INSTITUTO TECNOLÓGICO DE TUXTLA GUTIÉRREZ

NOMBRE DEL ALUMNO:

ÁLVAREZ HERNÁNDEZ JAVIER ARTURO

RESIDENCIA PROFESIONAL

NOMBRE DEL PROYECTO:

DESARROLLO DE APLICACIONES DINÁMICAS (DLL) EN LENGUAJE VISUAL C++

ASESOR CIO:

DR. MOISES CYWIAK GARBARCEWICS

ASESOR ITTG:

MC. RAÚL MORENO RINCÓN

CARRERA:

INGENIERÍA EN ELECTRÓNICA

Miércoles, 04 de diciembre del 2013.

León, Guanajuato.

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN

El proyecto que se presenta se desarrolla en las instalaciones del Centro de Investigaciones en Óptica (CIO) y servirá para ayudar en la parte de la comunicación, al proyecto de investigación del CONACYT, de nombre “Medición de peso en materiales textiles sin contacto tipo viajero para la industria mediante sensores de rayos X”, este es uno de los proyectos más importantes en estos momentos ya que está generando tecnología propia y además está financiado por la industria privada.

Se basa en la implementación de una Biblioteca de Enlace Dinámico (DLL), el cual es un archivo de código ejecutable que actúa como una biblioteca de funciones compartida, este término es exclusivo del sistema operativo Windows. Estas bibliotecas son códigos dedicados que contienen alguna funcionalidad pre-construida, la vinculación dinámica proporciona a los procesos o aplicaciones una forma de llamar a una función que no forma parte del código ejecutable de la aplicación.

El código ejecutable de la función está en otro archivo independiente, la cual contiene una o más funciones que se compilan, vinculan y almacenan de forma independiente de los procesos que hacen el llamado y la utilizan. Los archivos DLL también facilitan el uso compartido de datos y recursos. Distintas aplicaciones pueden tener acceso simultáneamente al contenido de una única copia de un archivo DLL cargado en la memoria.

Se presenta la estructura para crear, editar y utilizar una biblioteca de enlace dinámico, las partes que la conforman y como hacer uso de recursos de la API de Windows e interactuar directamente con las DLL prestablecidas en el Sistema Operativo de Windows.

Para el uso de una DLL no es necesario conocer los detalles específicos de su funcionamiento y la construcción del mismo, sino su propia interfaz. Esto se refiera a que resultados nos puede devolver y cómo es la manera correcta de preguntar o hacer el llamado a las funcionalidades de la librería.

JUSTIFICACIÓN

La creación de DLLs es de gran importancia, estos fragmentos de código pueden ser implementados en infinidad de aplicaciones, y agregar la funcionalidad de comunicación en una DLL, beneficiaría a los desarrolladores en ahorrar trabajo al escribir el código de aplicaciones que controlen aparatos electrónicos como lo son los sensores de rayos x.

En el proyecto se desarrolla una DLL de comunicación, ya sea entre aplicaciones que interactúan en una misma red de computadoras o en una misma computadora. Esto sirve para el control, análisis y supervisión de un sistema complejo. Permitiendo que el proceso que desarrolla el sistema pueda ser dividido en subprocesos más simples, así estos subprocesos se pueden llevar acabo en diferentes computadoras, de esta manera se puede conocer y acelerar el estado de cada proceso en el sistema. A la DLL creada en el proyecto se le pueden agregar funcionalidades, ampliando así el rango de aplicaciones que pueden hacer uso de ella.

La DLL ayuda también, a los que desarrollen una aplicación que haga uso de la DLL, ya que las funciones son de fácil comprensión en la sintaxis, pero el código de dicha función escrita dentro de la DLL puede interactuar con otro lenguaje y otras funciones más complejas y difíciles de entender, como la interacción constante con el sistema operativo e inclusive hacer el llamado a otras funciones, contenidas en un archivo de código diferente la cual puede ser otra DLL. También ayuda a la reutilización de código, uso eficaz de la memoria y espacio en disco reducido. Por lo tanto, el sistema y los programas se cargan más rápido, se ejecuten más rápidamente y necesitan menos espacio de disco en el equipo.

OBJETIVOS

- Objetivo general

El objetivo del proyecto es implementar los principios básicos de la comunicación de datos, entre aplicaciones y procesos con el uso de la API de Windows, aplicándolo en bibliotecas de enlace dinámicos (DLL), con el propósito de comunicar aplicaciones que hagan uso de una misma DLL y que corren en una computadora o en diferentes computadoras conectadas en una misma red, que controlen y consulten los estados de aparatos electrónicos como sensores o lasers.

- Objetivos específicos

- Comprensión de los mecanismos para la programación de las DLL.
- Programación básica de DLLs.
- Desarrollo e implementación de programas que hagan uso de DLLs.
- Mecanismos para el intercambio de datos entre aplicaciones de un mismo ordenador.
- Conocer los protocolos de comunicación.
- Comprender como funcionan los Sockets en la API de Windows.
- Introducción a la comunicación DHCP mediante sockets UDP.
- Creación de un socket UDP. Inicialización y monitoreo del socket.
- Comunicación entre aplicaciones de distintas computadoras mediante sockets.
- Comunicación cliente-servidor en una DLL mediante un canal de comunicación UDP.

CARACTERIZACIÓN DEL ÁREA EN LA QUE SE PARTICIPO

El presente proyecto “Desarrollo de Aplicaciones Dinámicas (dll) en Lenguaje Visual c++”, se desarrolló dentro de las instalaciones del Centro de Investigaciones en Óptica, A.C. Bajo la supervisión y asesoramiento del DR. Moisés Cywiak Garbarcewics. En el municipio de León, Guanajuato, México. Específicamente en el Laboratorio de Metrología Heterodina.

-Organigrama del CIO:

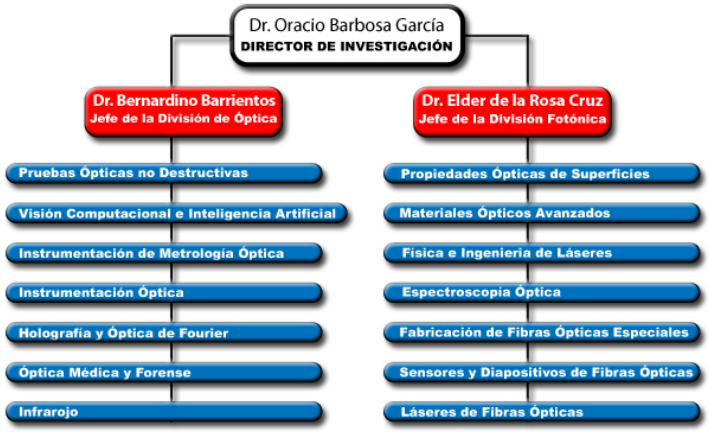

Figura 1. Líneas de Investigación.

Se desarrolló el proyecto en la División de Óptica. Dentro de la División de óptica se encuentra el área de Pruebas Ópticas no Destructivas que es el área en la que se enfoca el actual proyecto. El objetivo de esta línea de investigación es desarrollar nuevas técnicas en pruebas mecánicas e implementar nuevos métodos en el área de calibración de dispositivos y caracterización de superficies. Las pruebas no destructivas consisten en detectar cambios en toda la superficie de un objeto bajo análisis. El objeto es iluminado ya sea con luz láser o luz blanca de tal forma que se evita cualquier tipo interacción mecánica sobre su estructura y composición. Es posible medir deformación, esfuerzo, distancia, tensión, fracturamiento, densidad, temperatura, propagación de ondas mecánicas, flujo volumétrico, vibraciones, velocidad, vorticidad, concentración, rugosidad, forma, presión, etc., en la escala de nanómetros a centímetros cuando se trata de desplazamientos.

Se trabaja en la investigación y desarrollo de técnicas y métodos opto-electrónicos para estudiar deformaciones, estáticas y dinámicas, tanto en objetos con superficies especulares y no especulares y en objetos de fase.

PROBLEMAS A RESOLVER

- Establecer la estructura que tendrá el código fuente de la DLL.
- Escribir el código de programación de la DLL.
- Comprobar el funcionamiento de la DLL, creando una aplicación que vincule la DLL.
- Agregar a la DLL el código necesario para intercambiar datos entre aplicaciones.
- Comprobar el correcto funcionamiento en el intercambio de datos, agregando funcionabilidad a la aplicación que haga uso de la DLL.
- Elegir un protocolo de comunicación.
- Comprender el funcionamiento y estructura del protocolo de comunicación elegido.
- Agregar a la DLL la comunicación por red.
- Crear aplicación que compruebe la comunicación de datos por red establecida por la DLL.

-Cronograma de actividades:

Actividad	Semana															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Reconocimiento y acondicionamiento del área de trabajo	x	x	X													
Investigación de las DLLs	x	x	X	x	x	x	x									
Creación de DLLs con aplicación para la comunicación				x	x	x	x	x								
Crear aplicación de prueba para la DLL						x	x	x	x							
Modificaciones para la aplicación y DLL										x	x	x	x	x		
Pruebas y correcciones finales de la aplicación y Dll													x	x	x	x
Documentación final del proyecto								x	x	x	x	x	x	x	x	x

Descripción detallada de las actividades:

- Reconocimiento y acondicionamiento del área de trabajo:

En esta primera etapa se conocerá y acondicionara al equipo y herramientas necesarias que se utilizaran en el desarrollo del proyecto.

- Investigación de las DLLs:

Se pretende conocer el funcionamiento de las librerías de enlace dinámico, así también como desarrollar y crear una DLL para su uso correcto en una aplicación.

- Creación de DLLs con aplicación para la comunicación:

Se escribirá el código fuente para la creación de librería de enlace dinámico (DLL) con fines de comunicación estable entre aplicaciones.

- Crear de una aplicación de prueba para la DLL:

Etapa en la que se escribirá el código de una aplicación que utilice la DLL, para pruebas en su correcto funcionamiento.

- Modificaciones para la aplicación y DLL:

Se pretende corregir los errores e incluir modificaciones o mejoras necesarias en las librerías de enlace dinámico, así como también en las aplicaciones que pretenden utilizar las mismas.

- Pruebas y correcciones finales de la aplicación y DLL:

Después de terminar las aplicaciones y DLL, se realizaran pruebas y corregirán posibles errores que puedan surgir de improviso para su entrega final.

- Documentación final del proyecto:

Se documentara un reporte con los resultados obtenidos en la investigación y en el desarrollo del proyecto, se expondrán las características del funcionamiento, tablas e imágenes del desarrollo del proyecto.

ALCANCES Y LIMITACIONES

- Alcances

El programar DLLs con funcionalidades variadas en comunicación como lo es memoria compartida y sockets de Windows, permite enlazarse con aplicaciones que estén definidas para un aparato electrónico, permitiendo así que una aplicación pueda aparte de controlar un aparato electrónico, también tenga la funcionalidad de comunicar los datos a otro ordenador que analice los datos y de una respuesta.

Este proyecto permitirá la comunicación de datos, y con estos datos controlar aparatos electrónicos de forma remota. El código fuente de la DLL escrita en el proyecto es abierto, por lo que se le pueden ampliar las funcionalidades, haciendo que el proyecto pueda tener mayores alcances. También proveerá de una conexión en el que el usuario intervendrá en lo más mínimo posible, con los recursos mínimos.

- Limitaciones

Debido a que los aparatos que se van a usar en el proyecto del CONACYT, al cual está dirigido en apoyar este proyecto, aparatos como el emisor de rayos X (X123 de Amptek) y el receptor de rayos X (MiniX de Amptek), no estuvieron en el tiempo debido para desarrollar el proyecto se tuvo que avanzar sin el uso de estos aparatos, por lo que en la DLL solo se puede implementar en la comunicación del proyecto del CONACYT.

El proyecto que se realiza esta propuesto para establecer comunicación entre procesos, y tienen la limitación en salud, ya que está destinado para comunicar aparatos nocivos para la salud como lo es un sensor de rayos X, el cual deberá ser controlado por un ordenador pero a su vez hay que procurar una distancia considerable para evitar daños a la salud.

