

Alumno: Franco Ariel Figueroa

DNI:

Departamento: Departamento de desarrollo productivo y tecnológico

Carrera: Licenciatura en Sistemas

Año: 2021

Materia: Redes y Comunicaciones

Profesores:

Índice

Creación del servidor y del cliente……………………………………………………………….3

Conexión de un cliente al servidor y su correspondiente login…………………………………..3

Explicación de la navegación del cliente a través del menú y del tiempo de espera del servidor..4

Comunicación entre el servidor y el cliente……………………………………………………...5

Uso de los archivos binarios junto con sus respectivas estructuras………………………………5

Trabajo de creación y administración de datos en los archivos binarios………………………...6

Archivos .log……………………………………………………………………………………..7

Ver registro de actividades……………………………………………………………………….7

Aclaraciones……………………………………………………………………………………...7

Servidor a detalle…………………………………………………………………………………8

Archivos utilizados en el servidor……………………………………………………….8

Funcionamiento del main.cpp……………………………………………………………9

Funcionamiento del Servidor.h…………………………………………………………10

Funcionamiento del Servidor.cpp………………………………………………………12

Cliente a detalle…………………………………………………………………………………18

Archivos utilizados en el cliente………………………………………………………..18

Funcionamiento del main.cpp…………………………………………………………..18

Funcionamiento del Cliente.h…………………………………………………………..20

Funcionamiento del Cliente.cpp………………………………………………………..21

Lector binario a detalle…………………………………………………………………………25

Archivos utilizados en el cliente……………………………………………………….25

Funcionamiento del main.cpp………………………………………………………….25

CREACIÓN DEL SERVIDOR Y DEL CLIENTE

Los proyectos del servidor y cliente utilizan algunas librerías y comandos específicos de Windows, como la librería winsock.h y los comandos system(“pause”) y system(“cls”).

Para la creación del servidor y del cliente se utilizan clases y objetos. Se usa la clase Servidor (definida en Servidor.h) para crear un objeto para el servidor y la clase Cliente (definida en Cliente.h) para crear otro objeto para el cliente.

Ambas clases tienen al menos un atributo del tipo WSADATA, que contiene información para la implementación de sockets a través de la librería winsock.h, una variable del tipo SOCKET para poder trabajar con sockets, una variable del tipo SOCKADDR\_IN, para almacenar la direcciones que van a usar los sockets, una variable del tipo int para almacenar la dirección del puerto en el que deben escuchar o conectarse, y una variable char buffer[1024] para enviar y recibir información entre sockets.

CONEXIÓN DE UN CLIENTE AL SERVIDOR Y SU CORRESPONDIENTE LOGIN

El servidor solo es capaz de atender un cliente a la vez. Una vez que el cliente se conecta al servidor, el servidor deja de escuchar por otros clientes que se quieran conectar.

El servidor le pide al cliente que ingrese usuario y contraseña y verificara si son correctos o no. Si son correctos, el cliente tiene acceso a todas las funciones que ofrece el servidor (las opciones para crear y administrar tipos de medicamentos, las opciones para crear y administrar medicamentos, ver actividad y cerrar sesión). Si no son correctos, se le pide al cliente que vuelva a ingresar usuario y contraseña. Si el cliente ingresa tres veces mal usuario y contraseña, el servidor lo desconecta y vuelve a ponerse en escucha por otro cliente que se quiera conectar.

Para saber si los datos ingresados por el cliente durante el login son correctos, el servidor lee el archivo “Usuarios.txt” en donde se encuentra guardado todos los usuarios y sus respectivas contraseñas.

EXPLICACIÓN DE LA NAVEGACIÓN DEL CLIENTE A TRAVÉS DEL MENU Y DEL TIEMPO DE ESPERA DEL SERVIDOR

Explicando más a detalle las funciones a las que tiene acceso el cliente una vez que pasó por el login; el cliente puede ir a “Tipos de Medicamentos”, a “Medicamentos”, a “Ver registro de actividad” y a “Cerrar sesión”.

Tipos de Medicamentos funciona como acceso a un submenú en donde el cliente puede acceder a:

* “Crear”: Donde se puede crear nuevos tipos de medicamentos.
* “Administrar”: Donde se realiza una búsqueda con filtros y el cliente puede modificar los datos de los tipos de medicamentos.
* “Ver Todo”: Donde se puede ver toda la información de todos los tipos de medicamentos.
* “Volver”: Que sirve para volver al menú principal.

Medicamentos funciona igual que Tipos de medicamentos. Sirve como acceso a un submenú en donde el cliente puede acceder a:

* “Crear”: Donde se puede crear nuevos medicamentos.
* “Administrar”: Donde se realiza una búsqueda con filtros y el cliente puede modificar los datos de los medicamentos o eliminar medicamentos.
* “Ver Todo”: Donde se puede ver toda la información de todos los medicamentos.
* “Volver”: Que sirve para volver al menú principal.

Ver registro de actividades sirve para que el servidor muestre al cliente toda la actividades que realizó. Dicha información se guarda en un archivo de .log llamado Mi\_actividad del lado del cliente.

Cerrar sesión sirve para que el cliente se pueda desconectar del servidor. El servidor volverá a entrar en estado de escucha por algún otro cliente que se quiera conectar.

Cada vez que el cliente ingresa a una función, se le envía un mensaje de informe al servidor para que sepa a que función entro el cliente y, en base a eso, el servidor se prepara para realizar una función u otra. Si el cliente no ingresa a ninguna función por 120 segundos (2 minutos), se considera al cliente como inactivo y se lo desconecta del servidor, y el servidor vuelve a ponerse en escucha por otro cliente que se quiera conectar.

El control del tiempo de inactividad del cliente se realiza del lado del servidor dentro de los métodos recibir() y recibirInforme(). Se crea una estructura del tipo timeval y se setea su variable tv\_sec en 120 (es decir 2 minutos). Después limpió la configuración anterior que tenía el socket del cliente en su file descriptor con FD\_ZERO y FD\_SET, y finalmente realizo una consulta sobre el socket cliente para chequear el tiempo transcurrido.

COMUNICACIÓN ENTRE EL SERVIDOR Y EL CLIENTE

Como se dijo anteriormente, cada vez que el cliente ingresa a una función, se le envía un mensaje al servidor para que sepa a que función entró el cliente y, en base a eso, el servidor se prepara para realizar una función u otra. Este informe sobre la navegación del usuario es enviado por el programa del cliente al servidor sin que el usuario lo sepa a través del método “enviarInforme()”. Por su parte, el servidor también tiene un método específico para recibir los informes llamada “recibirInforme()”. Gracias a estos dos métodos, la navegación del menú, las funciones invocadas y sus correspondientes respuestas se sincronizan y trabajan en conjunto.

Además de estos métodos de informes, tanto el cliente como el servidor cuentan con sus respectivos métodos “void enviar()” y “string recibir()” que sirven para el intercambio de información entre ambos. Primero el cliente ingresa a la función enviar() para escribir algún mensaje. Este mensaje se almacena en la variable de la clase Cliente “char buffer[1024]” y luego es enviado al servidor que lo recibe con su método recibir(). El servidor guarda el mensaje enviado por el cliente en la variable de la clase Servidor “char buffer[1024]” y luego su contenido es pasado a un string debido a que es más fácil trabajar con un string que con un char array. El string con el contenido del mensaje es lo que devuelve el método recibir(), el cual es usado en alguna función que solicite el cliente, como crear o administrar.

