Tarea 3:
Comunicación cliente-servidor

Integrantes: Edson Herrera
Alvaro Pismant
Ignacio Tapia
Asignatura: Redes de Computadores

INTRODUCCIÓN:

En el contexto actual existen muchos medios de comunicación, de interacción y de organización social. En antaño la utilización se basaba en el teléfono, el periódico, radio. Pero a través del paso del tiempo la mayoría de personas en la actualidad utiliza Internet, esto significa que Internet es ya y será aún más el medio de comunicación y de relación esencial sobre el que se basa una nueva forma de sociedad que ya vivimos. Para establecer el funcionamiento de los nodos o interlocutores de dicha red se llega a la relación cliente/servidor. En la cual existe un cliente que realiza peticiones a través de un programa, y el servidor quien responde entregando el mensaje correspondiente, así se logra la correcta comunicación y funcionalidad de la red. Con la API de C ++, es posible la creación de nodos. El propósito de estos nodos es determinar cuándo y cómo se genera una serie de partículas, denominados en este caso, request y response. Los emisores controlan la cantidad de "partículas" que se crean y en qué dirección y a qué velocidad se mueven. En ésta oportunidad, es utilizada para la correcta implementación de traspaso de datos y parámetros, como es el socket, buffer, entre otros conceptos que se aplican en el código

"El modelo Cliente/Servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Las aplicaciones Clientes realizan peticiones a una o varias aplicaciones Servidores, que deben encontrarse en ejecución para atender dichas demandas". (Marini, 2012, p.1).

Objetivo

Conocer y aplicar el API de Networkning en C++. Implementar comunicación cliente/servidor

Análisis de la situación

A través de la comunicación cliente/servidor se debe implementar un sistema de consejos y/o tips aleatorios dedicados a la asignatura de Redes de Computadores I,haciendo énfasis a la óptima manera de cursar el ramo. Los cuales estarán regulados por la interacción request-response, en la cual el servidor, quien es el encargado de realizar la respuesta, tomará un papel importante, debido a que en este debe existir un archivo de texto que poseé los consejos que hacen posible la comunicación, éste archivo debe y puede ser modificable así permitiendo agregar o quitar texto. Podemos ver que la estructura del texto empleado en el archivo corresponde a que en cada línea se expresa un consejo. El fin de ello es lograr que el servidor envíe al cliente un consejo random cada vez que haya peticiones.

Diseño de la solución

Por medio de API en C++ se implementa un cliente que está pensado totalmente para que cumpla con la facilidad de establecer conexión con un servidor, guardando los datos que éste le proporciona, para finalmente mostrarlos en pantalla. Siempre se pretende evaluar la validez del mensaje, para ello se hace uso de sentencias lógicas, que funcionan como filtro.

El servidor en cambio abastece la información de forma iterativa, es decir, es el encargado de leer el archivo cuando es requerido y así seleccionar un consejo para posteriormente enviarlo. El programa encargado de generar al cliente además recauda los datos ingresados por el usuario, como son el servidor, el puerto y el nombre del archivo.

Cada código de éste proyecto está basado y guiado por los códigos proporcionados anteriormente en la tarea número 1 del ramo, se acude también al uso de librerías del lenguaje empleado para poder concretar el objetivo de una manera eficaz.

El funcionamiento de un sistema cliente-servidor sería:

1- Cliente solicita una información al servidor.

2- Servidor recibe la petición del cliente.

3- Servidor procesa dicha solicitud.

4- Servidor envía el resultado obtenido al cliente.

5- Cliente recibe el resultado y lo procesa.

Implementación de la solución

Para implementar la comunicación cliente/servidor debemos modificar los códigos fuentes que se facilitaron en “Tarea n°1”

En cuanto al código del “Servidor” posee y ejecuta la función leer el archivo que contiene los consejos para así ir guardandolos en un arreglo dinámico, el cual espera la conexión de un cliente (request), para así desarrollar un mensaje que luego será recepcionado por el cliente. El servidor le envía un consejo random desde el archivo, luego de enviar el texto permanece activo, para poder “escuchar” distintas peticiones nodo cliente .(Figura 1).

