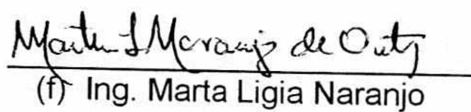
ARQUITECTURA DE SOFTWARE PARA EL ACCESO A MICROCONTROLADORES

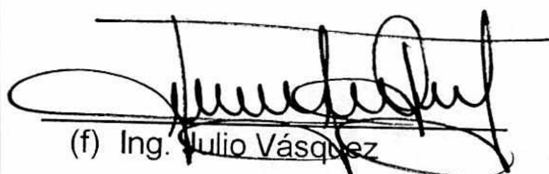
ASESOR:

(f) Ing. Marta Ligia Naranjo

TRIBUNAL:



(f) Ing. Marta Ligia Naranjo



(f) Ing. Julio Vásquez

(f) Ing. Luis Masaya

Fecha de aprobación: 17 de junio de 2003

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ciencias y Humanidades

ARQUITECTURA DE SOFTWARE PARA EL ACCESO A MICROCONTROLADORES

BIBLIOTECA
DE LA
UNIVERSIDAD DEL VALLE DE GUATEMALA

Trabajo Profesional presentado por Fredy Wingston Muñoz Godoy para optar al grado académico de Licenciatura en Ciencias de la Computación

Guatemala
2003

PREFACIO

Este trabajo surgió de la cooperación con los estudiantes de Ingeniería Electrónica en el proyecto CODEINE (Controller Development Internetworking) desarrollado para el curso de Arquitectura Digital 2.

El proyecto CODEINE se trabajó durante la segunda mitad del año 2002. Consistió en construir seis módulos independientes denominados esclavos. Cada módulo poseía un dispositivo controlador de procesos y realizaba una tarea específica. Existía un séptimo módulo denominado maestro que tenía la capacidad de enviar o pedir datos a los módulos esclavos. Este módulo maestro también podía recibir instrucciones a través de su puerto serial para ser enviadas a un esclavo específico.

Sobre esta plataforma se construyeron componentes de software que permitían enviar instrucciones o pedir información de los módulos desde cualquier navegador de Internet. Este software fue construido durante noviembre y diciembre del año 2002.

Aunque la versión inicial del software cumplía con su función, era poco robusta y presentaba varios problemas. Uno de ellos fue nombrado el minuto de silencio pues cuando el software dejaba de funcionar había que esperar un minuto para reiniciarlo.

Durante enero y febrero del año 2003 estos problemas fueron resueltos y se obtuvo la versión definitiva que se expone en este trabajo. La confiabilidad de esta última versión aún puede cuestionarse, pero constituye una base sobre la que se puede seguir trabajando y agregando funcionalidad.

RESUMEN

En este trabajo se expone una arquitectura para que los programas computacionales puedan acceder a la información los microcontroladores proveen. Este proyecto consta de módulos de software y protocolos establecidos para que esta comunicación pueda ser clara y efectiva.

La arquitectura propuesta permite que el programa que desea acceder al microcontrolador sea escrito en cualquier lenguaje de programación, ejecutado en cualquier tipo de computadora manejada por cualquier sistema operativo, siempre y cuando el programa tenga un medio para establecer una comunicación utilizando el conjunto de protocolos denominado Transmition Control Protocol/Internet Protocol (TCP/IP).

CONTENIDO

Página

PREFACIO i
RESUMEN ii
CONTENIDO iii
LISTA DE FIGURAS v

Capítulos

I. INTRODUCCIÓN 1

II. EL MICROCONTROLADOR 3

A. Origen del microcontrolador 3
B. Características típicas de un microcontrolador 4
C. Diferencias entre un microcontrolador y un microprocesador 6
D. La comunicación hacia el microcontrolador 6
E. El microcontrolador utilizado 8

III. DESCRIPCIÓN GENERAL DE LA ARQUITECTURA 10

A. Elementos que conforman la arquitectura 10
B. Comunicación de un programa cliente a un microcontrolador 11

IV. EL UNIFICADOR 14

A. Componentes del Unificador 14
1. La Antena 15
2. El Directorio 19
3. El Enlace 21
4. La Instrucción 23
5. La Respuesta 24
B. Funcionamiento del unificador 24

V. EL PROGRAMA CLIENTE 28

A. La comunicación hacia el unificador 28
B. Ejemplos de programas cliente 30
1. Programa Cliente: Monitor de ventilador 31
2. Programa Cliente: Monitor de osciloscopio 33

CONCLUSIONES 35

RECOMENDACIONES 37

BIBLIOGRAFÍA 38

APÉNDICE A: Traducción de la especificación de General Motors para un Controlador lógico programable (1968).. 39
APÉNDICE B: Página 1 del documento Microchip PIC16F87X Data Sheet 42
APÉNDICE C: Página 13 del documento Microchip PIC16F87X Data Sheet.... 44
APÉNDICE D: Código fuente del Unificador 46
APÉNDICE E: Código fuente de los Programas Cliente.................................... 89
APÉNDICE F: Instalación y manejo del Unificador; Instalación de los Programas Cliente.. 100

LISTA DE FIGURAS

Figura Página

1. Arquitectura para el acceso a microcontroladores 10

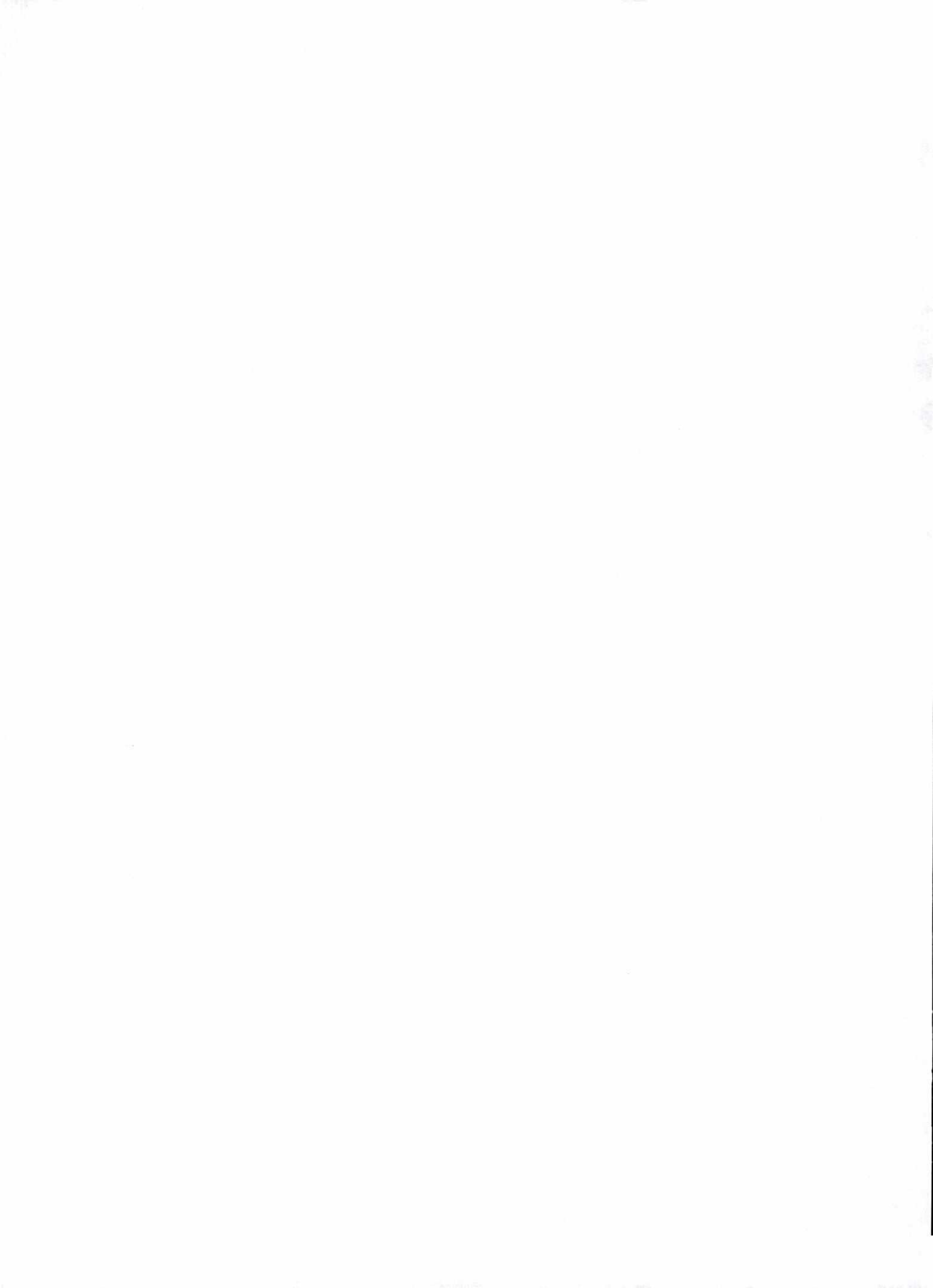
2. Acceso a múltiples microcontroladores utilizando más de un Unificador. 13