El servidor trabaja con la información enviada por el cliente, realizando los correspondientes chequeos (por ejemplo que la información sea del tipo solicitado o que no se repita, dichos chequeos de la función solicitada por el cliente) y envía su respuesta al cliente almacenando un mensaje en su buffer y enviándolo a través del método enviar(). El cliente recibe la respuesta del servidor en su método recibir(), la guarda en su buffer y lo convierte en un string.

Este ciclo se repite hasta que el servidor reciba toda la información que le solicita al cliente y hasta que el cliente haya recibido todo la información que le solicita al servidor.

USO DE LOS ARCHIVOS BINARIOS JUNTO CON SUS RESPECTIVAS ESTRUCTURAS

Para el manejo de los datos, se utiliza dos archivos binarios, un archivo binario para tipos de medicamentos (Datos\_Tipos\_De\_Medicamentos.dat) y otro archivo binario para los medicamentos (Datos\_Medicamentos.dat). Ambos archivos binarios se encuentran en la misma carpeta que el proyecto del servidor. Cuando se ingresa a la función crear o administrar, tanto para tipos de medicamento y para medicamentos, los archivos binarios son modificados según los datos que ingrese el cliente.

Para poder trabajar los archivos binarios se utilizan dos estructuras del lado del servidor, una estructura llamada “tipoMedicamento” (con los atributos int id, char tipo[20] y bool activo) que sirve para leer y escribir en el archivo binario Datos\_Tipos\_De\_Medicamentos.dat, que como su nombre lo indica, guarda toda la información de los tipos de medicamento; y la otra estructura llamada “medicamento” (con los atributos int id, char codigo[15], char nombre[20], char droga[20] y tipo[20]) que sirve para leer y escribir en el archivo binario Datos\_Medicamentos.dat, que, valga la redundancia, guarda toda la información de los medicamentos.

TRABAJO DE CREACIÓN Y ADMINISTRACIÓN DE DATOS EN LOS ARCHIVOS BINARIOS

Para trabajar con los archivos binarios primero se verifica que la información enviada por el cliente al servidor sea válida.

Si la información recibida del cliente no es válida, el servidor le envía un mensaje al cliente informándole del error, por ejemplo, si el usuario quiere crear un tipo de medicamento, pero ese tipo de medicamento ya existe, el servidor le manda un mensaje al cliente que dice “El tipo de medicamento ya existe, por favor ingrese otro”. Luego de enviar el mensaje, el cliente tiene que volver a mandar información al servidor, la cual se vuelve a chequear. Este ciclo se repite hasta que el cliente ingrese datos válidos. La verificación que se aplica varía de acuerdo a la función solicitada por el cliente.

Si la información recibida del cliente es válida, se procede a trabajar con la función solicitada por el cliente (ya sea crear o administrar). En el caso de crear, se abre el archivo binario correspondiente (puede ser Datos\_Tipos\_De\_Medicamentos.dat o Datos\_Medicamentos.dat, según lo que haya pedido el cliente) en modo escritura, y con la ayuda de las estructuras definidas del lado del servidor (tipoMedicamento y medicamento) se escribe al final del archivo binario la información ingresada por el cliente con un ID generado al azar entre el 1 y el 100 que no repite. En el caso de administrar, se abre el archivo binario correspondiente en modo lectura y se le envía al cliente la información de los tipos de medicamentos o medicamentos que cumplen con los filtros y se le pide que ingrese el ID del que quiera modificar (o eliminar si el cliente pidió administrar los medicamentos).

En el caso de modificar, una vez ingresado el ID, el cliente tiene que introducir los datos correspondientes a tipo de medicamentos o medicamentos (según lo que este administrado el cliente). Estos datos también pasan por un chequeo. Si no son válidos, se le informa al cliente de ello y se le pide que ingrese otros datos; si son válidos se le pregunta al cliente si quiere guardar los cambios o no. Si quiere guardar los cambios, se crea un archivo binario auxiliar y se lo abre en modo escritura para que guarde la información introducida por el cliente más los datos del archivo binario abierto previamente (a excepción de los datos que están siendo modificados). Después se cierren ambos archivos binarios, se elimina el archivo binario original y el archivo binario auxiliar es renombrado con el nombre del archivo binario eliminado. Si no quiere guardar los cambios simplemente se cierra el archivo binario abierto en modo lectura y se vuelve al menú.

En el caso de eliminar (solo para los medicamentos); una vez ingresado el ID, se le pregunta al cliente si quiere guardar los cambios. Si quiere guardar los cambios, se crea un archivo binario auxiliar y se lo abre en modo escritura para que guarde los datos del archivo binario abierto previamente (a excepción de los datos que se quieren eliminar). Después se cierren ambos archivos binarios, se elimina el archivo binario original y el archivo binario auxiliar es renombrado con el nombre del archivo binario eliminado. Si no quiere guardar los cambios simplemente se cierra el archivo binario abierto en modo lectura y se vuelve al menú.

ARCHIVOS .LOG

Toda la actividad del servidor y del cliente es guardada en archivos .log. En el caso del servidor, su actividad es guardada en el archivo “server.log”, y en el caso del cliente su actividad es guardada en un archivo .log cuyo nombre es el usuario con el que inició sesión.

Los archivos .log son modificados cuando el cliente inicia o cierra sesión, cuando crea algún tipo de medicamento o algún medicamento, cuando en la función administrar (tanto para tipos de medicamentos como para medicamentos) se elige la opción de guardar cambios y cuando se elige la función Ver todo.

VER REGISTRO DE ACTIVIDADES

Cuando el cliente solicita ver su registro de actividades, el servidor abre el archivo .log cuyo nombre sea igual que el del usuario en modo lectura. Después empieza a leerlo y envía la información del .log al cliente.

Por su lado, el cliente le muestra la información al usuario y crea un archivo .log llamado “Mi\_actividad”, en el que va escribiendo la información enviada por el servidor a medida que llega. Si ya existía un archivo “Mi\_actividad.log”, el cliente lo borra y lo vuelve a crear para que no haya datos repetidos, y entonces empieza la escritura.

ACLARACIONES

Aparte de los proyectos de servidor y cliente, hay otro proyecto llamado “Lector binario” que sirve para crear los archivos binarios, llenarlos con los datos que se requieran, y leer su contenido.

La función “Ver todo” que se encuentra en el submenú de tipos de medicamentos y en el submenú de medicamentos no se pedía en las consignas. Dicha función fue un agregado mío para que el cliente pueda ver todos los tipos de medicamentos o todos los medicamentos sin necesidad de ir a la función “Administrar” y escribir @ en todos los filtros de búsqueda.

SERVIDOR A DETALLE

ARCHIVOS UTILIZADOS POR EL SERVIDOR

El proyecto del servidor esta conformado por los archivos:

Servidor.cbp: El proyecto en sí. Relaciona los archivos .cpp, los archivos .h y demás para hacer funcionar todo.

include: Dentro de dicha carpeta se aloja los archivos .h utilizados por el proyecto servidor. En este caso, se uso un solo archivo .h, Servidor.h

src: Dentro de dicha carpeta se aloja los archivos .cpp (a excepción del main.cpp) utilizados por el proyecto servidor. Es este caso, dentro de dicha carpeta solo hay un archivo .cpp, Servidor.cpp

main.cpp: En él se define el puerto de escucha del servidor y se crea un objeto de la clase Servidor para que funcione como (valga la redundancia) servidor.

bin: Dentro de dicha carpeta se encuentra otra carpeta llamada Debug y dentro de ella se encuentra el ejecutable del proyecto.

obj: Dentro de dicha carpeta se encuentra lo que necesita el proyecto en cuanto a objetos.

