CSE422 Computer Networks Fall 2020
Project 1: Introduction to Socket Programming
Due: 23:59 Friday, October 2, 2020

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 project, you will implement a networking version of Bulls and Cows, which will comprise two C++ programs, one client and one server. This project 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 which can be downloaded from Mimir. (Note: using the skeleton code is not mandatory.)

This project is worth for 50 points. This project is due no later than 23:59 (11:59 PM) on Friday, October 2, 2020. No late submission will be accepted.

3 Specification

In this project, 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 project are defined as having only two fields: 1. message type: `unsigned int integer` and 2. message buffer: `char buffer[256]`. The protocol consists of several message types defined in `packet.h`. The protocol is visualized in Figure. 1.

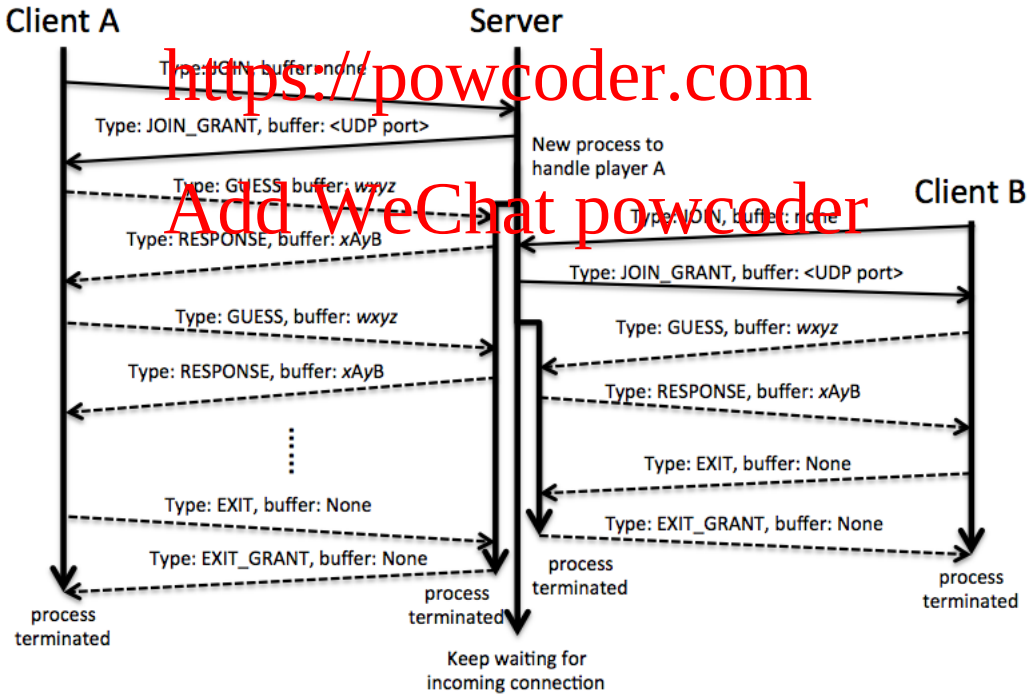

Figure 1: The protocol for project 1. The thick vertical lines denote processes. Note that the server creates new processes to handle individual client. The solid/dashed lines denote communication through a TCP/UDP socket.

3.3 The server program

Example invocation: `./proj1_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. If the recieved message is not one of these expected types, then the child process terminates.

Since the server is designed to normally `listen()` indefinitely, it will closed by posting a SIGINT signal against it (i.e. Ctrl-C).

In order to make sure no socket file descriptors are leaked, make sure to implement a SIGINT handler to close open sockets and then terminate. Additionally, this will be helpfull while developing/debugging the server.

A skeleton server code is provided: `proj1_server.cc`. A `parse_args()` function is also provided in `proj1_server.h`.

3.4 The client program

Example invocation: `./proj1_client -p 48192 -s localhost`

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` to the server, in order to obtain the UDP port number for gameplay. After the client obtains the UDP port number, a game process is created at the server side. The player either tries to guess the secret number by entering `GUESS` or exits the game by entering `EXIT`. As mentioned above, the server starts a new game (generate a new secret numbe) 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 **by the client program** right after the client's TCP connection to the server is established. The player does **NOT** need to issue a `JOIN` command. The player can only issue two types of commands: `GUESS` and `EXIT`. The parsing function `get_command()` that only accepts those two commands is provided in `proj1_client.h`.

Make sure to also add a SIGINT handler that properly closes all open sockets for the client as well. Although this process is not expected to run indefinitely, it is still important to have a clean way to terminate this process while developing/debugging.

A skeleton client code is provided: `proj1_client.cc`. Several helping functions, including command passing and argument parsing, are provided in `proj1_client.h`.

3.5 Error Handling

Note that many of the socket functions (`socket()`, `bind()`, etc.) can fail and return negative values accordingly. Make sure to check each of these return values for failure before moving on in the code. In case of failure notify the user, close sockets, and shutdown.