El concepto cliente-servidor tiene un inconveniente en el establecimiento de la comunicación, para entender este inconveniente se explica en un ejemplo, al navegar en el internet y dirigirse a una página, se escriben nombres inteligibles como www.google.com.mx y no direcciones IP, este concepto se le conoce como sistema de nombres de dominio (DNS), el cuál no está contenido en el proyecto, por lo que para establecer la comunicación se necesita la intervención del usuario.

CAPÍTULO 2

FUNDAMENTO TEÓRICO

- Borland C++ Builder

C++ Builder es un producto de desarrollo rápido de aplicaciones (RAD), con este software es posible escribir programas para Windows de una manera rápida y sencilla. Se pueden crear aplicaciones de consola o programas de GUI (Interfaz Gráfica de Usuario). Se puede crear la interfaz de usuario de un programa, esta interfaz consiste en los menús, cuadros de diálogo, ventana principal, entre otros; también se pueden colocar controles OCX Y ActiveX en formularios para crear programas especializados, como los navegadores Web. Todo esto brindado gracias al poder del lenguaje C++ [3].

- DLL´s

Windows proporciona un poderoso mecanismo de apoyo la reutilización de aplicaciones en el nivel binario a través de las Bibliotecas de Enlace Dinámico (DLLs). Una DLL normalmente representa una colección de funciones utilizadas en común, las capacidades y recursos empaquetados en un módulo que pueden ser enlazable aprovechado por otros programas. Este módulo vinculable típicamente tiene la extensión .dll [2].

Dicho en términos sencillos, una DLL consta de uno o más fragmentos de código almacenados en un archivo con extensión .dll. El código de las DLLs se puede llamar desde programas ejecutables, sin que la DLL misma sea un programa independiente. Se les considera como una especia de archivo auxiliar de un programa principal.

Hay muchos usos prácticos para una DLL. Por ejemplo, suponga que tiene un conjunto común de modelos matemáticos y cálculos que necesitan ser utilizados dentro de varias aplicaciones a desarrollar y mantener. En lugar de repetitivamente copiar o vincular el código para estos modelos y cálculos en cada programa, usted puede crear y compilar una DLL única que cada programa puede cargar y usar como si el código ya se hubiese incrustado dentro de cada programa. Esto tiene varias ventajas que se identifican como sigue:

- Un archivo DLL minimiza el código re-escritura, ya que puede ser compartida y utilizado por múltiples aplicaciones.

- Las aplicaciones serán más pequeñas en tamaño porque el código en la DLL está aislado de las aplicaciones que utilizan el código.
- Las aplicaciones se pueden actualizar sin necesidad de recompilar simplemente a través de actualizaciones de DLL.

Al crear una DLL se notara que se crea más de un archivo dependiendo de tipo de carga y los recursos usados en la misma, tenido así cuatro o más archivos [2]:

- Uno o más archivos de código fuente.
- uno o más archivos de encabezado de extensión .h, estos contendrán las declaraciones a las clases y funciones que contendrá la DLL, esta se deberá escribir e incluir en el archivo de código fuente.
- El archivo de Biblioteca de Importación con extensión .lib, este archivo permite al vinculador resolver llamadas en la aplicación con direcciones en la DLL, esta se crea automáticamente al compilar la DLL.
- El archivo DLL de extensión .dll.

Estos archivos conforman la DLL, y deberán incluirse en cualquier aplicación que haga uso de la misma, el de encabezado (.h), la biblioteca de importación (.lib) y la biblioteca de enlace dinámico (.dll).

La función principal que necesita ser incluida en el archivo de código de una DLL, es la función de entrada a la API de Windows, en el lenguaje de Microsoft la función es WinMain(). Pero para el lenguaje de Builder la función es DLLEntryPoint(), cambia por asuntos de compatibilidad en sus recursos, aunque también permite la compatibilidad con la función DLLMain(), pero es recomendable usar la establecida por Builder:

Después de la función entrada Main de la DLL, se puede agregar la funcionabilidad que se desea compartir por medio de la DLL.

- Tipos de DLLs

Existen dos tipos de DLL [1]: de código y de recursos, esta disposición no es excluyente, en una misma DLL es posible tener tanto código como recursos sin problema alguno, sin embargo es conveniente almacenar el código en una DLL y sus recursos en otra. Con la

llegada e implementación de .NET Framework[1] y/o Microsoft Foundation Class (MFC)[2] los tipos de DLLs se han extendido o más bien los tipos definidos con anterioridad han aumentado sus roles así como sus nombres y la división clara entre un tipo y otro se ha perdido en cierta manera. En general como ejemplo una DLL .NET ya no se llama así, sino que se llama ensamblado y puede cumplir uno o varios roles anteriores.

- DLL de código: Es un conjunto de funciones y funcionalidad más o menos destinada a ser utilizada por otros programas. El propio Windows está construido en base a esta característica, las DLLs kernel32.dll, gdi32.dll y user32.dll forman el núcleo del sistema operativo, al menos del subsistema de Win32/Win64.

El código contenido en una DLL puede tener una de dos formas primarias. La primera es una función independiente que puede llamar desde la aplicación principal, tanto LoadCursos() como DrawText() son funciones de la API de Windows, estas funciones están contenidas en alguna parte, en este caso las dos funciones están contenidas en la DLL de Windows llamada user32.dll.

La segunda forma de código contenida en una DLL consta de clases de c++. En lugar de acceder a funciones independientes, se crea un ejemplo de una clase y se hace la llamada a funciones miembros de esa clase, todas ellas contenidas en la DLL.

- DLL de recursos. Como su nombre indica, sólo contienen recursos y nada de código (o el menos posible). Un recurso puede ser un bitmap, una cadena de texto o la definición de un cuadro de diálogo. Este tipo de DLL es muy útil para construir aplicaciones internacionales en las que las cadenas en cada idioma están almacenadas en una DLL diferente, de modo que cuando se carga el programa podemos indicar qué DLL de recursos cargar, teniendo así nuestra aplicación, de forma casi automática, disponible en todos los idiomas que queramos. El propio Windows lo hace así con las versiones MUI[3] (Multilingual User Interface), que incluyen todos los textos y otros elementos en DLL normales pero renombradas con la extensión MUI. Usando este mismo paradigma, podemos tener soporte de temas y resulta mucho más fácil cambiar o actualizar los recursos de una aplicación.

[1] Hace un énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones.
[2] Es un conjunto de clases interconectadas por múltiples relaciones de herencia, que proveen un acceso más sencillo a las API de Windows.
[3] Interfaz de Usuario Multilenguaje, permite a un usuario seleccionar el lenguaje en el que quiere que se muestre su escritorio.

- Carga de DLLs

Existen dos formas por las que se puede cargar una DLL, estático y dinámico, con sus ventajas y desventajas.

El cargado estático se refiera a que la DLL se carga en el momento en que la aplicación o programa se ejecuta y hace el llamado a la DLL, esta tendrá funciones que exporta y las descripciones de estas funciones estarán contenidas en un Archivo de Biblioteca de Importación, este archivo especial deberá llamarse igual que el nombre de la DLL pero con la extensión .lib.

Para hacer el cargado estático de la DLL, se vincula el archivo de extensión .lib con el proyecto que hace la llamada a esta DLL, haciendo el cargado estático de la DLL permitiendo hacer el llamado a las funciones contenidas en la DLL como si fuera una función del programa principal [1].

El cargado dinámico se basa en que la DLL se carga solo cuando se necesita y se descarga cuando ya no se desea trabajar con ella, de esta manera la DLL solo se carga cuando se necesita haciendo el uso eficaz y ahorrando espacio en la memoria, también ayuda a que la aplicación se cargue más rápido debido a que no se carga todo el código del programa. La desventaja principal de este tipo de cargado es la dificultad que tiene a la hora de programarlo ya que se tiene que hacer el uso de funciones específicas de la API de Windows, para cargar la DLL se usa la función LoadLibrary(), y después de su uso para la descarga se usa FreeLibray() y para hacer el uso de las funciones contenidas en la DLL hace el uso de la función GetProcAddress(), esta última función devuelve un apuntador a la función que se hace referencia [1].

- Importar y exportar funciones

Las funciones contenidas en una DLL se pueden dividir en dos categorías, las funciones que son llamadas desde la DLL y las funciones llamadas desde fuera de la DLL. La primera, llamadas desde la DLL no requieren de algún tipo especial de llamado [2].

Para hacer un llamado desde fuera de la DLL se necesita un tipo de conversión a funciones públicas, de lo contrario se podrían considerar funciones privadas de la DLL, y así una aplicación que haga el llamado a una de las funciones dentro de esta DLL no podrá verlas ni usarlas, aunque este consiente de las funciones que contiene la DLL. Para que puedan ser de algún modo públicas se necesita exportar cada una de las funciones contenidas en

la DLL, considerándolas públicas, y así podrán ser llamadas desde otras DLLs o desde aplicaciones fuera de la DLL.

Para exportar se usa la palabra clave __export al crear la función en la DLL, esta deberá situarse entre el valor devuelto y el nombre de la función. Pero para que pueda ser usada por una aplicación externa se deberá importar en la DLL, para esto se usa la palabra clave __import, esta además deberá ser usada con la instrucción extern “C” que traslada la función de modo que pueda ser referenciado, de esta manera se le indica al compilador que se puede hacer referencia desde el archivo de biblioteca de importación (.lib).

La clave para que una DLL funcione está en realizar correctamente el código para poder exportar e importar las funciones.

Existe otra palabra clave __declspec() que hace el trabajo de importar funciones usando __declspec(dllimport) y para exportar funciones se usa __declspec(dllexport), así no se necesita tener demasiado cuidado en colocar en el lugar correcto las palabras clave __import() y __export(), brindando mayor flexibilidad en la escritura de las funciones.

- Encabezado de Bibliotecas de Enlace Dinámico

Un archivo de encabezado de extensión .h, es el que utilizamos para definir parámetros que se utilizan en el código o para definir todas las funciones que vamos a utilizar en todos los demás archivos, ya que las funciones deben declararse antes de usarse, esto es parte del preprocesamiento [4].

Después de saber que cada función y/o clase contenida en la DLL se debe exportar e importar, y que para esto se necesite de dos archivos de encabezados con extensión .h vinculados e incluidos a la DLL, una donde se declaren las funciones a exportar con la palabra clave __export() y otro donde se declaren las funciones a importar con __import(). Se destaca que no es necesario si se emplean los macros para usar solo un encabezado donde estén declaras las dos peticiones de exportar e importar, los macros permite realizar cálculos durante la compilación, en lugar de realizarlos durante la ejecución ahorrándose código.

- Memoria compartida

Es posible hacer que dos aplicaciones o procesos distintos puedan compartir una zona de memoria en común y, de esta manera, compartir o comunicarse datos. La memoria

compartida es aquel tipo de memoria que puede ser accedida por múltiples programas, ya sea para comunicarse entre ellos o para evitar tener copias de datos. Uno de los procesos asignara un área en la memoria a la que el otro pueda acceder, se le considera un método para conservar espacio en la memoria, haciéndolo un método eficaz para intercambiar datos entre aplicaciones [1].

Esta es una de las características que se ofrece a través de la construcción de una DLL el soporte para la memoria compartida entre múltiples aplicaciones.

Cuando un archivo DLL se carga mediante una aplicación dentro de Windows, está se encuentra diseñada para ser protegida dentro del espacio de memoria asignada para la aplicación. Si dos diferentes aplicaciones cargan y hacen uso de la misma DLL no se afectarán entre sí o mejor dicho no tendrán ningún impacto entre ellas, es decir, una instancia independiente de la DLL existirían para ambas aplicaciones. Esto es parte de la protección que ofrece el sistema operativo Windows.

Más sin embargo, se puede crear una DLL que pueda compartir un mismo segmento de la memoria entre múltiples aplicaciones. Esto puede ser muy útil si se quiere tener una aplicación que requiere datos de otras aplicaciones durante su ejecución si no se desea utilizar COM[4]. Esta capacidad es una forma de Comunicaciones entre procesos (IPC)[5].

Para declarar e inicializar un segmento de memoria compartida en Builder, se necesita de dos archivos el primero deberá estar definido en otro archivo fuente de extensión .cpp y que contendrá el código real del segmento. Para incluir opciones de línea de comando en el código del programa se usa #pragma option -zR.