Datos\_Medicamentos.dat: Es un archivo binario en donde se guarda toda la información de los medicamentos.

Datos\_Tipos\_De\_Medicamentos.dat: Es un archivo binario en donde se guarda toda la información de los tipos de medicamentos.

Usuarios.txt: Es un archivo de texto en donde se guardan los usuarios válidos y sus respectivas contraseñas.

FUNCIONAMIENTO DEL MAIN.CPP

En la variable int puerto se declara el puerto en el va a escuchar a servidor. A continuación, se llama a la función registrarInicioServidor(), que registra el arranque del servidor en el archivo server.log. Estas dos instrucciones son todo lo que va a hacer el main.cpp en el inicio del servidor.

Se procede a entrar a un while infinito que va a servir para que el servidor se quede escuchando/ejecutando todo el tiempo que queramos, hasta que lo cerremos manualmente. Dentro del while infinito, se crea el objeto servidor y se llama a la función login() dentro del if. Para llamar al login() primero se tiene que conectar un cliente al servidor. Cuando se conecta un cliente, se cierra el socket del servidor (el servidor va a seguir andando solo que ya no va a escuchar por más conexiones). Si el cliente ingresa el usuario y contraseña correcta, pasa el login; si no, tiene hasta 3 intentos para hacerlo y si se equivoca el cliente las 3 veces se lo desconecta. A continuación, se procede a destruir el objeto servidor y a volver a crearlo para que vuelva a quedarse en escucha (esto sería el equivalente a apagar y prender el servidor en la vida real, lo cual sería un error fatal, pero como es un caso especial en el que se atiende un solo cliente a la vez es la solución más sencilla de todas porque nos ahorra el uso de hilos para atender múltiples clientes y no tiene un impacto negativo en los demás clientes que estén en espera para conectarse al servidor).

Si el cliente pasa el login, el servidor entra a otro while que sirve para manejar la navegación del cliente a través del menú. Este while no es infinito y se saldrá de él cuando el cliente cierre sesión o se desconecte abruptamente. Dentro del segundo while, el servidor espera el informe sobre la navegación del cliente y, en base al contenido del informe, el servidor entra a una caso u otro de un switch case, el cual sirve para que el servidor sepa que pide el cliente.

FUNCIONAMIENTO DEL SERVIDOR.H

Dentro del .h se declara y define la clase Servidor. Esta clase es muy importante ya que tiene los atributos necesarios para realizar la conexión y en su constructor se establece la conexión con el cliente. La clase también tiene definida un destructor, el método string recibir(), el método void enviar(), el método char recibirInforme() y el método void cerrarSocket().

Propósito de los métodos:

* string recibir(): Chequea el tiempo de inactividad del cliente, si es igual a 120 segundos llama al método cerrarSocket(), y recibe los mensajes del cliente. El mensaje es recibido en el char buffer[1024] y luego convertido a un string para que sea más fácil de manejar.
* void enviar(): Envía el contenido del buffer del servidor al cliente en forma de mensaje. Como cada mensaje hacia el cliente puede ser diferente, se necesita cargar el buffer primero y después llamar a este método.
* char recibirInforme(): El programa del cliente envía informes sobre la navegación del cliente a través del menú sin que este lo sepa. Este método recibe dichos informes y permite al servidor realizar exactamente las funciones que solicita el cliente.
* void cerrarSocket(): Sirve para cerrar la conexión con el cliente.

Aparte de la clase Servidor, también se declaran funciones que luego serán definidas en el Servidor.cpp. Dichas funciones son:

Funciones para los registros:

* string fechaHoraActual: Chequea el tiempo y fecha de la maquina para devolverlo en un string.
* void registrarInicioServidor(int puerto): Registra en el archivo server.log el momento de

inicio del servidor.

Funciones para cuando un cliente intenta hacer login en el servidor:

* boolean chequearUsuarioYContrasenia(string usuarioIngresado, string contraseniaIngresada): Lee el archivo “Usuarios.txt” y chequea que el usuario y contraseña ingresado por el cliente sea correcto.
* boolean login(Servidor servidor): El cliente se conecto al servidor y por eso se le pide que ingrese un usuario y contraseña. Se chequea si son válidos llamando a la función chequearUsuarioYContrasenia. El cliente tiene 3 intentos para ingresar un usuario y contraseña válidos. Si los 3 intentos son inválidos se desconecta el cliente.

Funciones del submenú Tipo de medicamento:

* void atenderTipoMedicamento(Servidor servidor): Es el submenú de tipo de medicamento. Sirve para saber que opción del submenú elige el cliente.
* void crearTipoMedicamento (Servidor servidor): Sirve para que el cliente pueda crear un tipo de medicamento.
* void administrarTipoMedicamento(Servidor servidor): Sirve para que el cliente haga una búsqueda por filtro y modifique el tipo de medicamento que quiera (o no).
* void verTodoMedicamento(Servidor servidor): Sirve para ver todos los tipos de medicamentos que existen.

Función que se usa en otros métodos:

* string convertirAMayusculas(string s): Convierte todos los caracteres de un string a mayúsculas.

Funciones del submenú Medicamento:

* void atenderMedicamento(Servidor servidor): Es el submenú de medicamento. Sirve para saber que opción del submenú elige el cliente.
* void crearMedicamento (Servidor servidor): Sirve para que el cliente pueda crear un medicamento.
* bool validador\_de\_codigo(char código[15]): Chequea que el código tenga un formato valido y verifica el digito validador.
* void administrarMedicamento(Servidor servidor): Sirve para que el cliente haga una búsqueda por filtro y modifique o elimine el medicamento que quiera (o no).
* void verTodoMedicamento(Servidor servidor): Sirve para ver todos los medicamentos que existen.

Función independiente de los submenús:

* void transmitirArchivoDeActividades(Servidor servidor): Le envía al usuario el contenido de su respectivo archivo .log

FUNCIONAMIENTO DEL SERVIDOR.CPP

Al principio se definen dos estructuras, tipoMedicamento y medicamento, que servirán en varias funciones para la lectura y escritura de los archivos binarios Datos\_Tipos\_De\_Medicamentos.dat y Datos\_Medicamentos.dat. También se declara el string nombreArchivoCliente de forma global ya que se usará en varias funciones para registrar la actividad del usuario.

Funciones a detalle:

string fechaHoraActual():

Chequea el tiempo y fecha de la máquina, lo guarda en un char array y luego se convierte el dato a string porque es más fácil manejarlo de esa forma. Este string es el que devuelve la función.

Cada vez que se necesite actualizar el registro del cliente o del servidor, se llama a esta función para registrar también la fecha y hora del registro.

void registrarInicioServidor(int puerto):

Registra en el archivo server.log el momento de inicio del servidor. Esta función se va a ejecutar una sola vez en toda la ejecución del servidor. Pide la hora y fecha a fechaHoraActual() y crea un archivo ofstream que se usará para manejar el registro del servidor, server.log.

Abre el archivo server.log en modo escritura, y si no existe, lo crea. Finalmente escribe al final del server.log el registro de que se inició el servidor y el puerto en el que escucha.

boolean chequearUsuarioYContrasenia(string usuarioIngresado, string contraseniaIngresada):

Se abre el archivo Usuarios.txt en modo lectura, si no se puede, se envía mensaje de error. Esto es para comparar si el usuario y contraseña de los parámetros coinciden con el del archivo.

