Capítulo 6: Diseño del monitor central

6.1 Introducción

En este capítulo se describen las técnicas empleadas para desarrollar la interfaz del monitor central. Puesto que el funcionamiento de éste ya se ha explicado en el capítulo 3, aquí sólo se expondrán aquellos contenidos relativos a la programación de la interfaz.

6.2 La interfaz gráfica

La interfaz gráfica de este programa pretende ser lo más simple posible para facilitar su uso al operario. Está dividida en dos zonas. Una para la PMU y otra para la RMU. No obstante, ya que el objetivo de este software es el envío y recepción de ficheros independientemente del sistema destino, se ha intentado modularizar su funcionamiento en la medida de lo posible, usando para ello funciones en el envío y recepción con parámetros diferentes en cada caso.

En la siguiente figura se puede ver la estructura de ésta. El primer cuadro de texto consiste en la dirección IP del sistema al que queremos enviar o del que queremos recibir. En el caso de la RMU sólo existe la opción de “Mandar configuración” ya que la PMU es la única que calcula los resultados y presenta una interfaz donde elegir los resultados.

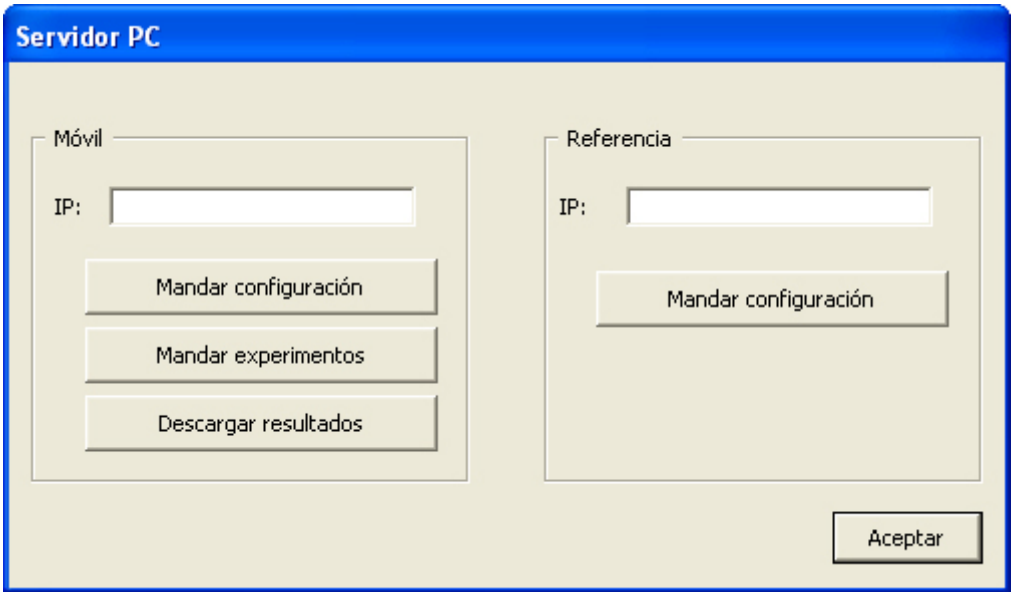

Figura 6.1 Interfaz del monitor PC

6.3 Envío de ficheros hacia la CPU

Si el operario pulsa sobre el botón “Mandar configuración” o “Mandar experimentos” se procederá al envío de un fichero, hacia la CPU determinada por la dirección IP introducida en el cuadro de diálogo.

Por lo tanto el primer paso es proporcionar al usuario un cuadro de diálogo donde éste pueda seleccionar el fichero que desea enviar. Se ha hecho uso de la clase CFileDialog especificando una serie de parámetros en su constructor para filtrar los ficheros *.txt y seleccionar el directorio inicial.

Si el usuario pulsa sobre la opción “Abrir” tras seleccionar un fichero, se llamará a la función ‘envia_fichero’ pasándole como parámetros la dirección IP de destino y el nombre del fichero. Esta función devolverá un valor TRUE si el envío se ha realizado de forma correcta o un valor FALSE en caso contrario.