Después de esta línea de código se declaran los tipos de datos a compartir por la DLL, de esta forma se le informa al compilador que el código que sigue será incluido en el segmento de memoria compartida.

El segundo archivo es un archivo de definición de módulos de extensión .def que proporcionan al vinculador información sobre exportaciones, atributos y otros datos del programa que se va a vincular, es un fichero de texto ASCII que proporciona al enlazador información relativa al contenido y requerimientos de sistema de una aplicación.

El fichero de definición de una DLL contiene información que debe ser conocida por el enlazador para construirla. Esta información contiene los siguientes datos: tipo de

[4] Component Object Model (COM) es una plataforma de Microsoft que permite la comunicación entre procesos y la creación dinámica de objetos.
[5] Siglas en ingles de Inter-Process Communication (IPC). Los procesos pueden comunicarse entre sí a través de memoria compartida.

aplicación, lista de funciones importables y exportables, atributos de los segmentos de código y de datos, tamaño de la pila y datos complementarios.

La información del fichero está clasificada en secciones que son las siguientes: CODE, DATA, DESCRIPTION, EXETYPE, EXPORTS, HEAPSIZE, IMPORTS, LIBRARY, NAME, SECTIONS, SEGMENTS, STACKSIZE, STUB, SUBSYSTEM.

Este archivo tendrá en la sección LIBRARY el mismo nombre que el archivo fuente principal y la sección que se necesita inicializar es la sección SEGMENTS para declarar el segmento de memoria compartida.

- Modelo OSI (Open System Interconecction)

El termino interconexión de sistemas abiertos (OSI), es el nombre asignado a un conjunto de reglas y normas establecidas en 1980, creado por la Organización Internacional de la Estandarización (ISO), siendo el objetivo principal de estas normas la de contar con un lineamiento estructural para intercambiar información entre computadoras, terminales y redes.

Es una normativa estándar muy útil debido a los aparatos tecnológicos creados por diferentes fabricantes y compañías dentro del mundo de las comunicaciones, y estas al estar en continua actualización y expansión, se optó por estandarizar un método que permitiera la comprensión entre todos de algún modo, incluso cuando las tecnologías no coincidieran. Permitiendo de este modo sin importar la localización geográfica o el lenguaje utilizado la comunicación.

Este modelo está dividido en siete capas o niveles, expuestos desde su nivel más alto hasta el más bajo [5]:

Nivel	Nombre	Categoría
Capa 7	Nivel de Aplicación	Aplicación
Capa 6	Nivel de Presentación	
Capa 5	Nivel de Sesión	
Capa 4	Nivel de Transporte	
Capa 3	Nivel de Red	Transporte de Datos
Capa 2	Nivel de Enlace de Datos	
Capa 1	Nivel de Físico	

Estos 7 niveles se pueden subdividir en dos categorías, las capas superiores y las capas inferiores. Las 4 capas superiores trabajan con problemas particulares a las aplicaciones, y las 3 capas inferiores se encargan de los problemas pertinentes al transporte de los datos.

- Capa Física:

En esta capa se definen las especificaciones eléctricas, mecánicas, de procedimiento y funcionales para activar, mantener y desactivar el enlace físico entre sistemas finales. En ella se especifican las características tales como niveles de voltaje, temporización de cambios de voltaje, velocidad de datos físicos, distancias de transmisión máximas, conectores físicos, entre otros atributos.

- Capa de Enlace de Datos:

La capa de enlace de datos proporciona el tránsito de datos confiable a través de un enlace físico. Esta capa se ocupa del direccionamiento físico, la topología de red, el acceso a la red, la detección y corrección de errores, entrega ordenada de tramas y control de flujo. También proporciona un medio para activar, mantener y desactivar el enlace de datos y facilita el flujo ordenado de datos entre nodos.

- Capa de Red:

Se encarga de identificar el enrutamiento existente entre una o más redes. Las unidades de información se denominan paquetes, y se pueden clasificar en protocolos enrutables y protocolos de enrutamiento.

Enrutables: viajan con los paquetes (IP, IPX, APPLETALK)

Enrutamiento: permiten seleccionar las rutas (RIP, IGRP, EIGRP, OSPF, BGP)

El objetivo de la capa de red es proporcionar conectividad y selección de una ruta entre dos sistemas para hacer que los datos lleguen desde el origen al destino, aun cuando ambos no estén conectados directamente o se encuentren ubicados en redes geográficamente distintas.

- Capa de Transporte:

Esta capa controla la integridad del mensaje, en todo su contenido, y en eso se incluye la ruta, la segmentación y la recuperación de errores para el mensaje. Así el objetivo final de la capa de transporte es proporcionar un servicio eficiente y confiable a los usuarios, que normalmente son procesos de la capa de aplicación. Para lograr este objetivo, la capa de transporte utiliza los servicios proporcionados por la capa de red.

Las responsabilidades principales que debe cumplir son:

- seguimiento de la comunicación individual entre aplicaciones en los hosts origen y destino: En cada host pueden haber diversas aplicaciones que se comunican a otras aplicaciones de host remotos, es responsabilidad de la capa de transporte mantener el flujo de datos entre estas aplicaciones.
- segmentación de datos y gestión de cada porción: Debido a que cada aplicación genera un flujo de datos para enviar a una aplicación remota, estos datos deben prepararse para ser enviados por los medios en partes entendibles establecidos en los protocolos de comunicación.
- reconstrucción de segmentos en flujos de datos de aplicación: En el host de recepción, cada sección de datos puede ser direccionada a la aplicación adecuada.
- identificación de las diferentes aplicaciones: Para poder transferir los flujos de datos a las aplicaciones adecuadas, la capa de Transporte debe identificarla aplicación de destino.

Los dos protocolos más comunes de la capa de Transporte del conjunto de protocolos TCP/IP son el Protocolo de control de transmisión (TCP) y el Protocolo de datagramas de usuario (UDP). Ambos protocolos son responsables de la comunicación de las múltiples aplicaciones.

En el servicio orientado a la conexión (TCP) consta de tres partes: establecimiento, transferencia de datos, y liberación; esto quiere decir que si un proceso quiere comunicarse con otro debe primero establecerse una conexión y debe cerrarse al finalizarse. Llevan al uso adicional de recursos para agregar funciones, las mismas que las hacen lentas. Las funciones adicionales son de entrega confiable y de control de flujo.

En el servicio no orientado a la conexión (UDP) se tratan los paquetes o el flujo de datos de forma individual. Es un protocolo simple, sin conexión, Cuenta con la ventaja de

proveer la entrega de datos sin utilizar muchos recursos, aunque se pueden detectar errores en los mensajes, no avisaría si un mensaje entero se pierde. Las porciones de comunicación en UDP se llaman datagramas.

- Capa de Sesión: Esta se encarga de establecer, administrar y cerrar la comunicación entre dos computadoras que se comunican, esta se encarga también de reanudar la conexión entre dos host si esta es interrumpida.

-Capa de Presentación: Esta capa se encarga de darle una presentación al flujo de datos enviado por una aplicación a otra en diferente computadora, haciendo que los datos lleguen de manera reconocible. Esta traduce entre varios formatos de datos utilizando un formato común.

-Capa de Aplicación: Esta permite acceder a las demás capas del modelo OSI y define los protocolos usados, esta se comunica de forma directa con el programa de aplicación del usuario. En esta capa aparecen diferentes protocolos y servicios como:

FTP (File Transfer Protocol), DNS (Domain Name Service - Servicio de nombres de dominio), DHCP (Dynamic Host Configuration Protocol - Protocolo de configuración dinámica de anfitrión), HTTP (HyperText Transfer Protocol) para acceso a páginas web, HTTPS (Hypertext Transfer Protocol Secure) Protocolo seguro de transferencia de hipertexto, POP (Post Office Protocol) para recuperación de correo electrónico, entre otros.

- DHCP

DHCP significa Protocolo de configuración de host dinámico. Es un protocolo que permite que un equipo conectado a una red pueda obtener su configuración (principalmente, su configuración de red) en forma dinámica (es decir, sin intervención particular). Sólo tiene que especificarle al equipo, mediante DHCP, que encuentre una dirección IP de manera independiente. El objetivo principal es simplificar la administración de la red [3].

El protocolo DHCP sirve principalmente para distribuir direcciones IP en una red, pero desde sus inicios se diseñó como un complemento del protocolo BOOTP (Protocolo Bootstrap), que se utiliza, por ejemplo, cuando se instala un equipo a través de una red (BOOTP se usa junto con un servidor TFTP donde el cliente encontrará los archivos que se

cargarán y copiarán en el disco duro). Un servidor DHCP puede devolver parámetros BOOTP o la configuración específica a un determinado host.

- API de Windows

Una interfaz de programación de aplicaciones (API) es un código fuente de un sistema informático o una biblioteca programada proporcionada con el fin de responder a las solicitudes de servicios que se realicen desde un programa [7].

El software que proporciona la funcionalidad descrita por una API se le llama implementación de la API. La API en sí es abstracto, ya que especifica un interfaz y no se involucra con los detalles en su implementación.

En términos generales, una API es un programa para un sistema operativo o medio ambiente, que consiste en un conjunto estandarizado de funciones y procedimientos. Gracias a esto los programadores pueden llamar a estas funciones y procedimientos desde sus programas para ganar funcionalidad extra, debido a que los programadores no tienen que escribir el código por ellos mismo de esta manera se ahorra tiempo.

Las funciones de la API residen en archivos DLL (como User32.dll , GDI32.dll , Shell32.dll , ...) en el directorio raíz del sistema operático de Windows. La API de Windows es el nombre dado por Microsoft para el conjunto básico de estas interfaces de programación. Está diseñado para ser usado por programas de C / C + + y es la forma más directa para interactuar con el sistema Windows para aplicaciones de software.

- Sockets de Windows

El socket de Windows (WinSockt) es la interfaz preferida para acceder a una gran variedad protocolos de red que existen en la actualidad y está disponible en diferentes formas en todas las plataformas. Winsock es una interfaz de programación para la red pero no es un protocolo. Es una biblioteca de enlaces dinámicos DLL para Windows que incluye el soporte de envío y recepción de paquete de datos. La interfaz se ha convertido finalmente en una verdadera interfaz independiente del protocolo, sobre todo con el lanzamiento de Winsock2. Este está basado en la Berkeley Software Distribution (BSD) de Unix. La funcionalidad de Windows Socket es proporcionada por wsock32.dll (WinSock versión 1.1) o ws2_32.dll (WinSock versión 2.0). Cuando compiles tu aplicación deberás incluir el encabezado winsock.h al igual que la librería wsock32.lib [6].

WinSock es la base de toda la actividad de Internet y sus diferentes aplicaciones como el correo electrónico, navegadores web, la difusión UDP, FTP, entre otros.

Cada máquina está identificada por medio de su dirección IP que debe ser única. Sin embargo, en cada máquina pueden estar ejecutándose múltiples procesos simultáneamente. Cada uno de estos procesos se asocia con un número de puerto, para poder así diferenciar los distintos paquetes que reciba la máquina a esto se le conoce como proceso de multiplexación. Un socket se identifica por la combinación de una dirección IP + número de puerto.

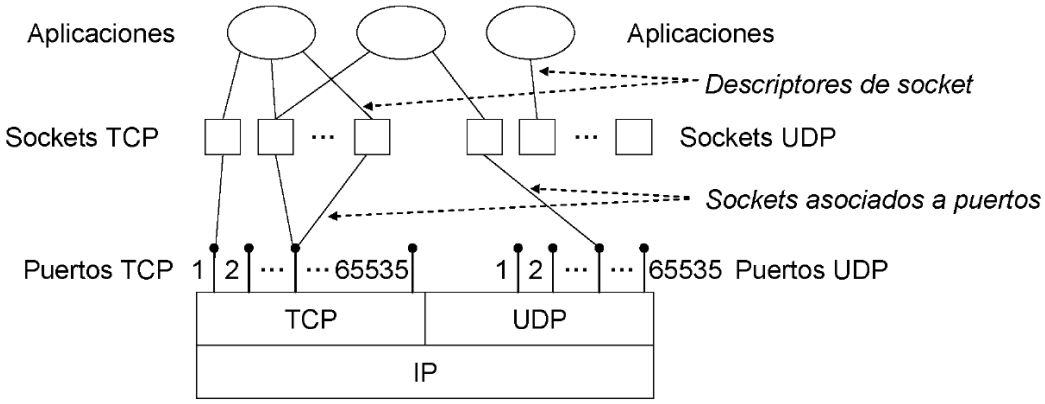

Figura 2. Sockets, protocolos y puertos

La figura anterior muestra un claro ejemplo de las relaciones que existen entre las aplicaciones, sockets, protocolos y puertos de un dispositivo. Cada socket está asociado a un puerto de protocolo TCP o UDP, una aplicación puede usar más de un socket al mismo tiempo.

