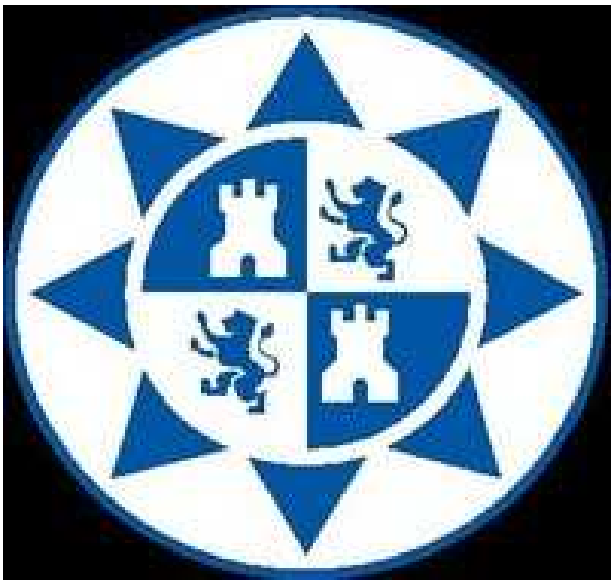
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Implementación windowsXp del protocolo de comunicaciones W2LAN



Profesor Francesc Burrull i Mestres

Alumno Fco Javier López Justo

INDICE:

1. INTRODUCION

El objetivo de este proyecto es la portabilidad del código del protocolo W2lan de un entorno Linux a un entorno Windows con entorno grafico. El protocolo ha sido diseñado previamente por el director del mismo. W2lan tiene como objetivo la distribución de contenidos multimedia en una red wireless del tipo Ad-hoc.

W2lan se basa en las ideas del protocolo MCDP-LAN para realizar una distribución del contenido multimedia pero en este caso consigue que se pueda implementar una red WLAN. Consiguiendo que las capas superiores sigan pensando que se encuentran en una LAN convencional.

W2lan esta diseñado en C, con entrada de parámetros mediante consola .El objetivo es realizar un programa donde quede resuelta la adaptación o conversión de la gran mayoría de funciones u operaciones que realiza W2lan, intentando mantener el mismo funcionamiento que el original. Además de añadir un interfaz de ventana para que los parámetros anteriormente introducidos por consola sean seleccionados por el usuario.

Para llevar a cabo el proyecto se ha utilizado en mismo lenguaje en el que esta escrito W2lan, y se ha apoyado en la api de Windows y Winpcap para conseguir el entorno grafico y buscar solución a funciones que no ha sido posible la conversión directa de Linux a Windows.

En la memoria se realizara una breve descripción de las tecnologías en las que se apoya para la adaptación a Windows y la descripción del programa W2lan resultante.

2. WINAPI

2.1. Introducion

WinAPI o Windows API es un conjunto de funciones (API) contenidas en bibliotecas que permiten que una aplicación se ejecute en el sistema operativo Microsoft Windows. Son diseñadas en C y C++.Sus versiones son Win16, *Win32*, Win32s, Win64.

Microsoft desarrolló un SDK (kit de desarrollo de software), que provee la documentación y las herramientas que permite a los desarrolladores crear software usando las API de Windows y otras tecnologías asociadas a este sistema operativo.

Las funciones WinAPI se pueden categorizar en:

-Servicios Base:

Provee acceso a los recursos fundamentales disponibles en sistemas Windows. Como son el sistema de archivos, dispositivos, procesos, acceso al registro de Windows, manejo de errores, etc. Estas funciones residen en los archivos kernel32.dll y advapi32.dll.

-Graphics Device Interface (interfaz gráfica):

Provee la funcionalidad para mostrar contenido gráfico a los monitores, impresoras y otros dispositivos de salida. Reside en el archivo gdi32.dll.

-Interfaz de usuario:

Provee la funcionalidad de crear y gestionar las ventanas y los controles más básicos como botones y barras de desplazamiento, entradas desde el mouse y el teclado, y otras funcionalidades asociadas a la parte GUI de Windows. Esta unidad funcional reside en user32.dll. Desde la versión Windows XP, los controles básicos residen en comctl32.dll, juntos con los controles comunes (Librería de Controles Comunes).

-Librería de cajas de diálogos comunes:

Provee a las aplicaciones las cajas de diálogo estándar para abrir y guardar ficheros, elegir colores y fuentes, etc. La librería reside en comdlg32.dll. Está agrupado junto a la categoría Interfaz de Usuario.

-Librería de controles comunes:

Da a las aplicaciones acceso a controles avanzados que provee el sistema operativo. Esto incluye la barra de estado, la barra de progreso, barra de herramientas, solapas, etc. La librería reside en el archivo en comctl32.dll. Es agrupado bajo la categoría Interfaz de Usuario.

-Shell de Windows:

Componente del API de Windows que permite a las aplicaciones acceder a la funcionalidad que provee el shell del sistema operativo. El componente reside en shlwapi.dll.

-Servicios de red:

Da acceso a varias capacidades de red del sistema operativo Windows. Sus subcomponentes incluyen NetBIOS, Winsock, NetDDE, RPC y muchos más.

Internet Explorer también ofrece muchos APIs que pueden ser usados por aplicaciones y pueden considerarse parte del API de Windows pues IE viene integrado con este sistema desde su versión 98.

Las API de Windows generalmente se concentran en la interacción entre el sistema operativo y una aplicación. Para la comunicación de diferentes aplicaciones entre ellas, Microsoft desarrolló una serie de tecnologías. Primero DDE, luego OLE, y más tarde COM.

La siguiente versión en la que se trabaja es WinFX, que está basada en nuevas tecnologías que se están probando en la versión de Windows llamada Vista. La interfaz gráfica de WinFX se llamaba Avalon y requiere tarjetas gráficas modernas pero luego Microsoft cambio el nombre de WinFX por el de .Net 3.0 que es el total de la suma de .Net 2.0 + Windows Presentation Fundation (Avalon) + Windows Comunication Fundation + Windows Cardspace + Windows Workflow Fundation.

También existe el proyecto WINE es un intento de que esta API esté disponible para plataformas tipo UNIX.

2.2 WinAPI en W2lan

Puesto que queremos realizar la conversión a Windows XP la WinAPI que utilizaremos en nuestro caso en la versión win32 .Los recursos que cogeremos de este lenguaje de programación serán para ayudarnos a diseñar el interfaz por el cual, el usuario acceda a las opciones del programa y vea su funcionamiento.

También utilizaremos alguna función de la categoría de servicios de red para adquirir información sobre nuestro dispositivo, ya que las funciones destinadas para ello en Linux no tienen su conversión en este sistema operativo.

Estas funciones se encuentran en Winsock2.dll archivo que Windows utiliza para todo lo relativo a sockets y manejo de interfaces .Y gracias a la estructura PPACKET_OID_DATA:

```
struct _PACKET_OID_DATA {

   ULONG Oid;                                    ///< OID code. See the Microsoft DDK documentation or
the file ntddndis.h
///< for a complete list of valid codes.

   ULONG Length;                            ///< Length of the data field

   UCHAR Data[1];                                ///< variable-lenght field that contains the information
passed to or received
///< from the adapter.

};
typedef struct _PACKET_OID_DATA PACKET_OID_DATA, *PPACKET_OID_DATA;
```

Y también la estructura LPADAPTER:

```
typedef struct _ADAPTER  {
        HANDLE hFile;                                 ///< \internal Handle to an open instance of the NPF
driver.

        CHAR  SymbolicLink[MAX_LINK_NAME_LENGTH]; ///< \internal A string containing the name of the
network adapter currently opened.

        int NumWrites;                                ///< \internal Number of times a packets written on this
adapter will be repeated
```

```
///< on the wire.

	HANDLE ReadEvent;				///< A notification event associated with the read calls on the adapter.
///< It can be passed to standard Win32 functions (like WaitForSingleObject
///< or WaitForMultipleObjects) to wait until the driver's buffer contains some
///< data. It is particularly useful in GUI applications that need to wait
///< concurrently on several events. In Windows NT/2000 the PacketSetMinToCopy()
///< function can be used to define the minimum amount of data in the kernel buffer
///< that will cause the event to be signalled.

	UINT ReadTimeOut;				///< \internal The amount of time after which a read on the driver will be released and
///< ReadEvent will be signaled, also if no packets were captured

	CHAR Name[ADAPTER_NAME_LENGTH];

	PWAN_ADAPTER pWanAdapter;

	UINT Flags;					///< Adapter's flags. Tell if this adapter must be treated in a different way, using the Netmon API or the dagc API.

#ifdef HAVE_AIRPCAP_API

	PAirpcapHandle	AirpcapAd;

#endif // HAVE_AIRPCAP_API

#ifdef HAVE_NPFIM_API

	void* NpfImHandle;

#endif // HAVE_NPFIM_API

#ifdef HAVE_DAG_API
	dagc_t *pDagCard;			///< Pointer to the dagc API adapter descriptor for this adapter

	PCHAR DagBuffer;				///< Pointer to the buffer with the packets that is received from the DAG card

	struct timeval DagReadTimeout;		///< Read timeout. The dagc API requires a timeval structure

	unsigned DagFcsLen;				///< Length of the frame check sequence attached to any packet by the card. Obtained from the registry

	DWORD DagFastProcess;			///< True if the user requests fast capture processing on this card. Higher level applications can use this value to provide a faster but possibly unprecise capture (for example, libpcap doesn't convert the timestamps).

#endif // HAVE_DAG_API

}  ADAPTER, *LPADAPTER
```

Podremos adquirir la Mac tal y como se puede apreciar en el código del programa.

A continuación se explicaran con profundidad el funcionamiento de las funciones utilizadas en el diseño de la interfaz grafica en W2lan.

2.3 GUI de W2lan

2.3.1 Programa principal

Para poder usar las funciones del API de Windows hay que incluir al menos un fichero de cabecera, este será windows.h que incluye a su vez la mayoría de los ficheros de cabecera corrientes que se utilizan.

La función principal es *WinMain*, en lugar de la que normalmente utilizamos en lo programas en C que es *main.*

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
  LPSTR lpszCmdParam, int nCmdShow)
```

La función *WinMain* tiene cuatro parámetros de entrada:

1. *hInstance* es el manipulador que Windows utilizara para la instancia del programa que estamos ejecutando.

2. *hPrevInstance* es un manipulador a instancias previas de la misma aplicación. Como Windows es multitarea, pueden existir varias versiones de la misma aplicación ejecutándose, varias instancias.La idea original era que este manipular nos ayudara a administrar los recursos ya que varias instancias utilizaban el mismo. Pero este segmento esta ya en desuso ya que en las nuevas versiones de winapi (Win32) se usa un segmento distinto para cada instancia , por lo tanto se mantiene por motivos de compatibilidad

3. *lpszCmdParam* es una cadena que contiene los argumentos de entrada del comando de línea.

4. *nCmdShow*, este parámetro especifica cómo se mostrará la ventana.

La estructura de esta función cambia muy poco de unas aplicaciones a otras. Siempre nos vamos a encontrar con algo parecido a esto:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
  LPSTR lpszCmdParam, int nCmdShow)
{
  /* Declaración: */
  HWND hwnd;
  MSG mensaje;
  WNDCLASSEX wincl;

  /* Inicialización: */
  /* Estructura de la ventana */
  wincl.hInstance = hInstance;
  wincl.lpszClassName = "NUESTRA_CLASE";
  wincl.lpfnWndProc = WindowProcedure;
  wincl.style = CS_DBLCLKS;
  wincl.cbSize = sizeof(WNDCLASSEX);

  /* Usar icono y puntero por defecto */
  wincl.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wincl.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
  wincl.hCursor = LoadCursor(NULL, IDC_ARROW);
  wincl.lpszMenuName = NULL;
  wincl.cbClsExtra = 0;
  wincl.cbWndExtra = 0;
  wincl.hbrBackground = (HBRUSH)COLOR_BACKGROUND;

  /* Registrar la clase de ventana, si falla, salir del programa */
  if(!RegisterClassEx(&wincl)) return 0;

  hwnd = CreateWindowEx(
        0,
        "NUESTRA_CLASE",
        "Ejemplo 001",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        544,
        375,
        HWND_DESKTOP,
        NULL,
        hThisInstance,
        NULL
  );

  ShowWindow(hwnd, SW_SHOWDEFAULT);

  /* Bucle de mensajes: */
  while(TRUE == GetMessage(&mensaje, 0, 0, 0))
  {
    TranslateMessage(&mensaje);
    DispatchMessage(&mensaje);
  }

  return mensaje.wParam;
}
```

Donde nos encontramos claramente dos partes, una donde inicializamos las variables necesarias para el funcionamiento de nuestra función y donde registramos la clase o clases de ventanas y otra que seria el un bucle que vendría la parte principal de la función ya que es la que atenderá los mensajes que vaya recibiendo el programa .

Para el registro de la clase de ventana debemos rellenar correctamente la estructura *WNDCLASSEX* que define las características que puede tener la ventana:

```
typedef struct _WNDCLASSEX {

   UINT    cbSize;
   UINT    style;
   WNDPROC lpfnWndProc;
   int     cbClsExtra;
   int     cbWndExtra;
   HANDLE  hInstance;
   HICON   hIcon;
   HCURSOR hCursor;
   HBRUSH  hbrBackground;
   LPCTSTR lpszMenuName;
   LPCTSTR lpszClassName;
   HICON   hIconSm;
} WNDCLASSEX;
```

Entre sus atributos podemos encontrar:

cbSize: Especifica el tamaño, en bytes, de esta estructura. Hay que asignar a este miembro el valor *sizeof (WINDOWCLASSEX)*.

Style: Es un entero de 16 bits que codifica el estilo de la clase de ventana. Estos bits se pueden combinar con la función OR, para obtener estilos con las características deseadas.

Lo valores posibles de Style serian:

Estilo	Acción
CS_BYTEALIGNCLIENT	Hace que el área de cliente coincida con el límite de un byte en la dirección de las x. Esto mejora las prestaciones a la hora de pintar en la pantalla. Este estilo afecta tanto al ancho de la ventana como a su posición en la pantalla.
CS_BYTEALIGNWINDOW	Lo mismo que el anterior, pero con el borde de la ventana, en lugar del área de cliente.

CS_CLASSDC	Crea un DC (Contexto de Dispositivo) que será compartido por todas las ventanas de la misma clase.
CS_DBLCLKS	Envía los mensajes de doble-clic al procedimiento de la ventana, cuando el usuario hace doble-clic sobre una ventana de esta clase.
CS_GLOBALCLASS	Permite a una aplicación crear una ventana de esta clase independientemente del valor de hInstance que se proporcione a CreateWindow o CreateWindowEx. Si no se especifica, el valor de hInstace debe ser el mismo que se uso para registrar la clase.
CS_HREDRAW	Redibuja toda la ventana cada vez que un movimiento o cambio de tamaño cambia la anchura del área de cliente.
CS_NOCLOSE	Deshabilita del comando de cerrar del menú del sistema.
CS_OWNDC	Crea un DC único para cada ventana de esta clase.
CS_PARENTDC	Asigna la región de recorte de la ventana hija a la de la ventana padre de modo que la hija puede pintar en la padre. Una ventana con el bit de estilo *CS_PARENTDC* recibe un contexto de dispositivo normal desde el caché de contextos de dispositivo del sistema. No proporciona a la ventana hija el contexto de dispositivo de la ventana padre o los valores del contexto de dispositivo. Especificar *CS_PARENTDC* mejora el comportamiento de la aplicación.
CS_SAVEBITS	Guarda como mapas de bits los trozos de pantalla tapados por la ventana. Windows usará estos mapas de bits para reconstruir la pantalla cuando la ventana se elimine.
CS_VREDRAW	Redibuja toda la ventana cada vez que un movimiento o cambio de tamaño cambia la altura del área de cliente.

lpfnWndProc: Apunta al procedimiento de ventana.

cbClsExtra: Especifica cuantos bytes extra se reservarán a continuación de la estructura de la clase. El sistema operativo inicializará estos bytes a cero.

cbWndExtra: Especifica cuantos bytes extra se reservarán a continuación de la instancia de la ventana. El sistema operativo inicializará estos bytes a cero.

hInstance: Identifica la instancia de la ventana a la que esta clase pertenece.

hIcon: Identificador del icono de la clase. Debe ser un manipulador de un recurso de tipo icono. Si es NULL, la aplicación debe mostrar un icono cuando el usuario minimice la ventana de la aplicación.

hCursor: Identificador del cursor de la clase. Debe ser un manipulador de recurso de tipo cursor. Si es NULL, la aplicación debe mostrar el cursor explícitamente cada vez que el usuario mueve el ratón sobre la ventana de la aplicación.

hbrBackground: Identificador del pincel para la clase. Puede ser un manipulador para un pincel físico que se usará para pintar el fondo de la ventana, o puede ser un valor de color. Si se trata de un valor de color debe ser uno de los colores estándar de

lpszMenuName: Puntero a una cadena terminada con cero que especifica un nombre de recurso de la clase menú, es el nombre con el que aparece en el fichero de recursos.

lpszClassName: Apunta a una cadena.

hIconSm: Manipulador a un icono pequeño que se asocia a la clase de ventana. Si este miembro es NULL, el sistema busca el recurso de icono especificado por el miembro *hIcon* un icono del tamaño adecuado para usarlo como icono pequeño.

Una vez rellenada la estructura adecuadamente llamaremos a la función *RegisterClassEx* para registrar la clase ventana con el fin de poder posteriormente usar dicha clase en la llamada a la función *CreateWindowEx* que utilizaremos inmediatamente después para crear la ventana.

Esta función es la que realmente crea la ventana, su síntesis seria:

```
HWND CreateWindowEx(
  DWORD dwExStyle,      // estilo extendido de ventana
  LPCTSTR lpClassName,  // puntero al nombre de la clase registrada
  LPCTSTR lpWindowName, // puntero al nombre de la ventana
  DWORD dwStyle,        // estilo de ventana
  int x,                // posición horizontal de la ventana
  int y,                // posición vertical de la ventana
  int nWidth,           // ancho de la ventana
  int nHeight,          // alto de la ventana
  HWND hWndParent,      // manejador de la ventana padre o propietaria
  HMENU hMenu,          // manejador de menu, o identificador de ventana hija
  HINSTANCE hInstance,  // manejador de la instancia de la aplicación
  LPVOID lpParam        // puntero a los datos de creación de la ventana
);
```

Con sus parámetros podemos especificar múltiples características de la ventana, posición, nombre…Al ser la ventana principal en *manejador de la ventana padre* pondremos *HWND_DESKTOP* que seria el manejador del escritorio por lo cual hereda de este. También en *estilo de la ventana* pondremos *WS_OVERLAPPEDWINDOW* que seria la ventana por defecto de Windows, ya que la clase es "inventada" por nosotros le llamamos "W2lan", no disponemos de estilos propios como si seria en caso de alguna ventana específica.

La función finalmente devuelve el manejador de la ventana que acabamos de crear con el podremos crear otras ventana y hacer que hereden de el, estas ventanas puedes ser controles como veremos mas adelante.

