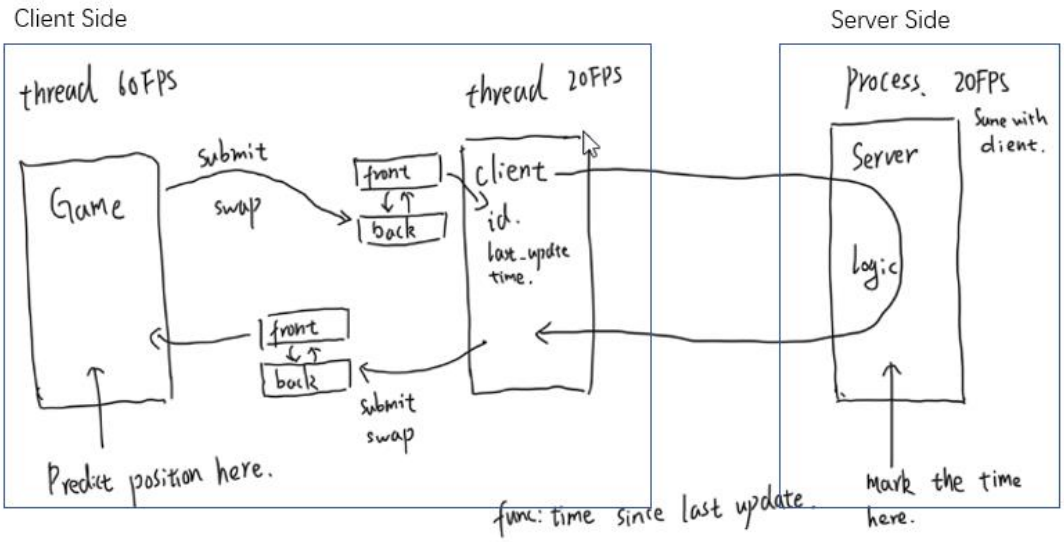
Networking System

What it does

I have implemented a network system based on TCP IPv4 asynchronized parallel server, and its responding client. This system is designed to transfer information between different players during runtime.



In each frame, the game would submit controller inputs information to the client. The client will store that data and communicate with the server. Later in the frame the game will get the returned data from the client to update game.

How to use this system.

You need to modify the following structure (Under Example Filter IO_Struct.h)

InputStruct contains the inputs data from user, and will be send to the server.

UpdateStruct contains the result of update function running on the server.

```
struct InputStruct {
    int input_x_axies = 0;
    int input_y_axies = 0;
};
struct UpdateStruct {
    eae6320::Math::sVector position[2]
    eae6320::Math::sVector speed[2]
    time_t last_time = 0;
};
```

Overwrite a function (Under Example Filter IO_Struct.cpp)

On the Server side:

Provide a value(Under Server.h)

You need to specifies the max number of clients to connect with the server.

The number need to be consistent with the UpdateStruct.

```
#define MAX_CLIENT_NUMBER 2
```

Overwrite a function (Under Example Filter IO_Struct.cpp)

The ServerLogic class is responsible for used gathered user inputs to update the state of the game in Update() function.

The serverLogic has a InputStruct containing inputs from client. You need tos tore the result in it's UpdateStruct.(The system guarantee to provide consistent inputs)

```
void Network::ServerLogic::Update()
```

You need to launch the server Using the following code.

You need to provide a port_num.(Such as 3333 as shown in the client code)

```
{
    Network::ServerLogic server_logic;
    Network::TCP::Server server;
    server.SetServerLogic(&server_logic);
    Network::network_error_code error_code;
    if (!server.Run(port_num, error_code)) {
        printf(error_code.code.c_str());
    }
}
```

On the Client Side

In game loop use the following functions(Under Client.h)

Use SubmitInputStruct to submit player's input.

Use GetUpdateStruct to obtain the result of server.

```
void SubmitInputStruct(const InputStruct& inputs);
UpdateStruct GetUpdateStruct();
```

Launch the client during initialization

The Ip address need to be the serve address(local server in following case)

The port_num need to be consistent with Server.

```
//Networking
{
    std::string local_host = "127.0.0.1";
    std::string port_num = "3333";
    m_client = Network::TCP::Client::Create_and_Run(local_host, port_num);
}
```

Shutdown the system
Call stop function to shut down the client. (The client will be destructed and the client thread will terminate)

```
void Stop();
```

What I used.

I separated the platform independent interface for the lower level operating system specific implementations.
I properly set up the Engine system as a static library that is correctly linked by the application.
I implemented a double buffer system for two threads to exchanging information's.

It is all related

The server will distribute as many ids at max_client_number defined to clients connect to the server. For now all clients need to be connected with server during Initialization of the server. The server will use the id mention above to provide consistent inputs to Serverlogic class.

Except for several atomic variables shared by multiple threads, other data are all protected by mutex. I used multiple threads on the server side to manage the connection with multiple clients and gather inputs from them. Then after update the data, the result will be send back to each clients by the main thread of the server in a sequential manner. It make sense cause using multiple threads and mutex to protect the result, will be as efficient as using only main thread. Because, only one thread is allowed to access the data. However, having multiple threads reading the same data concurrently won't lead to racing condition, as long as no threads attempts to modify. At this point of execution all threads modifying the data have already terminated. I joined them before running update function of Serverlogic class. Hence, it make sense to bypass the mutex and letting multiple thread access the data at the same time, if there is a mutex.

Actually, this system is very flexible, you can send whatever you want at any time. However, all the socket functions will block your execution until it returns, hence you are very likely going to need to manage multiple threads.

To setup this system

A project is provided as a static library on the website page. Download and unzip it to your solution. The headers you will need are client.h server.h and IO_Struct.h.