En referencia al código encargado de crear un “Cliente” éste se centra en realizar la conexión con el servidor que se asigne por consola, guardando la información, en éste caso el mensaje random, para luego verificar si es posible mostrar éste último o se ha llegado a un error. En caso de llegar a un error el programa y por ende la conexión se cierran.(Figura 2).

```
archivo.open(nomArchivo); //SE ABRE EL ARCHIVO

if (archivo.is_open())
{
        while (getline(archivo, linea))
        {                                                                   //MIENTRAS SE SIGAN LEYENDO LINEAS...
                consejos.push_back(linea);   //GUARDA LA LINEA EN EL ARREGLO
                cantLineas = cantLineas + 1; //SE AUMENTA EN 1 LA CANTIDAD DE LINEAS LEÍDAS
        }
        archivo.close(); //SE CIERRA EL ARCHIVO
}
else
{
        std::cout << "No se pudo abrir Archivo\n";
        return EXIT_FAILURE;                    //SI EL ARCHIVO NO SE PUEDE ABRIR O NO EXISTE, SALIR
}

try
{
        TCPServerSocket servSock(servAddress, servPort); // SE INSTANCIA EL SOCKET CON LA IP
        //SE MUESTRA INFO DE SERVER POR PANTALLA
        std::cout << "Servidor en espera de cliente en servidor: " << servAddress << " y en el puerto: " << servPort << std::endl;

        for (;;)
        { //SE EJECUTA POR SIEMPRE
                //EL METODO RECIBE COMO PARAMETRO LA CONEXIÓN ESTABLECIDA, EL ARREGLO DE CONSEJOS Y LA CANTIDAD DE LINEAS LEÍDAS
                HandleTCPClient(servSock.accept(), consejos, cantLineas); //ESPERA UNA CONEXIÓN DE ALGÚN CLIENTE
        }
}
catch (SocketException &e)
{
        std::cerr << e.what() << std::endl;
        exit(EXIT_FAILURE);
}
```

Figura 1.”Server.cc”

```
checkArgs *argumentos = new checkArgs(argc, argv);

std::string servAddress;
uint16_t servPort;
std::string mensaje;
//SE ALMACENAN LOS DATOS RECIBIDOS POR PARAMETROS
servAddress = argumentos->getArgs().SERVER;
servPort = argumentos->getArgs().PORT;

delete argumentos;

try
{
        // SE ESTABLECE CONEXIÓN CON EL SERVIDOR
        TCPSocket sock(servAddress, servPort);

        char mensajeBuffer[tamannoBuffer + 1]; // BUFFER PARA EL MENSAJE + 1 PARA EL VALOR /0
        uint32_t bytesRecibidos = 0;              // BYTES QUE SE LEEN POR CADA RECV()
        std::cout << std::endl;
        //SE RECIBE EL MENSAJE DESDE EL SERVIDOR Y SE GUARDA LA CANTIDAD DE BYTES RECIBIDOS DONDE MIENTRAS ESTE SEA DISTINTO DE 0...
        while ((bytesRecibidos = (sock.recv(mensajeBuffer, tamannoBuffer))) != 0)
        {
                if (bytesRecibidos < 0)
                { //EL METODO RECV RETORNA VALOR -1 CUANDO HAY ALGÚN ERROR AL RECIBIR DATOS
                        std::cerr << "Error al recibir datos." << std::endl;
                        return (EXIT_FAILURE);
                }
                mensajeBuffer[bytesRecibidos] = '\0'; // SE AGREGA /0 PARA MARCAR EL FIN DEL STRING
                std::cout << mensajeBuffer;                     //MUESTRA EL MENSAJE POR PANTALLA
        }
        std::cout << std::endl << std::endl;
}
```

Figura 2.”Client.cc”

Resultados

Finalmente se logra producir el objetivo deseado. A través de una serie de comandos, especificados en README.md, para lograr ejecutar el programa, dicho archivo que se encuentra en el repositorio en conjunto con los demás de ésta tarea.