El envío del fichero se realiza mediante una conexión TCP en un puerto determinado por la macro PUERTO_TCP_ENVIAR, especificada al comienzo del programa. El programa ejecutado en el PC actuará de servidor, por lo que se pondrá a la escucha mediante la función Listen perteneciente a la clase CSocket. A continuación se muestra el código utilizado en la función enviar.

```
bool CFicherosDlg::envia_fichero(CString nombre_fichero, CString ip)
{
	// Creamos el socket
	CSocket sockSrvr;
	sockSrvr.Create(PUERTO_TCP_ENVIAR);
```

```
	sockSrvr.Listen();

	// Usamos otro CSocket para aceptar la conexión
	CSocket sockConnection;
	sockSrvr.Accept(sockConnection);

	// Variables locales utilizas en la transferencia del fichero
	// Indica si la operación se ha producido correctamente o no
	BOOL bRet = TRUE;

	// Longitud del fichero y número de bytes restantes por enviar
	int fileLength, cbLeftToSend;

	// Puntero hacia el buffer de memoria
	// La memoria se reserva tras enviar  el tamaño del fichero
	BYTE* sendData = NULL;

	CFile sourceFile;
	CFileException fe;
	BOOL bFileIsOpen = FALSE;

	if( !( bFileIsOpen = sourceFile.Open( nombre_fichero,
		   CFile::modeRead | CFile::typeBinary, &fe ) ) )
	{
		TCHAR strCause[256];
		fe.GetErrorMessage( strCause, 255 );
		AfxMessageBox(strCause);
		/* Si hay algún error en la apertura del fichero lo manejamos aquí */
		bRet = FALSE;
		goto PreReturnCleanup;
	}

	// Primero enviamos la longitudo del fichero
	fileLength = sourceFile.GetLength();
	fileLength = htonl( fileLength );

	// El número de bytes a enviar es el tamaño de un entero
	cbLeftToSend = sizeof( fileLength );

	// Bucle para enviar la longitud del fichero
	do
	{
		int cbBytesSent;
		BYTE* bp = (BYTE*)(&fileLength) + sizeof(fileLength) - cbLeftToSend;
		cbBytesSent = sockConnection.Send( bp, cbLeftToSend );

		// Si ha habido algún error lo tratamos y salimos del bucle
		if ( cbBytesSent == SOCKET_ERROR )
		{
			int iErr = ::GetLastError();
			AfxMessageBox(_T("ERROR 2"));
			bRet = FALSE;
			goto PreReturnCleanup;
		}

		// Los datos se han enviado correctamente, por lo que se resta
		// de los bytes por enviar
		cbLeftToSend -= cbBytesSent;
	}
	while ( cbLeftToSend>0 );


	// Ahora enviamos los datos del fichero
	sendData = new BYTE[SEND_BUFFER_SIZE];

	cbLeftToSend = sourceFile.GetLength();

	// Bucle de envío
	do
	{
		// Leemos un conjunto de SEND_BUFFER_SIZE bytes desde el fichero
		int sendThisTime, doneSoFar, buffOffset;

		sendThisTime = sourceFile.Read( sendData, SEND_BUFFER_SIZE );
		buffOffset = 0;
```

```
                do
                {
                        doneSoFar = sockConnection.Send( sendData + buffOffset,

                sendThisTime );
                        // Comprobamos si hay error
                        if ( doneSoFar == SOCKET_ERROR )
                        {
                                int iErr = ::GetLastError();
                                bRet = FALSE;
                                goto PreReturnCleanup;
                        }

                        // Dato enviado satisfactoriamente
                        buffOffset += doneSoFar;
                        sendThisTime -= doneSoFar;
                        cbLeftToSend -= doneSoFar;
                }
                while ( sendThisTime > 0 );

        }
        while ( cbLeftToSend > 0 );

        // Etiqueta
        PreReturnCleanup:

        // Liberamos la memoria utilizada
        delete[] sendData;

        // Cerramos el descriptor si se ha abierto
        if ( bFileIsOpen )
                sourceFile.Close();

        // Cerramos el socket
        sockConnection.Close();

        return bRet;
}
```

Primero se definen una serie de variables necesarias para la comunicación. Como la longitud del fichero, el número de bytes restantes por enviar, el buffer de memoria, etc ..

A continuación se abre el fichero pasado como argumento y se almacena su longitud. Lo primero que vamos a enviar es la longitud del fichero, de esta forma el equipo que recibe los datos, conoce de antemano la longitud de este facilitando el bucle de recepción. Luego se procede al bucle de envío byte a byte hasta que no queden más bytes por enviar. Si hay algún problema, se muestra un mensaje al usuario por pantalla.

Lo siguiente es reservar espacio en memoria para almacenar el contenido del fichero antes de su envío. Para ello reservamos un buffer y se procede a rellenarlo mediante llamadas a la función ‘Read’ de la clase CFile. Cada vez que se lea una cantidad de bytes del fichero y se almacene en memoria, se procede a enviarlo. Para ello se actúa como en el caso anterior del envío de la longitud.