3. Componentes del Unificador 14

4. Diagrama de flujo general de ejecución del Unificador 25

5. Códigos de error emitidos por el Unificador 26

6. Interacción entre el navegador, el servidor web y un CGI 31



I. INTRODUCCIÓN

Este proyecto consiste en desarrollar una arquitectura que facilite la comunicación de programas ejecutándose en equipos de cómputo, como computadoras personales, hacia dispositivos controladores de procesos. Esta arquitectura oculta los aspectos relacionados con el transporte de la información desde y hacia los dispositivos controladores. De esta manera cualquier cambio a nivel del dispositivo controlador o del protocolo a seguir para comunicarse con él no será percibido por el programa.

Un dispositivo controlador de procesos, también llamado microcontrolador, es un circuito electrónico integrado, empacado en una sola pastilla. Éste es capaz de almacenar y ejecutar un programa escrito en un lenguaje propio de bajo nivel, recibir señales digitales desde el exterior y emitir señales digitales. Esto le permite controlar un proceso utilizando su programa interno.

Estos dispositivos pueden ser utilizados en procesos industriales como la fabricación de algún producto en donde se requiera controlar la velocidad a la que opera una faja transportadora, la temperatura de recintos o espacios de almacenamiento, los niveles de tanques de materia prima, etc. Pero también pueden ser utilizados en ambientes domésticos, donde se puede controlar el encendido o apagado de luces, vigilancia de puertas y ventanas para un sistema de alarma, manejo de aparatos eléctricos como el televisor o el equipo de sonido, entre otros usos. En el capítulo II se da una descripción general de los microcontroladores que ayuda a entender su funcionamiento y su lugar dentro de la arquitectura.

Existen diferentes marcas, tipos y modelos de microcontroladores. Se elige el que mejor se adecue al ambiente y las condiciones del trabajo para el que se requiere. Por esta razón en un ambiente pueden existir varios microcontroladores de diferentes características.

Esto exige que el programaque necesite comunicarse con un tipo de dispositivo en específico deba conocer la forma de conectarse y el protocolo a seguir.

Es necesario entonces definir una arquitectura de comunicación de tal manera que un programa acceda a la información que pueda proporcionar un microcontrolador sin importar su tipo y sin necesidad de conocer sus características específicas. De esta forma un solo programa es capaz de acceder a microcontroladores de diferentes características. Además podrá acceder a un nuevo tipo de microcontrolador que haya sido instalado después de haber sido construido el programa. En el capítulo III se describe esta arquitectura, sus partes y la interacción entre ellas para lograr este objetivo.

En el capítulo IV se expone el Unificador. Este es el elemento principal de la arquitectura. Es el canal de comunicación entre el programa y el microcontrolador. Está constituido en su totalidad de módulos de software.

El programa que requiera comunicarse con el microcontrolador podrá ser ejecutado sobre cualquier plataforma. Esto quiere decir que no importa el tipo de sistema de cómputo, el sistema operativo sobre el que se ejecute, ni el lenguaje de programación empleado para construir el programa. En el capítulo V se expone cómo se debe llevar a cabo la interacción entre el Unificador y el programa que desea comunicarse con el microcontrolador.

II. EL MICROCONTROLADOR

A. Origen del microcontrolador

Debido al gran desarrollo de la industria de la manufactura a mediados del siglo XX, era necesario contar con sistemas de automatización de procesos en las plantas de producción para aumentar su rendimiento. Este control industrial se llevaba a cabo por medio de relevadores. Un relevador (*relay*) es un dispositivo electromecánico utilizado para regular el paso de la corriente por un circuito eléctrico.

Estos sistemas contaban con gran cantidad de relevadores que generaban mucho calor y ruido. Ocupaban grandes espacios y su consumo eléctrico era alto. Si el proceso industrial cambiaba, también se tenía que cambiar el sistema de control. Este cambio representaba agregar o eliminar relevadores y además modificar las conexiones eléctricas entre ellos. Para realizar esta tarea era necesario invertir cantidades considerables de tiempo y personal.

Para las industrias cuyas plantas de producción debían sufrir cambios en períodos de tiempo relativamente cortos, esto significaba un alto costo. Por ejemplo, las empresas automotrices debían realizar modificaciones a estos sistemas de automatización con cada nuevo modelo que lanzaban al mercado.

Esto motivó a General Motors en 1968 a preparar una especificación detallando un controlador lógico programable que reemplazara el uso de sistemas de control por relevadores (*vid* Apéndice A). Éste sería un dispositivo que pudiera recibir impulsos eléctricos del exterior, procesar esos impulsos y emitir impulsos eléctricos de vuelta al exterior. El comportamiento del controlador debía estar regido por un programa o secuencia de operaciones lógicas almacenadas internamente y no por las conexiones entre elementos físicos externos, de tal forma que si el proceso industrial fuera modificado sólo se debería cambiar este programa.

Esta especificación generó interés en compañías fabricantes de sistemas de cómputo y equipo electrónico. Estas compañías trabajaron junto con General Motors para producir en 1970 un sistema comercial (Wilhelm, 1985:6).

Al principio, la forma de programar un controlador era quemar fusibles internos para definir dentro de él un circuito lógico equivalente a las operaciones lógicas que constituían el programa. Obviamente el controlador no podía ser reprogramado.

Se desarrollaron técnicas que permitían reprogramar el controlador. Una de ellas era usar luz ultravioleta para regresar al controlador a su estado original y reprogramarlo mediante impulsos eléctricos.

Otra solución fue incluir dentro del controlador una unidad operativa que fuera capaz de leer una instrucción, interpretarla y ejecutarla. Así reprogramar el controlador es simplemente reemplazar la secuencia de instrucciones que tiene almacenada internamente. Los controladores que siguen este modelo se llaman autómatas programables o microcontroladores (Mandado, 1999:115).

La National Electrical Manufacturers Association (NEMA) define al microcontrolador como:

> *«Un aparato electrónico operado digitalmente, que utiliza memoria programable para el almacenamiento interno de instrucciones, las cuales implementan funciones específicas tales como funciones lógicas, de secuencia, de cronometraje, de conteo y de aritmética para controlar, a través de módulos digitales o análogos de entrada y salida, diferentes tipos de máquinas o procesos.» (Wilhelm, 1985:7)*

B. Características típicas de un microcontrolador

Un microcontrolador es un circuito electrónico empacado en una sola pastilla de silicio a una gran escala de integración. De esta forma los componentes del microcontrolador se encuentran a distancias medidas en centésimas de centímetros, aumentando las velocidades de ejecución. Además, mientras más integrados estén los componentes del microcontrolador, éste será

más pequeño y consumirá menos recursos, como espacio y electricidad (Mandado, 1999:49).