In addition, some functions, like `recv()`, can return 0 to indicate that the other connected socket has been shutdown. This would indicate that the other process has experienced some error and had to shutdown unexpectedly. Make sure to notify the user, close sockets, and shutdown in this case as well.

3.6 Output Format

Make sure to label lines of output in brackets to indicate what it corresponds to (e.g. `[TCP]` for TCP setup/msgs, `[GAM]` for game messages, `[SYS]` for significant system events). If exiting early due to an unexpected error, make sure to output a descriptive message prefixed with `[ERR]`.‘

4 Deliverables

You will submit your project to Mimir (in the project 1 assignment). Please upload all files required to run your project and your README. If you start your project with the skeleton code, make sure to submit all files, even if they are not modified.

This project is due no later than 23:59 (11:59 PM) on Friday, October 2, 2020. No late submission will be accepted.

The compilation must be done using a makefile, which is also provided in the skeleton code. The Mimir IDE can be used to test compilation similar to that done in the test cases (make sure to generate all object files in Mimir so that they link 'correctly). You will not be awarded any points if your submission does not compile using a makefile in Mimir. Please test your programs in Mimir before submitting them.

A README file is required. Make sure to fill in your name, any comments you may have, and an estimate on the time it took to complete.

5 Example

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

1. Invoke the server.

```
>./proj1_server
[SYS] Parent process for TCP communication.
[TCP] Bulls and Cows game server started...
[TCP] Port: 56810
```

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:

```
>./proj1_client -s localhost -p 56810
[TCP] Bulls and Cows client started...
[TCP] Connecting to server: localhost:56810
[TCP] Sent: JOIN
[TCP] Rcvd: JOIN_GRANT 44379
[UDP] Guesses will be sent to: localhost at port:44379
[GAM] A new screct number is generated.
[GAM] Please start guessing!
[CMD]
```

   Server:

```
.........
[SYS] child process forked.
[TCP] New connection granted.
[TCP] Recv: JOIN
[UDP:44379] Gameplay server started.
[TCP] Sent: JOIN_GRANT 44379
[UDP:44379] A new game is started.
[UDP:44379] Secret number: 3027
```

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

```
.........
[CMD] GUESS 0123
[UDP] Sent: GUESS 0123
[UDP] Rcvd: RESPONSE 1A2B
[GAM] You guess 0123 and the response is 1A2B
[CMD]
```


   Server:

```
.........
[UDP:44379] Rcvd: GUESS 0123
[UDP:44379] Sent: RESPONSE 1A2B
```


4. The client exits.
   Client:


```
.........
[CMD] EXIT
[UDP] Sent: EXIT
[UDP] Rcvd: EXIT_GRANT
```

   Server:

```
.........
[UDP:44379] Rcvd: EXIT
[UDP:44379] Sent: EXIT_GRANT
[UDP:44379] 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
_____  2 points: Coding standard, comments ... etc
_____  1 points: README file
_____  2 points: Descriptive messages (guessing, winning ... etc)

Server: 30 points
_____  5 points: Error checking (return values < 0, return values == 0,
                                 invalid msg TYPE)
_____  5 points: OS assigns TCP and UDP port, use of htons()
_____  5 points: Handshaking over TCP (JOIN)
_____  5 points: Playing the game through UDP (GUESS/EXIT)
_____  5 points: Handling multiple clients
_____  5 points: Clean up, close sockets on SIGINT

Client: 15 points
_____  3 points: Error checking (return values < 0, return values == 0,
                                 invalid msg TYPE)
_____  2 points: Resolves hostname (gethostbyname())
_____  5 points: The protocol (JOIN/GUESS/EXIT)
                 The Client issues JOIN, not the player
_____  5 points: Clean up, close sockets on SIGINT
```


7 Notes

Please feel free to email TA Jonathon Fleck (fleckjo1@msu.edu) for questions or clarifications. Make sure to include the course name in the subject (e.g. "[CSE 422] Project 1 Questions").

7.1 Office Hours

Jonathon Fleck will also be holding office hours during the following times for this project:

Tuesday	(9/8)	2:00 pm - 3:00 pm,	Wednesday	(9/9)	2:00 pm - 3:00 pm
Tuesday	(9/15)	2:00 pm - 3:00 pm,	Wednesday	(9/16)	2:00 pm - 3:00 pm
Tuesday	(9/22)	2:00 pm - 3:00 pm,	Wednesday	(9/23)	2:00 pm - 3:00 pm
Tuesday	(9/29)	2:00 pm - 3:00 pm,	Wednesday	(9/30)	2:00 pm - 3:00 pm
Friday	(10/2)	5:00 pm - 6:00 pm			

Here is the Zoom [link]. The password is "422ta".