De estos parámetros algunos nos ofrecen varias opciones o interesa tenerlos más en cuenta:

dwExStyle: Especifica el estilo extendido de la ventana. Este parámetro puede ser uno de los siguientes valores:

Estilo	Significado
WS_EX_ACCEPTFILES	Especifica que una ventana creada con este estilo acepta arrastrar y soltar ficheros.
WS_EX_APPWINDOW	Fuerza una ventana de nivel superior sobre la barra de tareas cuando la ventana es visible.
WS_EX_CLIENTEDGE	Especifica que una ventana tiene un borde con un lado hundido.
WS_EX_CONTEXTHELP	Incluye un signo de pregunta en la barra de título de la ventana. Cuando el usuario hace un click en el signo, el cursor cambia a un signo de pregunta con un puntero. Si el usuario entonces hace un click en una ventana hija, ésta recibe un mensaje WM_HELP. La ventana hija debería pasara el mensaje al procedimiento de la ventana padre, el cual debería llamar a la función WinHelp usando el comando HELP_WM_HELP. La aplicación de Ayuda muestra una ventana pop-up que típicamente contiene ayuda para la ventana hija. WS_EX_CONTEXTHELP no puede ser usado con los estilos WS_MAXIMIZEBOX o WS_MINIMIZEBOX.
WS_EX_CONTROLPARENT	Permite al usuario navegar entre las ventanas hijas usando la tecla tab.
WS_EX_DLGMODALFRAME	Crea una ventana que tiene un borde doble; la ventana puede, opcionalmente, ser creada con una barra de titulo especificando el estilo WS_CAPTION en el parámetro dwStyle.
WS_EX_LEFT	La ventana tiene propiedades genéricas de alineamiento a la izquierda. Este es el valor por defecto.
WS_EX_LEFTSCROLLBAR	Si el lenguaje de la Shell es hebreo, Arábico u otro lenguaje que soporte alineamiento de orden de lectura, la barra de desplazamiento vertical (si está presente) está a la izquierda del área de cliente. Para otros lenguajes, el estilo es ignorado y no tratado como un error.
WS_EX_LTRREADING	El texto de la ventana es mostrado usando propiedades de orden de lectura de izquierda a derecha.
WS_EX_MDICHILD	Crea una ventana hija MDI.
WS_EX_NOPARENTNOTIFY	Especifica que una ventana hija creada con este estilo no envía un mensaje WM_PARENTNOTIFY a su ventana padre cuando es creada o destruida.
WS_EX_OVERLAPPEDWINDOW	Combina los estilos WS_EX_CLIENTEDGE y WS_EX_WINDOWEDGE.
WS_EX_PALETTEWINDOW	Combina los estilos WS_EX_WINDOWEDGE, WS_EX_TOOLWINDOW, y WS_EX_TOPMOST.
WS_EX_RIGHT	La ventana tiene propiedades genéricas de alineamiento a la derecha. Esto depende de la clase de ventana. Este estilo sólo tiene efecto si el lenguaje de la Shell es Hebreo, Arábico u otro lenguaje que soporte alineamiento de orden de lectura; de otra manera, el estilo es ignorado y no tratado como un error.

WS_EX_RIGHTSCROLLBAR	La barra de desplazamiento vertical (si está presente), se ubica a la derecha del área de cliente. Este es el valor por defecto.
WS_EX_RTLREADING	Si el lenguaje de la Shell es hebreo, Arábico u otro leguague que soporte alineamiento de orden de lectura, el texto de la ventana es mostrado usando propiedades de orden de lectura de Derecha a Izquierda. Para otros lenguajes el estilo es ignorado y no tratado como un error.
WS_EX_STATICEDGE	Crea una ventana con un estilo de borde tridimensional, hecho con la intención de ser usado con ítems que no aceptan entrada de datos del usuario.
WS_EX_TOOLWINDOW	Crea una ventana de herramienta, es decir, una ventana hecha con la intención de ser usada como una barra de herramientas flotante. Una ventana de herramienta tiene una barra de título más corta de lo normal, y la ventana es dibujada usando una fuente menor. Una ventana de herramienta no aparece en la barra de tareas o en el diálogo que aparace cuando el usuario presiona alt+tab. Si una ventana de herramienta posee un menú de sistema, su ícono no es mostrado en la barra de título. En todo caso, se puede mostrar el menú de sistema al hacer doble-click o presionando alt+tab.
WS_EX_TOPMOST	Especifica que una ventana creada con este estilo debería ser ubicada sobre todas las ventanas no-topmost y debería permanecer sobre ellas, incluso cuando la ventana es desactivada. Para añadir o quitar este estilo, use la función SetWindowPos.
WS_EX_TRANSPARENT	Especifica que la ventana creada con este estilo no debería ser pintada hasta que las hermanas debajo de ella (que fueron creadas por el mismo hilo) hayan sido pintadas. La ventana parece transparente porque los bits de las ventanas subyacentes ya han sido pintadas. Para obtener transparencia sin estas restricciones, use la función SetWindowRgn.
WS_EX_WINDOWEDGE	Especifica que la ventana tiene un borde con un lado alzado.

hWndParent: Identifica la ventana padre o propietaria de la ventana que es creada. Se debe suministrar un manejador de ventana válido cuando se crea una ventana hija o una ventana con dueño. Una ventana hija es confinada al área de cliente de su ventana padre. Una ventana con dueño es una ventana suerpuesta que es destruída cuando su ventana propietaria es destruida, o escondida cuando su propietaria es minimizada; siempre es mostrada encima de ésta.

hMenu: Para una ventana *hMenu* identifica el menú que se usará con la ventana; puede ser NULL si se usa el menú de la clase. Para una ventana hija, *hMenu* especifica el identificador de ventana hija, un valor entero que se usa por un control de cuadro de diálogo para informar a su ventana padre sobre eventos. Es la aplicación la que determina el identificador de la ventana hija; debe ser único para todas las ventanas hija con la misma ventana padre.

hInstance: identifica la instancia del módulo asociado con la ventana.

Finalmente se ha creado la ventana, pero cara al usuario sigue invisible, para que sea visible al usuario nos faltaría ejecutar la función *ShowWindow*:

```
BOOL ShowWindow(
  HWND hwnd, // manipulador de ventana
  int nCmdShow // modo de visualización de ventana
);
```

Entre sus parámetros deberemos especificarle el manipulador de la ventana que queremos que se vuelva visibles y el modo de visualización que por norma general siempre será SW_SHOWDEFAULT.

Llegamos a la segunda parte del programa que seria el bucle de mensajes este suele tener una síntesis parecida a esta:

```
while(TRUE == GetMessage(&mensaje, 0, 0, 0)) {
  TranslateMessage(&mensaje);
  DispatchMessage(&mensaje);
}
```

La misión del bucle es recoger los mensajes que son enviados a la ventana, traducirlos y enviarlos de nuevo a la pila de mensajes para que, ya traducidos, sean analizados también por el bucle.

Esta labor se consigue con la combinación de las tres funciones anteriormente expuestas. La función que esta en la sentencia del *while*, *GetMessage*, se encarga de recuperar los mensajes de la pila de mensajes de la aplicación y lo coloca en la estructura que le indicamos:

```
BOOL GetMessage(
  LPMSG lpMsg, // puntero a la estructura que contendrá el mensaje
  HWND hWnd, // manipulador de ventana
  UINT wMsgFilterMin, // primer mensaje
  UINT wMsgFilterMax // último mensaje
);
```

Los valores que devuelve esta función es *TRUE* si recibe un mensaje distinto de *WM_QUIT* y *FALSE* si recibe este mensaje. También puede tomar el valor de -1 si se ha producido algún error.

Una vez dentro del bucle el mensaje pasa por las funciones *TranslateMessage* y *DispatchMessage*.

```
BOOL TranslateMessage(
   CONST MSG *lpmsg  // puntero a la estructura con el mensaje
  );

LONG DispatchMessage(
  CONST MSG  *lpmsg       // puntero a una estructura con el mensaje
);
```

En primero se encarga de traducir en mensaje para que el procedimiento de ventana, que mas adelante explicaremos, pueda analizarlo, en mensaje una vez traducido vuelve a la pila de mensajes del proceso para que GetMessage lo vuelva a recuperar ya traducido .Una vez traducido DispatchMessage lo envía al procedimiento de ventana.

Para poder analizar los mensajes que recibe nuestra ventana es necesaria una función asociada a dicha ventana, esta es lo que llamamos el procedimiento de ventana y es la responsable de todo lo relativo al aspecto y al comportamiento de una ventana.

La función puede cambiar como el programador desee pero el valor de retorno y los parámetros de entrada deben ser los mismos

```
LRESULT CALLBACK WindowProcedure(
  HWND hwnd,     // Manipulador de ventana
  UINT msg,      // Código del mensaje
  WPARAM wParam, // Parámetro palabra, varía
  LPARAM lParam  // Parámetro doble palabra, varía
  );
```

Como hemos visto antes, para relacionar la ventana a su procedimiento lo conseguiremos con la asignación del puntero *lpfnWndProc,* que encontramos en la estructura WNDCLASSEX.

Normalmente, estas funciones están basadas en una estructura *switch* donde cada *case* corresponde aun determinado tipo de mensaje. Un ejemplo de procedimiento de ventana sencillo seria:

```
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
   switch (msg)              /* manipulador del mensaje */
   {
      case WM_DESTROY:
        PostQuitMessage(0);    /* envía un mensaje WM_QUIT a la cola de mensajes */
```

```
        break;
      default:              /* para los mensajes de los que no nos ocupamos */
        return DefWindowProc(hwnd, msg, wParam, lParam);
  }
  return 0;
}
```

En general, habrá tantos procedimientos de ventana como programas diferentes y todos serán distintos, pero también tendrán algo en común: todos ellos procesarán los mensajes que lleguen a una clase de ventana.

En este ejemplo sólo procesamos un tipo de mensaje, se trata de *WM_DESTROY* que es el mensaje que se envía a una ventana cuando se recibe un comando de cerrar, ya sea por menú o mediante el icono de aspa en la esquina superior derecha de la ventana.

Este mensaje sólo sirve para informar a la aplicación de que el usuario tiene la intención de abandonar la aplicación, y le da una oportunidad de dejar las cosas en su sitio: cerrar ficheros, liberar memoria, guardar variables, etc. Incluso, la aplicación puede decidir que aún no es el momento adecuado para abandonar la aplicación. En el caso del ejemplo, efectivamente cierra la aplicación, y lo hace enviándole un mensaje *WM_QUIT*, mediante la función *PostQuitMessage*.

El resto de los mensajes, si nosotros no los queremos procesarlos , lo hará en el caso "default", que simplemente se cede su tratamiento a la función de *WinApi* que hace el proceso por defecto para cada mensaje, *DefWindowProc*.

Este es el camino que sigue el mensaje *WM_QUIT* cuando llega, ya que el proceso por defecto para este mensaje es cerrar la aplicación.

Ejemplos de otros mensajes interesantes serian:

***WM_INITDIALOG**:* se envía al procedimiento de diálogo inmediatamente antes de que la ventana sea mostrado. El procedimiento de diálogo normalmente usa este mensaje para inicializar los controles y cumplimentar cualquier trabajo de inicialización que afecte a la apariencia de la ventana.

***WM_CREATE**:* se envía cuando una aplicación pide que sea creada una ventana mediante una llamada a la función CreateWindowEx. El procedimiento de ventana de la nueva ventana recibe este mensaje después de que la ventana ha sido creada, pero antes de que sea visible. El mensaje es enviado antes de que la función CreateWindowEx retorne.

***WM_PAINT**:* cuando Windows u otra aplicación hace una petición para pintar una porción de la ventana de la aplicación. El mensaje es enviado cuando las funciones

UpdateWindow o RedrawWindow son llamadas o por la función DispatchMessage cuando la aplicación obtiene un mensaje *WM_PAINT* tras el uso de las funciones GetMessage.

WM_COMMAND: es enviado cuando el usuario selecciona un comando de un ítem de un menú, cuando un control envía un mensaje de notificación a su ventana padre, o cuando una pulsación de un acelerador es traducida.

WM_MENUSELECT: se envía a la ventana dueña de un menú cuando el usuario selecciona un ítem del menú

WM_ACTIVATE: se envía cuando una ventana va a ser activada o desactivada. Este mensaje se envía primero al procedimiento de ventana de la ventana de mayor nivel que se va a desactivar; después es enviado al procedimiento de ventana de la ventana de mayor nivel que se va a activar.

WM_MOUSEMOVE: se envía a una ventana cuando el cursor se mueve. Si el ratón no ha sido capturado, el mensaje se envía a la ventana que contiene el cursor. En otro caso, el mensaje se envía a la ventana que haya capturado el ratón.

2.3.2 Ficheros de recursos

Para la formación de la ventana recurriremos a unos ficheros de recursos que nos ayudaran tanto a una forma de creación de menús para la ventana como a la asignación de identificadores para los comandos de los elementos o controles que crearemos dentro de la ventana como pueden ser listbox, combobox etc… los cuales enviaran mensajes de eventos específicos al procedimiento de ventana que podremos identificar y actuar en consecuencia.

El fichero de identificadores tendrá un aspecto parecido a este, en nuestro ejemplo llamaremos "ids.h":

```
#define CM_PRUEBA 100

#define CM_SALIR 101

#define ID_TEXTO 102
```

Este fichero deberá estar incluido tanto en el archivo principal como en el que utilizaremos para la creación del menú.

El fichero que nos servirá como ayuda para crear nuestro menú tendrá el mismo nombre que nuestro programa principal pero terminado en ".rc".

```
#include "ids.h"

Menu MENU
BEGIN
  POPUP "&Principal"
    BEGIN
      MENUITEM "&Prueba", CM_PRUEBA
      MENUITEM SEPARATOR
      MENUITEM "&Salir", CM_SALIR
    END
END
```

Este ejemplo crea una barra de menú con una columna "Principal", con dos opciones: "Prueba" y "Salir", y con un separador entre ellas.

La sintaxis es sencilla, definimos el menú mediante una cadena identificadora, sin comillas, seguida de la palabra *MENU*. Entre las palabras *BEGIN* y *END* podemos incluir items, separadores u otras columnas. Para incluir columnas usamos una sentencia del tipo *POPUP* seguida de la cadena que se mostrará como texto en el menú. Cada *POPUP* se comporta del mismo modo que un *MENU*.

Los ítems se crean usado la palabra *MENUITEM* seguida de la cadena que se mostrará en el menú, una coma, y el comando asignado a ese ítem, que puede ser un número entero, o, como en este caso, una macro definida. Los separadores se crean usando *MENUITEM* seguido de la palabra *SEPARATOR*.

Las cadenas que se muestran en el menú contienen un símbolo & en su interior, por ejemplo "&Prueba". Este símbolo indica que la siguiente letra puede usarse para activar la opción del menú desde el teclado, usando la tecla [ALT] más la letra que sigue al símbolo &. Para indicar eso, en pantalla, esa letra se muestra subrayada, en este ejemplo "Prueba".

Para asignar el menú, lo conseguiremos con una de las propiedades de *WNDCLASSEX*:

```
wincl.lpszMenuName = "Menu";
```

2.3.3 Controles o elementos de la GUI

En realidad, un control o elemento no es otra cosa que una ventana. Como ventana que es, tiene su propio procedimiento de ventana, y por supuesto, se pueden creado usando *CreateWindowsEx*.

El mensaje *WM_CREATE* nos permite realizar acciones cuando se crea la ventana principal, que es a la que pertenece el procedimiento de ventana.De esta forma podemos insertar controles en dicha ventana mediante *CreateWindowsEx* un ejemplo de seria:

```
HWND hctrl;
...
      case WM_CREATE:
        hInstance = ((LPCREATESTRUCT)lParam)->hInstance;

        /* Insertar control Edit */
        hctrl = CreateWindowEx(
          0,
          "EDIT",         /* Nombre de la clase */
          "",             /* Texto del título, no tiene */
          ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, /* Estilo */
          36, 20,         /* Posición */
          120, 20,        /* Tamaño */
          hwnd,           /* Ventana padre */
          (HMENU)ID_TEXTO, /* Identificador del control */
          hInstance,      /* Instancia */
          NULL);          /* Sin datos de creación de ventana */

        /* Inicialización de los datos de la aplicación */
        SetDlgItemText(hwnd, ID_TEXTO, "Inicial");
        SetFocus(hctrl);
        return 0;
```

Como vemos, usamos la misma sentencia que al crear la ventana padre pero en este caso en el parámetro de *nombre de la clase* utilizaremos "EDIT" que es el control que deseamos crear y en la ventana padre pondremos la ventana padre que hemos creado anteriormente y que es la que va a contener el control que estamos formando. También introduciremos los demás parámetros, estilo, posición, dimensiones...

El identificador del control se suministra a través del parámetro *hMenu*, por lo que será necesario hacer una conversión del identificador al tipo *HMENU*.

Ahora será nuestro procedimiento de ventana el encargado de procesar los mensajes o notificaciones de eventos procedentes del control o controles que hemos creado. Los eventos o comandos de los diferentes controles les llegaran mediante el mensaje *WM_COMMAND*. En la palabra de menor peso del parámetro wParam se envía el identificador del control. El manipulador del control se envía en el parámetro lParam y el código del mensaje de notificación en la palabra de mayor peso de wParam.

Una vez creado nos encontramos con múltiples comandos y funciones que nos ayudan a interactuar con el elemento pudiendo cambiar el estilo de la letra, propiedades durante la ejecución etc… Una función que nos será de gran ayuda para llevar a cabo esto es *SendMessage* que envía el mensaje que le indicamos a la pila de mensajes con la información que necesite para su funcionamiento.

```
LRESULT SendMessage(
   HWND hwnd,     // manipulador de la ventana de destino
   UINT uMsg,     // mensaje a enviar
   WPARAM wParam, // primer parámetro del mensaje
   LPARAM lParam  // segundo parámetro del mensaje
  );
```

Los controles poseen muchos tipos de mensajes para poder cambiar propiedades o visionar su estado durante la ejecución del programa mas adelante veremos algunos específicos de cada control, entre los cuales podremos cambiar la fuente de la letra , ver que elemento de la lista de un control esta seleccionado, seleccionar otro, etc…

Otra función que realiza lo mismo que la anterior pero con la que podemos especificar el control o elemento al que mandamos el mensaje es:

```
LONG SendDlgItemMessage(
```

```
  HWND hwndDlg,  // manipulador del cuadro de diálogo
  int idControl, // identificador del control
  UINT uMsg,     // mensaje a enviar
  WPARAM wParam, // primer parámetro del mensaje
  LPARAM lParam  // segundo parámetro del mensaje
 );
```

Con este tipo de función nos facilita el enviar un mensaje a un control especifico indicándole el manipulador e identificador. Nos es de utilidad por ejemplo cuando nos encontramos con que tenemos varios controles iguales dentro de la ventana principal. Windows cuando recibe un mensaje que no es para la ventana principal lo envía automáticamente al procedimiento de ventana del control al que se refiere, por ello si nos encontramos con dos controles del mismo tipo debemos diferenciarlos en el momento de mandar el mensaje, si conocemos el manipulador como es nuestro caso podríamos prescindir de esta función a la hora de enviar mensajes.