Finalmente se liberan los recursos utilizados y se cierra la conexión.

6.4 Recepción de fichero desde la CPU

En este caso se pretende descargar el fichero de resultados procedente de la PMU. Para ello, los primeros pasos son similares al caso del envío de un fichero salvo que ahora en vez de seleccionar un fichero para enviar, se selecciona la localización del fichero donde se grabarán los datos. En este caso también se ha hecho uso de una función 'recibe_fichero', que recibe como argumentos la ruta del fichero donde almacenar los datos y la dirección IP de donde proceden. Como en el caso anterior, se muestra el código de la función y se comentan los aspectos más generales de éste.

```
bool CFicherosDlg::recibe_fichero(CString nombre_fichero, CString ip)
{
	// Creamos el socket y nos conectamos
	CSocket sockClient;
	sockClient.Create();
	sockClient.Connect(ip, PUERTO_TCP_RECIBIR);

	// Variable de retorno
	BOOL bRet = TRUE;

	// Variables utilizadas en la operación de recepción
	int dataLength, cbBytesRet, cbLeftToReceive;

	// Puntero al buffer de recepción
	BYTE* recdData = NULL;

	// Fichero donde se almacenan los datos recibidos
	CFile destFile;
	CFileException fe;
	BOOL bFileIsOpen = FALSE;

	// Abrimos el fichero de recepción
	if( !( bFileIsOpen = destFile.Open( nombre_fichero, CFile::modeCreate |
		    CFile::modeWrite | CFile::typeBinary, &fe ) ) )
	{
		TCHAR strCause[256];
		fe.GetErrorMessage( strCause, 255 );
		AfxMessageBox("ERROR CON EL NOMBRE DEL FICHERO");
		/* Manejamos el error */

		bRet = FALSE;
		// Saltamos al final de la función
		goto PreReturnCleanup;
	}

	// Lo primero que recibimos es el tamaño del fichero
	cbLeftToReceive = sizeof( dataLength );

	// Bucle de recepción de bytes mientras haya bytes que recibir
	do
	{
		BYTE* bp = (BYTE*)(&dataLength) + sizeof(dataLength) - cbLeftToReceive;
		cbBytesRet = sockClient.Receive( bp, cbLeftToReceive );

		// Comprobamos errores
		if ( cbBytesRet == SOCKET_ERROR || cbBytesRet == 0 )
		{
			int iErr = ::GetLastError();
			AfxMessageBox("ERROR DE RECEPCION DE LONGITUD");
			/* Manejamos el error */

			bRet = FALSE;
			goto PreReturnCleanup;
		}

		// Decrementamos los bytes a recibir
		cbLeftToReceive -= cbBytesRet;
```

```
        }
        while ( cbLeftToReceive > 0 );

        dataLength = ntohl( dataLength );

        // Ahora recibimos tramas del fichero y la almacenamos en el buffer
        recdData = new byte[RECV_BUFFER_SIZE];
        cbLeftToReceive = dataLength;

        // Nuevo bucle mientras haya datos que recibir
        do
        {
                int iiGet, iiRecd;

                iiGet = (cbLeftToReceive<RECV_BUFFER_SIZE) ?
                                        cbLeftToReceive : RECV_BUFFER_SIZE ;
                iiRecd = sockClient.Receive( recdData, iiGet );

                // Comprobamos errores
                if ( iiRecd == SOCKET_ERROR || iiRecd == 0 )
                {
                        int iErr = ::GetLastError();
                        AfxMessageBox("ERROR A LA HORA DE RECIBIR EL FICHERO");
                        /* Manejamos el error */

                        bRet = FALSE;
                        goto PreReturnCleanup;
                }

                destFile.Write( recdData, iiRecd); // Write it
                cbLeftToReceive -= iiRecd;

        }
        while ( cbLeftToReceive > 0 );

        PreReturnCleanup: // labelled "goto" destination

        // Borramos la memoria utilizado, cerramos socket y fichero
        delete[] recdData;

        if ( bFileIsOpen )
                destFile.Close();

        sockClient.Close();

        return bRet;
}
```

El primer paso vuelve a se la declaración de aquellas variables implicadas en la recepción del fichero: longitud del fichero, bytes recibidos, bytes restantes, etc .. Luego se procede a abrir el fichero pero ahora en modo de escritura.

Como en el caso anterior, lo primero que recibimos es la longitud del fichero que procedemos a almacenar. Una vez conocido el número de bytes, realizaremos un bucle de recepción de bytes hasta alcanzar esta cifra.

Por último, liberaremos los recursos y cerraremos el socket utilizado.