El componente principal de un microcontrolador es la unidad operativa, que es equivalente a la Unidad Central de Procesamiento (CPU) de una computadora. De hecho algunos fabricantes realizan el desarrollo de sus microcontroladores en paralelo al desarrollo de microprocesadores. Los microprocesadores realizan la función de un CPU dentro de una computadora personal.

La unidad de procesamiento de un microcontrolador es capaz de interpretar un conjunto definido de instrucciones de bajo nivel. Un programa es una secuencia de instrucciones de este conjunto definido. Las instrucciones pueden ser operaciones aritméticas, de manipulación de memoria y de control de flujo del programa.

Los microcontroladores también cuentan con memoria de acceso volátil (RAM) para almacenamiento temporal y como espacio de trabajo para el programa. La cantidad de esta memoria es muy pequeña, pero se pueden añadir módulos externos de memoria. Esta memoria también es utilizada para el almacenamiento de variables de estado relacionadas con el funcionamiento del microcontrolador.

Se tiene también un espacio de almacenamiento permanente. Esta memoria puede ser modificada por impulsos eléctricos igual que la memoria RAM, pero al cortar la alimentación eléctrica no se pierde la información. Se utiliza para almacenar el programa que será ejecutado por la unidad operativa. También puede ser empleada por el programa para almacenar información del proceso que se está controlando.

Por último, un microcontrolador posee módulos de entrada o salida de señales digitales. Estos son utilizados para comunicar al microcontrolador con el exterior. Estos componentes también le permiten comunicarse con otros

dispositivos utilizando protocolos de comunicación definidos (Mandado, 1999:116).

C. Diferencias entre un microcontrolador y un microprocesador

Las características de un microcontrolador son similares a las de un Microprocesador, pero este último posee una capacidad y velocidad de procesamiento mayor. Esto hace pensar que un Microprocesador podría realizar la labor de un microcontrolador con mayor facilidad e incluso realizar tareas adicionales (Jones, 1996:12).

Es cierto que la superioridad de procesamiento del Microprocesador es una ventaja, pero el ambiente en el que opera un microcontrolador es hostil. Un microcontrolador está expuesto a altas temperaturas, extrema humedad, interferencia electromagnética y polvo. El microcontrolador fue diseñado para operar bajo estas circunstancias, el Microprocesador no.

Una de las características principales de un microcontrolador es su alta integración. La unidad operativa, la memoria RAM, la memoria permanente y algunos de sus módulos de entrada y salida se encuentran empacados en una sola pastilla. Un Microprocesador solamente contiene la unidad operativa, necesita memoria externa y todos sus módulos de entrada y salida son externos.

Otra diferencia entre estos dos dispositivos es que el microcontrolador fue diseñado para incluirse en sistemas que interactúan con máquinas u otros dispositivos. El Microprocesador fue diseñado para ser utilizados en sistemas que interactúan con humanos (Jones, 1996:12).

D. La comunicación hacia el microcontrolador

En ocasiones es necesario extraer información del microcontrolador que ha obtenido al controlar un proceso. Podría ser necesario saber la frecuencia con la que ocurre cierto evento, las mediciones hechas en determinados

momentos del proceso u otra información requerida para controlar o analizar un proceso a largo plazo.

Para extraer esa información es necesario establecer una comunicación con el microcontrolador. Generalmente se utiliza una computadora para realizar esta tarea, además de un protocolo de comunicaciones entre el microcontrolador y la computadora. Un protocolo de comunicación es un conjunto de reglas que los elementos a comunicarse deben seguir para que la transferencia de información se de en una forma ordenada, eficiente y efectiva (Wilhelm, 1985:385).

Los protocolos de comunicación más utilizados son el RS-232, el RS-449 y el IEEE-488. En realidad los dos últimos son versiones mejoradas del RS-232. Este estándar fue propuesto por la *Electronic Industries Association* (EIA) y era utilizado inicialmente para la comunicación entre una computadora y cualquier otro dispositivo externo (Wilhelm, 1985:403).

En su forma básica el RS-232 utiliza tres alambres para la comunicación. Uno para enviar datos, el segundo para recibir datos y el tercero es para establecer la referencia común que los dos equipos deben utilizar para interpretar las señales.

La unidad de información más pequeña que se puede transferir es un *byte* y se transmite en forma serial, es decir se envía *bit* por *bit*. Las velocidades de transmisión que se pueden alcanzar utilizando RS-232 van desde los 110 *bits* por segundo hasta los 921,600 *bits* por segundo. Las velocidades comúnmente utilizadas para la comunicación entre una computadora y un microcontrolador son de 57,600 o 115,200 *bits* por segundo (Wilhelm, 1985:410).

Las comunicaciones que utilizan RS-232 pueden experimentar algunos problemas. Si el cable utilizado para la comunicación pasa por alguna fuente de interferencia electromagnética, ésta podría alterar la información transmitida. Para detectar si un *byte* fue alterado se envía junto con él un *bit* de paridad.

Cuando el *byte* y su *bit* de paridad llegan a su destino, estos se chequean. Si no concuerdan, el *byte* se desecha, pero no se da la retransmisión automáticamente, esto es responsabilidad del equipo emisor.

El problema con el *bit* de paridad es que permite detectar la invalidez de un *byte* solamente si fue alterado en un *bit*. Si el efecto que produce la interferencia es mayor, el protocolo RS-232 no provee un método de detección. Por esta razón es necesario alejar de cualquier posible fuente de campos electromagnéticos el cable que se utiliza para la comunicación. Además se debe emplear un tipo de cable que posee un escudo contra interferencia.

Otra consideración que se debe tener al utilizar RS-232 es que a medida que la señal viaja a través del cable, ésta se debilita. Se sugiere que el cable no exceda los 15 metros. Si se necesita llegar a distancias más grandes, se pueden utilizar dispositivos que simplemente reciben la señal y la restauran para ser enviada nuevamente (Wilhelm, 1985:417).

A pesar de estos problemas el protocolo RS-232 es ampliamente utilizado y soportado por la mayoría de microcontroladores. El problema de la confiabilidad puede solucionarse construyendo sobre él otro protocolo que maneje retransmisiones.

E. El microcontrolador utilizado

El microcontrolador empleado en este trabajo es el modelo PIC16F877 fabricado por Microchip. Este está empacado en una pastilla de 40 pines. Opera a 20 Mhz. Tiene una memoria RAM de 368 *bytes,* además provee 256 *bytes* de memoria permanente. La memoria empleada para almacenar el programa es de 8192 *bytes*. Está orientado al uso comercial e industrial. Puede operar en un rango de temperatura de -55°C a 125°C (*vid* Apéndice B).

Tiene un conjunto de 32 instrucciones. Existen instrucciones para operaciones aritméticas, lógicas, de manipulación de memoria y de control de

flujo del programa. También cuenta con instrucciones para manipulación de *bits*. La ejecución de una instrucción toma 200ns.

A pesar de poseer 368 *bytes* de memoria RAM, estos no pueden ser accedidos de forma directa, pues las direcciones son de siete *bits*. La memoria RAM está organizada en cuatro bancos de 128 *bytes* cada uno. Además se utilizan dos bits para seleccionar uno de los cuatro bancos. De esta forma se tienen a disposición 512 posiciones de memoria RAM. Las primeras 32 posiciones del banco 0 y el banco 1, y las primeras 16 posiciones del banco 2 y 3 son utilizadas para acceder registros especiales y de control del microcontrolador. Además las últimas 16 posiciones de los bancos 1,2 y 3 son utilizadas para acceder los últimos 16 *bytes* del banco 0. El resto de posiciones permiten acceder a 368 *bytes* de propósito general (*vid* Apéndice C).