La función anterior en la abreviatura de utilizar:

```
SendMessage(GetDlgItem(hwnd,105), LB_ADDSTRING, 0, (LPARAM)tiempo);
```

Donde mediante la función *GetDlgItem* conseguimos el manipulador del objeto pasándoles el manipulador de la ventana padre y el identificador del control.

```
HWND GetDlgItem(
  HWND hDlg,      // manipulador del cuadro de diálogo
  int nIDDlgItem  // identificador del control
 );
```

Existen varios tipos de elementos o controles, pero en nuestro caso utilizaremos solo 3:

1. **Botones**
2. **Listbox**
3. **Combobox**

2.3.3.1 Botones

El botón es un control que nos sirve para que el usuario pueda ejecutar o activar opciones del programa, es un elemento muy recurrido y útil en todas las aplicaciones de ventana.

Los crearnos de la forma que hemos explicado especificando en el nombre de clase "BUTTON" .Existen varios estilos de botones: botones de pulsar (push buttons), botones de grupo, check boxes y radio buttons.

Para especificar el estilo de botón que queremos y otras propiedades utilizaremos el parámetro de estilos *dwStyle* de la función *CreateWindowsEx*, esos estilos pueden ser:

Estilo	Significado
BS_3STATE	Crea un button que es lo mismo que un check box, salvo que puede ponerse gris (grayed) además de ser marcado (checked) o desmarcado (unchecked). El estado gris se usa para mostrar que el check box está indeterminado.
BS_AUTO3STATE	Crea un button que es igual que el check box de tres estados, salvo que cambia de estado cuando el usuario lo selecciona. El estado cambia alternativamente entre checked, grayed y unchecked.
BS_AUTOCHECKBOX	Crea un button que es igual que un check box, pero que cuyo estado de selección oscila automáticamente entre checked y unchecked cada vez que el usuario selecciona el check box.
BS_AUTORADIOBUTTON	Crea un button que es igual que un radio button, pero que cuando es seleccionado por el usuario, Windows cambia su estado automáticamente a seleccionado y automáticamente deselecciona el resto de los radio buttons del mismo grupo.
BS_CHECKBOX	Crea un pequeño check box vacío con texto. Por defecto, el texto se muestra a la derecha del check box. Para mostrar el texto a la izquierda, hay que combinar esta bandera con el estilo BS_LEFTTEXT (o con su equivalente BS_RIGHTBUTTON).
BS_DEFPUSHBUTTON	Crea un botón normal que se comporta como uno del estilo BS_PUSHBUTTON, pero también tiene un borde negro y grueso. Si el botón está en un cuadro de diálogo, el usuario puede pulsar este botón usando la tecla ENTER, aún cuando el botón no tenga el foco de entrada. Este estilo es corriente para permitir al usuario seleccionar rápidamente la opción más frecuente, la opción por defecto.
BS_GROUPBOX	Crea un rectángulo en cuyo interior se pueden agrupar otros controles. Cualquier texto asociado con este estilo se mostrará en la esquina superior izquierda del rectángulo.
BS_LEFTTEXT	Coloca un texto a la izquierda de un radio button o check box cuando se combina con los estilos radio button o check box. Lo mismo que el estilo BS_RIGHTBUTTON.
BS_OWNERDRAW	Crea un botón owner-drawn. La ventana propietaria recibirá un mensaje WM_MEASUREITEM cuando el botón sea creado y un mensaje WM_DRAWITEM cuando algún aspecto visual del botón haya cambiado. No debe combinarse el estilo BS_OWNERDRAW con cualquier otro estilo de botón.

BS_PUSHBUTTON	Crea un botón corriente que envía un mensaje WM_COMMAND a su ventana padre cuando el usuario selecciona el botón.
BS_RADIOBUTTON	Crea un pequeño círculo con texto. Por defecto, el texto se muestra a la derecha del círculo. Para mostrar el texto a la izquierda, hay que combinar esta bandera con el estilo BS_LEFTTEXT (o con su equivalente BS_RIGHTBUTTON). Se usan para grupos de opciones relacionadas pero mutuamente exclusivas.
BS_USERBUTTON Obsoleto	Obsoleto, pero se mantiene por compatibilidad con versiones de Windows de 16 bits. Las aplicaciones basadas en Win32 deben usar en su lugar BS_OWNERDRAW.
BS_BITMAP	Indica que el botón muestra un mapa de bits.
BS_BOTTOM	Muestra el texto en la parte de abajo del área del botón.
BS_CENTER	Centra el texto horizontalmente en el área del botón.
BS_ICON	Indica que el botón muestra un icono.
BS_LEFT	Justifica a la izquierda el texto del botón. Sin embargo, si el botón es un check box o un radio button que no tiene el estilo BS_RIGHTBUTTON, el texto será justificado a la izquierda, pero al lado derecho del check box o radio button.
BS_MULTILINE	Divide el texto del botón en varias líneas si es demasiado largo para que quepa en una sola línea en el área del botón.
BS_NOTIFY	Permite al botón enviar mensaje de notificación BN_DBLCLK, BN_KILLFOCUS y BN_SETFOCUS a su ventana padre. Observa que los botones envían el mensaje BN_CLICKED a pesar de poseer este estilo.
BS_PUSHLIKE	Hace un botón (como un check box, three-state check box o radio button) que se comporta y tiene el mismo aspecto que un botón normal. El botón se muestra levantado cuando no está pulsado o unchecked y hundido cuando esté pulsado o checked.
BS_RIGHT	Justifica a la derecha el texto del botón. Sin embargo, si el botón es un check box o un radio button que no tiene el estilo BS_RIGHTBUTTON, el texto será justificado a la derecha, pero al lado derecho del check box o radio button.
BS_RIGHTBUTTON	Coloca el círculo del radio button o el cuadrado del check box al lado derecho del área del botón. Lo mismo que el estilo BS_LEFTTEXT.
BS_TEXT	Indica que el botón muestra un texto.
BS_TOP	Muestra el texto en la parte de arriba del área del botón.
BS_VCENTER	Muestra el texto en el centro, verticalmente, del área del botón.

Existen mensajes de eventos propios del control botón que nos podría interesar capturar en el proceso de ventana como serian:

Notificación	Evento
BN_CLICKED	Cada vez que el usuario hace clic sobre un botón
BN_DBLCLK	Cuando el usuario hace un doble clic sobre un botón. El botón debe tener el estilo BS_OWNERDRAW o BS_RADIOBUTTON.
BN_KILLFOCUS	Cuando un control pierde el foco del teclado
BN_SETFOCUS	Cuando un control gana el foco del teclado

También nos encontramos con mensajes que podemos mandar con la función *SendMessage o SendDlgItemMessage* para poder interactuar con el control*:*

Efecto	Codigo
Conseguimos cambiar el tipo de letra. Debemos ayudarnos de una variable hfont	static HFONT hfont; ... hfont = (HFONT)GetStockObject(DEFAULT_GUI_FONT); SendMessage(hctrl, WM_SETFONT, (WPARAM)hfont, MAKELPARAM(TRUE, 0));
Modificamos el estilo del botón durante la ejecución	SendDlgItemMessage(hwnd, ID_BOTON, BM_SETSTYLE, BS_PUSHBUTTON | BS_LEFTTEXT | WS_CHILD | WS_VISIBLE | WS_TABSTOP, MAKELPARAM(TRUE,0));
Determina el estado de marcado de un botón check box o radio button.	SendDlgItemMessage(hwnd, ID_BOTON2, BM_GETSTATE, 0, 0)

2.3.3.2 Listbox

Listbox consiste en una ventana rectangular con una lista de cadenas entre las cuales el usuario puede escoger una o varias según como hayamos configurado el *listbox* en el momento de su creación. Para nuestro caso lo utilizaremos únicamente para mostrar la información de lo que sucede en W2lan durante su ejecución por ello no capturaremos el mensaje que se envía cuando se selecciona una cadena ni utilizaremos muchas otras de sus características.

Para crear un control del tipo *listbox* nuevamente recurriremos a la función *CreateWindowsEx* utilizada de la forma que hemos comentado anteriormente y en nombre de clase tendremos que introducir "LISTBOX".

También como en el caso anterior nos encontramos con que *listbox* puede tener varios estilos u opciones en el momento de su creación:

Estilo	Significado
LBS_DISABLENOSCROLL	Muestra una barra de scroll vertical deshabilitada para el list box cuando no contiene suficientes elementos como para desplazarlos. Si no se especifica este estilo, la barra de scroll se oculta cuando el list box no contiene suficientes elementos.
LBS_EXTENDEDSEL	Permite que varios elementos puedan ser seleccionados usando la tecla SHIFT y el ratón o combinaciones especiales de teclas.
LBS_HASSTRINGS	Indica que el list box contiene elementos que son cadenas. El list box mantiene la memoria y las direcciones de las cadenas, así que la aplicación puede usar el mensaje LB_GETTEXT para recuperar el texto de un elemento en particular. Por defecto, todas los list boxes, menos los owner-drawn, tienen este estilo. Se puede crear un list box de tipo owner-drawn tanto con o sin este estilo.
LBS_MULTICOLUMN	Indica un list box multicolumna que es desplazado horizontalmente. El mensaje LB_SETCOLUMNWIDTH ajusta en ancho de las columnas.
LBS_MULTIPLESEL	Cambia la selección de cada cadena, cada vez que el usuario hace click o doble click sobre una cadena del list box. El usuario puede seleccionar cualquier número de elementos.
LBS_NOINTEGRALHEIGHT	Especifica que el tamaño del list box es exactamente el indicado por la aplicación cuando se creó el list box. Normalmente, Windows cambia el tamaño del list box para que no se muestren trozos de líneas.
LBS_NOREDRAW	Indica que la apariencia del list box no se actualiza cuando se hagan cambios. Se puede cambiar este estilo en cualquier momento enviando un mensaje WM_SETREDRAW.
LBS_NOSEL	Especifica qu el list box contiene ítems que pueden ser vistos, pero no seleccionados.

LBS_NOTIFY	Informa a la ventana padre con un mensaje de entrada cada vez que el usuario hace click o doble click sobre una cadena del list box.
LBS_OWNERDRAWFIXED	Especifica que el padre del list box es el responsable de actualizar su contenido en pantalla y de que los elementos del list box sean todos de la misma altura. La ventana padre recibirá el mensaje WM_MEASUREITEM cuando el list box sea creado y el mensaje WM_DRAWITEM cuando algún aspecto visual del list box haya cambiado.
LBS_OWNERDRAWVARIABLE	Especifica que el propietario del list box es el responsable de dibujar su contenido y que los elementos del list box tienen altura variable. La ventana padre recibirá un mensaje WM_MEASUREITEM para cada elemento del list box cuando el list box sea creado; y un mensaje WM_DRAWITEM cuando algún aspecto visual del list box haya cambiado.
LBS_SORT	Ordena alfabéticamente las cadenas en el list box.
LBS_STANDARD	Ordena alfabéticamente las cadenas en el list box. El list box tiene bordes en todos sus lados.
LBS_USETABSTOPS	Permite al list box reconocer y expandir los caracteres tab cuando muestra cadenas.
LBS_WANTKEYBOARDINPUT	Indica que la ventana padre del list box recibe mensajes WM_VKEYTOITEM cada vez que el usuario presiona una tecla y el list box tiene el foco de entrada. Esto permite a la aplicación realizar un procesamiento especial del teclado

Algunos de los mensajes que *listbox* envía automáticamente al procedimiento ventana principal, a través de *WM_COMMAND*, al realizarse alguna acción, y que nos podría interesar capturar para realizar alguna operación serian:

Notificación	Evento
LBN_DBLCLK	Cada vez que el usuario hace doble clic sobre uno de los ítems de un list box.
LBN_ERRSPACE.	Si no es posible conseguir memoria para completar una operación sobre el list box.
LBN_KILLFOCUS LBN_SETFOCUS	Cuando un control pierde el foco del teclado y cuando lo gana se generan estos mensajes respectivamente.
LBN_SELCHANGE.	Cada vez que la selección de un list box se modifica.
LBN_SELCANCEL.	Cuando el usuario cancela la selección de un ítem.

Y entre los posibles mensajes que podríamos mandar a este control nos encontramos entre los más interesantes:

Efecto	Codigo
Añade cadenas a un listbox, la dirección de la cadena se envía en el parámetro lParam.	sprintf(cad, "NUEVA CADENA"); SendMessage(hctrl, **LB_ADDSTRING**, 0, (LPARAM)cad);
Selecciona una cadena determinada. En el parámetro wParam se envía el índice en que debe comenzar la búsqueda y en lParam la dirección de la cadena a buscar	/* Seleccionar primera cadena que empiece por "E", después del 6º ítem */ int i=6; ... SendMessage(hctrl, **LB_SELECTSTRING**, (WPARAM)i, (LPARAM)"E");
Obtiene el índice del ítem actualmente seleccionado.	int i; ... i = SendMessage(hctrl, **LB_GETCURSEL**, 0, 0);

Obtenemos la longitud de la cadena de la posición que indicamos	int i=1;
	int l;
	l = SendMessage(hctrl, **LB_GETTEXTLEN**, (WPARAM)i, 0);

Recuperamos la cadena que se encuentra en la posición que le pasamos con i y lo guarda en la variable cad;	Char *cad ; Int i=1; SendMessage(hctrl, **LB_GETTEXT**, (WPARAM)i, (LPARAM)cad);
Eliminamos la cadena de la posición indicada	SendMessage(hctrl, **LB_DELETESTRING**, (WPARAM)i, 0);
Nos permite insertar un ítem en una posición determinada por el valor del parámetro wParam, y con el texto indicado en lParam	SendMessage(hctrl, LB_INSERTSTRING, (WPARAM)i, (LPARAM)cad);
Vacía el listbox	SendMessage(hctrl, LB_RESETCONTENT, 0, 0);
Selecciona el elemento en la posición que le indicamos en wParam	SendMessage(hctrl, LB_SETCURSEL, (WPARAM)i, 0);
Estas sentencias buscan en el listbox una cadena especifica, la diferencia es que la primera busca una cadena que coincida con el prefijo que viene en lParam y la segunda tiene que coincidir exactamente. En ambos casos nos devuelve la posición.	i = SendMessage(hctrl, LB_FINDSTRING, (WPARAM)i, (LPARAM)"CO"); i = SendMessage(hctrl, LB_FINDSTRINGEXACT, (WPARAM)-1, (LPARAM)"Portugal");
Nos sirve para obtener el número de ítems seleccionados actualmente en un list box.	nSeleccionados = SendMessage(hlista, LB_GETSELCOUNT, 0, 0);

2.3.3.3 Combobox

Los controles *combobox* consisten en un campo de selección, similar a un control *listbox* .El *combobox* puede ser mostrado todo el tiempo o puede desplegarse cuando el usuario selecciona el "pop box" que hay junto al campo de selección.

Dependiendo del estilo que seleccionemos del combobox, se podrá o no editar el contenido del campo de selección. Si el *listbox* es visible, escribir caracteres en el cuadro de selección hará que la primera entrada del *listbox* que coincida con el texto introducido sea seleccionada. Y la inversa, seleccionando un ítem de la lista se muestra el texto en el campo de selección. Estos y otros estilos se consiguen asignado el siguiente nombre según lo deseado:

Estilo	Significado
CBS_AUTOHSCROLL	Desplaza automáticamente a la derecha el texto en un control edit cuando el usuario escribe al final de la línea. Si este estilo no se selecciona, sólo puede introducirse el texto que cabe en los límites del cuadro de edición.
CBS_DISABLENOSCROLL	Muestra la barra de scroll vertical deshabilitada en el list box cuando no contiene suficientes elementos para desplazarlos. Sin este estilo la barra de scroll se oculta si no hay suficientes elementos en la lista.
CBS_DROPDOWN	Igual que CBS_SIMPLE, salvo que no se muestra el list box si el usuario no selecciona el icono cercano al control edit.
CBS_DROPDOWNLIST	Igual que CBS_DROPDOWN, salvo que el control edit se sustituye por un static text que muestra la selección actual del list box.
CBS_HASSTRINGS	Especifica que un combo box owner-drawn (actualizado por la ventana padre) contiene elementos que son cadenas. El combo box mantiene la memoria y las direcciones de las cadenas, así que la aplicación puede usar el mensaje CB_GETLBTEXT para recuperar el texto de un elemento en particular.
CBS_LOWERCASE	Convierte los caracteres introducidos en el control edit de un combo box a minúsculas.
CBS_NOINTEGRALHEIGHT	Especifica que el tamaño del combo box es exactamente el indicado por la aplicación cuando se creó el combo box. Normalmente, Windows cambia el tamaño del combo box para que no se muestren trozos de líneas.
CBS_OEMCONVERT	Convierte en texto introducido en control de edición del combo box. El texto se convierte de juego de caracteres de Windows a de OEM y después vuelve al juego de Windows. Esto asegura una correcta conversión cuando la aplicación llame a la función CharToOem para convertir una cadena Windows del combo box a una cadena OEM. Este estilo es comúnmente usado en combo boxes que contienen nombres de fichero y se aplica sólo a combo boxes creados con el estilo CBS_SIMPLE o CBS_DROPDOWN.

CBS_OWNERDRAWFIXED	Especifica que el padre del list box es el responsable de actualizar su contenido en pantalla y de que los elementos del list box sean todos de la misma altura. La ventana padre recibirá el mensaje WM_MEASUREITEM cuando el combo box sea creado y el mensaje WM_DRAWITEM cuando algún aspecto visual del combo box haya cambiado.
CBS_OWNERDRAWVARIABLE	Especifica que el propietario del combo box es el responsable de dibujar su contenido y que los elementos del list box tienen altura variable. La ventana padre recibirá un mensaje WM_MEASUREITEM para cada elemento del combo box cuando el combo box sea creado; y un mensaje WM_DRAWITEM cuando algún aspecto visual del combo box haya cambiado.
CBS_SIMPLE	Muestra el list box todo el tiempo. La selección actual del list box se muestra en el control edit.
CBS_SORT	Ordena automáticamente las cadenas introducidas en el list box.
CBS_UPPERCASE	Convierte los caracteres introducidos en el control edit de un combo box a mayúsculas.

Entre los mensajes de notificación ante un evento ocurrido en este nos encontramos con:

Notificación	Evento
CBN_DROPDOWN	Se envía cada vez que la lista de un combobox es desplegada
CBN_CLOSEUP	Se envía cada vez que la lista de un combobox vuelte a plegarse
CBN_DBLCLK	Al hacer doble click
CBN_EDITUPDATE.	Cada vez que el usuario modifica el texto de la parte del control de edición, y antes de que este nuevo texto se muestre en pantalla se envía este mensaje.