Primero se entra en un while que servirá para leer el archivo Usuarios.txt hasta que los datos sean válidos o hasta que ya no haya nada más para leer.

Se separa el usuario y contraseña del txt. Se lee carácter por carácter y cuando se encuentra con un punto y coma (;) significa que hasta llega el usuario. El usuario estará conformado por todos los caracteres leídos, menos el punto y el coma. Se procede a seguir con la lectura carácter por carácter y cuando se encuentre con un salto de línea (\n) significa que hasta llega la contraseña. La contraseña estará conformada por todos los caracteres leídos, menos el salto de línea.

Con el usuario y contraseña ya separados del archivo Usuarios.txt, se compara estos con los parámetros. Si coinciden, el usuario y contraseña son válidos y se sale del while; si no coinciden, no son válidos y se vuelve al inicio del while para leer la siguiente línea del archivo txt.

Finalmente se cierra el archivo y se devuelve el boolean que indica si los datos son válidos o no.

boolean login(Servidor servidor):

Si se ingresa a esta función significa que algún cliente se conecto al servidor. Se le pide al cliente que ingrese un usuario y contraseña, y los datos enviados por el cliente son verificados con la función chequearUsuarioYContrasenia, cuyos parámetros son cargados con la información recibida.

El cliente tiene tres intentos para ingresar un usuario y contraseña válidos. Si el cliente ingresó datos inválidos y todavía no supera los tres intentos, se le pide que los vuelva a ingresar.

Si en el primer o segundo intento los datos no son válidos, se le envía un mensaje al cliente indicándole que los datos que ingresó no son válidos.

Si en el tercer intento los datos no son válidos, se le envía un mensaje al cliente que le dice que se superó la cantidad de intentos de ingreso y se lo desconecta del servidor. El servidor se vuelve a poner en escucha por posibles conexiones.

void atenderTipoMedicamento(Servidor servidor):

Es el submenú de tipo de medicamento. Sirve para saber que opción del submenú elige el cliente.

Se entra en un while en el que hay un switch con cuatro casos. Cada caso llama a una función del submenú. Al principio del while, se recibe un informe sobre la navegación del cliente en su máquina y según este informe se accede a un caso u otro.

Si el cliente elige la opción 4, se sale del while y se vuelve al menú principal.

Si elige cualquier opción no válida, desde el lado del cliente se le informa al cliente que ingrese una opción válida.

void crearTipoMedicamento(Servidor servidor):

Sirve para que el cliente pueda crear un tipo de medicamento.

El servidor recibe el tipo de medicamento que el cliente quiere crear y lo guarda en la variable string tipo.

Se abre el archivo Datos\_Tipos\_De\_Medicamentos.dat en modo lectura binaria y con ayuda de la estructura tipoMedicamento definida al principio del Servidor.cpp, podemos leerlo.

Se crea un vector de int llamado IDexistentes en el que se guardara los ID de los tipos de medicamentos existentes en el archivo binario a medida que vayamos leyendo y también se crea un número al azar entre 1 y 100.

Se procede a entrar en un while con el que chequea si el tipo introducido por el cliente ya existe. Para saber esto se lee todo el archivo binario, y se compara el tipo obtenido en la lectura con el tipo ingresado por el cliente, ambos con cada carácter convertido a mayúsculas. Si coinciden, se le envía un mensaje al cliente informándole al cliente de dicha situación, y tanto el cliente como el servidor vuelven al submenú de tipo de medicamentos. Si no coinciden, se lee el siguiente fragmento del archivo binario y así hasta que ya no haya nada más para leer

Independientemente de si hubo un fallo durante la lectura o no, se cierra el archivo Datos\_Tipos\_De\_Medicamentos.dat que estaba abierto en modo lectura.

Si no se detectó ningún error durante el chequeo, se chequea que el número al azar creado anteriormente no coincida con algún número de lo que hay guardados en el vector IDexistentes. Si algún ID y el número al azar coinciden, se vuelve a crear un número al azar entre el 1 y el 100 y se vuelve a chequear el vector y el número al azar desde el principio. Así hasta que el número de azar no se repita.

Con la ayuda de una estructura tipoMedicamento, se escribe el tipo de medicamento creado con su id (que es el número al azar), su tipo (el ingresado por el cliente), y su activo (siempre en true durante la creación) en el archivo Datos\_Tipos\_De\_Medicamentos.dat., el cual se volvió a abrir en modo de escritura binaria.

Por último, se registra la creación en el archivo server.log y en el archivo .log del usuario.

string convertirAMayusculas(string s):

Se usa un for para recorrer el string del parámetro carácter por carácter y se lo convierte a mayúsculas. Se devuelve el string del parámetro en mayúsculas.

void administarTipoMedicamento(Servidor servidor):

Sirve para que el cliente haga una búsqueda por filtro y modifique el tipo de medicamento que quiera (o no).

El servidor recibe el tipo y el activo por el que quiere filtrar el cliente y crea un vector de int llamado IDexistentes en el que se guardara los ID de los tipos de medicamentos existentes en el archivo binario a medida que vayamos leyendo.

Se abre el archivo Datos\_Tipos\_De\_Medicamentos.dat en modo de lectura binaria y se entra en un while que servirá para recorrer y leer todo el archivo binario. Dentro del while hay varios if, los cuales sirven para mostrarle al cliente solamente los datos que cumplan con los datos que ingreso. Cada if funciona igual; se fija si los datos de la lectura del archivo binario cumplen con los filtros del cliente, y si coinciden, el servidor le envía un mensaje al cliente con la información de la lectura. Después se hace otra lectura del archivo binario y así se repite el proceso hasta que ya no haya más para leer, en cuyo caso el servidor le informa al cliente que ya no espere por más información.

Al final de la lectura se cierra el archivo binario que estaba abierto en modo lectura, independientemente de si se encontró algún error o no.

Si al final de la lectura no se encontró ningún tipo de medicamento que cumpla con los filtros, se el envía un mensaje al cliente informándoselo y tanto el cliente como el servidor vuelven al submenú de tipo de medicamentos.

Si al final de la lectura si hubo algún tipo de medicamento que cumpla con los filtros, se le pide al cliente que ingrese el ID del tipo de medicamento que quiera modificar. Esta petición está dentro de un while para que, en caso de que el cliente ingrese una ID no válida o que no exista, se le envíe un mensaje indicándole que ingrese otro ID. El chequeo del ID se hace gracias al vector int IDexistentes definido al principio, se compara lo ingresado por el cliente con cada elemento int del vector.

A continuación, el servidor recibe las modificaciones que el cliente quiere hacer a los atributos (a excepción del ID que nunca es modificado).

Se abre el archivo binario Datos\_Tipos\_De\_Medicamentos.dat en modo lectura y se crea un archivo binario auxiliar para abrirlo en modo de escritura binaria.

Hay varios if que sirven para realizar el tipo de modificación que quiera el usuario. Si en algún atributo a modificar el cliente escribió un @, dicho atributo quedara sin modificar. Todos los if funcionan igual, se lee el archivo Datos\_Tipos\_De\_Medicamentos.dat y a medida que se lee, se copia los datos en el archivo binario auxiliar, con excepción del tipo de medicamento que tenga el mismo ID que el que ingresó el cliente, en su lugar se escribe los datos modificados que ingresó el cliente.