Este microcontrolador cuenta con cinco puertos que pueden ser configurables como puertos de entrada o salida. Además cuenta con dos módulos de comunicación serial, uno sólo sincrónico y el otro puede ser configurado como sincrónico o asíncrono. En este proyecto se utilizó el segundo módulo de comunicación serial en modo asíncrono. Este módulo utiliza el protocolo RS-232, antes descrito.

Para la realización de este proyecto no fue necesario configurar, programar ni conocer a profundidad al microcontrolador utilizado. De estas tareas se encargaron los estudiantes de Ingeniería electrónica quienes tenían amplia experiencia manipulándolos. Lo único que fue necesario conocer acerca del microcontrolador es su limitada cantidad de memoria y la configuración del puerto serial para la comunicación con el protocolo RS-232.

III.DESCRIPCIÓN GENERAL DE LA ARQUITECTURA

La idea en la que se basa esta arquitectura es la de colocar una capa de abstracción entre el programa de computación (Programa Cliente) y el microcontrolador. De esta manera el Programa Cliente será capaz de enviar o solicitar información a esta capa sin preocuparse de establecer y manejar una comunicación serial directa con el microcontrolador. Desde el punto de vista del Programa Cliente esta capa es el microcontrolador en sí.

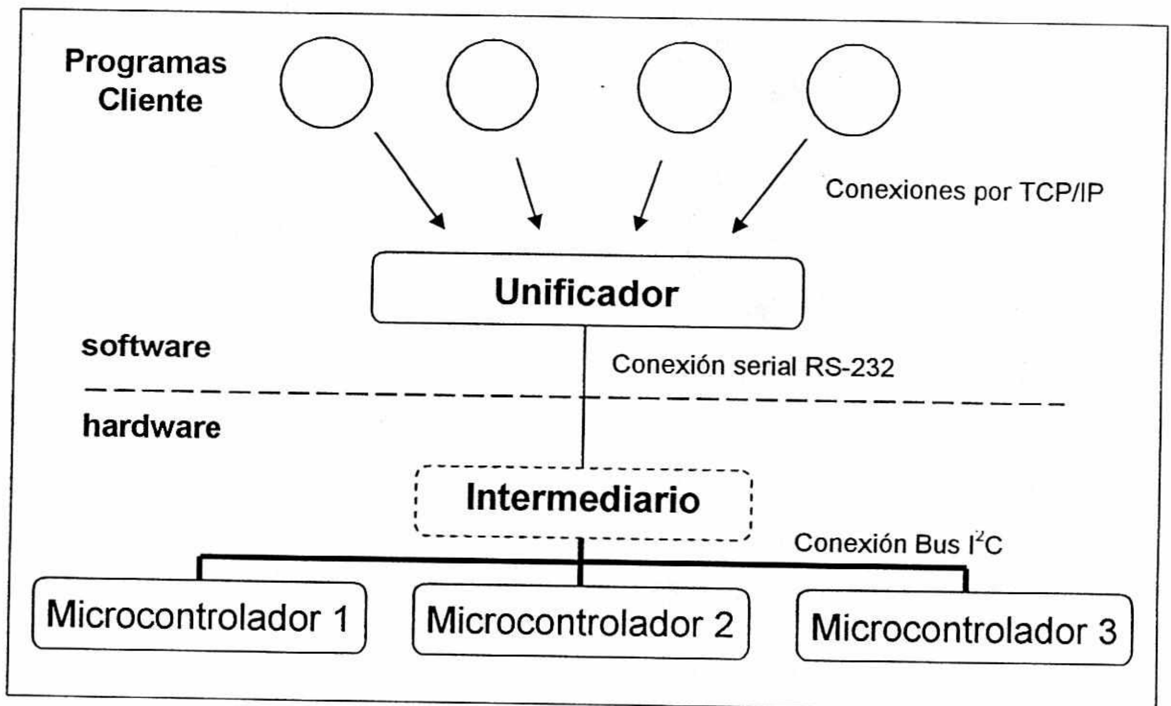


Figura 1 Arquitectura para el acceso a microcontroladores

A. Elementos que conforman la arquitectura

En la figura anterior se ilustran los elementos que toman parte en la arquitectura, así como las relaciones entre ellos. En el nivel más alto se encuentran los Programas Cliente que se ejecutan en sistemas de cómputo. Éstos, dependiendo de su función, necesitan enviar o solicitar información a microcontroladores.

En un nivel inferior se encuentra el Unificador, que es la capa de abstracción. Éste es un programa que se encuentra en ejecución permanente en una computadora específica. Los Programas Cliente ven al Unificador como el microcontrolador con el que desean intercambiar información. La comunicación al Unificador se logra a través de conexiones TCP/IP, que son los protocolos utilizados en Internet.

Al Unificador está conectado el Intermediario a través de una conexión serial que utiliza el protocolo RS-232. Este Intermediario es otro microcontrolador dedicado únicamente a permitir que el Unificador pueda comunicarse por una sola línea serial a varios microcontroladores.

Se utilizó el protocolo RS-232 porque el microcontrolador provee un módulo que lo maneja. Si en caso se desea cambiar este medio de comunicación el Unificador está diseñado de tal forma que solamente es necesario cambiar un segmento aislado de código.

Al Intermediario se pueden conectar hasta 126 microcontroladores a través de un bus de comunicación regido por el protocolo I^2C. Un bus de comunicación es un medio común al que muchos dispositivos se conectan a la vez. Este bus es controlado por el Intermediario. El Unificador provee acceso a los microcontroladores conectados al Intermediario.

Esta es la forma en que los elementos de la arquitectura se comunican entre sí y los medios que utilizan para lograrlo. Ahora se describirá un escenario típico en el que un Programa Cliente desea establecer comunicación con un microcontrolador.

B. Comunicación de un programa cliente a un microcontrolador

Un Programa Cliente envía al Unificador una instrucción. Como la comunicación entre ellos se lleva a cabo utilizando los protocolos TCP/IP, el programa debe conocer la dirección IP de la computadora donde está en

ejecución el Unificador. Si se cuenta con un servidor de resolución de nombres (DNS) el Programa Cliente únicamente tendrá que conocer el nombre de esta computadora.

Junto con la instrucción se envía el nombre del microcontrolador al que va dirigida la instrucción. Este nombre es una cadena de caracteres que el Unificador traduce a una dirección numérica. La dirección se adhiere a la instrucción y es enviada al Intermediario. El Intermediario utiliza la dirección numérica para enviar la instrucción al microcontrolador específico a través del bus I^2C.

En esta arquitectura un microcontrolador no puede iniciar la comunicación, sólo lo puede hacer un Programa Cliente. Esto porque el bus de comunicaciones utilizado solamente permite iniciar la comunicación al Intermediario (microcontrolador maestro). Los demás microcontroladores solamente reciben información (microcontrolador esclavo).

Las posibles instrucciones que un Programa Cliente puede enviar a un microcontrolador son enviar datos y solicitar datos. La instrucción de envío de datos está acompañada por los datos que se desean hacer llegar al microcontrolador. La instrucción de solicitud de datos se hace llegar al microcontrolador y este envía de vuelta la información solicitada.

El Unificador y el Intermediario se encargan de facilitar la comunicación entre un programa y un microcontrolador. Estos no tienen la capacidad de interpretar y procesar la información que se intercambia. Sin embargo existen límites con respecto a la cantidad de información que puede viajar a través del sistema. Estas limitaciones sí son conocidas tanto por el Unificador como por el Intermediario.

En ocasiones será necesario instalar grupos independientes de microcontroladores. Cada uno de estos grupos necesitará su propio Unificador que provea el acceso. De esta forma se pueden tener varios Unificadores

independientes. Un Programa Cliente es capaz de comunicarse a microcontroladores vinculados a Unificadores diferentes. Esto es posible porque el Programa Cliente debe conocer la dirección de la computadora donde está en ejecución el Unificador. Así solamente es necesario conocer las dirección de los otros Unificadores para acceder a todos los microcontroladores. Un ejemplo se muestra en la figura 2.