El programa realiza la entrega de mensajes a través del método response(Figura 4.), se ve claramente la comunicación cliente/servidor, ya que el programa también proporciona el estado de dicha conexión(Figura 3.), ésto es útil para saber qué está sucediendo mientras se espera, como cliente un Mensaje.

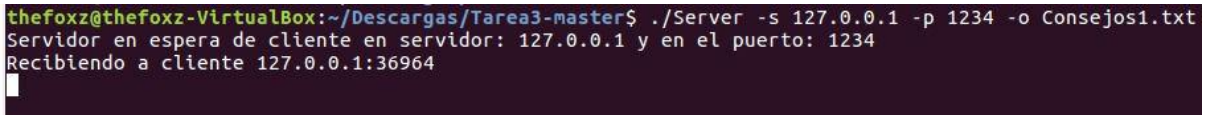


Figura 3.”cmd prueba”

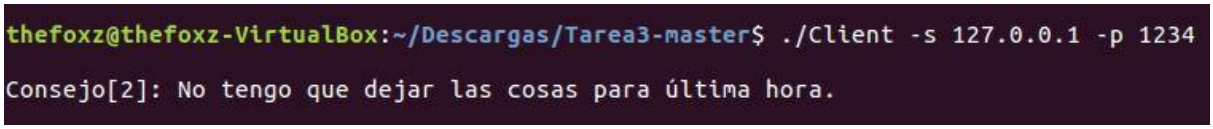


Figura 4.”cmd Mensaje”