Por último, el servidor recibe la confirmación o la negación para guardar los datos modificados. Si el cliente confirma guardar los datos, el archivo Datos\_Tipos\_De\_Medicamentos.dat es eliminado y el archivo binario auxiliar (el cual tiene los datos modificados por el cliente) pasa a llamarse Datos\_Tipos\_De\_Medicamentos.dat. La actividad es registrada en el server.log y en el .log del cliente. Si el cliente niega guardar los cambios simplemente se elimina el archivo binario auxiliar. La actividad no es registrada porque no hubo ningún cambio en los datos.

void verTodoTipoMedicamento(Servidor servidor):

Se abre el archivo Datos\_Tipos\_De\_Medicamentos.dat en modo de lectura binaria y, a medida que se va leyendo, se le envía la información de todos los tipos de medicamentos dentro del archivo al cliente. Al final de la lectura, se le envía un mensaje al cliente de que ya no espere por más información. Registra dicha consulta en el server.log y en el .log del cliente.

void atenderMedicamento(Servidor servidor):

Es el submenú de medicamento. Sirve para saber que opción del submenú elige el cliente.

Se entra en un while en el que hay un switch con cuatro casos. Cada caso llama a una función del submenú. Al principio del while, se recibe un informe sobre la navegación del cliente en su máquina y según este informe se accede a un caso u otro.

Si el cliente elige la opción 4, se sale del while y se vuelve al menú principal.

Si elige cualquier opción no válida, desde el lado del cliente se le informa al cliente que ingrese una opción válida.

void crearMedicamento(Servidor servidor):

Se crea un número al azar entre el 1 y 100 y se crea un vector de int llamado IDexistentes en el que se guardara los ID de los medicamentos existentes en el archivo binario Datos\_Medicamentos.dat a medida que lo vayamos leyendo.

El servidor recibe el código del nuevo medicamento que quiere crear el cliente y verifica que el digito verificador sea válido a medida que lee el archivo Datos\_Medicamentos.dat para verificar que no esté repetido el código en algún otro medicamento. Si el código es repetido o es inválido el digito verificador se le envía un mensaje al cliente informándole de dicha situación para que ingrese otro código.

Una vez el código paso el chequeo, se cierra el archivo Datos\_Medicamentos.dat que estaba abierto en modo lectura y el servidor recibe el nombre y el tipo del nuevo medicamento. Se abre el archivo Datos\_Tipos\_De\_Medicamentos.dat en modo de lectura binaria y se chequea que exista el tipo de medicamento ingresado por el cliente. Si no existe el tipo de medicamento en el archivo binario, se le pide al cliente que vuelva a introducir el tipo, y se chequea nuevamente.

Una vez que se obtiene un tipo de medicamento existente en Datos\_Tipos\_De\_Medicamentos.dat, se cierra dicho archivo y se pasa al siguiente chequeo, el cual consiste en verificar que la dupla nombre-tipo no este repetida. Para esto se abre el archivo Datos\_Medicamentos.dat en modo lectura y se lee todo el contenido del archivo. En cada lectura que se haga, se compara el nombre y tipo obtenidos de la lectura con los nombre y tipo ingresados por el cliente. Si se encuentra que tanto el tipo como el nombre se repiten se le envía un mensaje al cliente diciéndole que la dupla ya existe y que ingrese otro nombre y tipo. Al tipo que ingresa el usuario de vuelta se le vuelve a hacer el chequeo para verificar que exista en Datos\_Tipos\_De\_Medicamentos.dat.

El atributo droga que envía el cliente al servidor simplemente es recibida como viene ya que no se pedía que pase por ningún chequeo.

Si no se detectó ningún error durante el chequeo, se chequea que el número al azar creado anteriormente no coincida con algún número de lo que hay guardados en el vector IDexistentes. Si algún ID y el número al azar coinciden, se vuelve a crear un número al azar entre el 1 y el 100 y se vuelve a chequear el vector y el número al azar desde el principio. Así hasta que el número de azar no se repita.

Se guarda todos los datos recibidos y chequeados en una estructura del tipo medicamento y con ella, se escribe el nuevo medicamento al final del archivo Datos\_Medicamentos.dat. Para finalizar se guarda el registro sobre la creación del nuevo medicamento en el archivo server.log y en el .log del cliente.

bool validador\_de\_codigo(char codigo[15]):

Chequea que el código tenga un formato valido y verifica el digito validador. Primero se convierte los dígitos del código de char a int y se los suma. Si la suma es mayor o igual a 10, se separá la suma en su parte derecha y en su parte izquierda y las suma, y si esta suma también es mayor o igual a 10 se repite el proceso.

Se compara la suma con el digito verificador y si ambos son booleanos se pone a la variable bool coinciden en true. Por último, realizó un chequeo para comprobar que los números del código son realmente números y no algún carácter. Si se detecta que alguna posición hay un carácter en lugar de un número se pone a la variable bool coinciden en false.

Se devuelve la variable bool coinciden.

void administarMedicamento(Servidor servidor):

Sirve para que el cliente haga una búsqueda por filtro y modifique o elimine el medicamento que quiera (o no).

El servidor recibe del cliente el nombre y el tipo por el cual quiere filtrar la búsqueda. También crea un vector del tipo int llamado IDexistentes para guardar los ID de los medicamentos a medida que se lea el archivo Datos\_Medicamentos.dat.

Se abre el archivo Datos\_Medicamentos.dat en modo lectura binaria y se entra en un while que servirá para leer todo el archivo binario. Dentro del while hay varios if que servirán para filtrar el contenido del archivo según lo que ingresó el cliente. Todos los if se fijan si el contenido de la lectura actual cumple con los filtros del cliente y, en caso de que lo haga, el servidor envía todo el contenido del medicamento al cliente. Después se hace otra lectura del archivo binario y así se repite el proceso hasta que ya no haya más para leer, en cuyo caso el servidor le informa al cliente que ya no espere por más información.

Al final de la lectura se cierra el archivo Datos\_Medicamentos.dat que estaba abierto en modo lectura, independientemente de si se encontró algún error o no.

Si no se encontró a ningún medicamento que cumpla con los filtros, el servidor le informa al cliente del error y ambos son devueltos al submenú de medicamentos.

Si al final de la lectura si hubo algún medicamento que cumpla con los filtros, se le pide al cliente que ingrese el ID del medicamento que quiera modificar. Esta petición está dentro de un while para que, en caso de que el cliente ingrese una ID no válida o que no exista, se le envíe un mensaje indicándole que ingrese otro ID. El chequeo del ID se hace gracias al vector int IDexistentes definido al principio, se compara lo ingresado por el cliente con cada elemento int del vector.

El servidor recibe el informe de lo que quiere hacer el usuario con el medicamento con el mismo ID que ingresó, modificar o eliminar.

Si quiere modificar, el servidor empieza recibiendo el nuevo código del medicamento, al cual se le hace chequeos para comprobar que sea válido (a través de la función validador\_de\_codigo) y que no este repetido en algún otro medicamento. Si no cumple con alguno de estos dos requisitos, el servidor le informa al cliente del error y le pide que ingrese otra contraseña. Si el cliente ingresó @ se saltea el chequeo porque significa que no quiere que se modifique el atributo.

Una vez que el servidor recibió un código válido y único, procede a recibir el nombre y el tipo. Al tipo recibido se le hace un chequeo para verificar que exista. Para esto se abre el archivo Datos\_Tipos\_De\_Medicamentos.dat en modo lectura binaria y se lo lee. Si en algún momento de la lectura, el tipo ingresado por el cliente y el tipo leído coinciden, significa que el tipo es válido. Si el cliente ingresó @ se saltea el chequeo porque significa que no quiere que se modifique el atributo.