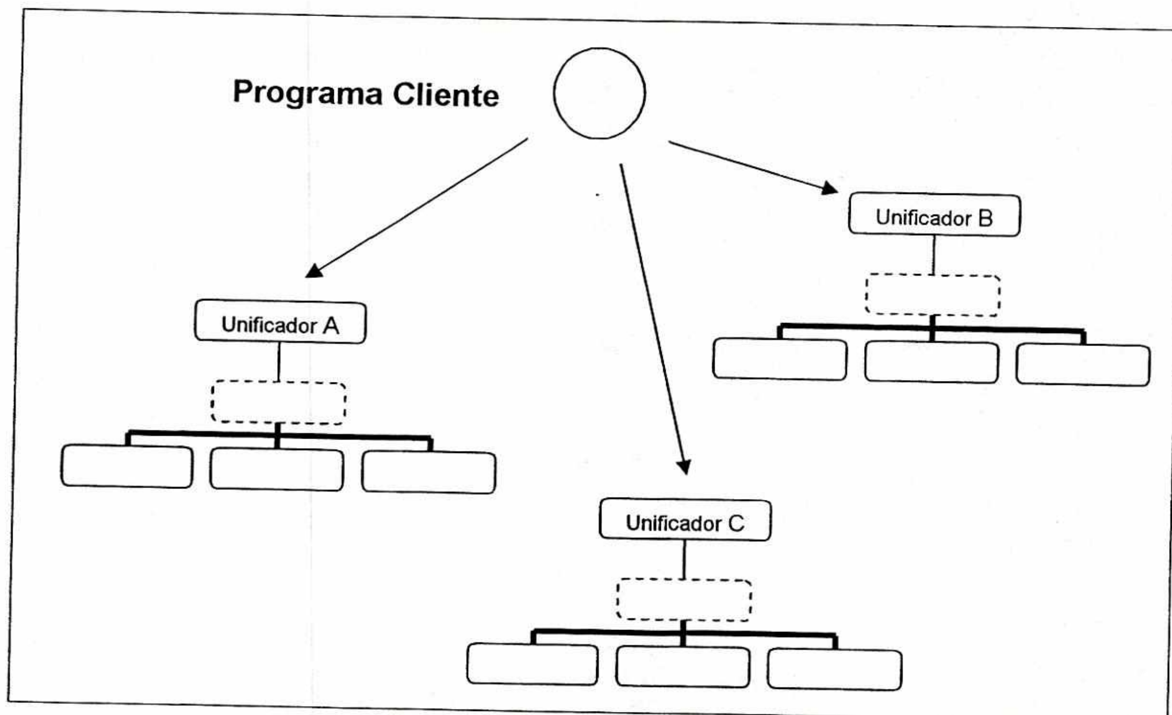


Figura 2 Acceso a múltiples microcontroladores utilizando más de un Unificador

En este caso existen tres grupos independientes de microcontroladores. Es posible que se encuentren en escenarios distantes. Cada grupo de microcontroladores está vinculado a su Unificador. El Programa Cliente puede acceder a un microcontrolador vinculado al Unificador A, al Unificador B o al Unificador C. Únicamente necesita conocer la dirección IP del Unificador con el que quiere establecer una comunicación.

IV.EL UNIFICADOR

El Unificador es un programa que se encuentra en ejecución permanente en una computadora específica. Su función es recibir instrucciones provenientes de Programas Cliente y enviarlas al Intermediario. Si un Programa Cliente solicitó datos de algún microcontrolador, el Unificador se encarga de recibir estos datos provenientes del Intermediario y de enviarlos al Programa Cliente.

Este programa fue escrito en lenguaje C++ para ser ejecutado en una computadora personal con sistema operativo Linux. El código fuente de este programa está listado en el Apéndice D.

A. Componentes del Unificador

El diseño del Unificador está totalmente orientado a objetos. Consta de varios componentes unidos en un solo programa. Esta característica le permite ser flexible y fácilmente modificable, pues un cambio en el funcionamiento de uno de los componentes no afectará a los demás, siempre y cuando la interfaz que provea se mantenga.

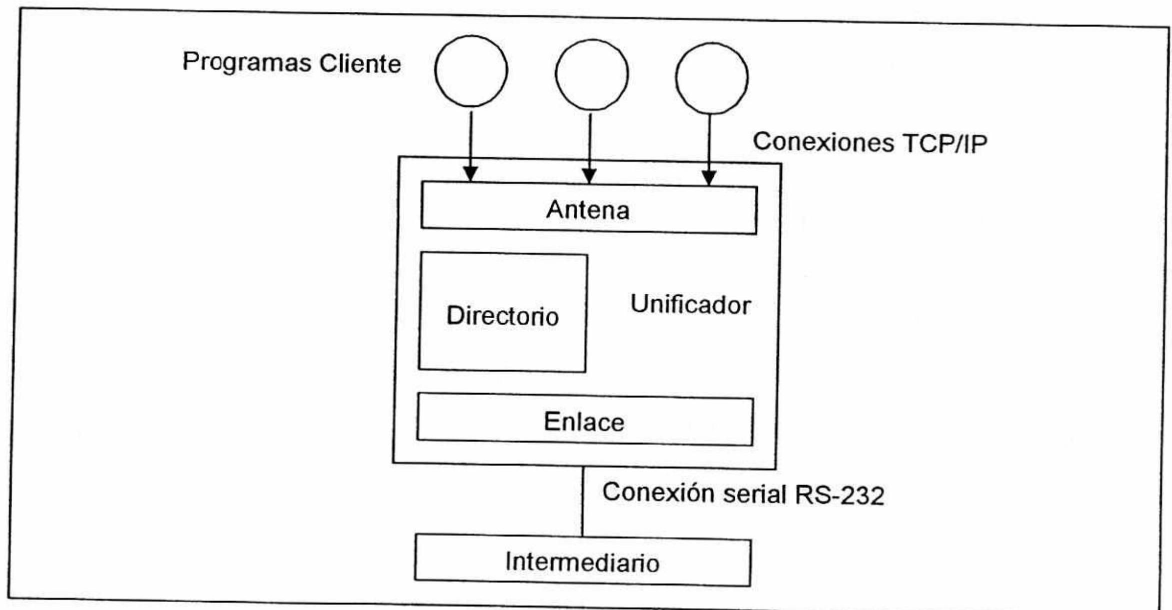


Figura 3 Componentes del Unificador

En la figura anterior se pueden observar los componentes que conforman al Unificador y cómo estos son utilizados para interactuar con los Programas Cliente y con el Intermediario. El Unificador contiene tres objetos: una Antena, un Directorio y un Enlace. Cada uno cumple una tarea específica y tienen una interfaz definida.

1. La Antena

La Antena es el objeto utilizado para interactuar con los Programas Cliente (*vid* Apéndice D.3). Esta permite esperar, aceptar y cerrar conexiones que utilizan el protocolo TCP/IP. Además es utilizada por el Unificador para recibir una instrucción, para recibir el nombre del microcontrolador al que se desea enviar la instrucción, para enviar mensajes de error o éxito en el procesamiento de la Instrucción y enviar al Programa Cliente el resultado del procesamiento de la instrucción.

a. Sockets

Para establecer las conexiones utilizando el protocolo TCP/IP los sistemas operativos de la familia UNIX utilizan una estructura denominada *socket*. Existen en estos sistemas otras formas, como el uso de Transport Layer Interface (TLI) o *streams*, pero los *sockets* proveen la interfaz más clara para el programador (Robbins, 1996:444).

Un *socket* es un canal de comunicación que permite el flujo continuo de *bytes* entre dos extremos. Este flujo puede ocurrir en ambas vías al mismo tiempo. Estas estructuras también pueden ser empleadas en la comunicación de procesos dentro de la misma computadora.

La interacción con un *socket* se da de la misma forma que con un archivo binario en un sistema UNIX. Se utiliza la llamada al sistema *read* para obtener *bytes* provenientes del otro extremo y se utiliza la llamada al sistema *write* para enviar *bytes* al otro extremo de la comunicación.

