Taller 3

Poner en marcha el proyecto

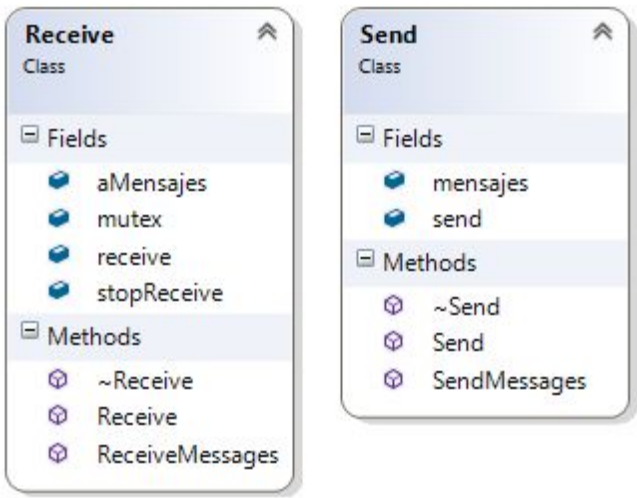
1. Abrir solución
2. Escribir la IP del servidor en el server.cpp del proyecto server y en el client.cpp en el proyecto client.

```
16      sf::IpAddress ip = sf::IpAddress::IpAddress("192.168.23.87"); //sf::IpAddress::getLocalAddress();
```

3. Ejecutar proyecto server (F5)
4. Ejecutar proyectos client (F5)
5. Client: Para salir, presionar Escapar
6. Server: Para salir, presionar Retroceder

Diagrama de clases

ChatLib:



Sockets Send y Receive para cliente

En el client.cpp

```
16      sf::TcpSocket socket;
```

Aceptar clientes

En el server.cpp, se hace un accept non blocking cada ejecución buscando una nueva conexión. Si encuentra una conexión, se establece en el sockettmp, que se pone en **non blocking** (hasta el final de la ejecución) y se añade al vector de sockets. A continuación se pide nueva memoria para el sockettmp para que continúe buscando.

```
51      while (serverOn)
52      {
53          if (sockets.size() < MAX_USERS) {
54              if (listener.accept(*sockettmp) == sf::Socket::Done) {
55                  std::cout << "New user" << std::endl;
56                  sockettmp->setBlocking(false);
57                  sockets.push_back(sockettmp);
58                  sockettmp = new sf::TcpSocket;
59              }
60          }
```

Lista de mensajes

En el client.cpp y el server.cpp

```
72      std::vector<std::string> aMensajes;
73      receiver.aMensajes = &aMensajes;
```

La clase recepción tiene un puntero vector<string> para acceder a la lista de mensajes.

Recibo de mensajes

En el Receive.cpp del proyecto ChatLib que comparten cliente y servidor

```
13      char data[1500];
14      std::size_t received;
15      std::string mensaje = "";
16      sf::Socket::Status status = socket->receive(data, 1500, received);
17      data[received] = '\0';
18
19      if (status == sf::Socket::Done) {
20          mensaje = data;
21          aMensajes->push_back(mensaje);// guarda mensaje a la llista de mensajes
22
23          if (aMensajes->size() > 25)
24          {
25              aMensajes->erase(aMensajes->begin(), aMensajes->begin() + 1);
26          }
27          return true;
28      }
29      return false;
```

Envío de mensajes

En el Send.cpp del proyecto ChatLib que comparten cliente y servidor

```
13      size_t sent;
14      sf::Socket::Status status;
15      do {
16          status = send->send(mensajes->c_str(), mensajes->size(), sent);
17          if (status != sf::Socket::Done)
18          {
19              std::cout << "Error al enviar" << mensajes << std::endl;
20          }
21      } while (status != sf::Socket::Done);
22
```

Blocking

En el client.cpp.

```
63      socket.setBlocking(false);
```

Al crear el socket, está por defecto en blocking y lo usamos para hacer el connect/accept. Justo antes de empezar el bucle de send/receive, lo ponemos en nonBlocking hasta el final del programa.

Desconexión de los sockets

En el client.cpp al final del main.

```
172     send->disconnect();
173     receive->disconnect();
174     Threceive.terminate();
175     return 0;
```

En el sever.cpp, si el servidor recibe el símbolo del dólar (que se envía cuando el cliente pulsa Escapar), desconecta el socket, borra la memoria del puntero y lo quita del vector).

```
67          if (receiver.ReceiveMessages()) {
68              if (aMensajes[aMensajes.size()-1] == "$") {
69                  sockets[i]->disconnect();
70                  std::cout << "User leaved" << std::endl;
71                  delete sockets[i];
72                  sockets.erase(sockets.begin() + i);
73              }
```

Postmortem

En este taller 3 hemos tenido diversos problemas.

El primero, ha sido la creación y implementación de la librería, la cual ha dado varios problemas, tuvimos problemas en los “include” de las librerías y eso dio resultado a que el proyecto se debe ejecutar en modo “Release”

El segundo problema ha sido la función que se encargaba de liberar los sockets. En un principio teníamos los sockets organizados en una Array estática, la cual nos permitía más facilidad a la hora de conectar los usuarios, però nos era muy complicado tratar una desconexión. Por ese motivo tuvimos que rehacer la organización de los sockets en una vector. Una vez realizado esto, tratar las desconexiones fue fácil.