Los mensajes para el combobox mas comunes serian:

Efecto	Codigo
Añade cadenas a un combobox, la dirección de la cadena se envía en el parámetro lParam.	sprintf(cad, "NUEVA CADENA"); SendMessage(hctrl, **CB_ADDSTRING**, 0, (LPARAM)cad);
Inserta cadenas en la lista en una	/* Insertar un ítem antes del seleccionado actualmente */

posición determinada, independientemetne de que el combo box tenga el estilo CBS_SORT, la cadena siempre se insertará en la posición especificada por el parámetro wParam. La cadena se indica en el parámetro lParam	strcpy(cad, "CADENA INSERTADA"); SendMessage(hctrl,**CB_INSERTSTRING**, (WPARAM)3, (LPARAM)cad);
Obtiene el número de elementos que contiene la lista de un combo box	/* Obtener número de ítems */ int i; ... i = SendMessage(hctrl, **CB_GETCOUNT**, 0, 0);
Obtiene el índice del ítem actualmente seleccionado.	int i; ... i = SendMessage(hctrl, **CB_GETCURSEL**, 0, 0);

Obtenemos la longitud de la cadena de la posición que indicamos	int i=1;
	int l;
	
	l = SendMessage(hctrl, **CB_GETLBTEXTLEN**, (WPARAM)i, 0);

Recuperamos la cadena que se encuentra en la posición que le pasamos con i y lo guarda en la variable cad;	Char *cad ; Int i=1; SendMessage(hctrl, **CB_GETLBTEXT**, (WPARAM)i, (LPARAM)cad);
Eliminamos la cadena de la posición indicada	SendMessage(hctrl, **CB_DELETESTRING**, (WPARAM)i, 0);
Selecciona una cadena determinada. En el parámetro wParam se envía el índice en que debe comenzar la búsqueda y en lParam la dirección de la cadena a buscar.	/* Seleccionar primera cadena que empiece por "E", después del 6º ítem */ int i=6; ... SendMessage(hctrl, **CB_SELECTSTRING**, (WPARAM)i, (LPARAM)"E");
Vacía el listbox	SendMessage(hctrl, **CB_RESETCONTENT**, 0, 0);
Selecciona el elemento en la posición que le indicamos en wParam	SendMessage(hctrl, **CB_SETCURSEL**, (WPARAM)i, 0);

Estas sentencias buscan en el listbox una cadena especifica, la diferencia es que la primera busca una cadena que coincida con el prefijo que viene en lParam y la segunda tiene que coincidir exactamente. En ambos casos nos devuelve la posición.	i = SendMessage(hctrl, **CB_FINDSTRING**, (WPARAM)i, (LPARAM)"CO");
	
	i = SendMessage(hctrl, **CB_FINDSTRINGEXACT**, (WPARAM)-1, (LPARAM)"Portugal");

3 .DESCRIPCION DE PCAP Y WINPCAP

Pcap es un Api open source que ofrece al diseñador un interfaz que le permite desde capturar paquetes a enviarlos a través de una red.

Esta escrita originalmente en C y en un entorno Unix .Aunque rápidamente ha sido llevada a otros sistemas operativos .Debido a su gran potencial y que es portable a un gran numero de sistemas operativos, esta siendo muy bien aceptada dentro del mundo de la programción.

Uno de los sistemas operativos a los cuales ha sido portada, como no, es Windows y gracias a esto el desarrollo de nuestra tarea se facillita bastante.

Nos encontramos que el nucleo de las funciones siguen igual en ambos sistemas operativos , solo nos encontramos con algunas funciones que no han sido partadas como seria *pcap_get_selectable_fd( )* función que nos devolveria un descriptor para su uso el la función select función que como no funciona bajo windows , por ahora , winpcap no ha facilitado su adaptación , en cambio encontramos con una nueva función que seria *HANDLE pcap_getevent (pcap_t *p)* que seria su homologa en Windows salvando las distancias ya que no sirven para las misma funciones.

De igual modo nos encontramos con bastantes funciones que son iguales que en Linux y otras muchas que son exclusivas para Windows y que, o vienen a suplir las diferencias entre Linux y Windows o dan otras nuevas funciones a pcap en Windows.

A continuación se esplicaren con mas detalle las funciones a las que hemos recurrido en la programación de W2lan. Estas serian:

- pcap_findalldevs_ex
- pcap_open_live
- pcap_datalink
- pcap_setmintocopy
- pcap_getevent
- pcap_loop

pcap_findalldevs_ex (char *source, struct pcap_rmtauth *auth, pcap_if_t **alldevs, char *errbuf)

Esta función nos devuelve todas las interfaces de red que pueden ser abiertas para capturar datos.

Para su funcionamiento correcto tendremos que proporcionar en su primer parámetro donde tiene que buscar es que maquina o desfino , nosotros en el código le asignamos la sentencia PCAP_SRC_IF_STRING , que pcap define como *"file://"* para que busque en el ordenador propio.Esta función podría buscar en diferentes sistemas de la red si se lo indicásemos.La estructura pcap_rmtauth nos servirá si utilidad si queremos buscar en un sistema con clave y usuario en tal caso se lo tendríamos que facilitar, tendría la siguiente forma:

```
struct pcap_rmtauth {

int         type  //Type of the authentication required.

char *      username  //Zero-terminated string containing the username that has to be used on the remote machine for authentication

char *      password  //Zero-terminated string containing the password that has to be used on the remote machine for authentication

};
```

Los siquientes parametros son variables que les devemos facilitar y donde, en caso de de encontrar algo nos guadara la información, **alldevs*, o en caso de error nos notificara cual es el que se ha producido,**errbuf.*

pcap_t* pcap_open_live (const char * device,int snaplen, int promisc,int to_ms,char *ebuf)

Nos devuelve un descriptor de captura para poder manejarlo posteriormente.

Para hacerlo funcionar debemos pasar el nombre de dispositivo de red en el primer parámetro, Any o Null, forzarían la captura de todos los dispositivos dispnilbles .

Los demás parámetros nos facilitan varias opciones como *int snaplen* donde podremos indicar el numero máximo de bytes a capturar,*int promisc* que indica si el modo de apertura es promiscuo (suministrándole un valor distinto de 0) o normal (con valor 0),*int to_ms* que le indicaría el tiempo ,en milisegundos, que queremos de espera hasta que los paquetes son pasados al usuario y *char *ebuf* que nos informara que cualquier error.

En caso de error además de indicar el acontecimiento por la variable *ebuf la función devolvería un NULL en lugar del descriptor deseado.

int pcap_ datalink (pcap t *p)

Esta función devuelve el tipo de enlace de datos asociado al interfaz de red que le estamos suminitrando a través del descriptor .El valor de retorno puede ser uno de los siguientes:

DLT_NULL → BSD loopback encapsulation

DLT _EN10MB → Ethernet (10Mb, 100Mb, 1000Mb)

DLT_IEEE802 →IEEE 802.5 Token Ring

DLT_ARCNET → ARCNET

DLT_ SLIP → SLIP

DLT_PPP → PPP

DLT FDDI → FDDI

DLT_ATM_RFC1483 → RFC 1483 LLCSNAP-encapsulated ATM

DLT_RAW→ raw IP, el paquete comienza con una cabecera IP

DLT_PPP_SERIAL_PPP → en modo HDLC-like framing (RFC 1662)

DLT_PPP_ETHER → PPPoE

DLT_C_HDLC → Cisco PPP con HDLC framing, definido en 4.3.1 RFC 1547

DLT_IEEE802 → 11 IEEE 802.11 wireless LAN

DLT_LOOP → OpenBSD loopback encapsulation

DLT_LINUX_SLL →Linux cooked capture encapsulation

LT_LTALK → Apple LocalTal

int pcap_setmintocopy (pcap_t *p, int size)

Pasandoles el descriptor de la interfaz que hemos abierto y el un valor , establece dicho valor como la cantidad de información que debe tener para proceder al volcado del buffer al interfaz .

Esta función viene a servinos de apoyo para no esperar hasta que se llene ya que por defecto tarda un tiempo considerable y el parámetro de la función pcap_open_live donde se asigamos un tiempo para que ejecute ese vaciado no parece cumplir del todo dicha tarea.

HANDLE pcap_getevent (pcap_t *p)

Esta funcion nos devuelve un handle que nos sirve como descriptor en funciones de ApiWindows como *WaitForSingleObject* donde podremos controlar eventos que ocurran en esa interfaz.

En nuestro caso el valor devuelto debemos forzarlo a un handle ya que la función parece devolver un entero en lugar de lo definido por la función .Una vez echa la transformación funciona correctamente.

int pcap_loop (pcap_t *p, int cnt, pcap_handler callback, u_char *user)

Esta función es la encargada de recoger los paquetes y analizarlo a través de la función donde los redirecciona que seria la indicada en el parámetro pcap_handler callback, esta función debe cumplir al el siguiente prototipo para que sea valida:

*typedef void(*) pcap_handler(u_char *user, const struct pcap_pkthdr *pkt_header, const u_char *pkt_data)*

La función prototipo de pcap_hander recibe el descriptor pcap por la variable u_char *user, la cual deberemos forzarla de nuevo al tipo (pcap_t *), y en *pkt_data nos llegara el paquete en cuestión .La estructura pcap_pkthdr nos viene a dar información sobre el tiempo en el que se ha capturado y su tamaño:

```
struct pcap pkthdr {
struct timeval ts; // time stamp (marca de tiempo)
bpf u int32 caplen; // tamano del paquete al ser capturado
bpf u int32 len; // tamano real del paquete en el fichero;
};
```

Ademas entre los parámetros de *pcap_loop* vemos el descriptor de pcapque se lo pasamos en el primer parámetro .El segundo, int *cnt*, nos sirve para indicar el numero máximo de paquetes que debe recoger antes de ejecutar la función y u_char *user que seria la informacion que pasamos.

4. W2LAN

4.1 Introducción:

W2lan es un protocolo de la 2 capa, diseñado para distribuir contenidos multimedia o información en una red Ac-hoc .El protocolo ofrece visibilidad total a los nodos pertenecientes en la red, estos nodos deberán estar en un modo Ad-Hoc y deberá existir una ruta entre cualquier par de nodos, sino, no impediría el funcionamiento de la re,d pero si llevara a la situación no deseada de dos redes fragmentadas.

4.2 Funcionamiento:

El protocolo W2lan se basa en la comunicación entre los nodos de la red con tres tipos de mensajes que son los que facilitaran que haya una conversación entre ellos con la finalidad de distribuir la información que alguno de los nodos posee.

Para ellos utiliza los mensajes {announce , request, data }:

Announce: Es utilizado cuando al nodo quiere transmitir información que posee. La estructura de este mensaje seria:

Announce

FF FF FF FF FF FF	MAC origen	69 01

Conv. ID	Eth.MAC destino	Eth.MAC origen	Eth.T

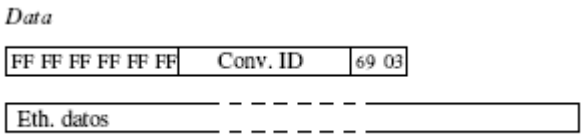
Request: Es utilizado por el nodo cuando le llega un anounce de una conversación que no posee e intenta informar al nodo que la genero que le interesa:

Request

MAC destino	MAC origen	69 02

Conv. ID

Data: Este mensaje se utiliza cuando después de anunciar alguna conversación se han recibido algún mensaje request para esta, lo cual significa que la información es deseada por algun nodo. En tal caso se manda la información deseada:



El protocolo en la red se pondría en funcionamiento cuando uno de los nodos que pertenece a la red, recibe desde capas superiores del protocolo o una trama W2lan de otro nodo.

Si recibe una trama de una capa superior, el nodo construirá una trama *announce* con un nuevo *ConversationId* y la emitirá. Después guardara el campo de datos en la lista de "pedingframes" correctamente identificada mediante la etiqueta *ConversationId* que acaba de crear también creara una entrada en la lista de "*timer list*" , que contendrá un tiempo (instante actual + tiempo de espera) que será el tiempo de caducidad, y otra en La lista de "*counters list*" ,para controlar las peticiones que se reciben de esa conversación , ambas identificadas con la etiqueta de *ConversionId* y en ambas lista se introducirá al final de la lista

Después el nodo esperara un cierto intervalo de tiempo, con la finalidad de recibir *request* para esa conversación .Si se produjese este evento la entrada de la lista de "*counters list*" vera incrementada su variable de contadores. Cuando finalice el periodo de espera la información pendiente se eliminara de la lista de "*pedingframes*" y si hubo algun incremento en la entrada de la lista de "*counters list*" para esa conversación el nodo construirá la correspondiente trama Data y la emitirá.

En otro caso a tratar es recibir una trama W2lan de otro nodo, en este caso tendremos dos posibilidades que nos llegue una trama *announce*, con la que comprobara la lista de "*conversationlist*" para ver si encuentra esa conversación. En caso de no encontrarla añadirá un nuevo elemento marcado como pendiente y mandara una trama *request* para pedir el envió de la información.

Si la trama fuese del tipo Data, buscaríamos igualmente en la lista de "*conversationlist*" la conversación de la cual nos han llegado los datos .Si se encuentra en la lista y esta marcada como pendiente, procederemos desmarcándola y enviado la trama con la cabecera almacenada en la lista de conversaciones y los datos que acabamos de recibir.

4.3 Código de W2lan:

4.3.1 Funciones de apoyo:

Dentro del funcionamiento de W2lan nos encontramos con funciones que son utilizadas sin importar el camino por donde se esta desarrollando el funcionamiento del programa, como serian las diferentes listas de las cuales hace uso el programa en diferentes partes y otras funciones que se utilizan de apoyo y para conseguir el funcionamiento deseado.

4.3.1.1 Listas

Las listas que maneja el programa son:

- *conversations list*
- *counters list*
- *pendingframes list*
- *timers list*

Estas listas son utilizadas por W2lan, para mantener y establecer conversaciones con otros nodos.

Con *conversations list,* W2lan consigue registrar todas las conversaciones que hay en la red, almacenando la ConversationId, la Mac origen y destino y el tipo de trama, también servirá para distinguir cuales están aun pendientes de recibir la información que se esta divulgando.

Las funciones para manejar esta lista serian:

```
//Crea la listas para su posterior usa

conversationslist conversationslistcreate(void) {
        conversationslist tmp[SIZE_CONVERSATIONSLIST];
        int i;

        for ( i=0 ; i < SIZE_CONVERSATIONSLIST; i++ ) {
                tmp[i]=(conversationslist)malloc(sizeof(struct conversationslistElement));
                assert( tmp[i] != NULL );
        }
```

```
		for ( i=0 ; i < SIZE_CONVERSATIONSLIST-1; i++ ) {
			tmp[i]->next = tmp[i+1];
		}
		tmp[SIZE_CONVERSATIONSLIST-1]->next = tmp[0];
		return (conversationslist)tmp[SIZE_CONVERSATIONSLIST-1];
}

//Busca en las listas existentes si existe alguna con la ConversationId que le proporcinamos

conversationslist conversationslistsearch( u_int8_t *conversationID , conversationslist list ) {

		int i;
		for ( i=0 ; i < SIZE_CONVERSATIONSLIST; i++ ) {
			if ( memcmp( list->conversationID, conversationID, ETH_P_W2LAN_CONVERSATIONID_LEN ) == 0 )
{
				return list;
			} else {
				list = list->next;
			}
		}
		return (conversationslist)NULL;
}

//Añadimos una nueva entrada en la lista


void conversationslistadd( u_int8_t *conversationID , u_int8_t * MACdst, u_int8_t * MACsrc ,  u_int16_t MACtype,
conversationslist *list ) {

		memcpy( (*list)->conversationID , conversationID , ETH_P_W2LAN_CONVERSATIONID_LEN );
		(*list)->pending = 1;
		memcpy ( (*list)->header.MACdst , MACdst , ETHER_ADDR_LEN );
		memcpy ( (*list)->header.MACsrc , MACsrc , ETHER_ADDR_LEN );
		(*list)->header.MACtype = MACtype;

		(*list) = (*list)->next;
}



/* helper functions */

//Recupera el valor de la variable pending donde nos informa si la conversación esta aun en estado pendiente

int conversationpending( conversationslist conv ) {
		return conv->pending;
}

//Establece el valor de pendiente a 1 con lo que la conversación aun no tiene la información necesaria

void conversationsetpending( conversationslist conv ) {
		conv->pending = 1;
}

//Establece que la conversación ya tiene la información necesaria

void conversationclearpending( conversationslist conv ) {
		conv->pending = 0;
}

//Recupera el valor de  la MAC destino

u_int8_t* conversationgetmacdst( conversationslist conv ) {
		return conv->header.MACdst;
}

//Recupera el valor de  la MAC origen


u_int8_t* conversationgetmacsrc( conversationslist conv ) {
		return conv->header.MACsrc;
}
```

```
//Recupera el tipo del mensaje

u_int16_t conversationgetmactype( conversationslist conv ) {
        return conv->header.MACtype;
}
```

Counter list , aporta una manera de llevar un control de los *request* que han llegado para una conversión en concreto. *Counter list* crea una entrada con el valor de *ConversationId* donde se le relaciona también un contador que inicialmente tendrá un valor de cero. La nueva entrada se introducirá como ultimo miembro de la lista en caso de que hubiera mas entradas e incrementara el contador por cada *request* recibido durante un tiempo de espera. Tras ese tiempo el programa recuperara el valor del contador y eliminara esa entrada poniendo la anterior en primer lugar de la lista.

```
//Crea las listas

counterslist counterslistcreate(void) {
        return (counterslist)NULL;
}


//Busca en la lista si ya existe una entrada con el valor de conversationId que le proporcionamos

counterslist counterslistsearch( u_int8_t *conversationID , counterslist list ) {

        if ( list == NULL ) {
                return NULL;
        } else {
                if ( memcmp( list->conversationID , conversationID, ETH_P_W2LAN_CONVERSATIONID_LEN ) == 0 )
{
                        return list;
                } else {
                        return counterslistsearch( conversationID, list->next );
                }
        }
}


//Introduce una nueva entrada en la lista , situandoesta en ultimo lugar , con el valor del conversationId que le
proporcionamos y el contador con valor a cero

void counterslistintoaslast( u_int8_t *conversationID , counterslist *list ) {

        assert( counterslistsearch(conversationID, *list) == NULL );

        if ( *list == NULL ) {
                *list = (counterslist)malloc( sizeof( struct counterslistElement ) );

                assert( *list != NULL );
                memcpy( (*list)->conversationID, conversationID, ETH_P_W2LAN_CONVERSATIONID_LEN );
                (*list)->requests = 0;
                (*list)->next = NULL;
        } else {
                counterslistintoaslast( conversationID, &((*list)->next) );
        }
```

```
}

//Recupera el valor del contador de la primera entrada que hay en la lista ,al mismo tiempo que elimina esa entranda

int counterslistfirstinqueue( u_int8_t *conversationID , counterslist *list ) {


        /* counterslist should not be empty */
        assert( (*list) != NULL );
        int requests;
        counterslist tmp = (*list);
        memcpy( conversationID , (*list)->conversationID , ETH_P_W2LAN_CONVERSATIONID_LEN );
        //   conversationID = (*list)->conversationID;
        requests = tmp->requests;
        (*list) = (*list)->next;
        free(tmp);
        return requests;
}


//Incrementa el contador de la entrada que tiene el valor de conversationId que le proporcionamos

void counterslistincrease( u_int8_t *conversationID , counterslist list ) {

        assert( counterslistsearch(conversationID, list) != NULL );

        if ( memcmp( list->conversationID , conversationID, ETH_P_W2LAN_CONVERSATIONID_LEN ) == 0 ) {

                list->requests = list->requests + 1;

        } else {
                counterslistincrease( conversationID , list->next );
        }
}
```