Con el tipo validado, se cierra el archivo Datos\_Tipos\_De\_Medicamentos.dat y se procede a chequear que la dupla nombre-tipo sea válida. Para hacer el chequeo se abre el archivo Datos\_Medicamentos.dat en modo lectura y, a medida que se lo va leyendo, compara el nombre y tipo ingresados por el cliente con el de la lectura.

Si en algún momento de la lectura coinciden, significa que la dupla ya existe, por lo tanto, no es válida. En dicho caso se le envía un mensaje al cliente informándole que la dupla que ingresó es repetida y que vuelva a introducir un nombre y tipo. El tipo vuelve a pasar por el chequeo (o no, si el cliente ingresó @) y la dupla vuelve a ser chequeada.

Si en ningún momento de la lectura de todo el archivo binario Datos\_Medicamentos.dat se encuentra ninguna coincidencia, se considera que la dupla ingresada por el cliente es válida y se cierra el archivo binario.

Próximamente, el servidor recibe la nueva droga del medicamento (si se recibe @ significa que no se quiere modificar dicho atributo) sin realizar ningún chequeo ya que este no es pedido en las consignas del trabajo práctico.

Se crea un archivo binario llamado Archivo\_Auxiliar.dat y se lo abre en modo escritura. También se abre el archivo binario Datos\_Medicamentos.dat en modo lectura. Se entra en un while en el que se lee todo el contenido de Datos\_Medicamentos.dat y, a medida que se lo lee, se copia su contenido en Archivo\_Auxiliar.dat, con excepción del medicamento que tenga el mismo ID que ingresó el cliente. En ese caso se escribe en Archivo\_Auxiliar los datos modificados que ingresó el cliente. Los atributos en los que el cliente ingresó un @ si son copiados de Datos\_Medicamentos.dat.

Por último, se cierran ambos archivos binarios y el servidor recibe el mensaje del cliente para guardar o no los cambios. En el caso de que se quieran guardar los cambios, se borra el archivo Datos\_Medicamentos.dat y Archivo\_Auxiliar.dat pasa a llamarse Datos\_Medicamentos.dat, y se registra la actividad en los archivos server.log y el .log del usuario. En el caso de que no se quiera guardar los cambios, se borra el archivo Archivo\_Auxiliar.dat y no se registra la actividad.

Si se quiere eliminar, se crea un archivo binario llamado Archivo\_Auxiliar.dat y se lo abre en modo escritura. También se abre el archivo binario Datos\_Medicamentos.dat en modo lectura. Se entra en un while en el que se lee todo el contenido de Datos\_Medicamentos.dat y, a medida que se lo lee, se copia su contenido en Archivo\_Auxiliar.dat, con excepción del medicamento que tenga el mismo ID que ingresó el cliente. En ese caso simplemente no se escribe nada en el archivo binario auxiliar.

Se cierran ambos archivos binarios y el servidor recibe el mensaje del cliente para guardar o no los cambios. En el caso de que se quieran guardar los cambios, se borra el archivo Datos\_Medicamentos.dat y Archivo\_Auxiliar.dat pasa a llamarse Datos\_Medicamentos.dat, y se registra la actividad en los archivos server.log y el .log del usuario. En el caso de que no se quiera guardar los cambios, se borra el archivo Archivo\_Auxiliar.dat y no se registra la actividad.

void verTodoMedicamento(Servidor servidor):

Se abre el archivo Datos\_Medicamentos.dat en modo de lectura binaria y, a medida que se va leyendo, se le envía la información de todos los medicamentos dentro del archivo al cliente. Al final de la lectura, se le envía un mensaje al cliente de que ya no espere por más información. Registra dicha consulta en el server.log y en el .log del cliente.

void transmitirArchivoDeActividades(Servidor servidor):

Se abre el archivo .log del cliente en modo de lectura, y a medida que se lo lee, se le envía su contenido al cliente. Al final de la lectura, se le envía un mensaje al cliente de que ya no espere por más información.

CLIENTE A DETALLE

ARCHIVOS UTILIZADOS EN EL CLIENTE

El proyecto del cliente está conformado por los archivos:

Cliente.cbp: El proyecto en sí. Relaciona los archivos .cpp, los archivos .h y demás para hacer funcionar todo.

Cliente.h: En dicho archivo se declara y define la clase Cliente y se declara las funciones que usara el programa del cliente.

Cliente.cpp: En dicho archivo se definen las funciones declaradas en Cliente.h

main.cpp: En él se crea un objeto de la clase Cliente para que funcione como el cual se conectara al servidor.

bin: Dentro de dicha carpeta se encuentra otra carpeta llamada Debug y dentro de ella se encuentra el ejecutable del proyecto.

obj: Dentro de dicha carpeta se encuentra lo que necesita el proyecto en cuanto a objetos.

FUNCIONAMIENTO DEL MAIN.CPP

Se crea un objeto Cliente. A continuación, se entra en un while en el cual se crea un objeto cliente auxiliar que se conecta al servidor e intenta pasar el login. Si logra pasar el login, los datos del objeto auxiliar y del objeto original pasan a ser los mismos y se borra el objeto auxiliar; si no logra pasar el login, se borra el objeto auxiliar y se repite el proceso del while. Esto se hace así porque, en caso de que el cliente falle el login, pueda volver a intentar hacer login de vuelta.

Una vez pasado el login, el cliente llama a la función menu(), la cual hace que quede dentro de un while en donde pueda llamar a todas las funciones que quiera hasta que elija cerrar sesión para cerrar el programa.

FUNCIONAMIENTO DEL CLIENTE.H

Dentro del .h se declara y define la clase Cliente. Esta clase es importante ya que tiene los atributos necesarios para realizar la conexión y en su constructor se establece la conexión con el servidor. La clase también tiene definida un destructor, el método string recibir(), el método void enviar(), el método char enviarInforme() y el método void cerrarSocket().

Propósito de los métodos:

* string recibir(): Recibe los mensajes del servidor. El mensaje es recibido en el char buffer[1024] y luego convertido a un string para que sea más fácil de manejar. También imprime el mensaje recibido en pantalla.
* void enviar(): El cliente escribe un mensaje, su contenido es guardado en el buffer del cliente, y es enviado al servidor.
* char enviarInforme(): Envía informes sobre la navegación del cliente a través del menú sin que este lo sepa.
* void cerrarSocket(): Sirve para cerrar el socket del cliente.

Aparte de la clase Cliente, también se declaran funciones que luego serán definidas en el Cliente.cpp. Dichas funciones son:

Funciones para que el cliente se conecte al servidor:

* void conectarseAlServidor(Cliente cliente): Como su nombre lo indica, esta función sirve para conectarse al servidor. En caso de que no pueda conectarse al servidor, le permite al cliente elegir si intentar conectarse de vuelta o cerrar el programa.
* boolean login(Cliente cliente): Esta función sirve para cuando el cliente se conecte al servidor se solicite un usuario y contraseña para mandárselo al servidor y, en base a la respuesta del servidor, dejar pasar al cliente, solicitarle que ingrese otro usuario y contraseña o desconectarlo del servidor.