+

ANEXOS

Diagrama de clases

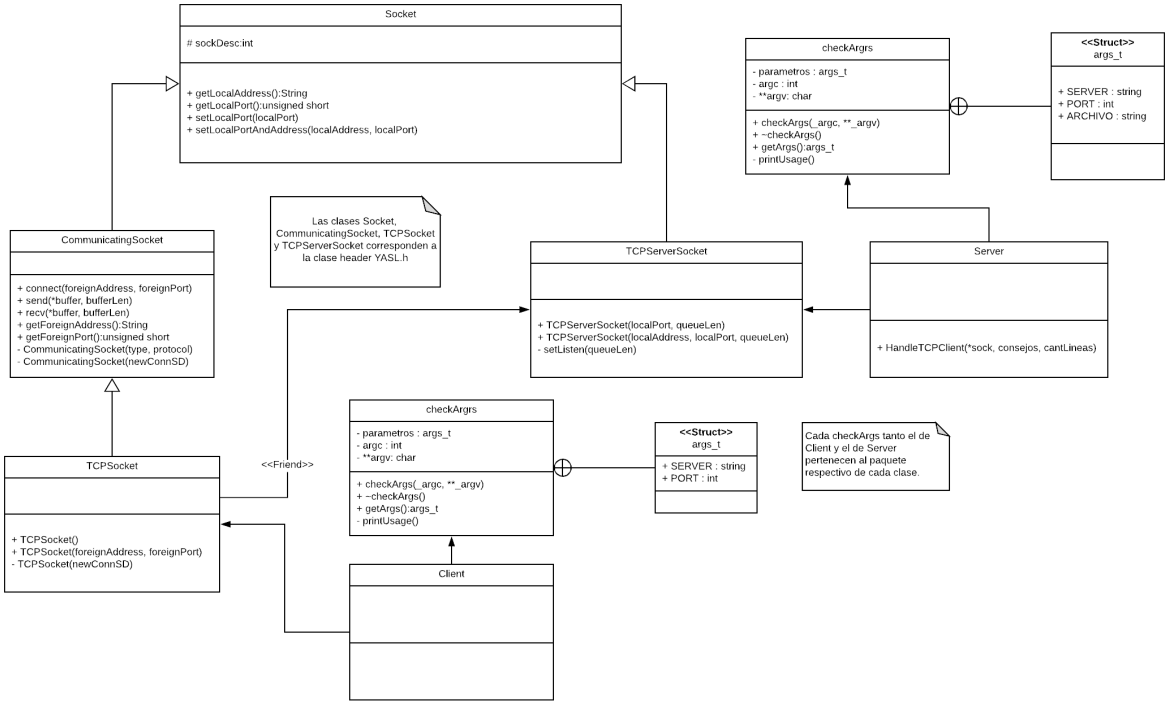


Diagrama de flujo

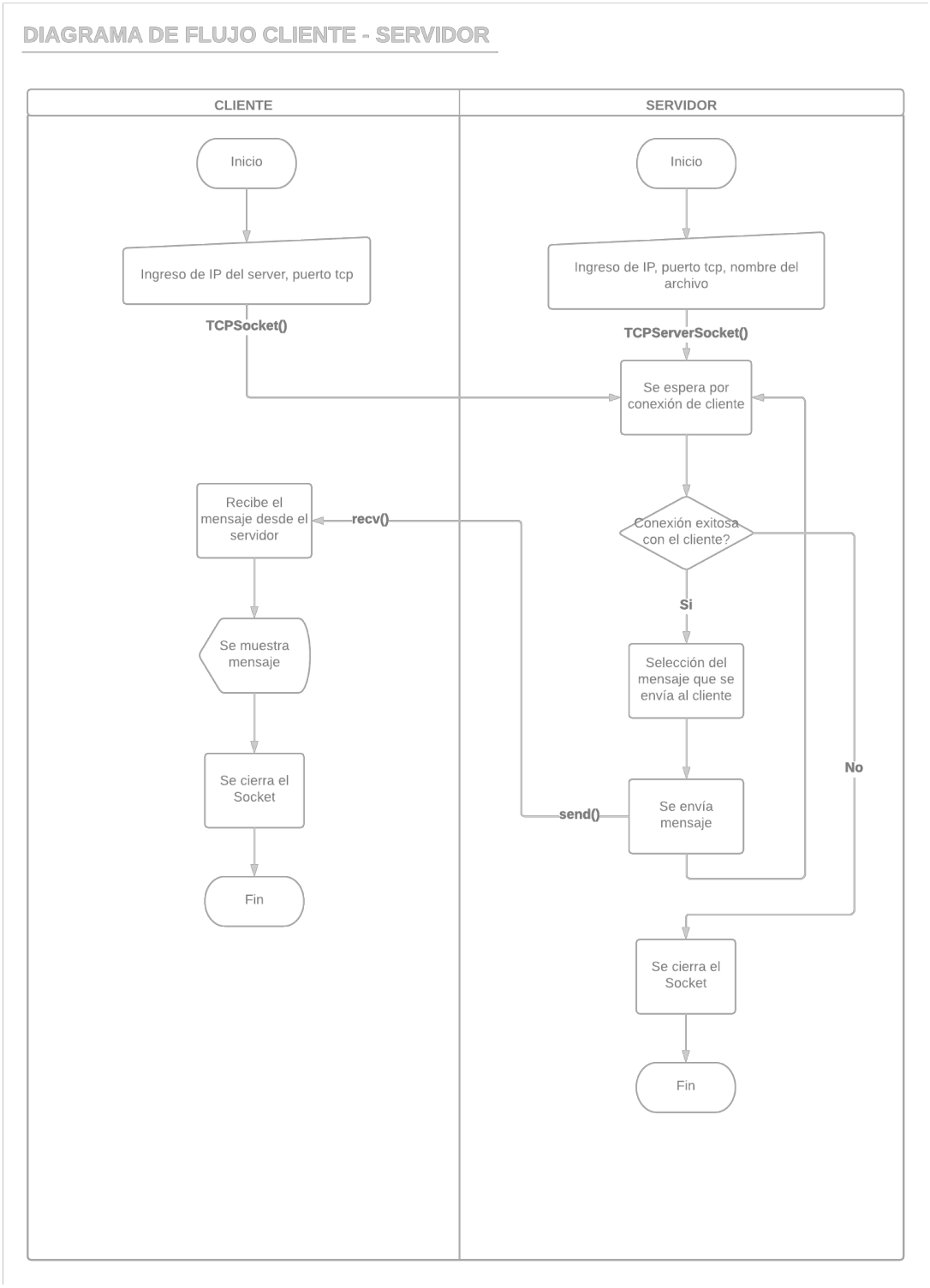
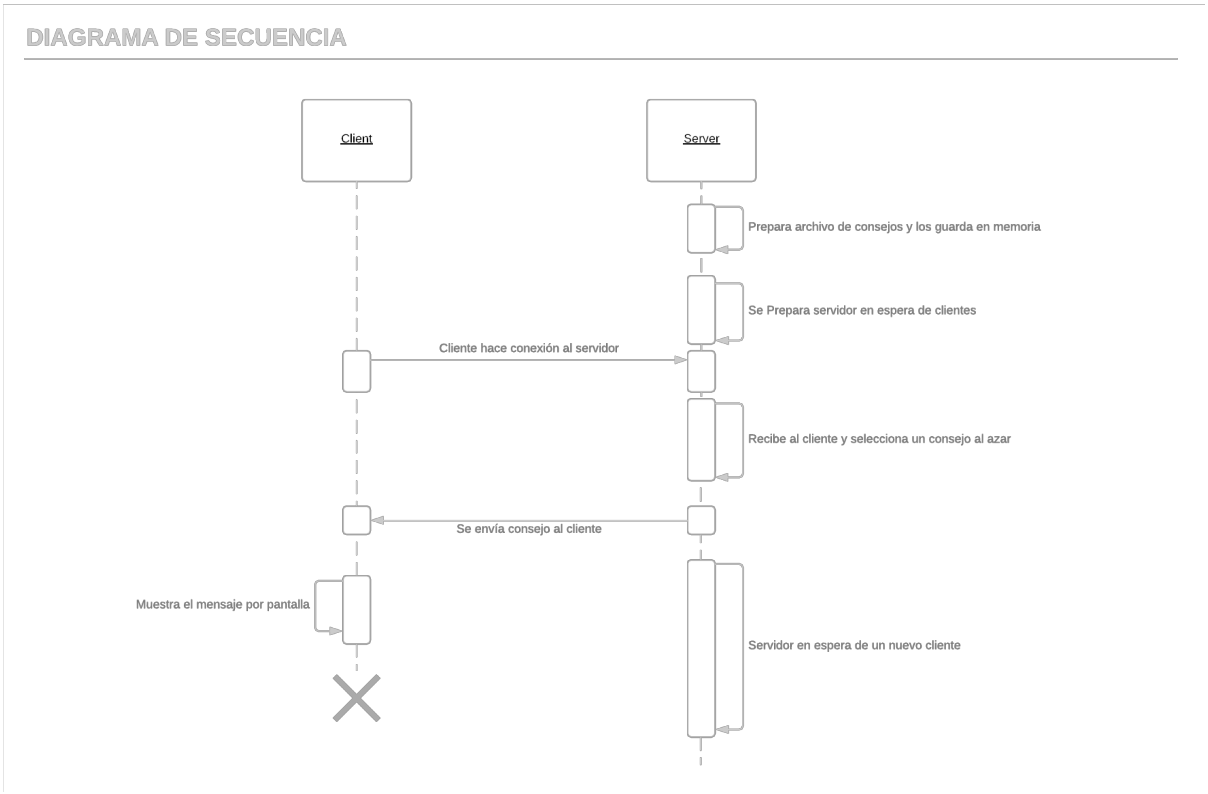


Diagrama de secuencia



BIBLIOGRAFÍA

Marini, E. (2012). El Modelo Cliente/Servidor. Recuperado https://www.linuxito.com/docs/el-modelo-cliente-servidor.pdf

David, G. (2003).Complete Maya Programming: An Extensive Guide to MEL and C++ API. Recuperado https://books.google.cl/books?hl=es&lr=&id=i_4S0tBcraAC&oi=fnd&pg=PP1&dq=mayas+c%2B%2B+api+nodes&ots=BJ8_jdYsKJ&sig=OK2cCCWoXaTNQheKV9BWbjCFKdM#v=onepage&q=mayas%20c%2B%2B%20api%20nodes&f=false

Gabriel, A.(2018). Redes de computadores, tarea 1. Universidad de Valparaiso, Valparaiso, Chile.