- DLL del winsock:

Cada aplicación que use sockets tendrá que cargar apropiadamente la versión DLL del socket de Windows, antes de crear un socket de lo contrario marcara un error. La carga de la DLL del socket de Windows es realizado por la llamada a la función WSAStartup() [5].

Cuando la aplicación haya terminado por usar la interfaz del socket de Windows, se debe llamar a la función WSACleanup, que permite liberar los recursos asignados por el socket de Windows y cancelar cualquier socket en espera de las llamadas solicitadas, esta se define como:

int WSACleanup (void);

Por cada llamada a la función WSAStartiup, deberá haber una llamada de terminado WSACleanup.

- Dominios de comunicación y tipos de sockets

Los sockets no contienen información suficiente para describir la comunicación entre procesos. Los sockets operan dentro de los dominios de comunicación, estos definen si las dos aplicaciones que se comunican se encuentran en el mismo sistema o en sistemas diferentes de redes distintas, y cómo pueden ser direccionados. Los dominios de comunicación más usados son los siguientes:

- AF_UNIX: es el dominio que se caracteriza por la comunicación entre aplicaciones que se ejecutan dentro de un mismo sistema.
- AF_INET: este dominio se utiliza por las aplicaciones que se comunican a través de cualquier tipo de red. Dentro de este dominio se definen los siguientes sockets:
    - Socket Stream: este tipo de socket hace uso del protocolo TCP.
    - Socket Datagram: se caracteriza por hacer uso del protocolo UDP.
    - Socket Raw: este permite el acceso a un nivel más bajo, es usado para definir nuevos protocolos de comunicación.

-Orden de los Bytes

Diferentes microprocesadores representan los números de dos maneras big-endian y Little-endian, en la primera forma de representarlos primero va el byte más significativo y luego el menos significativo, o de la segunda forma que es contraria a la primera ósea al revés. Debido a este suceso en el mundo de las redes se puede provocar que una computadora interprete correcta o incorrectamente los paquetes de datos recibidos. Para resolver esto se implementó un formato común llamado Orden de los Bytes en la Red (Network Byte Order), este es un formato que ordena los datos para ser enviado por red.

- Servicios TCP y UDP

El modelo para aplicaciones más usado es el modelo cliente-servidor, este modelo puede elegir usar entre dos protocolos, el protocolo TCP y UDP. A continuación se describe los

pasos a seguir para obtener un servicio orientado al protocolo TCP y UDP, después se describirán cada una de las funciones que se mencionan.

Servicio orientado a conexión (TCP)

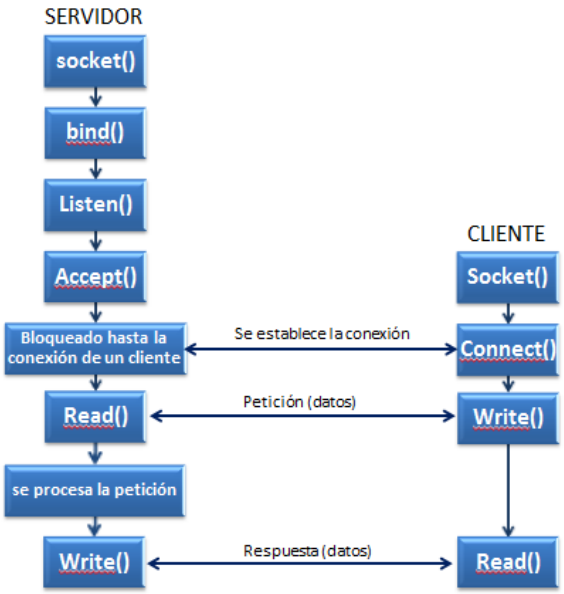

Figura 3. Protocolo TCP con llamadas al socket de Windows.

Para el servicio no orientado a conexión (UDP), se usa el siguiente esquema de funcionamiento:

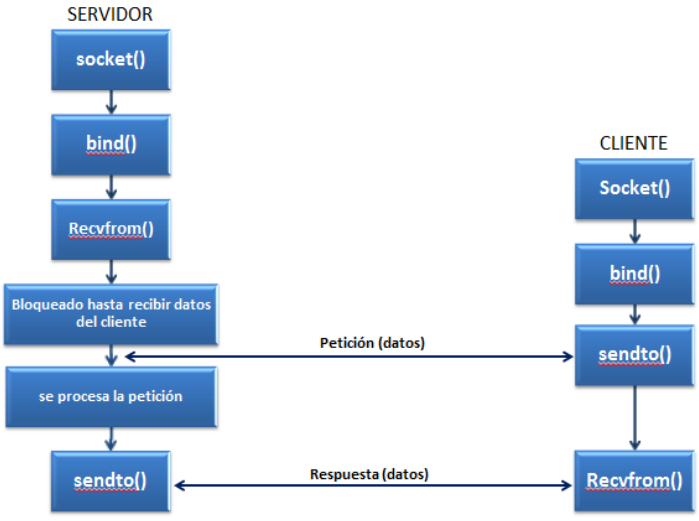

Figura 4. Protocolo UDP con llamadas al socket de Windows

-Estructura de datos

La función socket(), solamente crea un socket, pero no asigna ningún valor lo que quiere decir que no lo asocia con ninguna IP y tampoco a ningún puerto. Las estructuras de datos son usadas en la programación de sockets para almacenar información sobre direcciones, la cual contendrá información del socket [5].

```
struct sockaddr
{
        unsigned short sa_family; // familia de la dirección
        char sa_data[14]; // Dirección de protocolo de 14 bytes.
};
```

Existe otra estructura la cual nos ayuda a ser referencia a los elementos del socket:

```
struct sockaddr_in
{
        short int sin_family; // familia de la dirección
        unsigned short sin_port; // Numero de puerto.
        struct in_addr sin_addr; // Dirección IP.
        unsigned char sin_zero[8]; // Relleno.
};
```

- Funciones del socket de Windows

-Función socket:

Un socket es un identificador de un proveedor de transporte. En Windows, un socket no es lo mismo que un descriptor de archivo. A continuación se describe brevemente la creación de socket para cada uno de los protocolos disponibles. Por simplicidad, se describirá brevemente que, para crear un socket se llama a la función socket, esta devuelve el identificador del socket creado de tipo entero, si se produce un error devuelve el valor -1.

```
int socket(int domain, int type, int protocol);
```

-Función bind: Una vez creado el socket de un protocolo en particular, se debe enlazar a una dirección conocida una dirección IP y un número de puerto. La función de enlace que asocia estos parámetros es bind. Esta función se declara como:

int bind(int socket, struct sockaddr *addr, int addrlen);

-Función listen:

Esta función se usa si se quiere esperar conexiones entrantes, lo cual significa, si se quiere, que alguien pueda conectarse a nuestra máquina. Marcará un socket en modo conexión, especificado por el puerto, como la aceptación de las conexiones entrantes.

int listen(int socket, int backlog);

-Función accept: ahora está listo para aceptar conexiones de clientes. Esto se logra con la llamada a esta función es utilizada por un servidor, para aceptar una solicitud de conexión de un cliente. Cuando hay una conexión disponible, el socket creado está listo para su uso para leer los datos del proceso que solicita la conexión. La llamada acepta la primera conexión en la cola de las conexiones pendientes. La llamada crea un nuevo descriptor de socket o estructura sockaddr con las mismas propiedades que toma y lo devuelve a la persona que llama. Si no encuentra solicitudes de conexión pendientes, esta se bloquea.

int accept(int socket, struct sockaddr *addr, socklen_t *addrlen);

-Función sendto: la función se aplica a los sockets de protocolo orientado a conexión (TCP) o al protocolo no orientado a conexión (UDP). La función enviará un mensaje a través de una conexión del socket. Si el socket está en modo sin conexión, se enviará el mensaje a la dirección especificada por la estructura sockaddr. Si la toma es en modo conexión, la estructura sockaddr se ignorará.

ssize_t sendto(int socket, const void *buffer, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);

-Función recvfrom:

La función recvfrom recibe datos de un conector llamado por la estructura sockaddr del socket descrito y lo almacena en un búfer. Está función se aplica a cualquier socket de tipo TCP o UDP.

ssize_t recvfrom (int socket, void *bufffer, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);

CAPÍTULO 3

PROCEDIMIENTO Y DESCRIPCIÓN DE LAS ACTIVIDADES REALIZADAS

El proyecto se llevó acabo en un proceso ordenado, lo que nos permitió finalizar el proyecto definido en 4 meses por etapas, las etapas definidas para la creación de la DLL, son las siguientes:

- Crear DLL en Builder.
- Agregando memoria compartida a la DLL.
- Declarar funciones en la DLL.
- Funciones para el uso de la memoria compartida.
- Funciones de conexión de la DLL.

Los recursos usados en el proyecto fueron proporcionados a todo momento, hasta donde se pudo, alcanzando así un proceso casi sin interrupciones. En los materiales necesarios, no se contaba con las computadoras necesarias para establecer una red con más de 2 computadoras, atrasando el desarrollo del proyecto y creando una limitante. Con la ayuda de la supervisión por mi asesor en el CIO, el Dr. Cywiak Garbarcewicz Moises que estuvo al pendiente del proyecto en lo que llevo todo su desarrollo.

- Crear DLL en Builder

Para crear una DLL en Builder hay que seleccionar DLL Wizard del menú New Items y especificar el lenguaje que se usará y activar el casillero de la opción Multi Threaded, para tener más de un hilo en ejecución para multiples tareas dentro de la DLL. Como lo indica la siguiente figura.

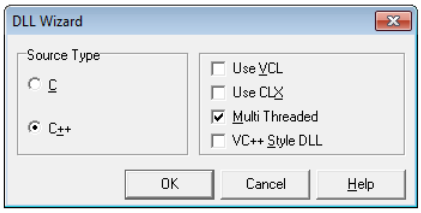

Figura 5. Crear DLL

Generando de esta manera el siguiente código:

```
#include <windows.h>
#pragma argsused
//Función de entrada de la DLL, más información Anexo 1.
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
      return 1;
}
...
```

En este punto guardamos el archivo con el nombre LibCio.cpp y el proyecto con el nombre LibCio.bpr, y partir de este código generado ya podemos escribir las funciones y funcionabilidad que tendrá la DLL.

- Agregando memoria compartida a la DLL

El código necesario que definirá el segmento de memoria compartida, tiene que estar escrito en otro archivo de extensión .cpp que estará vinculado al archivo LibCio.dll. Entonces creamos un nuevo archivo y en el menú New Items seleccionamos Unit.

Dentro de este nuevo archivo escribimos el siguiente código:

```
#pragma option –zRSHSEG   //Información en el Anexo 3
#pragma option –zTSHCLS

//A partir de esta sección se pueden declarar las variables que se desean compartir
...
```

Este archivo se guarda con el nombre MemCom.cpp y se vincula al archivo LibCio.dll en el Project Manager. El segmento SHSEG y SHCLS llamados en las líneas de código anterior necesitan de un vinculador que informe sobre los datos a enlazar en el segmento de memoria compartida al compilador. El vinculador es otro archivo de extensión .def, a continuación escrito:

```
LIBRARY LibCio

SEGMENTS    //más información en el Anexo 4
      SHSEG CLASS 'SHCLS' SHARED
```

Así se termina por declarar el segmento de memoria compartida, clases y los datos a compartir, llamamos a este archivo LibCio.def. Para que los datos puedan ser compartidos en la DLL, se deberán declarar en el archivo de código fuente de la DLL antes de la función DLLEntryPoint() como datos externos de esta manera:

```
...
extern int var1;
extern int ser_cli;
extern char datos_buffer[100];
extern char IPser[20],IPcli[20];
...
```

- Declarar funciones en la DLL

Para que las aplicaciones realicen el llamado a una de las funciones declaradas en la DLL, necesitan ser exportadas e importadas para su correcto funcionamiento, esto para poder compartir las funciones, entonces centralizamos y definimos los prototipos o moldes de las funciones para compartir dichas funciones; este archivo se le conoce como archivo de cabecera donde también se exportaran e importaran las funciones con ayuda de macros.

