Etapa 04
Servidores intermedios y de piezas

Juan Aguilar Torres
Gabriel Gonzáles Flores
Sebastián Rodríguez Tencio

Métodos en Común

Split

```cpp
std::vector<std::string> split(std::string buffer){

    while (std::getline(iss, info, char(29))) {
        std::istringstream iss2(info);
        std::string subinfo;
        while (std::getline(iss2, subinfo, ':')) {
            info.push_back(subinfo);
        }
    }
    return info;
}
```

Get IP

```cpp
/// @brief Obtiene la dirección IP
/// @return Dirección IP obtenida
std::string getIPAddress() {
```

Servidor de Piezas

```cpp
class MethodsPieces{

    public:

    /**
     * @brief Indica su dirección IP y las figuras disponibles al
     * servidor solicitante por medio de un BROADCAST.
     */
    void sendPresentBroadcast();  // LEGO PRESENT

    /**
     * @brief Indica su dirección IP y las figuras disponibles
     * al servidor solicitante.
     */
    void sendPresent(std::string buffer);  // LEGO PRESENT

    /**
     * @brief Responde con las piezas de la figura solicitada en
     * formato HTML
     */
    std::string sendResponse(const std::string& request);

    /**
     * @brief Indica al servidor intermedio que va a dejar de dar servicios
     */
    std::string handleRelease(const std::string& figureName);

    /**
     * @brief Obtiene los nombres de las figuras del servidor de piezas.
     */
    std::vector<std::string> getFigureNames(const std::string& folderPath);

    private:

    /**
     * @brief Contiene los nombres de las figuras del servidor de piezas.
     */
    std::vector<std::string> figureNames;
};
```

MethodsPieces

PiecesServer

```
MethodsPieces mp;
std::thread * workerUDP;
std::thread * workerTCP;
char buffer[1024];

/*----------- CREATES SOCKET UDP ------------*/
/*-------- BINDS TO PIECES_UDP_PORT ---------*/
Socket* socketUDP;
socketUDP = new Socket('d');
struct sockaddr_in other;
socketUDP->Bind(PIECES_UDP_PORT);
printf("Pieces (LOCAL): Socket UDP bind a %d\n", PIECES_UDP_PORT);

/*-------- UDP THREAD TO RECEIVE MESSAGES ------------*/
workerUDP = new std::thread( taskUDP, socketUDP );



/*----------- CREATES SOCKET TCP ------------*/
/*-------- BINDS TO PIECES_TCP_PORT ---------*/
Socket * socketTCP, * client;
socketTCP = new Socket('s');
socketTCP->Bind( PIECES_TCP_PORT );
socketTCP->Listen( 5 );
socketTCP->SSLInitServer( "cert/ci0123.pem", "cert/ci0123.pem" );
printf("Pieces (LOCAL): Socket TCP bind a %d\n", PIECES_TCP_PORT);

/*-------- TCP THREAD TO RECEIVE MESSAGES ------------*/
for( ; ; ) {
    printf("Pieces (LOCAL): Socket TCP esperando a recibir mensajes
    en el puerto %d\n", PIECES_TCP_PORT);
  client = socketTCP->Accept();
  client->SSLCreate( socketTCP );
  workerTCP = new std::thread( taskTCP, client );
}

workerUDP->join();
workerTCP->join();
}
```

TaskTCP

TaskUDP

Servidor Intermedio

MethodsIntermediate

```
/**
 * @brief Realiza un broadcast a traves de UDP el cual solicita a los servidores de piezas
 * que reciban este mensaje que se reporten respecto al servidor emisor.
 */
void sendDiscover();

/**
 * @brief Procesa la información de un Present, añadiendo a la tabla.
 *
 * @param buffer Información del Present.
 */
void handlePresent(std::string buffer);

/**
 * @brief Create a Request object
 *
 * @param figure Desired Figure
 * @return std::string
 */
std::string createRequest(std::string figure);

/**
 * @brief Sends a Request to Pieces Server
 *
 * @param figure Desired Figure
 * @param request_string The request created string
 * @return std::string
 */
std::string sendRequest(std::string ip_piezas, std::string request_string);

/**
 * @brief Parsea el HTML para la respuesta que se envia al cliente
 */
std::string parseResponse(const std::string& html);

/**
 * @brief Se agrega una entrada en el mapa relacionando una figura con una IP.
 * @param figure Figura que pertenece a un servidor de piezas.
 * @param ip IP que pertenece a un servidor de piezas.
 */
void addMapEntry(std::string figure, std::string ip);
```

```
/**
 * @brief Se elimina un servidor de piezas del mapa.
 * @param ip IP que pertenece a un servidor de piezas.
 */
void removeMapEntry(std::string ip);

/**
 * @brief Se obtiene la figura y las ip's que estan relacionadas.
 * @param ip IP que pertenece a un servidor de piezas.
 * @return Devuelve la figura y las ip's que estan relacionadas.
 */
std::string getMapEntry(std::string figure);

/**
 * @brief Obtiene el mapa con las figuras y las ip's relacionadas.
 * @return Returna la variable mapTable con toda su informacion.
 */
std::map< std::string, std::vector<std::string> > getMap();

bool containsFigure(const std::string& figure);

private:

/**
 * @brief Mapa que contiene a las figuras y las relaciona con las
 * ip's de cada servidor de piezas.
 */
std::map< std::string, std::vector<std::string> > mapTable;
```

IntermediateServer

```
int main(int argc, char** argv) {
    MethodsIntermediate mi;
    std::thread * workerUDP;
    std::thread * workerTCP;
    char buffer[1024];

    /*----------- CREATES SOCKET UDP ------------*/
    /*----- BINDS TO INTERMEDIARY_UDP_PORT ------*/
    Socket * socketUDP;
    socketUDP = new Socket( 'd' );
    socketUDP->Bind( INTERMEDIARY_UDP_PORT);
    printf("Intermediate Server (LOCAL): Socket UDP bind a %d\n", INTERMEDIARY_UDP_PORT);

    /*-------- UDP THREAD TO RECEIVE MESSAGES ------------*/
    workerUDP = new std::thread( taskUDP, socketUDP, std::ref(mi));

    /*----------- CREATES SOCKET TCP ------------*/
    /*-------- BINDS TO PIECES_TCP_PORT ---------*/
    Socket * socketTCP, * client;
    socketTCP = new Socket('s');
    socketTCP->Bind( INTERMEDIARY_TCP_PORT );
    socketTCP->Listen( 5 );
    socketTCP->SSLInitServer( "cert/ci0123.pem", "cert/ci0123.pem" );
    printf("Intermediate Server (LOCAL): Socket TCP bind con %d\n", INTERMEDIARY_TCP_PORT);

    /*-------- TCP THREAD TO RECEIVE MESSAGES ------------*/
    for( ; ; ) {
        printf("Intermediate Server (LOCAL): Socket TCP esperando
        a recibir mensajes en el puerto %d\n", INTERMEDIARY_TCP_PORT);
        client = socketTCP->Accept();
        client->SSLCreate( socketTCP );
        workerTCP = new std::thread( taskTCP, client, std::ref(mi) );
    }

    workerUDP->join();
    workerTCP->join();
}
```

TaskTCP

TaskUDP

Problemas encontrados

Gracias.