits correct. The client program will also output a message, showing that the player has won the game.

Note that the message `JOIN` is sent after the client's TCP connection to the server is established. The player can only issue two types of commands: `GUESS` and `EXIT`. The parsing function `get_command` provided only accepts those two commands.

A skeleton client code is provided: `lab1_client.cc`. Several helping functions, including command parsing and argument parsing, are provided in `lab1_client.h`.

# 4 Deliverables

You will submit your lab using the *handin* utility. Please submit all files in your project directory. If you start your lab with the skeleton code, submit all files, even if the file is not modified.

This lab is due no later than 23:59 (11:59 PM) on Monday, October 1, 2012. No late submission will be accepted.

The compilation must be done using makefile. The code should compile and link on `black.cse.msu.edu` and `arctic.cse.msu.edu`. You will not be awarded any points if your submission does not compile using makefile. Please test your programs before handing them in.

A README file is required. You will run your server program and client program and paste the log in your README file. A sample README file is also included in the skeleton code. You are also encouraged to include any comment in the README file.

# 5 Example

Follows is an example of output from the client and server. Your output may differ.

1. Invoke the server

   ```
   >./lab1_server
   [SYS] Parent process for TCP communication.
   [TCP] Bulls and Cows game server started...
   [TCP] Port: 42856
   ```

2. Invoke the client. The client connects to the server. The server creates a child process and a UDP socket for this client and sends the UDP port to this client.
   Client:

```
>./lab1_client -s adriatic.cse.msu.edu -p 42856
[TCP] Bulls and Cows client started...
[TCP] Connecting to server: adriatic.cse.msu.edu:42856
[TCP] Sent: JOIN
[TCP] Rcvd: JOIN_GRANT 38119
[UDP] Guesses will be sent to: adriatic.cse.msu.edu at port:38119
[GAM] A new secret number is generated.
[GAM] Please start guessing!
[CMD]
```

Server:

```
.........
[SYS] child process forked.
[TCP] New connection granted.
[TCP] Recv: JOIN
[UDP:38119] Gameplay server started.
[TCP] Sent: JOIN_GRANT 38119
[UDP:38119] A new game is started.
```

3. The client tries to guess the secret number.
   Client:

```
.........
[CMD] GUESS 0123
[UDP] Sent: GUESS 0123
[UDP] Rcvd: RESPONSE 1A0B
[GAM] You guess 0123 and the response is 1A0B
[CMD]
```

   Server:

```
.........
[UDP:38119] Rcvd: GUESS 0123
[UDP:38119] Sent: RESPONSE 1A0B
```

4. The client exits.
   Client:

```
.........
[CMD] EXIT
[UDP] Sent: EXIT
[UDP] Rcvd: EXIT_GRANT Exit granted, goodbye.
```

   Server:

```
.........
[UDP:46763] Rcvd: EXIT
[UDP:46763] Sent: EXIT Exit granted, goodbye.
[UDP:46763] Player has left the game.
[SYS] child process terminated.
```

# 6 Grading

You will not be awarded any points if your submission does not compile.

```
General requirements: 5 points
_____  1 points: Coding standard, comments ... etc
_____  2 points: README file
_____  2 points: Descriptive messages (guessing, winning ... etc).

Server: 30 points
_____  5 points: Error checking (Handle return values < 0).
_____  5 points: OS assigns TCP and UDP port
_____  5 points: Handshaking over TCP (JOIN)
_____  5 points: Playing the game through UDP (GUESS/EXIT)
_____  5 points: Clean up. Close sockets when done using them.
_____ 10 points: Handling multiple clients

Client: 15 points
_____  5 points: Error checking (Handle return values < 0)
_____  5 points: Resolves hostname (gethostbyname(())
_____  5 points: The protocol (JOIN/GUESS/EXIT)
```

# 7 Notes

Please feel free to mail TA Chin-Jung Liu liuching AT cse.msu.edu for questions or clarifications. Additional notes and FAQ willl be posted on the website as well.