Pendingframes list, almacena el campo de datos de las conversaciones de las cuales estamos esperando recibir request por si algun nodo nos pide esa información .Por cada conversación habrá una entrada en la lista que tendra como etiqueta la ConversationId, un campo donde guardaremos el campo de datos y otro donde nos informara el tamaño del campo en cuestión.

```
//Crea las listas

pendingframeslist pendingframeslistcreate(void) {
        return (pendingframeslist)NULL;
}


pendingframeslist pendingframeslistsearch( u_int8_t *conversationID , pendingframeslist list ) {
        if ( list == NULL ) {
                return NULL;
        } else {
                if ( memcmp( list->conversationID , conversationID,
```

```
ETH_P_W2LAN_CONVERSATIONID_LEN ) == 0 ) {
                        return list;
                } else {
                        return pendingframeslistsearch( conversationID, list->next );
                }
        }
}

//Crea una entrada con la Id que le pasamos y su campo de datos

void pendingframeslistput( u_int8_t *conversationID , u_int8_t *ethernetdatafield , u_int16_t size ,
pendingframeslist *list ) {

        /* the pending frame should not be in the list */



        assert( pendingframeslistsearch(conversationID, *list) == NULL );


        pendingframeslist tmp = *list;
        *list = (pendingframeslist)malloc( sizeof( struct pendingframeslistElement ) );

        assert( *list != NULL );
        memcpy( (*list)->conversationID, conversationID, ETH_P_W2LAN_CONVERSATIONID_LEN );
        memcpy( (*list)->ethernetdatafield, ethernetdatafield, size );
        (*list)->size = size;
        (*list)->next = tmp;
}

//Elimina de la  lista la entrada que le indicamos

void pendingframeslistextract( u_int8_t *conversationID , u_int8_t *ethernetdatafield , u_int16_t *size ,
pendingframeslist *list ) {

        /* the pending frame should be in the list */

        pendingframeslist pendingframe = pendingframeslistsearch(conversationID, *list);
        assert( pendingframe != NULL );

        if ( ( (*list) == pendingframe) ) {

}
                (*size) = (*list)->size;
                memcpy( ethernetdatafield , (*list)->ethernetdatafield , (*list)->size );
                pendingframeslist tmp = (*list)->next;
                free(*list);
                *list=tmp;
        } else {
                pendingframeslistextract( conversationID, ethernetdatafield, size, &((*list)->next) );
        }
}
```

Timers list ayudara a mantener en control de los tiempos en el cual nos van llegando las conversaciones, en la entradas de la lista se almacenaran el conversationId como etiqueta, y el tiempo que será en el que hemos recibido la conversación mas el tiempo de espera que tenemos asignado.

```
//Crea las listas

timerslist timerslistcreate(void) {
        return (timerslist)NULL;
}

//Nos informa si tenemos algun elemento en la lista

int timerslistempty( timerslist list ) {


        if ( list == NULL) {
                return 1;
        } else {
                return 0;
        }
}

//Busca en la lista la entrada que coincida con la ConversatioId que le proporcionamos

timerslist timerslistsearch( u_int8_t *conversationID , timerslist list ) {
        if ( list == NULL ) {
                return NULL;
        } else {
                if ( memcmp( list->conversationID , conversationID, ETH_P_W2LAN_CONVERSATIONID_LEN ) == 0 )
{
                        return list;
                } else {
                        return timerslistsearch( conversationID, list->next );
                }
        }
}

//Crea una entrada etiquetándola con la conversationId que le proporcionamos y guardando el valor del tiempo que le
proporcionamos. Dicha entrada la situa al final de la lista

void timerslistintoaslast( u_int8_t *conversationID , struct timeval* t , timerslist *list ) {


        /* the timer should not be in the list */
        assert( timerslistsearch(conversationID, *list) == NULL );

        if ( *list == NULL ) {
                *list = (timerslist)malloc( sizeof( struct timerslistElement ) );
                assert( *list != NULL );
                memcpy( (*list)->conversationID, conversationID, ETH_P_W2LAN_CONVERSATIONID_LEN );
                (*list)->timer.tv_sec = t->tv_sec;
                (*list)->timer.tv_usec = t->tv_usec;
                (*list)->next = NULL;
        } else {
                timerslistintoaslast( conversationID, t, &((*list)->next) );
        }
}

//Extrae  la primera entrada de la lista facilitándonos la información y luego la elimina

void timerslistfirstinqueue( u_int8_t *conversationID , struct timeval* t , timerslist *list ) {



        /* timerslist should not be empty */
        assert( (*list) != NULL );

        timerslist tmp = (*list);
        memcpy( conversationID , (*list)->conversationID , ETH_P_W2LAN_CONVERSATIONID_LEN );
        //   conversationID = (*list)->conversationID;

        t->tv_sec = tmp->timer.tv_sec;
        t->tv_usec = tmp->timer.tv_usec;
        (*list) = (*list)->next;
```

```
		free(tmp);
		return;
}

//Nos facilita los tiempos del siguiente al primero de la lista

void timerslistsuccessor( u_int8_t *conversationID , struct timeval *t, timerslist list ) {

		/* timerslist should not be empty */
		assert( list != NULL );

		t->tv_sec = list->timer.tv_sec;
		t->tv_usec = list->timer.tv_usec;
		return;
}
```

4.3.1.2 Otras funciones

Otras funciones que se utilizan en W2lan y que son necesarias para su funcionamiento o para informarnos serian:

- gettimeofday
- ether_aton_r
- ether_aton
- timeval_subtract
- timeval_add
- mostrar

En lo referente a gettimeofday, ether_aton_r y ether_aton, nos encontramos con funciones que son provienen de librerias de C en linux y que se han adaptado a entorno windows incluyéndolas como funciones de W2lan

. En el caso de ether_aton_r y ether_aton, el código que tienen ambas sentencias si funcionan en windows aunque no el resto de la cabecera a la que pertenecen, *ether.h*, por lo tanto las incluimos como funciones propias en W2lan , estas tienen la misma ejecución que sus homologas en Linux:

```
struct ether_addr * ether_aton_r (const char *asc, struct ether_addr *addr)
{
	size_t cnt;

	for (cnt = 0; cnt < 6; ++cnt)
	{
		unsigned int number;
		char ch;

		ch = _tolower (*asc++);
		if ((ch < '0' || ch > '9') && (ch < 'a' || ch > 'f'))
			return NULL;
		number = isdigit (ch) ? (ch - '0') : (ch - 'a' + 10);

		ch = _tolower (*asc);
		if ((cnt < 5 && ch != ':') || (cnt == 5 && ch != '\0' && !isspace (ch)))
		{
			++asc;
			if ((ch < '0' || ch > '9') && (ch < 'a' || ch > 'f'))
				return NULL;
			number <<= 4;
			number += isdigit (ch) ? (ch - '0') : (ch - 'a' + 10);

			ch = *asc;
			if (cnt < 5 && ch != ':')
				return NULL;
```

```
                }

                /* Store result.  */
                addr->octet[cnt] = (unsigned char) number;

                /* Skip ':'.  */
                ++asc;
        }

        return addr;
}
```

```
struct ether_addr * ether_aton (const char *asc)
{
        static struct ether_addr result;

        return ether_aton_r (asc, &result);
}
```

Algo parecido nos pasa con *gettimeofday* esta función que en Linux nos devuelve los segundos que han pasado desde cierta fecha que viene definida por defecto o que nosotros podemos asignar.No tiene la misma ejecución en Windows dando un dato erróneo .Para no tener que modificar el programa y mantener la idea original se ha utilizado este código que da unos resultados optimos:

```
int gettimeofday(struct timeval *tv)/*, struct timezone *tz)*/
{

        union {
                long long ns100;
                FILETIME ft;
        } now;

        GetSystemTimeAsFileTime (&now.ft);
        tv->tv_usec = (long) ((now.ns100 / 10LL) % 1000000LL);
        tv->tv_sec = (long) ((now.ns100 - 116444736000000000LL) / 10000000LL);

        return (0);
}
```

Las funciones timeval_subtract y timeval_add , son utilizadas para sumar o restas lo tiempos que les pasamos devolviendo posteriormente el resultado de la operación en otra variable que también es suministrada , estas dos funciones son especialmente usadas en la listas de W2lan para calcular si el tiempo de espera de una conversación a expirado o para añadir al tiempo de llegada de la conversación el tiempo de espera.

```
int timeval_subtract( struct timeval *result, struct timeval *x, struct timeval *y ) {

        /* Perform the carry for the later subtraction by updating y. */

        sprintf(cad," X: %d %d \n Y: %d %d",x->tv_usec,x->tv_sec,y->tv_usec,y->tv_sec );

        if (x->tv_usec < y->tv_usec) {
                int nsec = (y->tv_usec - x->tv_usec) / 1000000 + 1;
                y->tv_usec -= 1000000 * nsec;
                y->tv_sec += nsec;
        }
        if (x->tv_usec - y->tv_usec > 1000000) {
                int nsec = (y->tv_usec - x->tv_usec) / 1000000;
                y->tv_usec += 1000000 * nsec;
                y->tv_sec -= nsec;
        }
        sprintf(cad,"Final X: %d %d \n Y: %d %d",x->tv_usec,x->tv_sec,y->tv_usec,y->tv_sec );
        mostrar(cad);

        /* Compute the time remaining to wait. tv_usec is certainly positive. */
        result->tv_sec = x->tv_sec - y->tv_sec;
        result->tv_usec = x->tv_usec - y->tv_usec;

        sprintf(cad,"%d %d",result->tv_sec,result->tv_usec);
        mostrar(cad);


        /* Return 1 if result is negative. */
        return x->tv_sec < y->tv_sec ;

}
```

```
void timeval_add( struct timeval *result, struct timeval *x, struct timeval *y ) {

        int usec = ( y->tv_usec + x->tv_usec );
        int sec  = (  y->tv_sec +  x->tv_sec );
        if ( usec >= 1000000 ) {
                usec = usec - 1000000;
                sec = sec + 1;
        }
        result->tv_usec = usec;
        result->tv_sec = sec;
        return;
}
```

Por ultimo durante el programa se recurre a una función para comunicarse con el interfaz de usuario creado, enviándole información acerca de lo que esta pasando en la ejecución del programa.

```
void mostrar(char * cad_m)
{

        char tiempo[200];
        DWORD tiempo_d;



        SendMessage(GetDlgItem(hwnd,105), WM_SETFONT, (WPARAM)hfont, MAKELPARAM(TRUE, 0));

        tiempo_d = time(NULL);
        tmPtr = localtime(&tiempo_d);
        sprintf(tiempo,"%s : %s",asctime(tmPtr),cad_m);


        SendMessage(GetDlgItem(hwnd,105), LB_ADDSTRING, 0, (LPARAM)tiempo);
   SendMessage(GetDlgItem(hwnd,105), LB_SETCURSEL, (WPARAM)SendMessage(GetDlgItem(hwnd,105),
LB_GETCOUNT, 0, 0)-1, 0);

}
```

Comentar en esta parte que al estar mostrar en el hilo que hemos creado no podemos recurrir a los manejadores que hemos utilizado en la creación de la interfaz de usuario por lo tanto para poder comunicarnos con el listbox ,que nos mostrara la información ,debemos pedir su manejador.Para ello utilizamos *GetDlgItem(hwnd,105)* donde pasaremos el manejador de la ventana principal y el identificador del listbox que si sabemos puesto que se lo hemos asignado nosotros.

4.3.2 Funciones principales W2lan

El código del programa lo podemos diferenciar en dos partes claras, una seria la referente al entorno Windows donde el objetivo es ofrecer al usuario las opciones e información del programa así como la información durante su ejecución. Y una segunda que seria el W2lan en si.

4.3.2.1 Creación del interfaz para el usuario.

Para la primera parte se centra en los archivos:

- *ids.h*
- *w2lan_i.rc*
- *w2lan_i.c*

De los cuales w2lan_i.c seria el programa principal y del que realmente empieza la ejecución del programa W2lan.Los archivos *ids.h y w2lan_i.rc* son requeridos a la hora de crear y hacer funcionar la ventana que manejaremos.

Ids.h define el valor de los identificadores de control que se asignaran en la creación de los distintos elementos que existirán en la ventana principal y en su menú. Estos valores servirán para identificar la acción que ha desencadenado una llamada a la función que rige el interfaz, *WindowProcedure*, que se comentara mas adelante.

ids.h:

```
/* Identificadores de comandos */
#define CM_DIALOGO      105

/* Identificadores de diálogo */
#define ID_LISTA        100
#define ID_COMBO3       101
#define ID_BOTON        102
#define CM_VERSION      103
#define CM_SALIR        104
#define ID_LISTA2       105
#define ID_BOTON2       106
#define ID_BOTON3       107
#define ID_BOTON4       108
```

Dentro de la ventana se realizara un menú horizontal para el cual es necesitario el archivo *w2lan_i.rc* .Éste, mediante una asignación en el momento de la creación de la

ventana principal, servir de guía al programa para saber como es el menú horizontal que se creara.

w2lan_i.rc:

```
#include <windows.h>
#include "IDS.H"

Menu MENU
BEGIN
 POPUP "&Principal"
 BEGIN
  MENUITEM "&Version", CM_VERSION
  MENUITEM "&Salir", CM_SALIR
 END
END
```

En *w2lan_i.c* se encuentra el código que nos generara la ventana y su funcionamiento, también, la función principal con la que se inicia el programa. Al tratarse de una aplicación Windows (winapi) en C la función principal con la que se iniciara el programa será *WinMain* en lugar del *main* caracteristico.

El código muestra la creación de la ventana principal de la cual heredaran el resto de ventanas que insertaremos. Primero indica los diferentes parámetros y características de la ventana principal. Se rellena adecuadamente la estructura *WNDCLASSEX (wincl)*, que define algunas características que serán comunes a todas las ventanas de una misma clase, como color de fondo, icono, menú por defecto, el procedimiento de ventana, etc. Después se llama a la función *RegisterClassEx* quedando registrada la clase ventana .Una vez registrada se crea la ventana con *CreateWindowEx* indicándole características como nombre de la ventana, tamaño, de quien es hija etc…

w2lan_i.c:

```
int WINAPI WinMain (HINSTANCE hThisInstance,HINSTANCE hPrevInstance,LPSTR lpszArgument,int
nFunsterStil)
{

        /* Estructura de la ventana */
        wincl.hInstance = hThisInstance;
        wincl.lpszClassName = "W2lan";
        wincl.lpfnWndProc = WindowProcedure;      /* Esta función es invocada por Windows */
        wincl.style = CS_DBLCLKS;                 /* Captura los doble-clicks */
        wincl.cbSize = sizeof (WNDCLASSEX);

        /* Usar icono y puntero por defector */
        wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
        wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
```

```
wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
wincl.lpszMenuName = "Menu";
wincl.cbClsExtra = 0;                   /* Sin información adicional para la */
wincl.cbWndExtra = 0;                   /* clase o la ventana */
/* Usar el color de fondo por defecto para la ventana */
wincl.hbrBackground = GetSysColorBrush(COLOR_BACKGROUND);

/* Registrar la clase de ventana, si falla, salir del programa */
if(!RegisterClassEx(&wincl)) return 0;

/* La clase está registrada, crear la ventana */
hwnd = CreateWindowEx(
        0,              /* Posibilidades de variación */
        "W2lan",    /* Nombre de la clase */
        "W2lan",      /* Texto del título */
        WS_OVERLAPPEDWINDOW, /* Tipo por defecto */
        CW_USEDEFAULT,      /* Windows decide la posición */
        CW_USEDEFAULT,      /* donde se coloca la ventana */
        900,            /* Ancho */
        800,            /* Alto en pixels */
        HWND_DESKTOP,       /* La ventana es hija del escritorio */
        NULL,           /* Sin menú */
        hThisInstance,      /* Manipulador de instancia */
        NULL            /* No hay datos de creación de ventana */
        );

/* Mostrar la ventana */
ShowWindow(hwnd, SW_SHOWDEFAULT);
```

Para la inclusión del menú horizontal mediante el fichero w2lan_i.rc se incluye el parámetro *wincl.lpszMenuName = "Menu" ,* especificando el nombre estructura que debe buscar en el archivo.

```
wincl.lpszMenuName = "Menu";
```

Finalmente la función se quedaría en el bucle a la espera de mensajes originados por la ventana o entradas por teclado, en el caso de recibir algún mensaje desde la ventana este la re direcciona al procedimiento de ventana, en este caso seria la función *WindowProcedure* , que será la encargada de analizar el mensaje y actuar en consecuencia.

```
/* Bucle de mensajes, se ejecuta hasta que haya error o GetMessage devuelva FALSE */
        while(TRUE == GetMessage(&mensaje, NULL, 0, 0))
        {
                /* Traducir mensajes de teclas virtuales a mensajes de caracteres */
                TranslateMessage(&mensaje);
                /* Enviar mensaje al procedimiento de ventana */
                DispatchMessage(&mensaje);
        }

        /* Salir con valor de retorno */
```

```
        return mensaje.wParam;
}
```

WindowProcedure recibe el tipo de mensaje (*msg*), e información adjunta en *wParam* y *lParam*.

La función analiza el tipo de mensaje que hemos recibido y actúa en consecuencia, esto lo realiza mediante un switch.

```
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
        static HINSTANCE hInstance;
        static HINSTANCE hInstance2;
        static char datos [80];
        static HWND hctrl;
        static HWND hctrl2;
        static HWND hctrl3;
        static HWND hctrl4;
        static HWND hctrl5;
        static HWND hctrl6;
        static HWND hctrl7;
        int pid;
        int i=0;
        int *pasar;
        int indice=0;
        char mode[10];
        char mode2[500] , cad[500];
        char select;
        int num_device;
        char errbuf[PCAP_ERRBUF_SIZE];
        HDC hdc;
        PAINTSTRUCT ps;

        switch (msg)            /* manipulador del mensaje */
        {
```

Dentro de los posibles tipos mensajes que recibe trata *WM_CREATE* que se origina cuando se crea la ventana principal, esto será aprovechado para crear elementos dentro de la ventana principal o padre.

Para ello se recurrirá a la misma función que utilizo para crear la ventana principal, pero en este caso se crearan elementos (ventanas) que se integraran en la ventana principal formando una única interfaz.

```
case WM_CREATE:

                hInstance = ((LPCREATESTRUCT)lParam)->hInstance;

                hfont = CreateFont(14, 0, 0, 0, 300,
```

```
                                FALSE, FALSE, FALSE, DEFAULT_CHARSET,
                                OUT_TT_PRECIS, CLIP_DEFAULT_PRECIS,
                                PROOF_QUALITY, DEFAULT_PITCH | FF_ROMAN,
                                "Times New Roman");



                                hwnd,           /* Ventana padre */
                                (HMENU)ID_COMBO3,/* Identificador del control */
                                hInstance,      /* Instancia */
                                NULL);          /* Sin datos de creación de ventana */


                        SendMessage(hctrl2, WM_SETFONT, (WPARAM)hfont, MAKELPARAM(TRUE, 0));

                        if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
                        {

                                MessageBox (hwnd,"Error in pcap_findalldevs" ,"W2lan" , MB_OK);
                                exit(1);

                        }
        /* Buscamos los device posibles*/
                        /* Print the list */
                        for(apoyo=alldevs; apoyo; apoyo=apoyo->next)
                        {
                                sprintf(mode2,"%d. %s", ++i, d->description);
                                SendMessage(hctrl2, CB_ADDSTRING, 0,(LPARAM)mode2 );
                        }
```

Los elementos creados, tienen el fin de facilitar la elección de opciones de nuestro programa y mostrar la información que se ira generando durante su funcionamiento. Para conseguir esto utilizaremos dos elementos *listbox* y uno *combobox*.