Agregamos un nuevo archivo al proyecto, en el menú New Items seleccionamos Header File, aquí declaramos los prototipos de funciones para exportar e importar con ayuda de los macros.

```
//las siguientes líneas de código están definidas en el Anexo 2.
#ifndef _PRUEBA1_H
#define _PRUEBA1_H

#ifdef __DLL__
#   define DLL_EXP __declspec(dllexport)
#else
#   define DLL_EXP __declspec(dllimport)
#endif

//declaración de funciones como externas para ser llamadas desde otros archivos
extern "C" void DLL_EXP intcomp(int num1);
extern "C" int DLL_EXP outcomp();
extern “C” char DLL_EXP *ipservidor();
```

```
extern “C” char DLL_EXP *ipcliente();

#endif
```

Este archivo de cabecera se guarda con el nombre LibCio.h y se vincula al archivo principal de código LibCio.dll desde el Project Manager, así estará listo para incluirla en el código fuente de la DLL en el archivo LibCio.cpp con la instrucción #include “LicCio.h”, y poder definir el código de cada función.

- Funciones para el uso de la memoria compartida

Al declarar las funciones en el archivo de cabecera pasamos a definir estas funciones en el archivo LibCio.cpp de código fuente de la DLL, después de la función DLLEntryPoint(), como se muestra a continuación:

La función intcomp, hace el uso de la memoria compartida cambiando el valor interno en la DLL, como único parámetro solicita el valor que se actualizara.

```
...
void DLL_EXP intcomp(int num1)
{
  var1=num1;
}
...
```

Outcomp, esta función devuelve el valor que se encuentra en la memoria compartida, estableciendo así comunicación entre aplicaciones.

```
...
int DLL_EXP outcomp()
{
  return var1;
}
...
```

La siguiente función regresa un apuntador a la IP del servidor, contenida en la memoria compartida.

```
...
char DLL_EXP *ipservidor()
```

```
{
  char *ip=IPser;
  return ip;
}
...
```

La función ipcliente, regresa un apuntador a la ip del cliente, contenida en la memoria compartida.

```
...
char DLL_EXP *ipcliente()
{
  char *ip=IPcli;
  return ip;
}
...
```

Con estas funciones podemos cambiar u obtener el valor de un dato en la memoria compartida, haciendo el llamado correspondiente a cualquiera de las funciones anteriores.

-Funciones de conexión de la DLL

Teniendo la memoria compartida y sus funciones terminadas se plantea el intercambio de datos por petición entre computadoras conectadas en una misma red, para esto se usa la API de Windows específicamente las funciones descritas en los sockets de Windows que están definidas en la DLL wsock32.dll. Se escribe el código del servidor y cliente en la misma DLL, dependerá de la aplicación que llame a la DLL por cuál de las dos optara. Primero se declara el prototipo de las funciones en el archivo de cabecera LibCio.h, declarando las siguientes funciones:

```
...
extern "C" void DLL_EXP comunicar(int, char[20]);
extern "C" void DLL_EXP enviar(char[100]);
extern "C" char DLL_EXP *recibir();
extern "C" char DLL_EXP *mensaje();
...
```

Después se procede a definir el código de cada función declarada en el archivo LibCio.h, en el archivo LibCio.cpp de código fuente de la DLL, como sigue:

La función comunicar, establece la comunicación cliente-servidor, además crea dos estructuras de tipo sockaddr donde guarda los parámetros del cliente y del servidor como la IP y el puerto.
No regresa ningún valor, el primer parámetro solicitado es de tipo entero que especifica si es cliente o servidor la aplicación que hace el llamado a esta función, 0 = servidor y un número mayor si es cliente. El segundo parámetro solicitado es una cadena de caracteres donde se especifica el número de IP del servidor, solo si la aplicación que hace el llamado funcionara como cliente.

```
...
void DLL_EXP comunicar(int tipo, char numip[20])
{
  if(tipo==0)
  {
    //Configuración del servidor
    resp=WSAStartup(MAKEWORD(1,0),&wsaData);   //Cargar DLL del WinSock. Anexo 5
    if(resp!=0)
    {
      MessageBox(NULL,"Error de carga","ERROR de apertura de socket",MB_OK);
      WSACleanup();
      return;
    }
    memset(Host,0,50);
    gethostname(Host,50);   //Obtener nombre del sistema donde se ejecuta. Anexo 14
    hp=gethostbyname(Host);   //Dirección IP del sistema donde se ejecuta. Anexo 15
    if (hp == NULL)
    {
      MessageBox(NULL,"Error de carga","ERROR, no se encuentra host",MB_OK);
      WSACleanup();
      return;
    }
    sprintf(direccion,"%d.%d.%d.%d",UC(hp->h_addr[0]),UC(hp->h_addr[1]),UC(hp-
>h_addr[2]),UC(hp->h_addr[3]));   //Conversion. Anexo 16
    //Comienza la estructura de comunicación UDP/IP
    sockprueba=socket(AF_INET,SOCK_DGRAM,17);   //Función definida en el Anexo 8
    if (sockprueba < 0)
    {
      MessageBox(NULL,"Error","ERROR de apertura de socket",MB_OK);
      WSACleanup();
      return;
    }
```

```
    //Creación de la estructura con la dirección IP del servidor
    server.sin_family=AF_INET;
    server.sin_port=htons(5001);
    server.sin_addr.s_addr=inet_addr(direccion);
    //Agregando dirección al socket
    If(bind(sockprueba,(struct sockaddr *) &server,sizeof(server))<0)   //Anexo 9
    {
      MessageBox(NULL,"Error","ERROR en la conexión",MB_OK);
      WSACleanup();
      return;
    }
    //Recibiendo datos del cliente
    ancho=sizeof(client);
    recvfrom(sockprueba,msj_cli,100,0,(struct sockaddr *)&client,&ancho);   //Anexo 13
    //Enviando datos al cliente
    strcpy(msj_cli2,msj_cli);
    strcpy(datos_buffer,msj_cli);
    sprintf(msj_cli,"Conexion exitosa...");
    //Función definida en el Anexo 12
    sendto(sockprueba,msj_cli,strlen(msj_cli)+1,0,(struct sockaddr *)&client,sizeof(client));
    closesocket(sockprueba);   //Cerrar socket
    WSACleanup();   //Descargar DLL del WinSock
  }
  else
  {
    //Configuración del cliente
    resp=WSAStartup(MAKEWORD(1,0),&wsaData);   //Cargar DLL del WinSock. Anexo 5
    if(resp!=0)
    {
      MessageBox(NULL,"Error de carga","No fue posible cargar DLL API",MB_OK);
      WSACleanup();
      return;
    }
    memset(Host,0,50);
    gethostname(Host,50);   //Obtener nombre del sistema donde se ejecuta. Anexo 14
    hp=gethostbyname(Host);   //Dirección IP del sistema donde se ejecuta. Anexo 15
    if (hp == NULL)
    {
      MessageBox(NULL,"Error de carga","ERROR, no se encuentra host",MB_OK);
      WSACleanup();
      return;
    }
    sprintf(direccion,"%d.%d.%d.%d",UC(hp->h_addr[0]),UC(hp->h_addr[1]),UC(hp-
>h_addr[2]),UC(hp->h_addr[3]));   //Conversión definida en el Anexo 16
```

```
    //Comienza la estructura de comnunicacion UDP/IP
    sockprueba=socket(AF_INET,SOCK_DGRAM,17);   //Función definida en el Anexo 8
    if (sockprueba < 0)
    {
      MessageBox(NULL,"Error","ERROR de apertura de socket",MB_OK);
      WSACleanup();
      return;
    }
    //Creación de la estructura con la dirección IP del servidor
    server.sin_family=AF_INET;
    server.sin_port=htons(5001);
    server.sin_addr.s_addr=inet_addr(numip);
    //Creación de la estructura con la dirección IP del cliente maquina actual
    client.sin_family=AF_INET;
    client.sin_port=htons(5001);
    client.sin_addr.s_addr=inet_addr(direccion);
    //Agregando dirección al socket
    If(bind(sockprueba,(struct sockaddr *) &client,sizeof(server))<0)   //Anexo 9
    {
      MessageBox(NULL,"Error","ERROR en la conexión",MB_OK);
      WSACleanup();
      return;
    }
    //Enviando datos al servidor
    sprintf(msj_cli,"Envio de datos cliente...");
    //Función definida en el Anexo 12
    sendto(sockprueba,msj_cli,strlen(msj_cli)+1,0,(struct sockaddr *)&server,
sizeof(server));
    //Recibiendo datos del servidor
    ancho=sizeof(server);
    recvfrom(sockprueba,msj_cli2,100,0,(struct sockaddr *)&client,&ancho);   //Anexo 13
    //Aumentar el número de clientes
    ser_cli++;    //Aumentar para declarar que se conectó un cliente
    strcpy(datos_buffer,msj_cli2);
    closesocket(sockprueba);   //Cerrar socket
    WSACleanup();   //Descargar DLL del WinSock
  }
  strcpy(IPser,numip);      //Guardar la dirección IP del servidor en la memoria compartida
  strcpy(IPcli,direccion);   // Guardar la dirección IP del cliente en la memoria compartida
}
```

...

La siguiente función enviar, manda datos por el socket establecido dependiendo si es servidor o cliente, la aplicación que hace el llamado a la función.

No devuelve ningún valor, el único parámetro que solicita es una cadena de caracteres con el mensaje a enviar.

```
...
void DLL_EXP enviar(char msj[100])
{
  //Declaracion de tipo de datos
  WSADATA wsa;
  int sock1,regreso;
  regreso=WSAStartup(MAKEWORD(1,0),&wsa);   //Cargar DLL WinSock. Anexo 5
    if(regreso!=0)
    {
      MessageBox(NULL,"Error de carga","No fue posible cargar DLL API",MB_OK);
      WSACleanup();
      return;
    }
  sock1=socket(AF_INET,SOCK_DGRAM,17);
  bind(sock1,(struct sockaddr *) &server,sizeof(server));
  //Enviar datos, si es servidor lo envía al cliente, si es cliente al contrario
  if(ser_cli==0)
    sendto(sock1,msj,strlen(msj)+1,0,(struct sockaddr *)&client,sizeof(client));
  else
    sendto(sock1,msj,strlen(msj)+1,0,(struct sockaddr *)&server,sizeof(server));
  closesocket(sock1);   //Cerrar socket
  WSACleanup();    //Liberar DLL
  strcpy(datos_buffer,msj);
}
...
```

Función recibir, esta función recibe datos especificando desde que dirección IP se espera recibir la información y se almacena en el buffer, se bloquea hasta recibir datos.
Devuelve un apuntador a una cadena que es el mensaje recibido.

```
...
char DLL_EXP *recibir()
{
  //Declaracion de tipo de datos
  struct sockaddr_in recv;
  char mensaje[100];
  char *msj;
  int tam;
  WSADATA wsa;
  int sock2,regreso;
```

```
  regreso=WSAStartup(MAKEWORD(1,0),&wsa);   //Cargar DLL WinSock. Anexo 5
    if(regreso!=0)
    {
      MessageBox(NULL,"Error de carga","No fue posible cargar DLL API",MB_OK);
      WSACleanup();
      sprintf(mensaje,"No fue posible cargar DLL Winsock");
      msj=mensaje;
      return msj;
    }
  //Crear socket y definer los parametros IP y puerto
  sock2=socket(AF_INET,SOCK_DGRAM,17);
  bind(sock2,(struct sockaddr *) &server,sizeof(server));
  tam=sizeof(recv);
  recvfrom(sock2,mensaje,100,0,(struct sockaddr *)&recv,&tam);      //Recibe datos
  strcpy(datos_buffer,mensaje);   //Guardar en el buffer
  msj=datos_buffer;
  closesocket(sock2);   //Cerrar socket
  WSACleanup();   //Liberar DLL Winsock
  return msj;   //Regresar los datos recibidos
}
```

...