Funciones para navegar a través del menú, los submenús y solicitar tareas al servidor:

* void menu(Cliente cliente): Es el menú principal por el que navega el cliente al pasar por el login. De acá el cliente puede llamar a las funciones que se detallan a continuación.
* void submenu\_TipoDeMedicamento(Cliente cliente): Es el submenú por el que navega el cliente al elegir la opción “Tipo de Medicamento” en el menú principal. Acá el cliente puede elegir distintas opciones que involucran el manejo de los tipos de medicamentos.
* void submenu\_Medicamento(Cliente cliente): Es el submenú por el que navega el cliente al elegir la opción “Medicamento” en el menú principal. Acá el cliente puede elegir distintas opciones que involucran el manejo de los medicamentos.
* void verActividad(Cliente cliente): Permite ver al cliente su registro de actividad.

FUNCIONAMIENTO DEL CLIENTE.CPP

A diferencia del Servidor.cpp, el archivo Cliente.cpp no necesita ninguna variable global.

Funciones a detalle:

void conectarseAlServidor(Cliente cliente):

Se entra while del que no se sale hasta que el cliente logre conectarse con el servidor.

Dentro del while, el programa del cliente intenta conectarse al servidor. Si el cliente logra conectarse sale del while inmediatamente. Si el cliente no logra conectarse, puede intentar conectarse al servidor todas las veces que quiera hasta que lo logre o puede elegir cerrar el programa.

boolean login(Cliente cliente):

Una vez establecida la conexión con el servidor, el cliente ingresa un usuario y contraseña para mandárselo al servidor. Después, el cliente recibe una respuesta del servidor. Si la respuesta es “Acceso concedido”, el cliente pasa por el login, si la respuesta es “Usuario o contraseña incorrecto” el cliente tiene que volver a ingresar un usuario y contraseña, y si la respuesta es “"Se superó la cantidad máxima de intentos de ingreso” el cliente es desconectado del servidor.

void menu(Cliente cliente):

Es el menú principal por el que navega el cliente al pasar por el login. Se entra en un while dentro del cual el el cliente elige una de las siguientes cuatro opciones:

1. Tipo de Medicamento: Si el cliente elige esta opción se llama a la función void submenu\_TipoDeMedicamento(Cliente cliente)
2. Medicamento: Si el cliente elige esta opción se llama a la función void submenu\_Medicamento(Cliente cliente)
3. Ver registro de actividades: Si el cliente elige esta opción se llama a la función void verActividad(Cliente cliente)
4. Cerrar sesión: Si el cliente elige esta opción se cierra la conexión con el servidor y también se cierra el programa

El switch case dentro del while sirve para realizar específicamente la tarea que pidió el cliente. Si el cliente no ingresa una opción válida, se le informa del error pidiéndole que ingrese una opción válida.

void submenu\_TipoDeMedicamento(Cliente cliente):

Es el submenú por el que navega el cliente al elegir la opción “Tipo de Medicamento” en el menú principal. Se entra en un while dentro del cual el cliente elige una de las siguientes cuatro opciones:

* Crear: Sirve para crear un nuevo tipo de medicamento. El cliente simplemente ingresa el el tipo a crear y este es enviado al servidor para crearlo y guardarlo en un archivo binario.
* Administrar: Se realiza una búsqueda con filtros y el cliente puede elegir si modificar o no un tipo de medicamento. Primero el cliente escribe el nombre y el tipo por el cual filtrar la búsqueda. Después se entra en un while que sirve para que el cliente se quede en espera de información todo el tiempo que haga falta. Dentro de este while, se recibe la información proveniente del servidor sobre los tipos de medicamentos que cumplen los filtros y se la imprime en pantalla.

Si el servidor envía el mensaje “No se encontró ningún resultado para los criterios seleccionados” no se muestra ninguna información porque ningún tipo de medicamento paso el filtro y se vuelve al submenú de los tipos de medicamentos.

Si el servidor envía el mensaje “” (vacío) significa que ya no espere por más información y que si se encontraron medicamentos que pasaron el filtro. A continuación, se le pide al cliente que ingrese un ID. Este pasa por un chequeo para comprobar que sea un int y no un carácter y si pasa el chequeo este es enviado al servidor para comprobar que el ID sea una opción válida. El servidor envía el mensaje “No existe el ID ingresado” si no es opción válida, por lo que el cliente tiene que volver a ingresar un ID y se vuelve a hacer el chequeo para volver a mandárselo al servidor.

Una vez que se obtenga un ID válido, se le pide al cliente que ingrese el nuevo estado del atributo activo (s para true, n para false y @ para que no se modifique). El atributo ID y el atributo tipo no son modificados.

Por último, el cliente indica si quiere guardar los cambios (s para si guardar y n para no guardar). Lo escrito por el cliente pasa por un chequeo para comprobar que se haya escrito una ‘s’ o una ‘n’, y si no pasa el chequeo, el cliente escribe de vuelta.

Si el cliente escribió ‘s’, recibe la información del tipo de medicamento modificado para mostrar los cambios y se vuelve al submenú. Si el cliente escribió ‘n’, simplemente se vuelve al submenú.

* Ver todo: Sirve para ver toda la información de los tipos de medicamentos. El cliente le informa al servidor de dicha consulta y se pone en modo de escucha. A medida que el cliente recibe la información, esta se imprime en pantalla para mostrarla al usuario. Cuando el cliente recibe “” (vacío) de parte del servidor significa que ya no espere por más información y el cliente deja de escuchar.
* Volver: Simplemente se vuelve al menú principal.

El switch case dentro del while sirve para realizar específicamente la tarea que pidió el cliente. Si el cliente no ingresa una opción válida, se le informa del error pidiéndole que ingrese una opción válida.

void submenu\_Medicamento(Cliente cliente):

Es el submenú por el que navega el cliente al elegir la opción “Medicamento” en el menú principal. Se entra en un while dentro del cual el cliente elige una de las siguientes cuatro opciones:

* Crear: Sirve para crear un nuevo medicamento. El cliente ingresa un código y se recibe como respuesta del servidor “\n” (salto de línea) significa que el código es válido. Si no recibe dicho mensaje tiene que volver a escribir un código.

Después ingresa el nombre y el tipo del nuevo medicamento. Si el cliente recibe “” (vacío), el tipo es válido; si recibe otro mensaje, el tipo no es válido y tiene que escribir otro tipo. El servidor realiza el chequeo de la dupla nombre-tipo. Si el cliente recibe el mensaje “\n” (salto de línea) por parte del servidor, la dupla es válida; si no lo recibe, tiene que volver a escribir el nombre y el tipo.

Por último, el cliente le envía la droga del medicamento al servidor. El servidor guarda el medicamento en un archivo binario y le muestra al cliente como quedo el medicamento creado.

* Administrar: Se realiza una búsqueda con filtros y el cliente puede elegir si modificar o eliminar algún medicamento. Primero el cliente escribe el nombre y el tipo por el cual filtrar la búsqueda. Después se entra en un while que sirve para que el cliente se quede en espera de información todo el tiempo que haga falta. Dentro de este while, se recibe la información proveniente del servidor sobre los medicamentos que cumplen los filtros y se la imprime en pantalla.

Si el servidor envía el mensaje “No se encontró ningún resultado para los criterios seleccionados” no se muestra ninguna información porque ningún medicamento paso el filtro y se vuelve al submenú de los medicamentos.