El primer *listbox* servirá para poder elegir el modo de funcionamiento del *W2la*n y el siguiente *listbox* para mostrar la información que genere en programa durante la ejecución.

El *combobox* se utilizara para mostrar las interfaces que hemos encontrado mediante la utilización de la función *pcap_findalldevs_ex ,* que devuelve los interfaces que posee la maquina , con el objetivo de que puedan ser seleccionadas por el usuarios. Esta función devolverá todas las existentes, no solo las que podamos utilizar .Algunas si no se tienen los permisos oportunos es posible que no puedan ser utilizadas.

```
/* Listbox con los modos de funcionamiento*/
                        hctrl = CreateWindowEx(
                                0,
                                "LISTBOX",      /* Nombre de la clase */
                                "",             /* Texto del título, no tiene */
                                LBS_STANDARD | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, /* Estilo */
                                10, 20,         /* Posición */
                                70, 60,         /* Tamaño */
                                hwnd,           /* Ventana padre */
```

```
                    (HMENU)ID_LISTA, /* Identificador del control */
                    hInstance,      /* Instancia */
                    NULL);          /* Sin datos de creación de ventana */
                /* Inicialización de los datos de la aplicación */


                SendMessage(hctrl, LB_ADDSTRING, 0, (LPARAM)cadena[2]);
                SendMessage(hctrl, LB_ADDSTRING, 0, (LPARAM)cadena[1]);
                SendMessage(hctrl, LB_ADDSTRING, 0, (LPARAM)cadena[0]);
                SendMessage(hctrl, LB_SETCURSEL, (WPARAM)0, 0);//por defecto seleccionamos la 1º opcion


/* Listbox donde mostraremos la información de la ejecucion de W2lan*/
                hctrl4 = CreateWindowEx(
                    0,
                    "LISTBOX",      /* Nombre de la clase */
                    "",             /* Texto del título, no tiene */
                    WS_VSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,/* Estilo */
                    140, 80,        /* Posición */
                    700,600,       /* Tamaño */
                    hwnd,           /* Ventana padre */
                    (HMENU)ID_LISTA2, /* Identificador del control */
                    hInstance,      /* Instancia */
                    NULL);          /* Sin datos de creación de ventana */

                //reducimos el tamaño de la letra

                SendMessage(hctrl4, WM_SETFONT, (WPARAM)hfont, MAKELPARAM(TRUE, 0));


/* Combobox donde estaran las posible device para que se puedan seleccionar*/

                hctrl2 = CreateWindowEx(
                    0,
                    "COMBOBOX",     /* Nombre de la clase */
                    NULL,           /* Texto del título */
                    CBS_DROPDOWNLIST |
                    WS_VSCROLL |
                    WS_CHILD | WS_VISIBLE | WS_TABSTOP, /* Estilo */
                    140, 20,        /* Posición */
                    700, 250,       /* Tamaño */
```

Finalmente en el caso de *WM_CREATE* crea varios botones como son , *Go* ,que pondrá en marcha la función W2lan ,*Stop* ,que la detendrá , *Clear* ,que limpiara las entradas del listox que nos muestra la información en ejecución y por ultimo un botón de la clase autocheckbox que nos servirá para indicarnos si estamos en *Debug Mode* o no.

```
                //BOTONES

                hctrl3 = CreateWindowEx(
                    0,
                    "BUTTON",       /* Nombre de la clase */
                    "Go",      /* Texto del título */
                    BS_TEXT |
                    WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, /* Estilo */
```

```
                        10, 100,          /* Posición */
                        91, 24,         /* Tamaño */
                        hwnd,           /* Ventana padre */
                        (HMENU)ID_BOTON,/* Identificador del control */
                        hInstance,      /* Instancia */
                        NULL);          /* Sin datos de creación de ventana */

                hctrl5 = CreateWindowEx(
                        0,
                        "BUTTON",       /* Nombre de la clase */
                        "Stop",      /* Texto del título */
                        BS_TEXT |
                        WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, /* Estilo */
                        10, 140,          /* Posición */
                        91, 24,         /* Tamaño */
                        hwnd,           /* Ventana padre */
                        (HMENU)ID_BOTON2,/* Identificador del control */
                        hInstance,      /* Instancia */
                        NULL);          /* Sin datos de creación de ventana */

                //Este boton lo deshabilitamos al empezar
                EnableWindow(hctrl5, FALSE);

                hctrl6 = CreateWindowEx(
                        0,
                        "BUTTON",       /* Nombre de la clase */
                        "Clear",      /* Texto del título */
                        BS_TEXT |
                        WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, /* Estilo */
                        10, 180,          /* Posición */
                        91, 24,         /* Tamaño */
                        hwnd,           /* Ventana padre */
                        (HMENU)ID_BOTON3,/* Identificador del control */
                        hInstance,      /* Instancia */
                        NULL);          /* Sin datos de creación de ventana */

                hctrl7 = CreateWindowEx(
                        0,
                        "BUTTON",       /* Nombre de la clase */
                        "Debug Mode",      /* Texto del título */

                        BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP| BS_LEFTTEXT, /*
Estilo */
                        10, 220,          /* Posición */
                        100, 24,         /* Tamaño */
                        hwnd,           /* Ventana padre */
                        (HMENU)ID_BOTON4,/* Identificador del control */
                        hInstance,      /* Instancia */
                        NULL);          /* Sin datos de creación de ventana */

                return 0;
```

En *WindowProcedure* también analiza los mensajes que saltan al apretar alguno de los botones creados. Estos llegaran a traves del mensaje *WM_COMMAND*. Cuando la función entra en este caso del swicth ejecuta otro switch donde el elemento que se analiza es la información que viene en *LOWORD(wParam)* donde viene el identificador del elemento que a producido el mensaje que se analiza.

Se desarrollaran dentro de la estrucutra los mensajes originados al apretar los botones del menú horizontal, como son salir y versión, cuya acción es sencilla, uno cierra del todo la ventana cerrando también el programa y el otro muestra información sobre W2lan.

Mas importancia tienen los botones propios del interfaz, el primero que se muestra en el código es el de *Go*, el cual pondrá el programa en funcionamiento. Antes de ello recoge y trata la información que le indica el *listbox* que se utiliza para mostrar y seleccionar el modo de funcionamiento, también el del *combobox* donde se habrá seleccionado la interfaz con la que se quiere trabajar y el marcado o no del botón de *Debug mode*. Una vez recogida esta información anulamos los botones Go y Debug mode para que no puedan ser ejecutados de nuevo y habilitamos el botón de Stop para poder para el funcionamiento del programa cuando queramos. Finalmente lanzamos un hilo donde se ejecutara el paralelo el W2lan, y de esa forma quedara libre la ventana para que pueda comunicarse y mostrar los mensajes que W2lan le envía.

```
case WM_COMMAND:
                        switch(LOWORD(wParam)) {

                                case CM_SALIR:
                                        PostQuitMessage(0);
                                        break;

                                case CM_VERSION:
                                        MessageBox(hwnd, "W2lan 1.0 \n Copyright (C) 2007 by Francesc Burrull i
Mestres. \n francesc.burrull@upct.es", "W2lan", MB_OK);
                                        break;

                                case ID_BOTON://boton Go

//recogemos el modo de funcionamiento que ha sido recogido

                                        indice = SendMessage(hctrl,LB_GETCURSEL, 0, 0);

                                SendMessage(hctrl, LB_GETTEXT, (WPARAM)indice, (LPARAM)mode);

                                // de igual modo recogemos la device que ha sido seleccionada

                                        i = SendMessage(hctrl2,CB_GETCURSEL, 0, 0);

                                        //Verificamos que se ha seleccionado alguna interfaz
                                if(i==-1)
                                {
                                MessageBox(hwnd,"No ha elegido niguna interfaz\nPor favor seleccione
una","W2lan",MB_OK);
                                break;
                                }

                                        SendMessage(hctrl2,CB_GETLBTEXT,(WPARAM)i, (LPARAM)mode2);

                                        w2lanmode=(char*) malloc(strlen(mode));

                                //Guardamos de forma global el modo de funcionamiento elegido

                                        strcpy(w2lanmode,mode);
```

```
                                        //revisamos el si estamos en debug mode

                                        if(SendDlgItemMessage(hwnd, ID_BOTON4, BM_GETSTATE, 0, 0) ==
BST_CHECKED)
                                                DEBUG=1;
                                        else
                                                DEBUG=0;


                        //deshabilitariamos los botones  Go y Debug Mode , y habilitariamos el de Stop
                                        EnableWindow(hctrl3, FALSE);
                                        EnableWindow(hctrl5, TRUE);
                                        EnableWindow(hctrl7, FALSE);



                                        /*sprintf(cad, "mode: %s y debug: %i  -->%i ",w2lanmode,DEBUG,i);
                                        SendMessage(hctrl4, LB_ADDSTRING, 0, (LPARAM)cad);*/

                        //una vez recogidos los datos procedemos a lanzar la funcion W2lan pasandole el numero de
la
                        //interfaz seleccionada
                                        hilo=CreateThread(0,0,(void*)w2lan,i,0,0);

                                        if(hilo==NULL)
                                                {
                                MessageBox(hwnd,"Error al crear Thread" ,"W2lan", MB_OK);
                                }
                        break;
```

En el caso del botón *Stop* sencillamente se habilita el botón de *Go* y *Debug Mode*, deshabilita el propio botón de Stop y terminaría el hilo.

```
case ID_BOTON2:
                        //habilitamos el boton de Go y el de Debug Mode
                                        EnableWindow(hctrl3, TRUE);
                                        EnableWindow(hctrl7, TRUE);
                                        //eliminamos el funcionamiento del hilo
                                        TerminateThread(hilo,0);
                                         free(hilo);
                        //Deshabilitamos el boton de Stop
                                        EnableWindow(hctrl5, FALSE);
                                        break;
```

En el botón clear lanzara un mensaje para que el procedimiento de ventana elimine todas las entradas del listbox que se utiliza como ventana informativa.

```
case ID_BOTON3:  //Boton de Clear
                        //Eliminamos todas las entradas del listbox dejandolo vacio
                                        SendMessage(hctrl4, LB_RESETCONTENT, 0, 0);
                        break;
```

```
                                }//fin switch (Loword)
                                break;
```

Para terminar la función *WindowProcedure* también analiza los mensajes utililes para la información al usuario y buen funcionamiento del programa.

WM_PAINT salta cuando se dibuja la ventana en el ordenado por ello se utiliza para poner información por la ventana en este caso lo utiliza para indicarle el tipo de letra que habrá en toda la ventana y dos textos de información que irán cerca de su correspondiente elemento , "Mode" encima de su *listbox* y "Device" encima del *combobox.*

También es obligado para el correcto funcionamiento el tipo *WM_DESTROY* que cerrara la ventana y el caso default para los mensajes que no se ocupa el código escrito.

```
case WM_PAINT:

                hdc = BeginPaint(hwnd, &ps);
                SetTextColor(hdc, RGB(100,98,98));
                SetBkMode(hdc, TRANSPARENT);
                sprintf(cad, "Device:");
                TextOut(hdc, 140, 4, cad, strlen(cad));
                sprintf(cad, "Mode:");
                TextOut(hdc, 10, 4, cad, strlen(cad));
                EndPaint(hwnd, &ps);
                break;

        case WM_DESTROY:

                PostQuitMessage(0);    /* envía un mensaje WM_QUIT a la cola de mensajes */
                DeleteObject(hfont);

                break;



        default:                /* para los mensajes de los que no nos ocupamos */
                return DefWindowProc(hwnd, msg, wParam, lParam);
        }// find del switch general
        return 0;
}
```

5.3.2.2Código del W2lan

Una vez lanzado el hilo que ejecuta la función W2lan entraría en la segunda parte del código que seria la referenciada a la implementación del w2lan propiamente dicho. En esta implementación entra los archivos:

- *W2lan.c*
- *W2lan.h*
- *W2lan_callback.c*
- *W2lan_callback.h*
- *W2lan_core.c*
- *W2lan_core.h*
- *W2lan_list.c*
- *W2lan_list.h*
- *W2lan_help.c*
- *W2lan_help.h*

Todo el proceso empezaría en la función W2lan() que se encuentra en W2lan.c .La función recoge y prepara el nombre del interfaz que le pasara a la función *pcap_open_liv e* que abre y devuelve un descriptor del interfaz para poder controlarlo. También comprueba si la interfaz abierta es una Ethernet mediante la función pcap_datalink.

```
int w2lan (int numero)
{
	/* Beginning of parameter processing */

	/* don't let getopt() write to stderr */
	LPADAPTER	lpAdapter = 0; //Para abrir la interfaz mediante winscok2.dll
	int	j=0;
	int  segundos_wfso, msegundos_wfso;
	// int	fd;
	DWORD		dwErrorCode;
	char		AdapterName[8192];
	char		*temp,*temp1;
	int			AdapterNum=0,Open;
	ULONG		AdapterLength;
	PPACKET_OID_DATA  OidData; //estructura que utilizaremos para conseguir la mac
	BOOLEAN			tiempo_wfso= FALSE;
	char		AdapterList[Max_Num_Adapter][1024];


	/* To store getopt results from function main */
	char c;

	/* Device to be opened */
	char *device ;
	char device2 [500];


	/* libpcap error handling buffer */
	char errbuf[PCAP_ERRBUF_SIZE];
```

```
	/* data link type */
	int datalink = 0;

	/* promiscuous mode = 1 */
	int promisc_flag = 1;

	/* local packet capture descriptor */
	pcap_t *pd;

	/* counter for testing purposes */
	count = 1;
	int i;
	int cadena;
	DWORD tempo,res_wait;
	HANDLE fd_windows;

	/*Para controlar el tiempo transcurrido al empezar el wfso*/
	struct timeval time_wfso_start , time_wfso;

	for(apoyo=alldevs, i=0; i<numero ; apoyo=apoyo->next, i++);

	device=apoyo->name;
	cadena=strlen(device);


	for (i=7;i<cadena;i++)
	{
		device2[i-8]=device[i];
	}

	i=cadena-8;
	device2[i]='\0';


	/* open the user specified device */

	if ( ( pd = pcap_open_live(device2, SNAPLEN, 0,1, errbuf ))==NULL) //para windows el mode promiscuo debe estar a
0 osea desactivado
	{
		//mostrar();
		sprintf(cad, "ERROR: pcap_open_live error: %s",errbuf);
		mostrar(cad);
		return 0;
	}



	/* and keep the descriptor address */



	/* determine link layer type */
	if ( ( datalink = pcap_datalink( pd ) ) < 0 )
	{

		sprintf(cad, "ERROR_PCAP_DATALINK");
		mostrar(cad);
		MessageBox(hwnd,cad,"W2lan" , MB_OK);
		pcap_close(pd);
		//exit(ERROR_PCAP_DATALINK);
	}




	if ( datalink != DLT_EN10MB )
	{
		//mostrar();
		sprintf(cad, "ERROR: interface is %s, not DLT_EN10MB (aka Ethernet)",pcap_datalink_val_to_name(
datalink ));
		mostrar(cad);
```

```
                MessageBox(hwnd,cad,"W2lan" , MB_OK);
                pcap_close(pd);
                return 0 ;
                //exit(ERROR_DATALINK_TYPE);
        }



        /* if here, we have Ethernet */



        /* Determining if there is a wireless network underneath */

        /* number of wireless interfaces */
        int     interfaces = 0;

        /* temporary storage of wireless info */
        char    tmp[0x20]; /* it should be long enough */
        char    *lp;


        /* indicates user specified wireless device found*/
        int found = 0; /* not found yet */

        /* are wireless extensions present? */
        /* scan all the wireless adapters */
        /* if here, we have found the -user specified- underneath wireless adapter */
        /* is it configured in Ad-hoc mode ? */



        /* obtain our mac address */
```

La adquisición de la mac de la interfaz se realiza mediante *PacketOpenAdapter* que también abre la interfaz (Winsock2.dll) y la estructura *PPACKET_OID_DATA* que rellenada con la funcion *PacketRequest ,* conseguise facilemente la mac de la interfaz elegida , por medio de una de sus variables.

```
lpAdapter = PacketOpenAdapter(device2);

        if (!lpAdapter || (lpAdapter->hFile == INVALID_HANDLE_VALUE))
        {
                dwErrorCode=GetLastError();

                sprintf(cad,"Unable to open the adapter, Error Code : %lx",dwErrorCode);
                mostrar(cad);
    MessageBox(hwnd,cad,"W2lan" , MB_OK);
                return -1;
        }

        // Allocate a buffer to get the MAC adress

        OidData = malloc(6 + sizeof(PACKET_OID_DATA));
        if (OidData == NULL)
        {
                sprintf(cad,"error allocating memory!");
```

```
                mostrar(cad);
                 MessageBox(hwnd,cad,"W2lan" , MB_OK);
                PacketCloseAdapter(lpAdapter);
                return -1;
        }

        // Retrieve the adapter MAC querying the NIC driver


        OidData->Oid = OID_802_3_CURRENT_ADDRESS;
        OidData->Length = 6;
        ZeroMemory(OidData->Data, 6);
        PacketRequest(lpAdapter, FALSE, OidData);
        myMAC=(u_int8_t*) malloc(ETHER_ADDR_LEN);
        myMAC = OidData->Data;

        sprintf(cad,"The MAC address of the adapter is %.2x:%.2x:%.2x:%.2x:%.2x:%.2x",
                myMAC[0],
                myMAC[1],
                myMAC[2],
                myMAC[3],
                myMAC[4],
                myMAC[5]);
        mostrar(cad);
```

Una vez abierta la interfaz y adquirida la Mac, se procede a preparar el programa para dejarlo en escucha. Se preparan las listas, y algunas variables que serán de utilidad.

Las listas las se inicializan rápidamente llamando a la función correspondiente para su creación.

```
convl = (conversationslist *)conversationslistcreate();
        countl = (counterslist *)counterslistcreate();
        pendingframesl = (pendingframeslist *)pendingframeslistcreate();
        timersl = (timerslist *)timerslistcreate();
```

Antes de ejecutar la función que lo dejara esperando la respuesta, *WaitForSingleObject,* se consigue el descriptor o handle que utiliza Windows para sus funciones y se reducir el tamaño del buffer para que no tenga que esperar a que lleguen un número determinado de tramas para empezar a analizarlas. Esto se lleva a cabo con la función *pcap_getevent, que* facilita el handle de la interfaz abierta, y con la función *pcap_setmintocopy* asignando el valor de 15 que será el tamaño de tramas mínimo.

```
        fd_set set_read;
        int retvalue;
        int selectretvalue;

        struct pcap_pkthdr *cabecera;
        const u_char *paquete;
```

```
// Initialize the timer //

timerrequestglobal = (struct timeval *)malloc(sizeof(struct timeval));
timerrequest = NULL:


// Default starting value in reactive mode


counteractivewindow = 0;
activewindow = 1; // starting "speaking" w2lan



//Asignamos el valor maximo en bits deseado para que el buffer de la interfaz se descarge.
//Por defecto tiene uno muy grande y no nos interesa esperar a que se llene .15 es la longitud
//minima de uno de nuestros paquetes.

pcap_setmintocopy(pd,15);

//recogemos el handle de la funcion para que podemos ponernos en escucha.

fd_windows=(HANDLE) pcap_getevent(pd);

// Permanent regime first line //
```

Una vez preparado todo lo anterior el programa entraria en el bucle infinito que realializa la captura de paquetes y recibe la información para su posterior análisis.