La Función mensaje, permite conocer los datos contenidos en el buffer de la DLL, devuelve un apuntador a una cadena que contendrá el buffer.

...

```
char DLL_EXP *mensaje()
{
  char *msj;
  msj=datos_buffer;
  return msj;
}
```

...

Definido el código de las funciones declaradas se procede a construir la DLL, nos vamos a Build LibCio en el menú Project, esperando a que nos muestre la siguiente ventana:

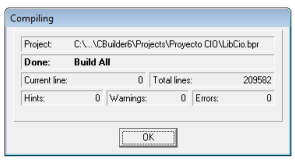

Figura 6. DLL compilada exitosamente.

CAPÍTULO 4

RESULTADOS

Para poder obtener resultados de la DLL compilada, se necesita crear una aplicación que haga uso de esta DLL y comprobar si las funcionalidades devuelven los resultados deseados, empezamos a crear una aplicación:

- Crear aplicación para DLL

Se procede a crear dos aplicaciones que carguen la DLL y hagan el llamado a sus funciones, una aplicación con funciones de servidor y la otra de cliente, estas aplicaciones son solo para fines de comprobación en el correcto funcionamiento de la DLL, en la carga de la misma y el llamado a sus funciones.

Para Comenzar se crean las dos nuevas aplicaciones, estas se crean en el grupo del proyecto. Nos vamos a File-New, y elegimos Application, y se agrega la aplicación al proyecto del grupo, realizamos los mismos pasos para la segunda aplicación; el Project Manager quedara de esta manera:

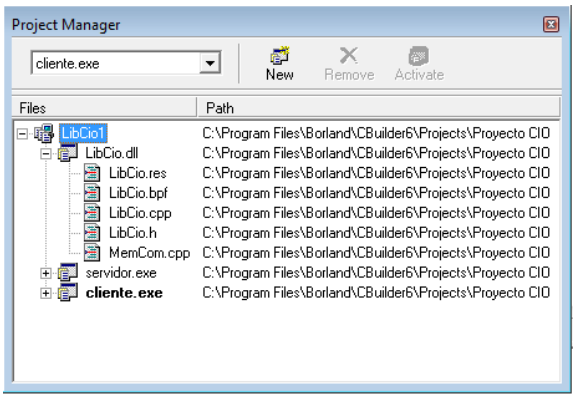

Figura 7. Vista del Project Manager

Teniendo las dos aplicaciones creadas, se empieza a escribir el código correspondiente a cada aplicación.

-Cargando DLL a la aplicación

Para cargar la DLL en la aplicación se necesita agregar al proyecto el archivo de extensión .lib en nuestro caso el archivo LibCio.lib, e incluir el archivo de encabezado de la DLL, el

archivo LibCio.h, una vez establecido los requisitos para cargar la DLL, se procede a personalizar la aplicación, agregando Button, Label, Edit y demás, necesarios para hacer uso de las funciones de la DLL. Las aplicaciones lucen de esta manera:

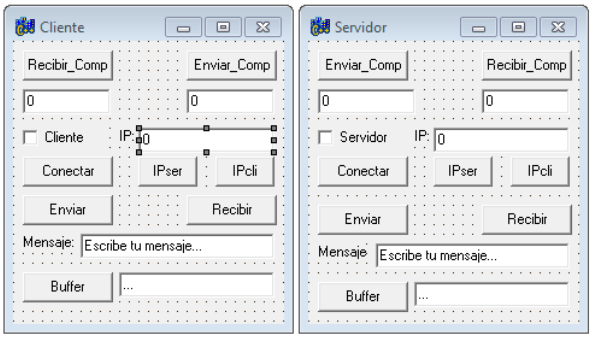

Figura 8. Diseño de las Aplicaciones

- Código de las aplicaciones:

```
//se declaran los tipos de datos a usar
int entrada=0,ser_cli=0;
char numip[20], buffer[100];
//Introducir dato a la memoria compartida
void __fastcall TServidor::Button3Click(TObject *Sender)
{
  intcomp(StrToInt(Edit3->Text));   //Función de la DLL
}
//Obtener dato de la memoria compartida
void __fastcall TServidor::Button4Click(TObject *Sender)
{
  Edit4->Text=IntToStr(outcomp());   //Función de la DLL
}
//Declarar si la aplicación funcionara como Servidor o Cliente
void __fastcall TServidor::CheckBox1Click(TObject *Sender)
{
  if(CheckBox1->Checked){
    ser_cli=1;   //Entero que declara si funcionara como servidor o cliente
    CheckBox1->Caption="Cliente";
    Label1->Visible=true;
    Edit5->Visible=true;
  }
  else{
    ser_cli=0;
```

```
    CheckBox1->Caption="Servidor";
    Label1->Visible=false;
    Edit5->Visible=false;
  }
}
//Establece la comunicación
void __fastcall TServidor::Button5Click(TObject *Sender)
{
  memset(numip,0,sizeof(numip));   //pone a ceros la cadena numip
  strcpy(numip,Edit5->Text.c_str());   //Copia el contenido del Edit5 (IP) a numip.
  comunicar(ser_cli,numip);   //Función de la DLL
}
//Recibe datos por la conexion
void __fastcall TServidor::Button7Click(TObject *Sender)
{
  char *texto;   //Crea un puntero tipo char
  texto=recibir();   //Función de la DLL
  Edit6->Text=texto;  //Muestra el mensaje en el Edit6
}
//Obtiene el contenido del buffer de comunicación
void __fastcall TServidor::Button8Click(TObject *Sender)
{
  char *msj=mensaje();   //Crea un punter y recibe el contenido del buffer
  Edit7->Text=msj;   //Muestra el contenido del mensaje en el Edit7
}
//Envia mensaje por el puerto de comunicación
void __fastcall TServidor::Button6Click(TObject *Sender)
{
  char msj[100];
  memset(msj,0,sizeof(msj));
  strcpy(msj,Edit6->Text.c_str());
  enviar(msj);   //Función de la DLL
}
//Optiene la dirección IP del servidor
void __fastcall TServidor::Button2Click(TObject *Sender)
{
char *IP;   //Crea un apuntador
  IP=ipservidor();   //Función de la DLL
  Edit7->Text=IP;
}
//Optiene la dirección IP del cliente
void __fastcall TServidor::Button1Click(TObject *Sender)
{
  char *IP;   //Crea un apuntador
```

```
  IP=ipcliente();   //Función de la DLL
  Edit7->Text=IP;
}
```

Para la segunda aplicación es el mismo código que el de la primera aplicación, lo único que cambia es la declaración de ser_cli=1, para establecer que inicie como cliente.

Al usar los botones Enviar_comp y Recibir_comp, se hace uso de la memoria compartida, permitiendo que se compartan datos entre aplicaciones, se logra obtener lo deseado, como se aprecia en la siguiente imagen:

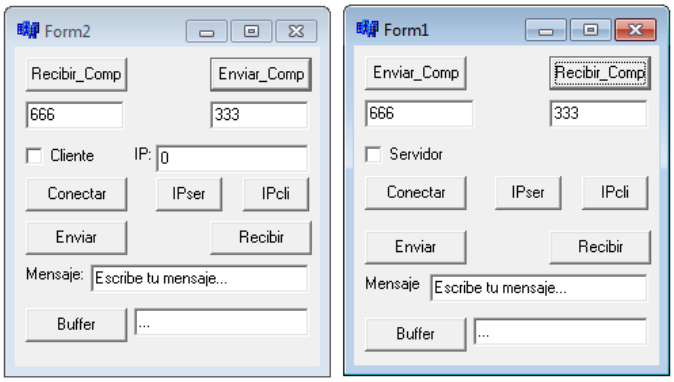

Figura 9. Compartir datos en el segmento compartido.

Establecer conexión, para conectar dos aplicaciones con el uso de sockets, se hace uso de los botones Conectar, en las dos aplicaciones, como se logra apreciar en la imagen, una aplicación tiene como valores prestablecidos en el CheckBox, funcionar como servidor y la otra como cliente, en la aplicación cliente, se necesita escribir la dirección IP del servidor como un parámetro necesario. Al presionar el botón conectar en ambas aplicaciones se logra la conexión:

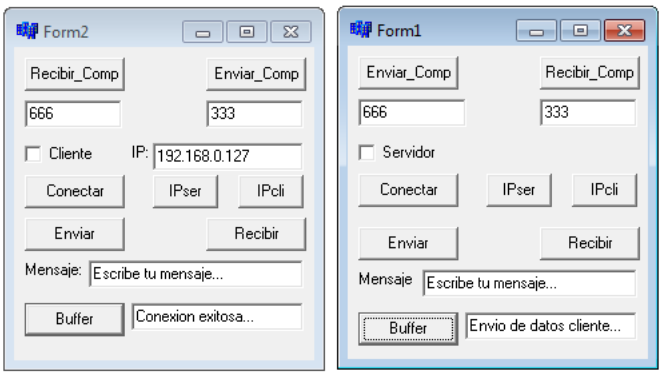

Figura 10. Conexión establecida.

Se puede conocer la dirección IP del servidor o la dirección IP del cliente, para esto se usan los botones IPser e IPcli de las aplicaciones que son obtenidas de la memoria compartida de la DLL.

Figura 11. Obtener dirección IP

Para enviar y recibir mensajes entre aplicaciones ejecutadas en diferentes computadoras, se usan los botones Enviar y Recibir, el mensaje se escribe y se muestra en el campo Mensaje de las aplicaciones.

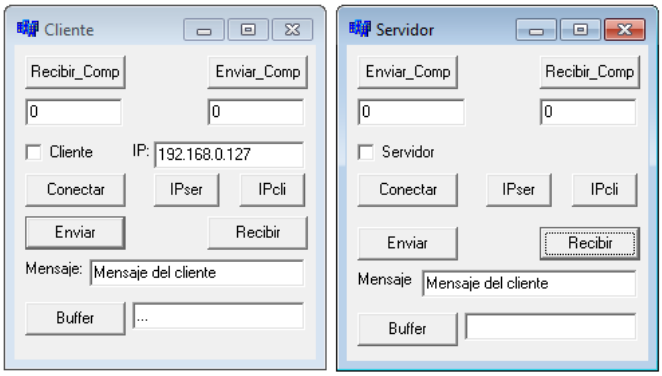

Figura 12. Mensaje recibido desde el cliente.

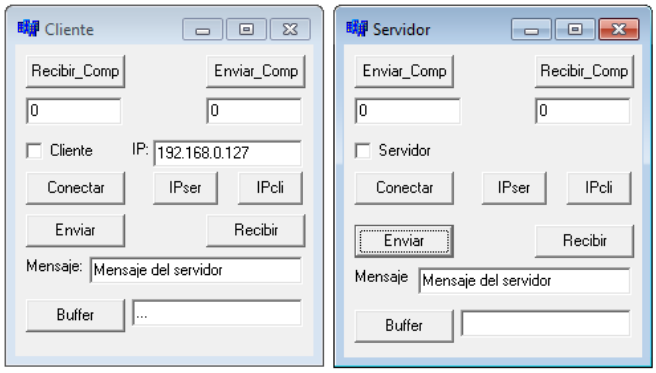

Figura 13. Mensaje recibido desde el servidor.

Al hacer uso de las aplicaciones y hacer el llamado a cada una de la las funciones llamadas desde la DLL, comprobamos que funcionan correctamente, por lo que se concluye con un éxito.

CONCLUSIONES Y RECOMENDACIONES

Se comprendió y explico la estructura de una DLL, los archivos que se deben crear para su funcionamiento, también cómo funciona la carga y descarga de las funciones y de la propia DLL.

Escribimos el código de programación de la propia DLL y se crearon aplicaciones que hagan el uso de la DLL, en el transcurso del proyecto para probar el correcto funcionamiento de las mismas, se pudo establecer la comunicación entre aplicaciones que son ejecutadas en un mismo ordenador, gracias a la funcionalidad de compartir datos en un segmento de memoria compartida dentro de la DLL.