Al igual que un archivo binario, el *socket* debe ser abierto. Esto se logra con la llamada al sistema *socket* que devuelve un manejador de archivo (*file handler*) como ocurre con la apertura de un archivo con la llamada al sistema *open* (Robbins, 1996:447).

Luego de abrir el *socket* este puede ser empleado para realizar una conexión hacia un objetivo. Para ello se emplea la llamada al sistema *connect*, indicando la dirección IP y el puerto TCP al que se quiere dirigir la conexión. El *socket* también puede ser utilizado para esperar conexiones provenientes de otros programas, como lo hace un servidor web y como lo hará la Antena del Unificador. Para lograrlo, se invoca a la llamada al sistema *listen*, junto con el puerto TCP en el que se desea esperar conexiones (Robbins, 1996:447).

b. Iniciación

La iniciación del objeto Antena consiste en crear un *socket* para esperar, a través de él, que arriben las conexiones de los Programas Cliente[1]. Si al crear el *socket* se da algún error, se informa al dueño de la Antena, en este caso el Unificador, que la Antena no pudo ser inicializada correctamente. Esta inicialización debe realizarse antes de utilizar los servicios que presta el objeto Antena.

c. Finalización

El *socket* creado para esperar conexiones debe ser liberado para que el puerto del Transport Control Protocol (TCP) pueda ser reutilizado. En la finalización se le indica al sistema operativo que destruya este *socket*[2]. Después de invocar la finalización de la Antena, esta no podrá ser utilizada nuevamente sino hasta ser iniciada.

---

[1] Apéndice D.4, líneas 31 a 67.
[2] Apéndice D.4, líneas 76 a 79.

d. Esperar una conexión

Cuando un Programa Cliente solicita una conexión a la Antena, el sistema operativo coloca esa solicitud en una cola asociada al *socket* de espera creado en la iniciación. El servicio de Esperar Conexión solicita al sistema operativo que se acepte la conexión que se encuentre a la cabeza de esa cola[1].

Es posible que la cola esté vacía. En este caso la ejecución del proceso que utiliza el objeto Antena es suspendida. La ejecución se restablece, en el mismo punto en el que fue suspendida, al haber alguna solicitud disponible en la cola.

El objeto Antena mantiene abierta la conexión aceptada, hasta que el Unificador le indique explícitamente que la cierre. El Programa Cliente también puede cerrar la conexión en cualquier momento. Esto invalidaría la conexión y cualquier intento de la Antena de enviar o solicitar información a esa conexión, resultaría en un error que es apropiadamente manejado por la Antena.

e. Cerrar una conexión

Al concluir una conversación entre el Unificador y el Programa Cliente, se debe cerrar la conexión utilizada para dar cabida a una nueva. Para ello el objeto Antena provee el servicio de cerrar conexión[2].

Este servicio solicita al sistema operativo que destruya la conexión. Si la conexión fue previamente cerrada por el Programa Cliente, el sistema operativo ignora esta solicitud pues la conexión ya no existe.

---

[1] Apéndice D.4, líneas 89 a 97.
[2] Apéndice D.4, líneas 105 a 108.

f. Recibir el nombre

Este servicio permite recibir el nombre del microcontrolador hacia el que el Programa Cliente desea dirigir una instrucción[1]. Este nombre se recibe en forma de cadena de caracteres.

La recepción consiste en solicitar caracter por caracter a la conexión abierta por el Programa Cliente, hasta recibir el indicador de terminación de cadena ('\0'). Si por alguna razón el nombre no pudo ser recibido correctamente, esto se hace saber al Unificador.

g. Recibir la instrucción

Este servicio permite recibir la instrucción que se enviará al microcontrolador[2]. La recepción consiste en solicitar a la conexión abierta por el Programa Cliente los *bytes* que conforman la instrucción y construir un objeto Instrucción. El objeto Instrucción se explicará más adelante en este capítulo. Si ocurre algún error al recibir la Instrucción, se indica al Unificador.

h. Enviar mensajes de estado

Es posible que al procesar la instrucción el Unificador se encuentre con errores o bien la instrucción sea procesada exitosamente. Estos estados del procesamiento de la instrucción deben darse a conocer al Programa Cliente. El servicio de Envío de Mensajes de Estado permite enviar un *byte* a través de la misma conexión por donde se recibió la instrucción. Éste *byte* indica al Programa Cliente el estado del procesamiento de la instrucción que ha enviado[3].

---

[1] Apéndice D.4, líneas 120 a 138.
[2] Apéndice D.4, líneas 150 a 176.
[3] Apéndice D.4, líneas 185 a 188.

i. Enviar respuesta

Si el Programa Cliente envía una instrucción de solicitud de datos, el microcontrolador deberá devolver los datos solicitados. Esta respuesta del microcontrolador se representa dentro del Unificador por un objeto Respuesta que se explicará más adelante en este capítulo. Este servicio permite enviar al Programa Cliente la respuesta del microcontrolador[1].

2. El Directorio

Por medio del Intermediario se pueden conectar varios microcontroladores a un solo Unificador. Para diferenciar uno de otro se les asigna una dirección que se graba dentro de cada microcontrolador. La dirección consiste en un número entero dentro del rango de 1 a 126. Sin embargo, para un ser humano es muy difícil recordar un número. Resulta mucho más fácil utilizar un nombre que además sea descriptivo.

Por esto el Programa Cliente envía al Unificador un nombre en forma de cadena de caracteres. Esta cadena puede estar formada por los caracteres de la 'a' a la 'z', de la 'A' a la 'Z', los diez dígitos, el punto (.) y el guión bajo (_). La cadena no puede contener espacios en blanco y debe tener al final el carácter de terminación de cadena ('\0').

El Directorio es el objeto utilizado para traducir los nombres de microcontroladores a direcciones numéricas (*vid* Apéndice D.7). Internamente almacena la correspondencia entre nombres y direcciones en una tabla. Provee un único servicio que es el de traducción.

a. El archivo directorio

Este es un archivo de texto que es utilizado para almacenar y establecer las correlaciones entre nombres y direcciones. Estas

---

[1] Apéndice D.4, líneas

correlaciones serán utilizadas posteriormente por el objeto Directorio para realizar su función.

Una correlación se almacena en una sola línea del archivo. Primero se coloca la dirección numérica del microcontrolador, luego se coloca un espacio en blanco y por último el nombre del microcontrolador. Se pueden colocar tantas correlaciones como se deseen, una en cada línea.

b. Iniciación

La iniciación del objeto Directorio consiste en cargar a una estructura en memoria la información contenida en el archivo directorio[1]. Esta estructura es la Tabla de Nombres. Si no se encuentra el archivo directorio o este contiene alguna correlación mal escrita, se hace saber al Unificador. Si el archivo 'directorio' es modificado se debe reiniciar el Directorio para que los cambios tengan efecto.

c. Finalización

En la finalización el Directorio únicamente devuelve al sistema operativo la memoria utilizada por la Tabla de Nombres[2].

d. Traducción de nombres

Por medio de este servicio el Unificador envía al Directorio el nombre de microcontrolador en forma de cadena de caracteres y recibe la dirección numérica[3]. El Directorio busca en la Tabla de Nombres una correlación que involucre al nombre solicitado. Si lo encuentra, devuelve la dirección numérica asociada a este nombre. Si no la encuentra se lo hace saber al Unificador.

---

[1] Apéndice D.8, líneas 29 a 78.
[2] Apéndice D.8, líneas 87 a 100.
[3] Apéndice D.8, líneas 113 a 130.