Antes de ejecutar la función *WaitForSingleObject ,* debemos asignar a la variable *tempo* , el tiempo de espera para la función *WaitForSingleObject* , en el caso de que nos encontremos a la escucha de peticiones.

El problema con la función *WaitForSingleObject* radica en que el tiempo que nosotros le asignamos no lo descuenta, por lo que nosotros deberemos llevar el proceso de descuento e ir reasignándolo a la variable, para ello recurriremos a la función *gettimeofday* la cual nos dará el tiempo actual y lo guardaremos.

En caso de que no estemos en el momento de escucha le pasaremos a la función *WaitForSingleObject* ,a través de la variable *tempo* ,el valor INFINITE, el cual hará que la función no salte salvo que reciba alguna trama por la interfaz.

```
do
        {
    if (timerrequest != NULL)
                {

      if( tiempo_wfso)//vemos si estando en tiempo de espera debemos descontar tiempo
      {
      gettimeofday(&time_wfso);// recogemos el tiempo actual

      //se lo restamos a al valor anterior calculando el tiempo que ha pasado
      segundos_wfso=time_wfso.tv_sec - time_wfso_start.tv_sec;
      msegundos_wfso=time_wfso.tv_usec - time_wfso_start.tv_usec;
```

```
        //la diferencia es lo descontamos al tiempo de espera actual
        timerrequest->tv_sec= timerrequest->tv_sec-segundos_wfso;
        timerrequest->tv_usec= timerrequest->tv_usec-msegundos_wfso;


        }

        //transformamos los valores para asignarselo a WFSO
                                tempo = (int)(timerrequest->tv_sec);

                                tempo = tempo*1000000;

                                tempo = tempo + (int)(timerrequest->tv_usec);

                                //Comprobamos si el tiempo es negativo lo hacemos antes de la división puesto que esta vuelve el
valor a positivo
                                if(((int)(tempo))<0)
                                {
                                 //estando aqui quiere decir que hemos cumplido el tiempo de espera
                                 //ponemos tempo a 0 como precaucion
                                 tempo = (int) 0;//para que sea un 0 real


                                        sprintf(cad,"Tempo:Timerrequest has expired",tempo);
                                        mostrar(cad);

                                        timer_expired(pd);

                               continue;

                                MessageBox (hwnd,"no continue, nunca aqui" ,"W2lan" , MB_OK);
                                pcap_close(pd);
                                //Paramos el programa activando el boton stop
                                SendMessage(hwnd, WM_COMMAND , (WPARAM)106, 0);


        }
        else
        {
                                tempo= tempo/1000;//ponemos el tiempo el milisegundos en lugar de micro
        }


                                sprintf(cad,"Tiempo espera %d useg",tempo);
                                //Recogemos el tiempo actual antes de seguir y activamos tiempo_wfso para
                                //la proxima vez que entremos descuente el tiempo que haya transcurrido
                                gettimeofday(&time_wfso_start);
                                tiempo_wfso=TRUE;

                }
                else
                {
                                //Ponemos un tiempo de espera infinito
                                tempo = INFINITE;
                                sprintf(cad,"Tiempo espera infinito");
                                //ponemos a false ya que la proxima vez que entremos en (timerrequest != NULL)
                                //sera la primera vez y no tendremos que descontar el tiempo
                                tiempo_wfso=FALSE;
                }
```

Se percibe en el código anterior que el programa recoge un valor del tiempo antes de empezar la espera, en el caso que tengamos puesto un valor distinto a infinito, con la intención de al volver a pasar por ese punto, que será cuando reciba una trama, vuelva a recoger el valor del tpo en ese momento y, realizando una resta , hallar el tiempo deseado.

En caso que el tiempo sea negativo o cero el código hace que "salte" directamente a la función *timer_expired(pd)* ,evitando pasar por *WaitForSingleObject*, ya que este da errores cuando se le asigna un valor de 0 al tiempo, que seria la orden directa de que saltara por tiempo de espera cumplido, de esta forma elimina procedimiento innecesario y va a la función deseada , en este caso ,después reinicia el bucle ya que habrá tratado una situación como si hubiera pasado por *WaitForSingleObject*.

Llegado a este punto solo queda ejecutar la función *WaitForSingleObject* y esperar a que se den tres de los posibles resultados de la espera:

- Que la función de un fallo desconocido, en tal caso infamorma del error mediante *GetLastError* y saldría de w2lan.

```
res_wait=WaitForSingleObject(fd_windows,tempo);//hFile,tempo);


                    if(res_wait==WAIT_FAILED)
                    {
                              sprintf(cad, "Error WFSO , %d",GetLastError());
                              mostrar(cad);
                              MessageBox(hwnd,cad,"W2lan",MB_OK);
                              return 0;
                    }
```

- Que reciba una trama , la cual procedería a analizar mediante *w2lan_callback*.

```
if (res_wait == WAIT_OBJECT_0 )
                    {
                              // the fourth field is to indicate the network socket to the callback function
                              // many people say "useless field", but it is useful i.e. for the callback function
                              // to send packets in the same socket we opened to receive packets
                              retvalue = pcap_loop( pd , 1 , w2lan_callback, (unsigned char *)pd );

                    }
```

- O que es tiempo de espera se cumpla (en el caso de que tengamos alguno), que llevaría a ejecutar *timer_expired.*

```
                    else if ( res_wait == WAIT_TIMEOUT)

                    {
```

```
                                if (DEBUG)
                                  {
                                mostrar();
                                printf(cad,"DEBUG: w2lan.c: selectretvalue = %d. timerrequest has expired.",
selectretvalue );
                                mostrar(cad);
                                  }
                                sprintf(cad,"DEBUG: w2lan.c: timerrequest->tv_sec  -> %d", (int)(timerrequest-
>tv_sec) );
                                mostrar(cad);
                                sprintf(cad,"DEBUG: w2lan.c: timerrequest->tv_usec -> %d", (int)(timerrequest-
>tv_usec) );
                                mostrar(cad);

                                //Ejecutamos la funcion para resolver la expiracion de tiempo
                           if(  timer_expired(pd)==2)
                           {

          return 0;

        }


                        }
                        //else if ( res_wait == WAIT_ABANDONED )
                        //  {

                        //
                        //  }
                        else   //A ELIMINAR CUANDO VAYA BIEN EL PROGRAMA y poner solo else arriba
                        {
                                MessageBox(hwnd,"That should not happen :-(" ,"W2lan", MB_OK);
                        //  pcap_close(pd);
                        //  return 0;
                        }



          }while (1);

          MessageBox(hwnd,"NEVER_HERE!Stable version should never arrive here." ,"W2lan", MB_OK);
          return 0;

}
```

El segundo caso, que será el mas común de los tres, recibir alguna trama, esto lleva a la función *w2lan_callback* la cual analizara que tipo de trama se ha recibido llevándo a 4 casos posibles:

- Un anuncio w2lan, la cual se analizara en *announce_received*.
- Una petición w2lan, que recogera en *request_received.*
- Datos w2lan, que analizara en *data_received.*

```
void w2lan_callback( unsigned char *extra , const struct pcap_pkthdr *pkthdr , const unsigned char *packet )
{
	struct ether_header *eptr;
	eptr = (struct ether_header *) packet;

	u_int8_t  *macdst;
	macdst = (u_int8_t *)malloc( ETHER_ADDR_LEN*sizeof(u_int8_t) );

	u_int8_t  *macsrc;
	macsrc = (u_int8_t *)malloc( ETHER_ADDR_LEN*sizeof(u_int8_t) );

	u_int16_t mactype = 0;

	u_int8_t  *macpayload;
	macpayload = (u_int8_t *)malloc( ETH_DATA_LEN*sizeof(u_int8_t) );


	/* take the Ethernet destination address */
	memcpy( macdst , (const struct ether_addr *)&eptr->ether_dhost , ETHER_ADDR_LEN );
	/* take the Ethernet source address */
	memcpy( macsrc , (const struct ether_addr *)&eptr->ether_shost , ETHER_ADDR_LEN );
	/* take the Ethernet frame type in host format */
	mactype = ntohs( eptr->ether_type );
	/* take the Ethernet payload */
	memcpy( macpayload, packet+14 , (pkthdr->len)-14 );



	// packet capture descriptor
	pcap_t *pd = (pcap_t *)extra;

	if ( mactype == ETH_P_W2LAN_ANNOUNCE )
	{
		// W2LAN Announce received

		// activate active window for reactive mode
		activewindow = 1;
		counteractivewindow = 0;

		announce_received( extra , pkthdr , packet );

	}
	else if ( mactype == ETH_P_W2LAN_REQUEST )
	{
		// W2LAN Request received


		// activate active window for reactive mode
		activewindow = 1;
		counteractivewindow = 0;



		request_received( extra , pkthdr , packet );

	}
	else if ( mactype == ETH_P_W2LAN_DATA )
	{
		// W2LAN data received

		// activate active window for reactive mode
		activewindow = 1;
		counteractivewindow = 0;

		data_received( extra , pkthdr , packet );

	}
```

Y por último, una trama que no obedece a ningún caso de los anteriormente citados. En tal caso y según el método que se ha seleccionado en la interfaz se realizaran las siguientes operaciones:

- Modo Reactivo: Generara un anuncio mediante *generate_announce*, solo en el caso de haber recibido alguna trama del tipo w2lan antes, en cuyo caso se habría ejecutado *activewindow = 1;* dando la opción en el modo reactivo a actuar generando un anuncio.Tambien se debe cumplir que la trama recibida tenga como origen la interfaz donde trabaja el programa.

- Modo Passive: En este caso no efectuaría ninguna acción.

- Modo Always: Al llegar cualquier trama que no fuese w2lan, se generaría automáticamente un anuncio.

```
else
 {
            // non W2LAN frame received

            if ( strcmp( w2lanmode, "reactive" ) == 0 )
          {

                        // Typical for clients, i.e. windows/linux in mobile PCs
                        // in reactive mode, if we are in active window and source=myMAC generate announce



                        if ( counteractivewindow == MAX_NON_W2LAN_FRAMES )
                        {
                                    counteractivewindow = 0;
                                    activewindow = 0;
                        }

                        counteractivewindow = counteractivewindow + 1;



                        if ( ( activewindow == 1) && ( memcmp( macsrc, myMAC, ETHER_ADDR_LEN ) == 0 ) )
                        {


                                    generate_announce( extra , pkthdr , packet );


                        }
            }
            else if ( strcmp( w2lanmode, "passive" ) == 0 )
            {

                        // Typical for non-gateway, helper machines, coverage expanders
                        // in passive mode, do nothing ( done :-) )
            }
            else if ( strcmp( w2lanmode, "always" ) == 0 )
            {

                        // Typical for gateway, linux server (dhcp+nat+...)
                        // in always mode, if source=myMAC generate announce

                        if ( memcmp( macsrc, myMAC, ETHER_ADDR_LEN ) == 0 )
                        {
```

```
                                        generate_announce( extra , pkthdr , packet );

                                }
                        }
                }
                free( macdst );
                free( macsrc );
                free( macpayload );

}
```

Una vez recogida la trama y analizado el tipo, o habiendo cumplido el tiempo de espera se encontraría en uno de las siguientes situaciones o funciones:

- announce_received
- request_received
- data_received
- generate_announce
- timer_expired

Announce_received: Comprueba que sea un anuncio w2lan correctamente formado, mira que el tamaño sea el correcto y que sea un broadcast.

```
void announce_received( unsigned char *extra , const struct pcap_pkthdr *pkthdr , const unsigned char *packet ) {

        /* packet capture descriptor */
        pcap_t *pd = (pcap_t *)extra;

        /* cast to ethernet header */
        struct ether_header *eptr;
        eptr = (struct ether_header *) packet;

        int pcap_sendpacket_retcode;
        /* if malformed return */
        if ( pkthdr->len != ETH_P_W2LAN_ANNOUNCE_LEN ) {

                return;
        }

        /* if not broadcast return */
        if ( memcmp( eptr->ether_dhost, ether_aton( "FF:FF:FF:FF:FF:FF" ), ETHER_ADDR_LEN ) != 0 ) {

                return;
        }

        /* here we have a valid w2lan announce */
```

Una vez comprobado esto recupera la Id de conversación de la trama y mira si la esta ya en la lista de *conversationslist*.En caso de no tener dicha conversación la añade con dicha Id a la lista de *conversationslist* y crea una trama *request* W2Lan para enviarla.

En caso de poseer dicha Id en esa lista no se realizaría ninguna acción.

```
        // check if conversation is in conversationslist return else ...(lots)
        /* packet+28 is the position in packet where conversationID appears */
        if ( conversationslistsearch( (u_int8_t *)(packet+28) , (conversationslist)convl ) == NULL ) {


                // not in the list. add to the conversationslist

                u_int16_t ethernettypeaux;
                memcpy( (&ethernettypeaux), packet+26, 2);
                conversationslistadd( (u_int8_t *)(packet+28) , (u_int8_t *)(packet+14), (u_int8_t *)(packet+20), ntohs(
ethernettypeaux ) , (conversationslist *)&convl );

        } else {
                // in the list, do nothing else

                return;
        }

        unsigned char *w2lanreqpacket;


        w2lanreqpacket = (unsigned char *)malloc(ETH_P_W2LAN_ANNOUNCE_LEN*sizeof(unsigned char));
        assert(w2lanreqpacket != NULL);
        /* Initialize reqpacket */
        memcpy( w2lanreqpacket, "", 1);


        memcpy(  w2lanreqpacket , eptr->ether_shost , ETHER_ADDR_LEN );

        memcpy( w2lanreqpacket + ETHER_ADDR_LEN , myMAC , ETHER_ADDR_LEN );

        u_int16_t reqtype = htons( ETH_P_W2LAN_REQUEST );
        memcpy( w2lanreqpacket+ETHER_ADDR_LEN+ETHER_ADDR_LEN , (unsigned char *)(&reqtype) , 2 );

        memcpy( w2lanreqpacket+ETHER_ADDR_LEN+ETHER_ADDR_LEN+2 , packet+28 ,
ETH_P_W2LAN_CONVERSATIONID_LEN );


        if ( ( pcap_sendpacket_retcode = pcap_sendpacket( pd , w2lanreqpacket , ETH_P_W2LAN_REQUEST_LEN ) ) < 0 )
        {
          sprintf(cad,"ERROR sending the packet: %s", pcap_geterr( pd ));
                mostrar(cad);

        }

        free( w2lanreqpacket );

}
```

Request_received: Comprueba si se encuentra con un paquete de *request* valido, comprobando su tamaño. Una vez verificado recoge el Id de conversación de la trama y busca en la lista de *counterslist*, en caso de que poseer una entrada en esta lista con esa Id incrementamos el contador de la esa entrada.

```
void request_received( unsigned char *extra , const struct pcap_pkthdr *pkthdr , const unsigned char *packet ) {

	/* packet capture descriptor */
	//   pcap_t *pd = (pcap_t *)extra;

	/* cast to ethernet header */
	struct ether_header *eptr;
	eptr = (struct ether_header *) packet;

	//   int pcap_sendpacket_retcode;

	/* if malformed return */
	if ( pkthdr->len != ETH_P_W2LAN_REQUEST_LEN ) {
		if (DEBUG) {
			sprintf(cad,"DEBUG: w2lan_core.c: request_received(): malformed request packet (size = %d
bytes).\n", pkthdr->len );
			mostrar(cad);
		}
		return;
	}

	/* here we have a valid w2lan request */

	if ( counterslistsearch( (u_int8_t *)(packet+14) , (counterslist)countl ) == NULL ) {

		// not in the list. do nothing
		return;
	} else {

		// in the list. increase counter associated with conversationid
		counterslistincrease( (u_int8_t *)(packet+14) , (counterslist)countl );
	}

}
```

Data_received : Comprueba que se encuentra con una trama de datos W2lan bien formada, con una longitud que no exceda el máximo y que no se trate de una trama broadcast.

```
void data_received( unsigned char *extra , const struct pcap_pkthdr *pkthdr , const unsigned char *packet ) {

	/* packet capture descriptor */
	pcap_t *pd = (pcap_t *)extra;

	/* cast to ethernet header */
	struct ether_header *eptr;
	eptr = (struct ether_header *) packet;

	int pcap_sendpacket_retcode;

	if ( ( (pkthdr->len) >= ETH_P_W2LAN_DATA_MAX_LEN ) ) {

		return;
	}

	/* if not broadcast return */
	if ( ( memcmp( eptr->ether_dhost, (u_int8_t *)ether_aton( "FF:FF:FF:FF:FF:FF" ), ETHER_ADDR_LEN ) ) != 0 ) {
```

```
                return;
        }
```

Con la Id de la trama recibida busca en la lista de *conversationslist* para ver si existe, en caso de no existir no haría nada y el función acabaría, al contrario en caso de que exista verifica si poseemos ya dicha información guardada o aun esta pendiente de recibirla .En caso de que aun este pendiente de recibir la información , la marca como no pendiente.

```
/* here we have a w2lan data packet.  */
        // see if conversation is pending...
        /* packet+6 is the position in packet where conversationID appears */
        conversationslist conv = conversationslistsearch( (u_int8_t *)(packet+6) , (conversationslist)convl );

        if ( conv == NULL ) {
                // conversation  not in list. do nothing

                return;
        }
        else {
                // see if the conversation is pending

                if ( conversationpending( conv ) ) {
                        // conversation is pending. we have to clear its pending field

                        conversationclearpending( conv );

                        }

                        char multicast;
                        multicast = conversationgetmacdst( conv )[0];

                        multicast = multicast & 0x01;
```