Se estudiaron los protocolos de comunicación establecidos en el modelo OSI, en la capa 4, la capa de transporte, el protocolo UDP/IP fue el elegido. Entonces se estudió el funcionamiento del protocolo UDP y como se podía agregar esta funcionalidad a la DLL. Se encontraron los sockets de Windows que son funcionalidades de la API de Windows, por lo que se optó programar con la DLL de los socket de Windows, brindando la solución para agregar a la DLL, así se completó la comunicación de datos en una DLL y se implementó la nueva funcionalidad en una aplicación. Así se pudo establecer la conexión entre aplicaciones que son ejecutadas en diferentes computadoras de una misma red. Cumpliendo así todos los objetivos planteados al inicio del proyecto.

Se recomendaría, si se quiere ampliar más el proyecto. La creación de una DLL que puedan usar programadores que usen otros IDE y compiladores, debido a problemas de compatibilidad en el formato del archivo LIB, de esta manera se puede ampliar el rango de aplicaciones que hagan uso de la DLL. También se podría agregar conexiones más seguras y estables como lo proporciona el protocolo TCP/IP.

REFERENCIAS BIBLIOGRÁFICAS

[1] Swart Bob, Cashman Mark, Gustavson Paul, Hollingworth Jarrod. *Borland C++ Builder 6 Developer´s Guide* (1ª. Edición), Sams Publishing. Indianapolis: 2002.

[2] Reisdorph, Kent. *Aprendiendo Borland C++ Builder en 21 Días* (Edición en Español), Prentice Hall. México: 1999.

[3] Calvert´s, Charlie. *Borland C++ Builder Unleashed* (1ª. Edición), Sams Publishing. Estados Unidos de America: 1997.

[4] Swan, Tom. *Mastering Borland C++ 4.5* (2ª. Edición), Sams Premie. Estados Unidos de America: 1995.

[5] Anthony Jones, Jim Ohlund. *Network Programming for Microsoft Windows* (1ª. Edición), Microsoft Press. Estados Unidos de America: 1999.

[6] Petzold, Charles. *Programming Windows* (6ª Edición), Microsoft Press. Redmond, Washington: 2012.

[7] Ayres John, Bowden David, Diehl Larry, Dorcas Phil, Harrison Ken, Mathes Rod, Reza Ovais, Tobin Mike. *Los Tomos de Delphi: El API gráfico Win32*, Danysoft Internacional. Madrid, España: 2000.

ANEXOS

Anexo 1:

```
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
return 1;
}
```

HINSTANCE: es usado para cargar recursos y cualquier otra tarea que sea realizada en una base por-modulos. Un módulo puede ser el archivo .EXE

Los tres parámetros que se encuentran en la función de entrada DLLEntryPoint, el primer parámetro *hinst*, identifica el handle de la DLL; el segundo parámetro identifica el tipo de actividad asociada al proceso de carga o descarga; el tercer parámetro identifica si la DLL es cargada de forma dinámica o estática.

Anexo 2:

```
#ifdef __DLL__
#   define EXP_IMP __declspec(dllexport)
#else
#   define EXP_IMP __declspec(dllimport)
#endif
```

La directiva # define se utiliza principalmente para asociar identificadores significativos con constantes, palabras clave y manifestaciones o expresiones de uso común.

Cuando se crea una DLL se define automáticamente el símbolo __DLL__ y no se define cuando se crea una aplicación.

Si está definido el símbolo __DLL__, se define el símbolo EXP_IMP y se expande a __declspec(dllexport), de lo contrario si no está definido el símbolo __DLL__, el símbolo EXP_IMP, se expande a __declspec(dllimport). Todo esto implica que se use solamente un encabezado.

Anexo 3:

Se Utiliza #pragma option para incluir las opciones de línea de comandos en el código del programa. Para crear el segmento y clase compartida se agrega los atributos:

```
#pragma option -zR[SEGMENT NAME]   //segmento compartido
#pragma option -zT[CLASS NAME]          //clase compartida
```

Anexo 4:

Esta sección SEGMENTS, define los atributos de segmentos adicionales de código y datos.

Sintaxis: SEGMENTS

```
SegmentName [CLASS 'ClassName']
 [MinAlloc]
 [SHARED | NONSHARED]
 [PRELOAD | LOADONCALL]
```

SegmentName: Cadena de caracteres que identifica a cada segmento.

ClassName: Este identificador es opcional, es el nombre de clase del segmento.

MinAlloc: Este dato es opcional, un número entero que indica el tamaño mínimo de memoria que se asignará al segmento.

SHARED: Una copia del segmento es compartida entre todos los procesos.

NONSHARED: Es el valor por defecto para .EXEs y .DLLs.

PRELOAD: Significa que el segmento es cargado inmediatamente.

LOADONCALL: Indica que el segmento es cargado cuando es accedido o invocado.

Anexo 5:

int WSAStartup(WORD wVersionRequested, WSADATA lpWSAData)

wVersionRequested: especifica la versión de la DLL del socket de Windows que se requiere cargar, se puede usar la macro MAKEWORD(x,y) para obtener el correcto valor de la versión.

lpWSAData: es un apuntador a una estructura de tipo WSAData.

La sintaxis de la estructura WSAData esta descrita a continuación:

```
typedef struct WSAData
{
        WORD            wVersion;
        WORD            wHighVersion;
        char            szDescription[WSADESCRIPTION_LEN + 1];
        char            szSystemStatus[WSASYS_STATUS_LEN + 1];
        unsigned short  iMaxSockets;
        unsigned short  iMaxUdpDg;
        char FAR *      lpVendorInfo;
} WSADATA, * LPWSADATA;
```

El primer parametro wVersion , a la versión que va a utilizar . El parámetro wHighVersion se refiere a la versión más alta de la biblioteca disponible. El parámetro szDescription y szSystemStatus son establecidos por la aplicación particular del socket de windows. Los campos iMaxSockets y iMaxUdpDg definen el número máximo de sockets y tamaño máximo de un paquete UDP. El ultimo parámetro lpVendorInfo informa específica del proveedor en cuanto a la ejecución del socket de windows.

Anexo 6:

Funciones definidas de Network Byte Order:

Htonl:
unsigned long int htonl(unsigned long int hostlong);

La función htonl convierte el entero de 32 bits dado por hostlong desde el orden de host-bytes al orden de red-bytes.

Htons:
unsigned short int htons(unsigned short int hostshort);

La función htons convierte el entero de 16 bits dado por hostshort desde el orden de host-bytes al orden de red-bytes.

Ntohl:
unsigned long int ntohl(unsigned long int netlong);

La función ntohl convierte el entero de 32 bits dado por netlong desde el orden de

Red-bytes al orden de host-bytes.

Ntohs:
unsigned short int ntohs(unsigned short int netshort);

La función ntohs convierte el entero de 16 bits dado por netshort desde el orden de

Red-bytes al orden de host-bytes.

Anexo 7:

Estructuras de datos más usadas:

```
struct sockaddr
{
        unsigned short sa_family; // familia de la dirección
        char sa_data[14]; // Dirección de protocolo de 14 bytes.
};

struct sockaddr_in
{
        short int sin_family; // familia de la dirección
        unsigned short sin_port; // Numero de puerto.
        struct in_addr sin_addr; // Dirección IP.
        unsigned char sin_zero[8]; // Relleno.
};
```

Anexo 8:

-Función socket: Sirve para crea un socket o abrir una conexión. Sintaxis:

int socket(int domain, int type, int protocol);

domain: dominio donde se realiza la conexión, eligiendo entre AF_UNIX o AF_INET.

Type: especifica el tipo de socket a crear y toma los valores SOCK_STREAM, SOCK_DGRAM o SOCK_RAW.

Protocol: hace referencia al protocolo a establecer.

Anexo 9:

-Función bind: esta función asocia los parámetros de dirección IP y número de puerto al socket. Sintaxis:

int bind(int socket, struct sockaddr *addr, int addrlen);

socket: identificador del socket.

addr: puntero a una estructura sockaddr, esta estructura contiene la dirección IP y el puerto que se asignara al socket.

addrlen: tamaño de la estructura sockaddr.

Anexo 10:

-Función listen: habilita un socket para recibir peticiones de conexión y poder establecer la conexión. Sintaxis:

int listen(int socket, int backlog);

socket: Identificador del socket.

blacklog: Número máximo de conexiones que se pueden establecer en la cola de entrada.

Anexo 11:

-Función accept: esta función espera hasta que un cliente establezca la conexión con el servidor, esta función es bloqueante por lo que no finalizara hasta establecer una conexión o sea interrumpida. Sintaxis:

int accept(int socket, struct sockaddr *addr, socklen_t *addrlen);

socket: identificador al socket.

addr: puntero a una estructura sockaddr, en la cual se almacenaran los datos del cliente.

addrlen: puntero a un valor entero que contendrá el tamaño de la estructura sockaddr.

-Función closesocket: cierra o termina el socket, es importante cerrar un socket cuando ya no se utilice más, indicándole al sistema que lo libere. Sintaxis:

int close(int socket);

socket: Identificador al socket.

Anexo 12:

-Función sendto: envía datos a través de sockets no orientado a la conexión, esta función permite indicar quien es el destinatario. Sintaxis:

ssize_t sendto(int socket, const void *buffer, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);

socket: identificador al socket.

buffer: untero a los datos que se desean enviar.

len: tamaño de los datos en bytes.

to: puntero a una estructura sockaddr que contiene los datos del destino.

tolen: tamaño de la estructura sockaddr.

Anexo 13:

-Función recvfrom: recibe datos a través de sockets no orientados a la conexión, permite indicar quien es que envía los datos. Sintaxis:

ssize_t recvfrom (int socket, void *bufffer, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);

socket: Identificador del socket.

buffer: puntero en donde se almacenaran los datos recibidos.

len: número máximo de bytes a recibir.

from: puntero a una estructura sockaddr, en donde se guardaran los datos de la máquina que envía los datos.

fromlen: tamaño de la estructura sockaddr.

Anexo 14:

- función gethostname, devuelve el nombre del sistema donde se está ejecutando el programa. Devuelve 0 cuando se ha ejecutado con éxito. Sintaxis:

int gethostname (char *hostname, size_t size);

hostname: Puntero a un array de caracteres donde se almacenará el nombre de la máquina.

size: Longitud en bytes del array hostname.

Anexo 15:

-Función gethostbyname, se utiliza para convertir un nombre de máquina en una dirección IP, a través de una estructura de tipo hostent. Sintaxis:

struct hostent *gethostbyname (const char *name);

name: Nombre de la máquina que se quiere conocer la dirección IP.

La función devuelve un puntero a una estructura de tipo hostent. En el caso de producirse algún error, devuelve NULL y establece la variable h_errno. Sintaxis:

```
struct hostent{
char *h_name;
char **h_aliases;
int h_addrtype;
int h_length;
char **h_addr_list;
};
```

h_name: Nombre oficial de la máquina.

h_aliases: Array de nombres alternativos.

h_addrtype: Tipo de dirección de retorno (AF_INET).

h_length: Longitud de la dirección en bytes.

h_addr_list: Array de direcciones de red para la máquina.

h_addr: La primera dirección en h_addr_list.

Anexo 16:

La primera dirección en h_addr_list de la estructura hosted es h_addr, Cada byte de la dirección IP figura en esta lista, que podemos acceder por su índice. Sin embargo, estos datos están contenidos en un número entero. Así que, se aplica la siguiente macro, que borra todos los valores superiores a 255 para darnos un byte, este macro se escribe en el archivo de cabecera.

```
# define UC(b) (((int)b)&0xff)
```

Para finalizar se usa la función sprintf(), que convierte el valor de la dirección IP en su notación normal, de modo que pueda ser mostrado al usuario.