3. El Enlace

El objeto Enlace provee los servicios necesarios para que el Unificador pueda comunicarse con el Intermediario (*vid* Apéndice D.11). La comunicación hacia el Intermediario se logra a través de una conexión serial que utiliza el protocolo RS-232. Si fuera necesario cambiar el medio de comunicación entre el Unificador y el Intermediario sólo se requiere cambiar la implementación de este objeto.

a. Puerto Serial

Puesto que la comunicación hacia el Intermediario se logra a través del puerto serial, el objeto Enlace utiliza un objeto Puerto Serial que se encarga de manejar este puerto y proveer acceso a él.

El objeto Puerto Serial provee dos servicios básicos: enviar un *byte* y recibir un *byte*. Además este objeto debe ser inicializado para abrir y configurar apropiadamente el puerto serial de la computadora (*vid* Apéndice D.9).

En un sistema operativo Linux, el puerto serial es tratado como un archivo y tiene su representación en el sistema de archivos. Se accede a los puertos seriales a través de los archivos tipo '/dev/ttyS'. Para el caso particular del Unificador se utiliza el primer puerto serial de la computadora. Este puerto serial es representado por el archivo '/dev/ttyS0'. Este archivo debe tener los permisos de lectura y escritura apropiados para que el proceso del Unificador lo pueda acceder.

La iniciación del puerto serial consiste en abrir el archivo que representa al puerto serial y establecer sus parámetros[1]. Para la comunicación hacia el Intermediario la velocidad utilizada es de 115,200 *bits* por segundo y no se utiliza el *bit* de paridad.

---

[1] Apéndice D.10, líneas 79 a 120.

b. Iniciación

La iniciación del objeto Enlace consiste en crear e inicializar el objeto Puerto Serial[1]. Si se da algún problema en la inicialización del Puerto Serial, esto se hace saber al Unificador.

c. Finalización

En la finalización el objeto Puerto Serial es destruido para liberarlo apropiadamente y así quede disponible para ser utilizado por otra aplicación[2].

d. Enviar instrucción

Este servicio permite enviar al Intermediario una instrucción dirigida a un microcontrolador específico y recibir la respuesta dada a esa instrucción[3]. Este servicio recibe la dirección numérica del microcontrolador, un objeto Instrucción y devuelve un objeto Respuesta.

Como se mencionó anteriormente, el microcontrolador posee una cantidad limitada de memoria con la que se puede trabajar. Por esto, la cantidad de datos que se le pueden enviar o solicitar tiene un límite. Estas cantidades máximas son diferentes para ambos casos y sus valores actuales pueden cambiar, pero no sobrepasan los 100 *bytes*.

Antes de enviar la instrucción al Intermediario se realizan algunas verificaciones. Si la instrucción es de lectura, no se deben solicitar más *bytes* que la cantidad máxima soportada para lectura. Si la instrucción es de escritura, no se deben enviar más *bytes* que la cantidad máxima soportada para la escritura. La dirección debe estar dentro del rango permitido de 1 a 126. Si cualquiera de estos requisitos no se cumplen, se hace saber al Unificador.

---

[1] Apéndice D.12, líneas 53 a 63.
[2] Apéndice D.12, líneas 72 a 75.
[3] Apéndice D.12, líneas 87 a 208.

La Instrucción es enviada *byte* por *byte* a través del puerto serial hacia el Intermediario. Si es una instrucción de escritura no hay forma de saber si el Intermediario la recibió, el Unificador asume que llegó. Si la instrucción es de lectura, se espera que los datos solicitados arriben por el puerto serial para colocarlos en un objeto Respuesta. Como medida de seguridad, a cada *byte* de respuesta se le da un máximo de un segundo para arribar. Esto para impedir que el Unificador se quede esperando eternamente la respuesta. Si este límite de tiempo se vence, la operación se da por inválida y se le hace saber al Unificador.

4. La Instrucción

El objeto Instrucción es utilizado para representar internamente la instrucción que un Programa Cliente envía hacia un microcontrolador (*vid* Apéndice D.14). No es un objeto activo como los demás que componen al Unificador, sino que funciona como una estructura de datos.

La Instrucción contiene un campo llamado Tipo que indica si la instrucción es de lectura o escritura. Otro campo llamado Cantidad indica la cantidad de datos a leer o escribir, dependiendo del tipo de instrucción. Además contiene un arreglo de *bytes* que contiene los datos a enviar al microcontrolador, si se trata de una instrucción de escritura.

El objeto Antena es la responsable de recibir del Programa Cliente la información para construir un objeto Instrucción. El primer *byte* que recibe la Antena es el tipo de instrucción, cero (0) si es una instrucción de lectura y uno (1) si es de escritura. El segundo *byte* indica la cantidad de datos a escribir o leer. Si es una instrucción de escritura se esperan también los *bytes* a enviar al microcontrolador, tantos *bytes* como lo indique el *byte* de cantidad.

Para hacer llegar esta instrucción al Intermediario, se le entrega al objeto Enlace. Este objeto construye una trama que será enviada por el puerto serial. El primer *byte* de esta trama es el caracter especial STX del conjunto de caracteres

ASCII (valor 2). El segundo *byte* es el que indica el tipo de instrucción, cero (0) si es una instrucción de lectura y uno (1) si es de escritura. El tercer *byte* indica la cantidad de datos a leer o escribir. A continuación se colocan los datos si se trata de una instrucción de escritura. La trama concluye con el carácter especial ETX del conjunto de caracteres ASCII (valor 3).

5. La Respuesta

Al solicitar al objeto Enlace que envíe una Instrucción hacia el Intermediario, el resultado de la operación es devuelto a través de un objeto Respuesta (*vid* Apéndice D.13). Al igual que el objeto Instrucción, éste funciona como una estructura de datos.

El objeto Respuesta contiene un campo llamado Resultado. Este indica el resultado de la operación de lectura o escritura. La operación pudo haber sido exitosa o se pudo dar algún problema. Las posibles causas de fallo son: sobrepaso de los límites de transferencia de datos, dirección inválida o recepción incompleta de datos provenientes del Intermediario, si se trata de una instrucción de lectura.

También contiene un arreglo de *bytes* para colocar los datos recibidos del Intermediario si se ha operado una instrucción de lectura. Además existe un campo llamado Cantidad que indica la cantidad de datos que se recibieron.

El Unificador recibe del objeto Enlace la Respuesta y envía al Programa Cliente en un *byte* el campo Resultado. Si la instrucción fue de lectura, se envían los *bytes* en el arreglo de datos, luego del *byte* de Resultado.

B. Funcionamiento del unificador

El Unificador se encuentra ejecutando un ciclo infinito que espera una conexión de un Programa Cliente. Al abrirse una conexión, el Unificador recibe el nombre del microcontrolador y la traduce a la dirección del

microcontrolador. Luego recibe la Instrucción y la envía junto con la dirección al Intermediario. Espera la respuesta del Intermediario, la envía al Programa Cliente y cierra la conexión.

Este ciclo se repite hasta que se le indique al Unificador que se detenga. Solamente se puede procesar una instrucción a la vez. En la figura siguiente se muestra el diagrama de flujo general para el Unificador.