A continuación comprueba si se trata de una conversación que el mismo haya originado, en el caso de que sea propia, la volverá a reconstruir y enviar y con ello finalizara la función.

```
if ( ( memcmp( conversationgetmacdst( conv ), ether_aton( "FF:FF:FF:FF:FF:FF" ), ETHER_ADDR_LEN ) == 0 ) || ( memcmp(
conversationgetmacdst( conv ), myMAC , ETHER_ADDR_LEN ) == 0) || ( multicast == 1 ) ) {
                                        // the frame is ours.  we have to reconstruct the ethernet frame and sendpacket it

                                        unsigned char *ethernetframe;

                                        ethernetframe = (unsigned char *)malloc(ETHER_MAX_LEN*sizeof(unsigned char));
                                        assert(ethernetframe != NULL);

                                        /* Initialize datapacket */
                                        memcpy( ethernetframe, "", 1);

                                        memcpy( ethernetframe , conversationgetmacdst( conv ) , ETHER_ADDR_LEN );

                                        memcpy( ethernetframe + ETHER_ADDR_LEN , conversationgetmacsrc( conv ) ,
ETHER_ADDR_LEN );

                                        u_int16_t ethernettype = htons( conversationgetmactype( conv ) );
                                        memcpy( ethernetframe+ETHER_ADDR_LEN+ETHER_ADDR_LEN , (u_int8_t
*)(&ethernettype) , 2 );
```

```
                                        memcpy( ethernetframe + ETHER_HDR_LEN , packet + ETHER_HDR_LEN ,
(pkthdr->len)-ETHER_HDR_LEN );

                                        if ( ( pcap_sendpacket_retcode = pcap_sendpacket( pd , ethernetframe , pkthdr->len ) )
< 0 ) {

            sprintf(cad,"\nError sending the ethernet frame: %s\n", pcap_geterr( pd ));
            mostrar(cad);

                                        }

                                        free( ethernetframe );

                                }
```

Sino, en caso de que no sea propia procede a guardar dicha información en la lista de *pendingframes* con la Id de la conversación y crea un nuevo miembro en la lista de *counterslist* también con dicha Id.

```
else {
                                        // ethernet frame not ours. do nothing


                                // conversation was pending. we prepare next announce adding ethernet payload to pending list, a
new counter to counterslist, and a new timer to timerslist

                                // TODO check the lists

                                pendingframeslistput( (u_int8_t*)(packet+6) , (u_int8_t*)(packet+14) , (pkthdr->len)-
ETHER_HDR_LEN , (pendingframeslist *)&pendingframesl );

                               counterslistintoaslast( (u_int8_t*)(packet+6) , (counterslist *)&countl );
```

Luego captura el tiempo actual y le añade el tiempo de espera asignado por defecto. Comprueba si la lista de tiempos esta vacía, en tal caso asigna el valor de tiempo espera actual lo igualaría al de por defecto. Y por ultimo introduce al final de la lista de *timerslist* el tiempo que hemos calculado para esta información junto a la Id correspondiente.

```
                               //  Timers...

                                struct timeval t1;
                                int gettimeofdayretcode = gettimeofday( &t1);
                                if ( gettimeofdayretcode == 0 ) {

                                } else {
                                        sprintf( cad, "ERROR: gettimeofday failed (ret. code %d).\n", gettimeofdayretcode );
                                        mostrar(cad);
                                        MessageBox(hwnd,cad,"W2lan" , MB_OK);
                                        exit(GET_TIME_OF_THE_DAY_ERROR);
                                }
                                ////////////////

                                // prepare the local timer
                                struct timeval aux;
                                aux.tv_sec = TIMEOUT_REQUEST_SECONDS;
                                aux.tv_usec = TIMEOUT_REQUEST_USECONDS;

                                timeval_add( &t1, &t1, &aux );

                                if ( timerslistempty( timersl ) == 1 ) {
                                        // list empty. set the global timer
```

```
                                timerrequest = timerrequestglobal;
                                timerrequest -> tv_sec = TIMEOUT_REQUEST_SECONDS;
                                timerrequest -> tv_usec = TIMEOUT_REQUEST_USECONDS;
                        }

                        timerslistintoaslast( (u_int8_t*)(packet+6) , &t1 , (timerslist *)&timersl );

                        ////////////////////
```

Una vez iniciadas las listas para dicha Id e iniciado el tiempo de espera lanza un anuncio W2Lan con el fin para recibir request W2Lan para esa información recibida.

```
                        // after updating lists,  we reannounce the ethernet frame.
                        unsigned char *w2lanannouncepacket;

                        w2lanannouncepacket = (unsigned char
*)malloc(ETH_P_W2LAN_ANNOUNCE_LEN*sizeof(unsigned char));
                        assert(w2lanannouncepacket != NULL);

                        /* Initialize announce packet */
                        memcpy( w2lanannouncepacket, "", 1);

                        memcpy( w2lanannouncepacket , (u_int8_t *)ether_aton("FF:FF:FF:FF:FF:FF") ,
ETHER_ADDR_LEN );

                        memcpy( w2lanannouncepacket + ETHER_ADDR_LEN , myMAC , ETHER_ADDR_LEN );


                        u_int16_t announcetype = htons( ETH_P_W2LAN_ANNOUNCE );
                        memcpy( w2lanannouncepacket+ETHER_ADDR_LEN+ETHER_ADDR_LEN , (u_int8_t
*)(&announcetype) , 2 );


                        memcpy( w2lanannouncepacket + ETHER_HDR_LEN , (u_int8_t
*)conversationgetmacdst(conv) , ETHER_ADDR_LEN );


                        memcpy( w2lanannouncepacket + ETHER_HDR_LEN + ETHER_ADDR_LEN, (u_int8_t
*)conversationgetmacsrc(conv) , ETHER_ADDR_LEN );


                        u_int16_t storedethernettype = htons( conversationgetmactype( conv ) );
                        memcpy(
w2lanannouncepacket+ETHER_HDR_LEN+ETHER_ADDR_LEN+ETHER_ADDR_LEN , (u_int8_t *)(&storedethernettype) , 2
);
                        /* packet+6 is the position in packet where conversationID appears */

                        memcpy(
w2lanannouncepacket+ETHER_HDR_LEN+ETHER_ADDR_LEN+ETHER_ADDR_LEN+2 , packet+6 ,
ETH_P_W2LAN_CONVERSATIONID_LEN );

                        if ( ( pcap_sendpacket_retcode = pcap_sendpacket( pd , w2lanannouncepacket ,
ETH_P_W2LAN_ANNOUNCE_LEN ) ) < 0 ) {
                                sprintf(cad,"\nError sending the packet: %s\n", pcap_geterr( pd ));
                                mostrar(cad);
                        }

                        free( w2lanannouncepacket );
```

```
                        } else {
                                // conversation not pending. do nothing



                                return;
                        }  // else - conversation not pending
                }  // else - not in conversationslist

}  // datareceived()
```

generate_announce : Esta función solo se ejecuta en el caso de recibir una trama que no sea w2lan y el programa este en el modo passive o always .

Su objetivo es crear trafico y que haya un intercambio de tramas para comprobar el correcto funcionamiento del programa. Por ello que la función genera un anuncio de una información donde la Id de la conversación se crea aleatoriamente y se registra en las listas.

La función debe crear una conversación nueva, por ello crea los datos y la Id de la conversación aleatoriamente, al mismo tiempo que guarda la Id en la lista de *conversaciones*, poniéndola como pendiente.

```
void generate_announce( unsigned char *extra , const struct pcap_pkthdr *pkthdr , const unsigned char *packet ) {


        /* packet capture descriptor */
        pcap_t *pd = (pcap_t *)extra;

        /* cast to ethernet header */
        struct ether_header *eptr;
        eptr = (struct ether_header *) packet;

        int pcap_sendpacket_retcode;


        unsigned char *ethernetframe;

        ethernetframe = (unsigned char *)malloc(ETHER_MAX_LEN*sizeof(unsigned char));
        assert(ethernetframe != NULL);
        /* Initialize datapacket */
        memcpy( ethernetframe, "", 1);

        memcpy( ethernetframe , packet+ETHER_HDR_LEN , (pkthdr->len)-ETHER_HDR_LEN );

        // Random numbers are XX XX XX XX XX XX
        long longconversationID = rand(); // for the 4 least significative bytes
        long longconversationIDhigh = rand(); // for the 2 most significative bytes

        u_int8_t conversationID[ETH_P_W2LAN_CONVERSATIONID_LEN] = { 0xFF ,0xFF ,0xFF ,0xFF ,0xFF ,0xFF };
        memcpy( conversationID , (u_int8_t *)(& longconversationIDhigh), ETH_P_W2LAN_CONVERSATIONID_LEN );
        memcpy( conversationID+2 , (u_int8_t *)(& longconversationID), ETH_P_W2LAN_CONVERSATIONID_LEN );

        // Add conversationID to the conversationslist convl as pending
        u_int16_t ethernettypeaux;
        memcpy( (&ethernettypeaux), packet+12, 2);
        conversationslistadd( conversationID , (u_int8_t *)(packet) , (u_int8_t *)(packet+6) , ntohs(ethernettypeaux) ,
(conversationslist *)&convl );
```

Seguidamente añade en la lista de *pendingframes*, con la información creada, y en la lista de *contadores*, registrándola en la última posición, sendas entradas con la Id generada.

También tras recoger el tiempo actual y añadirle el de espera por defecto, lo registra en la lista de *tiempo,* introduciéndola en el último lugar de esta. Durante este proceso, y antes de almacenar la nueva entrada, comprueba si la lista de tiempos esta vacía, si lo esta inicializa el tiempo de espera con el tiempo de espera por defecto.

```
// Add conversationID+frame to the pending list pendingframesl
        pendingframeslistput( conversationID , (u_int8_t*)(ethernetframe) , (pkthdr->len)-ETHER_HDR_LEN ,
(pendingframeslist *)&pendingframesl );
        // Add new counter to countl
        counterslistintoaslast( conversationID , (counterslist *)&countl );

        // Add new timer to timersl

        struct timeval t1;
        int gettimeofdayretcode = gettimeofday( &t1);
        if ( gettimeofdayretcode == 0 ) {

        } else {
                sprintf( cad, "ERROR: gettimeofday failed (ret. code %d).\n", gettimeofdayretcode );
                mostrar(cad);
                MessageBox(hwnd,cad,"W2lan" , MB_OK);

                exit(GET_TIME_OF_THE_DAY_ERROR);
        }

        // prepare the local timer
        struct timeval aux;
        aux.tv_sec = TIMEOUT_REQUEST_SECONDS;
        aux.tv_usec = TIMEOUT_REQUEST_USECONDS;
        timeval_add( &t1, &t1, &aux );

        if ( timerslistempty( timersl ) == 1 ) {
                // list empty. set the global timer
                assert(timerrequest == NULL);

                timerrequest = timerrequestglobal;
                timerrequest -> tv_sec = TIMEOUT_REQUEST_SECONDS;
                timerrequest -> tv_usec = TIMEOUT_REQUEST_USECONDS;
        }


        timerslistintoaslast( conversationID , &t1 , (timerslist *)&timersl );
```

Una vez finalizado todo el proceso de creación y registro procede a elaborar y mandar una trama de anuncio W2lan.

```
		// TODO send announce
		unsigned char *w2lanannouncepacket;

		w2lanannouncepacket = (unsigned char *)malloc(ETH_P_W2LAN_ANNOUNCE_LEN*sizeof(unsigned char));
		assert(w2lanannouncepacket != NULL);

		// Initialize announce packet
		memcpy( w2lanannouncepacket, "", 1);

		memcpy( w2lanannouncepacket , (u_int8_t *)ether_aton("FF:FF:FF:FF:FF:FF") , ETHER_ADDR_LEN );


		memcpy( w2lanannouncepacket + ETHER_ADDR_LEN , myMAC , ETHER_ADDR_LEN );


		u_int16_t announcetype = htons( ETH_P_W2LAN_ANNOUNCE );
		memcpy( w2lanannouncepacket+ETHER_ADDR_LEN+ETHER_ADDR_LEN , (u_int8_t *)(&announcetype) , 2 );

		memcpy( w2lanannouncepacket + ETHER_HDR_LEN , (u_int8_t *)(packet) , ETHER_ADDR_LEN );


		memcpy( w2lanannouncepacket + ETHER_HDR_LEN + ETHER_ADDR_LEN, (u_int8_t
*)(packet+ETHER_ADDR_LEN) , ETHER_ADDR_LEN );

		memcpy( w2lanannouncepacket+ETHER_HDR_LEN+ETHER_ADDR_LEN+ETHER_ADDR_LEN , (u_int8_t
*)(packet+ETHER_ADDR_LEN+ETHER_ADDR_LEN) , 2 );

		memcpy( w2lanannouncepacket+ETHER_HDR_LEN+ETHER_ADDR_LEN+ETHER_ADDR_LEN+2 ,
conversationID , ETH_P_W2LAN_CONVERSATIONID_LEN );

		if ( ( pcap_sendpacket_retcode = pcap_sendpacket( pd , w2lanannouncepacket , ETH_P_W2LAN_ANNOUNCE_LEN
) ) < 0 ) {
			sprintf(cad,"\nError sending the packet: %s\n", pcap_geterr( pd ));
			mostrar(cad);
		}

		free( w2lanannouncepacket );
		free( ethernetframe );
		return;

}
```

Timer_expired: Esta función se ejecuta solo al vencer el tiempo de espera que hemos inicializado con el fin de recibir tramas W2Lan request para información que posee ya el programa.

Timer_expired recoge la Id y el tiempo calculado del primer miembro que tiene en la lista y del cual, se supone, que ha vencido el tiempo de espera asignado, eliminando al mismo tiempo dicho miembro de la lista de tiempos. Después comprueba si la lista ha quedado vacía. En el caso de que aun posea algún miembro recoge el tiempo inicial calculado de este y procede a restarlo con el actual con el fin de comprobar que tiempo de espera le queda a dicha información hasta que venza. Una vez calculado lo asigna al tiempo de espera. En el caso de que la lista de tiempos este vacía pondríamos el tiempo de espera infinito.

```
int timer_expired(pcap_t *pd) {


        // Timers...

        struct timeval t;
        struct timeval t1;
        struct timeval t2;
        u_int8_t conversationID[ETH_P_W2LAN_CONVERSATIONID_LEN];


        // extract expired timer

        timerslistfirstinqueue( conversationID , &t1, (timerslist *)&timersl );


        if ( timerslistempty( timersl ) == 0 )
        {
                timerslistsuccessor( conversationID , &t1 , timersl );

                int gettimeofdayretcode = gettimeofday( &t2);

                if ( gettimeofdayretcode == 0 )
                {

                }
                else
                {

                        sprintf( cad, "ERROR: gettimeofday failed (ret. code %d).\n", gettimeofdayretcode );
                        mostrar(cad);
                        MessageBox(hwnd,"gettimeofday failed!","W2lan" , MB_OK);
                        return 2 ;
                }


                if ( ( timeval_subtract ( &t , &t1 , &t2 ) ) != 0 )
                {
                        MessageBox(hwnd,"timeval_subtract fail!","W2lan" , MB_OK);
                        sprintf( cad, "ERROR: timeval_subtract returned a negative time.\n");
                        mostrar(cad);
                        return 2 ;
                        //exit(TIMEVAL_SUBSTRACT_ERROR);
                }


                if ( timerrequest == NULL )
                {
                        timerrequest = timerrequestglobal;
                }
                timerrequest -> tv_sec = t.tv_sec;
                timerrequest -> tv_usec = t.tv_usec;
                //Al ser un nuevo tiempo de espera inicializamos el tiempo de inicio para
                //calcular el transcurrido
                 gettimeofday(&time_wfso_start);
        }
        else
        {
                timerrequest = NULL;
        }
```

Una vez resuelto la asignación del próximo tiempo de espera, el programa continua con el tratamiento de la Id por la cual ha saltado la función. Recoge de la lista de *contadores*, el número de peticiones que ha recibido para esa Id en su tiempo de espera, al mismo tiempo que lo elimina de la lista. Y recupera la información de la lista de *pendingframes,* eliminando la entrada de dicha id*,* con el fin de mandar luego dicha

información si es necesario. Finalmente comprueba si hay alguna petición, en caso afirmativo manda la una trama W2lan de datos con la información de esa conversación.

```
	// counters. Here we also obtain conversationID


	int numrequests;
	numrequests = counterslistfirstinqueue( conversationID , (counterslist *)&countl );

	// pending frames lists...

	int pcap_sendpacket_retcode;
	u_int16_t w2landatapacketsize;
	unsigned char *w2landatapacket;
	w2landatapacket = (unsigned char *)malloc(ETHER_MAX_LEN*sizeof(unsigned char));
	assert(w2landatapacket != NULL);
	/* Initialize datapacket */
	memcpy( w2landatapacket, "", 1);


	pendingframeslistextract( conversationID , w2landatapacket+ETHER_HDR_LEN , &w2landatapacketsize,
(pendingframeslist *)&pendingframesl );

	if ( numrequests > 0 ) {

		memcpy( w2landatapacket , ether_aton( "FF:FF:FF:FF:FF:FF" ) , ETHER_ADDR_LEN );


		memcpy( w2landatapacket+ETHER_ADDR_LEN , conversationID , ETHER_ADDR_LEN );


		u_int16_t datatype = htons( ETH_P_W2LAN_DATA );
		memcpy( w2landatapacket+ETHER_ADDR_LEN+ETHER_ADDR_LEN , (unsigned char *)(&datatype) , 2
);


		// check ethernet data field


		if ( ( pcap_sendpacket_retcode = pcap_sendpacket( pd , w2landatapacket ,
w2landatapacketsize+ETHER_HDR_LEN ) ) < 0 ) {
			sprintf(cad,"\nError sending the packet: %s\n", pcap_geterr( pd ));
			mostrar(cad);
		}


	}

	free(w2landatapacket);
	return;

}
```

Llegado a este punto el programa habrá recorrido todas las opciones posibles dentro de los hilos de funcionamiento del mismo , y solo le quedara repetir las diferentes acciones cada vez que le llegue algun paquete.El programa solo finalizara cuando el usuario lo indique.

6. Conclusiones y líneas futuras:

La conversión de Linux a Windows ha sido llevada casi en su totalidad, consiguiendo un programa que casa con el entorno de Windows y cumple con las mismas funciones esenciales que las que poseía en Linux .Se ha manteniendo el mismo lenguaje y todo el procemiento principal del programa.

Sin embargo algunos aspectos podrían ser comtemplados para una mejora mas adelante como podría ser el control de tiempo de espera, ya que al tener que prescindir de la función select se ha tenido que recurrir a un control de tiempo algo rudimentario y que en ocasiones ocasiona un retardo algo mayor que el deseado en esto influye la cantidad de trafico existente ,llegando incluso a ,en ocasiones de recuperar los nuevos tiempos de espera de peticiones en cola , dar valores de espera negativos, nunca muy altos, los cuales en esta adaptación se ha optado por tomar como tiempo vencido y proceder inmediatamente a analizar si hubo peticiones.

Una posible solución es el trabajo que se esta realizando para poder utilizar la función select en Windows que esta en un estado avanzado pero no termina de funcionar correctamente en el entorno.No descuenta bien los tiempos ni funcionan correctamente los descriptores que se le facilitan.

Uno de los objetivos que no se han cumplido ha sido la comprobación de que nos encontamos en modo ad-hoc, el cual es esencial para el funcionamiento de nuestro protocolo ya que se basa en este tipo de redes no siendo valido las de tipo infrastructura.Por lo tanto era necesario una comprovacion antes de la puesta en funcionamiento de que la interfaz funcionara en ese modo.

Como desarrollo del propio W2lan nos encontraríamos con una posible evolución que seria su intregracion dentro de la pila de protocolos de Windows de forma que no fuese necesario su puesta de funcionamiento ni manipulación del usurario , esta integración seria útil para dipositivos comerciales que necesitaran llevar a cabo la distribución de información multimedia de forma que el dispositivo , lo realizara automáticamente sin que el usuario tuviera que interferir en ningún momento.