Anexo 17:

- Código fuente de la DLL:

Código del archivo LibCio.h:

```
//---------------------------------------------------------------------------
#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#pragma hdrstop

//---------------------------------------------------------------------------
#pragma argsused
#include "prueba1.h"
```

```
extern int var1;
extern int ser_cli;
extern char datos_buffer[100];
extern char IPser[20],IPcli[20];

int resp;
WSADATA wsaData;
int sockprueba;
struct sockaddr_in server,client;
char Host[50];
char msj_cli[100], msj_cli2[100];
struct hostent *hp;
char direccion[20];
int ancho,sercli;
int WINAPI DllEntryPoint(HINSTANCE, unsigned long, void*)
{
      return 1;
}
//---------------------------------------------------------------------------
void DLL_EXP intcomp(int num1)
{
  var1=num1;
}
//---------------------------------------------------------------------------

int DLL_EXP outcomp()
{
  return var1;
}
//---------------------------------------------------------------------------

char DLL_EXP *ipservidor()
{
  char *ip=IPser;
  return ip;
}
//---------------------------------------------------------------------------

char DLL_EXP *ipcliente()
{
  char *ip=IPcli;
  return ip;
}
//---------------------------------------------------------------------------
```

```
void DLL_EXP comunicar(int tipo, char numip[20])
{
  if(tipo==0)
  {
    resp=WSAStartup(MAKEWORD(1,0),&wsaData);
    if(resp!=0)
    {
      MessageBox(NULL,"Error de carga","ERROR de apertura de socket",MB_OK);
      WSACleanup();
      return;
    }
    memset(Host,0,50);
    gethostname(Host,50);
    hp=gethostbyname(Host);
    if (hp == NULL)
    {
      MessageBox(NULL,"Error de carga","ERROR, no se encuentra host",MB_OK);
      WSACleanup();
      return;
    }
    sprintf(direccion,"%d.%d.%d.%d",UC(hp->h_addr[0]),UC(hp->h_addr[1]),UC(hp-
>h_addr[2]),UC(hp->h_addr[3]));
    sockprueba=socket(AF_INET,SOCK_DGRAM,17);
    if (sockprueba < 0)
    {
      MessageBox(NULL,"Error","ERROR de apertura de socket",MB_OK);
      WSACleanup();
      return;
    }
    server.sin_family=AF_INET;
    server.sin_port=htons(5001);
    server.sin_addr.s_addr=inet_addr(direccion);
    If(bind(sockprueba,(struct sockaddr *) &server,sizeof(server))<0)
    {
      MessageBox(NULL,"Error","ERROR en la conexión",MB_OK);
      WSACleanup();
      return;
    }
    ancho=sizeof(client);
    recvfrom(sockprueba,msj_cli,100,0,(struct sockaddr *)&client,&ancho);
    strcpy(msj_cli2,msj_cli);
    strcpy(datos_buffer,msj_cli);
    sprintf(msj_cli,"Conexion exitosa...");
```

```
    sendto(sockprueba,msj_cli,strlen(msj_cli)+1,0,(struct sockaddr *)&client,sizeof(client));
    closesocket(sockprueba);
    WSACleanup();
  }
  else
  {
    resp=WSAStartup(MAKEWORD(1,0),&wsaData);
    if(resp!=0)
    {
      MessageBox(NULL,"Error de carga","No fue posible cargar DLL API",MB_OK);
      WSACleanup();
      return;
    }
    memset(Host,0,50);
    gethostname(Host,50);
    hp=gethostbyname(Host);
    if (hp == NULL)
    {
      MessageBox(NULL,"Error de carga","ERROR, no se encuentra host",MB_OK);
      WSACleanup();
      return;
    }
    sprintf(direccion,"%d.%d.%d.%d",UC(hp->h_addr[0]),UC(hp->h_addr[1]),UC(hp-
>h_addr[2]),UC(hp->h_addr[3]));
    sockprueba=socket(AF_INET,SOCK_DGRAM,17);
    if (sockprueba < 0)
    {
      MessageBox(NULL,"Error","ERROR de apertura de socket",MB_OK);
      WSACleanup();
      return;
    }
    server.sin_family=AF_INET;
    server.sin_port=htons(5001);
    server.sin_addr.s_addr=inet_addr(numip);
    client.sin_family=AF_INET;
    client.sin_port=htons(5001);
    client.sin_addr.s_addr=inet_addr(direccion);
    If(bind(sockprueba,(struct sockaddr *) &client,sizeof(server))<0)
    {
      MessageBox(NULL,"Error","ERROR en la conexión",MB_OK);
      WSACleanup();
      return;
    }
    sprintf(msj_cli,"Envio de datos cliente...");
```

```
    sendto(sockprueba,msj_cli,strlen(msj_cli)+1,0,(struct sockaddr *)&server,
sizeof(server));
    ancho=sizeof(server);
    recvfrom(sockprueba,msj_cli2,100,0,(struct sockaddr *)&client,&ancho);
    ser_cli++;
    strcpy(datos_buffer,msj_cli2);
    closesocket(sockprueba);
    WSACleanup();
  }
  strcpy(IPser,numip);
  strcpy(IPcli,direccion);
}
//---------------------------------------------------------------------------
void DLL_EXP enviar(char msj[100])
{

  WSADATA wsa;
  int sock1,regreso;
  regreso=WSAStartup(MAKEWORD(1,0),&wsa);
    if(regreso!=0)
    {
      MessageBox(NULL,"Error de carga","No fue posible cargar DLL API",MB_OK);
      WSACleanup();
      return;
    }
  sock1=socket(AF_INET,SOCK_DGRAM,17);
  bind(sock1,(struct sockaddr *) &server,sizeof(server));
  if(ser_cli==0)
    sendto(sock1,msj,strlen(msj)+1,0,(struct sockaddr *)&client,sizeof(client));
  else
    sendto(sock1,msj,strlen(msj)+1,0,(struct sockaddr *)&server,sizeof(server));
  closesocket(sock1);
  WSACleanup();
  strcpy(datos_buffer,msj);
}
//---------------------------------------------------------------------------
char DLL_EXP *recibir()
{
  struct sockaddr_in recv;
  char mensaje[100];
  char *msj;
  int tam;
  WSADATA wsa;
  int sock2,regreso;
```

```
  regreso=WSAStartup(MAKEWORD(1,0),&wsa);
    if(regreso!=0)
    {
      MessageBox(NULL,"Error de carga","No fue posible cargar DLL API",MB_OK);
      WSACleanup();
      sprintf(mensaje,"No fue posible cargar DLL Winsock");
      msj=mensaje;
      return msj;
    }
  sock2=socket(AF_INET,SOCK_DGRAM,17);
  bind(sock2,(struct sockaddr *) &server,sizeof(server));
  tam=sizeof(recv);
  recvfrom(sock2,mensaje,100,0,(struct sockaddr *)&recv,&tam);
  strcpy(datos_buffer,mensaje);
  msj=datos_buffer;
  closesocket(sock2);
  WSACleanup();
  return msj;
}
//---------------------------------------------------------------------------
char DLL_EXP *mensaje()
{
  char *msj;
  msj=datos_buffer;
  return msj;
}
```

Código del archivo MemCom.cpp:

```
//---------------------------------------------------------------------------
#pragma option -zRSHSEG
#pragma option -zTSHCLS

  int var1=0;
  int ser_cli=0;
  char datos_buffer[100];
  char IPser[20],IPcli[20];
```

Código del archivo LibCio.h:

```
//---------------------------------------------------------------------------
#ifndef _LIBCIO_H
#define _LIBCIO_H
```

```
#ifdef __DLL__
#   define DLL_EXP __declspec(dllexport)
#else
#   define DLL_EXP __declspec(dllimport)
#endif
#define UC(b) (((int)b)&0xff)

extern "C" void DLL_EXP intcomp(int);
extern "C" int DLL_EXP outcomp();
extern "C" void DLL_EXP comunicar(int, char[20]);
extern "C" void DLL_EXP enviar(char[100]);
extern "C" char DLL_EXP *recibir();
extern "C" char DLL_EXP *mensaje();

#endif
```

Código del archivo Definir.def:

```
//---------------------------------------------------------------------------
LIBRARY LibCio

SEGMENTS
        SHSEG CLASS 'SHCLS' SHARED
```

- Código fuente de la aplicación Servidor:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Servidor.h"
#include "LibCio.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TServidor *Servidor;
int entrada=0,ser_cli=0;
char numip[20], buffer[100];
//---------------------------------------------------------------------------
__fastcall TServidor::TServidor(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

```
/*void __fastcall TForm1::Button1Click(TObject *Sender)
{
  meter(StrToInt(Edit1->Text));
}*/
//---------------------------------------------------------------------------
/*void __fastcall TForm1::Button2Click(TObject *Sender)
{
  entrada=sacar();
  Edit2->Text=IntToStr(entrada);
}*/
//---------------------------------------------------------------------------
void __fastcall TServidor::Button3Click(TObject *Sender)
{
  intcomp(StrToInt(Edit3->Text));
}
//---------------------------------------------------------------------------
void __fastcall TServidor::Button4Click(TObject *Sender)
{
  Edit4->Text=IntToStr(outcomp());
}
//---------------------------------------------------------------------------
void __fastcall TServidor::CheckBox1Click(TObject *Sender)
{
  if(CheckBox1->Checked){
    ser_cli=1;
    CheckBox1->Caption="Cliente";
    Label1->Visible=true;
    Edit5->Visible=true;
  }
  else{
    ser_cli=0;
    CheckBox1->Caption="Servidor";
    Label1->Visible=false;
    Edit5->Visible=false;
  }
}
//---------------------------------------------------------------------------
void __fastcall TServidor::Button5Click(TObject *Sender)
{
  memset(numip,0,sizeof(numip));
  strcpy(numip,Edit5->Text.c_str());
  comunicar(ser_cli,numip);
}
```

```
//---------------------------------------------------------------------------
void __fastcall TServidor::Button7Click(TObject *Sender)
{
  char *texto;
  texto=recibir();
  Edit6->Text=texto;
}
//---------------------------------------------------------------------------
void __fastcall TServidor::Button8Click(TObject *Sender)
{
  char *msj=mensaje();
  Edit7->Text=msj;
}
//---------------------------------------------------------------------------
void __fastcall TServidor::Button6Click(TObject *Sender)
{
  char msj[100];
  memset(msj,0,sizeof(msj));
  strcpy(msj,Edit6->Text.c_str());
  enviar(msj);
}
//---------------------------------------------------------------------------
void __fastcall TServidor::Button2Click(TObject *Sender)
{
char *IP;
  if(Edit5->Visible==false)
  Edit5->Visible==true;
  IP=ipservidor();
  Edit7->Text=IP;
}
//---------------------------------------------------------------------------
void __fastcall TServidor::Button1Click(TObject *Sender)
{
char *IP;
  if(Edit5->Visible==false)
  Edit5->Visible==true;
  IP=ipcliente();
  Edit7->Text=IP;
}
```

-Código fuente de la aplicación Cliente:

```
//---------------------------------------------------------------------------
```

```
#include <vcl.h>
#pragma hdrstop

#include "Cliente.h"
#include "Libcio.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TCliente *Cliente;
int entrada=0,ser_cli=1;
char numip[20], buffer[100];
//---------------------------------------------------------------------------
__fastcall TCliente::TCliente(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button1Click(TObject *Sender)
{
  Edit1->Text=IntToStr(outcomp());
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button2Click(TObject *Sender)
{
  intcomp(StrToInt(Edit2->Text));
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button3Click(TObject *Sender)
{
  memset(numip,0,sizeof(numip));
  strcpy(numip,Edit3->Text.c_str());
  comunicar(ser_cli,numip);
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button4Click(TObject *Sender)
{
  char msj[100];
  memset(msj,0,sizeof(msj));
  strcpy(msj,Edit4->Text.c_str());
  enviar(msj);
}
//---------------------------------------------------------------------------
void __fastcall TCliente::CheckBox1Click(TObject *Sender)
{
```

```
  if(CheckBox1->Checked){
    ser_cli=0;
    CheckBox1->Caption="Servidor";
    Label1->Visible=false;
    Edit3->Visible=false;
  }
  else{
    ser_cli=1;
    CheckBox1->Caption="Cliente";
    Label1->Visible=true;
    Edit3->Visible=true;
  }
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button6Click(TObject *Sender)
{
  char *texto;
  texto=mensaje();
  Edit5->Text=texto;
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button5Click(TObject *Sender)
{
  char *texto;
  texto=recibir();
  Edit4->Text=texto;
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button7Click(TObject *Sender)
{
  char *IP;
  IP=ipservidor();
  Edit3->Text=IP;
}
//---------------------------------------------------------------------------
void __fastcall TCliente::Button8Click(TObject *Sender)
{
  char *IP;
  IP=ipcliente();
  Edit3->Text=IP;
}
//---------------------------------------------------------------------------
```