Si el servidor envía el mensaje “” (vacío) significa que ya no espere por más información y que si se encontraron medicamentos que pasaron el filtro. A continuación, se le pide al cliente que ingrese un ID. Este pasa por un chequeo para comprobar que sea un int y no un carácter y si pasa el chequeo este es enviado al servidor para comprobar que el ID sea una opción válida. El servidor envía el mensaje “No existe el ID ingresado” si no es opción válida, por lo que el cliente tiene que volver a ingresar un ID y se vuelve a hacer el chequeo para volver a mandárselo al servidor.

Una vez que se obtenga un ID válido, se le pide al cliente que ingrese que elija si quiere modificar los datos del medicamento con dicho ID o eliminarlo. También se hace un chequeo para asegurarse de que el cliente elija una opción válida.

Si el cliente elige modificar, el primer atributo que se le pide que introduzca es el nuevo código. Este es enviado al servidor para verificar que sea válido y que no este repetido. Si el código no es válido o es repetido, el servidor le manda al cliente un mensaje sobre el error y el cliente tiene que introducir otro código. Si el código es válido y único, el servidor le envía como mensaje al cliente un salto de línea (\n) para que este introduzca los demás atributos.

A continuación, el cliente escribe y envía el nombre y el tipo al servidor. El servidor realiza el chequeo del tipo. Si este no es válido, el cliente recibe un mensaje del servidor indicándolo y tiene que enviar otro tipo al servidor. Si el tipo es válido, el cliente recibe el mensaje \n (salto de línea) de parte del servidor. Después el servidor chequea que la dupla nombre-tipo sea válida. Si no lo es, el cliente recibe un mensaje del servidor pidiéndole que ingrese otra vez los atributos. Si la dupla es válida, el cliente recibe el mensaje “” (vacío) y procede a escribir el último atributo.

El último atributo que define el cliente es la droga. El cliente la escribe y la envía al servidor. A este no se le realiza ningún chequeo.

Por último, el cliente indica si quiere guardar los cambios (s para si guardar y n para no guardar). Lo escrito por el cliente pasa por un chequeo para comprobar que se haya escrito una ‘s’ o una ‘n’, y si no pasa el chequeo, el cliente escribe de vuelta.

Si el cliente escribió s, recibe la información del medicamento modificado para mostrar los cambios y se vuelve al submenú. Si el cliente escribió n, simplemente se vuelve al submenú.

Si en alguno de los atributos el cliente escribió @, este atributo no será modificado, independientemente de los cambios que se hagan en los demás atributos.

Si el cliente quiere eliminar el medicamento, se le pregunta si quiere guardar los cambios (s para si guardar y n para no guardar). Lo escrito por el cliente pasa por un chequeo para comprobar que se haya escrito una ‘s’ o una ‘n’, y si no pasa el chequeo, el cliente escribe de vuelta.

Si el cliente escribió ‘s’ o ‘n’, se vuelve al submenú ya que no hay cambios para mostrar.

* Ver todo: Sirve para ver toda la información de los medicamentos. El cliente le informa al servidor de dicha consulta y se pone en modo de escucha. A medida que el cliente recibe la información, esta se imprime en pantalla para mostrarla al usuario. Cuando el cliente recibe “” (vacío) de parte del servidor significa que ya no espere por más información y el cliente deja de escuchar.
* Volver: Simplemente se vuelve al menú principal.

El switch case dentro del while sirve para realizar específicamente la tarea que pidió el cliente. Si el cliente no ingresa una opción válida, se le informa del error pidiéndole que ingrese una opción válida.

void verActividad(Cliente cliente):

Permite ver al cliente su registro de actividad. Para esto el cliente le informa al servidor de que quiere ver su actividad y se fija si existe el archivo Mi\_actividad.log. Si existe lo borra y lo vuelve a crear para que, de esta forma al recibir toda la información de la actividad del usuario, no se repitan datos. Si este no existe simplemente lo crea. Después el cliente se pone en modo de escucha. El servidor abre el registro de actividades del cliente y le informa de su contenido al cliente. El cliente a medida que recibe la información, la muestra en pantalla y la escribe en el archivo Mi\_actividad.log.

LECTOR BINARIO A DETALLE

ARCHIVOS UTILIZADOS EN EL LECTOR BINARIO

Lector binario.cbp: El proyecto en sí. Relaciona los archivos.

main.cpp: El único archivo programable que utiliza el proyecto.

bin: Dentro de dicha carpeta se encuentra otra carpeta llamada Debug y dentro de ella se encuentra el ejecutable del proyecto.

FUNCIONAMIENTO DEL MAIN.CPP

Dentro de dicho archivo hay dos estructuras, tipomedicamento y medicamento. Ambas serán útiles para la lectura y escritura de los archivos binarios. También hay 6 funciones que se detallan a continuación:

void crearArchivosVacios():

Sirve para crear los archivos binarios Datos\_Medicamentos.dat y Datos\_Tipos\_De\_Medicamentos.dat completamente vacíos. Para hacer esto simplemente se le indica al programa que abra los dos archivos con dichos nombres y terminaciones en modo de escritura binaria y que se cierren inmediatamente después. Si ya existían los archivos, simplemente los abre y los cierra sin modificar su contenido.

void crearMedicamento(medicamento m1):

Sirve para escribir un medicamento en el archivo binario Datos\_Medicamentos.dat. Para hacer esto se abre el archivo Datos\_Medicamentos.dat en modo escritura binaria y se escribe en él la estructura que llega como parámetro. Después, se cierra el archivo binario.

void crearTipoMedicamento(tipomedicamento t):

Sirve para escribir un tipo de medicamento en el archivo binario Datos\_Tipos\_De\_Medicamentos.dat. Para hacer esto se abre el archivo Datos\_Tipos\_De\_Medicamentos.dat en modo escritura binaria y se escribe en él la estructura que llega como parámetro. Después, se cierra el archivo binario.

void leer\_todo\_medicamentos():

Sirve para mostrar en pantalla todos los medicamentos que existen y su información.

Se crea una estructura del tipo medicamento y se abre el archivo binario Datos\_Medicamentos.dat en modo de lectura binaria. Con la ayuda de la estructura, se puede recorrer y leer el archivo, al mismo tiempo que se imprime los datos en pantalla. Cuando ya no hay nada más para leer se cierra el archivo binario.

void leer\_todo\_tipoMedicamentos():

Sirve para mostrar en pantalla todos los tipos de medicamentos que existen y su información.

Se crea una estructura del tipo tipomedicamento y se abre el archivo binario Datos\_Tipos\_De\_Medicamentos.dat en modo de lectura binaria. Con la ayuda de la estructura, se puede recorrer y leer el archivo, al mismo tiempo que se imprime los datos en pantalla. Cuando ya no hay nada más para leer se cierra el archivo binario.

void consulta\_Medicamento(int id):

Sirve para hacer una búsqueda de medicamento filtrando por id.

Se crea una estructura del tipo medicamento y se abre el archivo binario Datos\_Medicamentos.dat en modo de lectura binaria. Con la ayuda de la estructura, se puede recorrer y leer el archivo. Cuando se lea un medicamento que tenga el mismo ID que el del parámetro, se lo imprime en pantalla. Cuando ya no hay nada más para leer se cierra el archivo binario.

Solo hay una función de lectura con filtros porque el propósito de esta función era practicar la lectura binaria de cualquier archivo binario con algún filtro.

Las funciones descritas son llamadas dentro del int main() del archivo. Para llamar a alguna función en especifico hay que programarlo manualmente ya que el proyecto de lector binario no tiene un menú por el que el usuario pueda navegar y llamar funciones.