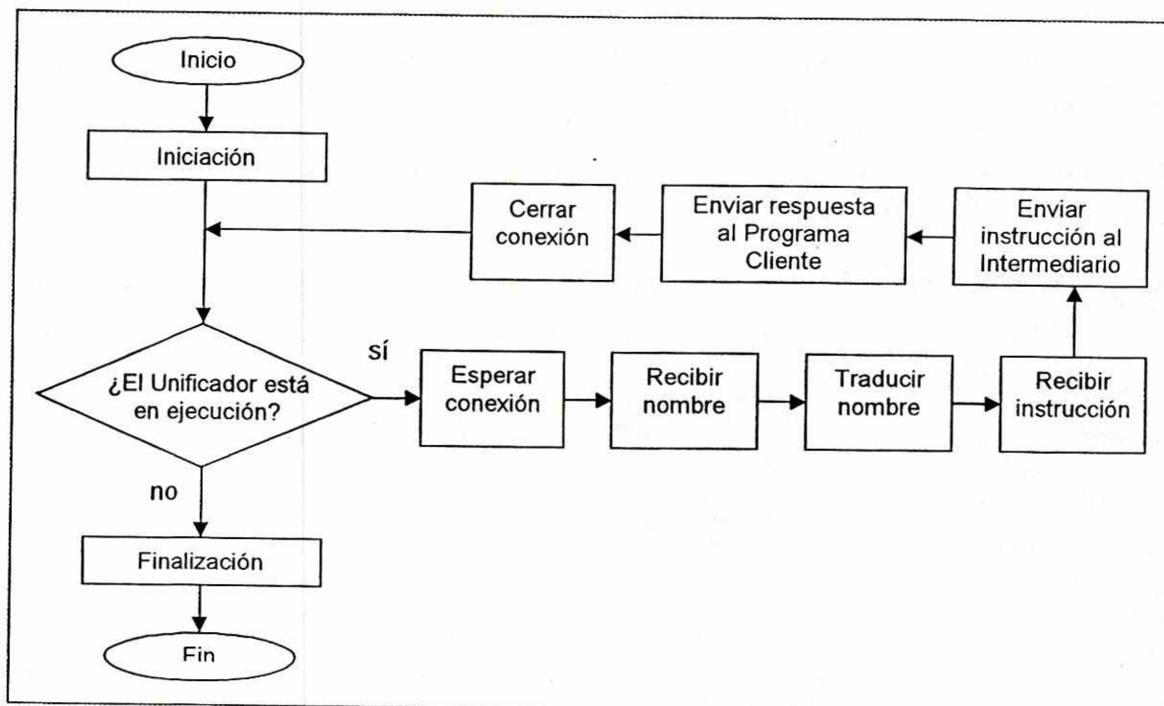


Figura 4 Diagrama de flujo general de ejecución del Unificador

En el proceso de iniciación se crean e inician la Antena, el Directorio y el Enlace. Además se abre el archivo de bitácora llamado logfile[1]. Este archivo lleva un registro de las acciones del Unificador. Si no existe al iniciar el Unificador, este archivo es creado.

Si ocurre algún error en la inicialización de cualquiera de los objetos, se emite un mensaje de error y se interrumpe la ejecución. En el proceso de finalización se destruyen todos los objetos utilizados.

[1] Apéndice D.1, líneas 99 a 156.

UNIVERSIDAD DEL VALLE DE GUATEMALA

Si la inicialización fue exitosa, se inicia el ciclo principal del Unificador. Se le indica a la Antena que espere una conexión[1]. Cuando haya alguna disponible, se el pide a la Antena que reciba el Nombre del microcontrolador[2].

El nombre no tiene una cantidad definida de caracteres, así que la Antena utiliza el caracter de terminación de cadena ('\0') para determinar cuando se ha capturado todo el nombre. Si este caracter nunca llegara, la Antena se quedaría esperándolo por tiempo indefinido, interrumpiendo la ejecución del Unificador. Para evitar que esto suceda, se establece un límite de tiempo de dos segundos. Si al pasar este límite no se recibe el nombre completo, ha ocurrido un error en la recepción del nombre. Esto se hace saber al Programa Cliente enviándole el código de error tres (3). Los códigos de error que se mencionarán en esta sección están listados en la figura siguiente. La conexión al Programa Cliente se cierra y se reinicia el ciclo principal esperando otra conexión.

Códigos de error en la recepción de la instrucción

0	La instrucción fue recibida correctamente.
1	La instrucción no se recibió correctamente.
2	Código no utilizado
3	El nombre del microcontrolador es incorrecto.

Códigos de error en la ejecución de la instrucción

0	La instrucción se ejecutó con éxito.
1	La cantidad de datos enviados o solicitados sobrepasa el máximo.
2	Dirección inválida o inexistente.
3	No se pudo establecer comunicación con el Intermediario.

Figura 5 Códigos de error emitidos por el Unificador

Si el nombre se recibió correctamente se le entrega al Directorio para que lo traduzca a la dirección correspondiente[3]. Si el Directorio no tiene registrado ese nombre, ha ocurrido un error en la traducción del nombre. Se envía al Programa Cliente el código de error tres (3). La conexión se cierra y se reinicia el ciclo principal.

---

[1] Apéndice D.1, línea 184.
[2] Apéndice D.1, líneas 209 a
[3] Apéndice D.1, líneas 221 a 228.

Ahora el Unificador solicita a la Antena que reciba la instrucción[1]. La Antena lo hace y construye un objeto Instrucción. Como en el caso de la recepción del nombre, en la recepción de la instrucción es posible que la Antena se quede esperando datos y estos no arriben. La solución en este caso es también establecer un límite de tiempo de dos segundos. Si la Antena no recibe la instrucción completa en ese límite de tiempo, ha ocurrido un error en la recepción de la instrucción. Se envía al Programa Cliente el código de error uno (1). La conexión se cierra y se reinicia el ciclo principal.

A este punto el Unificador ha recibido correctamente la instrucción y sabe a qué microcontrolador enviarla. Esta condición del procesamiento se hace saber al Programa Cliente enviándole el código cero (0).

El Unificador envía el objeto Instrucción y la dirección al Enlace para que éste, a su vez, los envíe al Intermediario[2]. Del Enlace se recibe un objeto Respuesta, que representa la respuesta del Intermediario. Al Programa Cliente se le envía el código de Resultado contenido en la Respuesta[3], que indica si la instrucción pudo ser procesada por el Intermediario o si hubo algún error. Si la instrucción es de lectura, se le envían al Programa Cliente los *bytes* retornados por el Intermediario en la Respuesta[4]. La conexión se cierra y se reinicia el ciclo principal.

[1] Ver Apéndice D.1, líneas 231 a 240.
[2] Ver Apéndice D.1, línea 248.
[3] Ver Apéndice D.1, línea 252.
[4] Ver Apéndice D.1, línea 266.

V.EL PROGRAMA CLIENTE

El Programa Cliente es un programa escrito en cualquier lenguaje y que se ejecuta en cualquier computadora manejada por cualquier sistema operativo. El único requisito indispensable que debe cumplir un Programa Cliente es que debe ser capaz de comunicarse a la computadora donde está en ejecución el Unificador[1], utilizando los protocolos TCP/IP.

Este Programa Cliente puede estar dedicado a controlar un proceso, puede estar simplemente colectando información para análisis estadísticos posteriores o puede servir como un puente para otro programa que desea acceder a uno o varios microcontroladores. Para realizar su tarea puede enviar tantas instrucciones al Unificador como necesite. Para cada instrucción a enviar deberá abrir una nueva conexión.

A. La comunicación hacia el unificador

Cada sistema operativo provee el medio para establecer comunicaciones TCP/IP. Windows por ejemplo utiliza un mecanismo denominado WinSock, un conjunto de librerías que proveen servicios de comunicación. En Unix se utilizan los *sockets*, explicados en el capítulo anterior. Pero no importa el medio utilizado, si se siguen los protocolos TCP/IP, la comunicación podrá realizarse sin problemas.

El primer paso es establecer la conexión hacia el Unificador. Para ello el Programa Cliente debe conocer la dirección IP del Servidor. Es posible que se maneje un esquema de nombres para las computadoras y que solamente se conozca el nombre del Servidor. Entonces será necesario solicitar al servicio de resolución de nombres (DNS) que traduzca el nombre del Servidor a su dirección IP.

[1] A la computadora donde está en ejecución el Unificador se le llamará Servidor a lo largo de este capítulo.

Una vez conocida la dirección, se debe pedir al medio de comunicación establecer una conexión hacia esa dirección por el puerto 3232. Si la conexión no se pudo realizar, el Unificador no está en ejecución. Esto asumiendo que el Servidor es accesible desde la computadora que está ejecutando el Programa Cliente y no hay problemas con la configuración de la red.

El siguiente paso es enviar el nombre del microcontrolador al que se quiere enviar la instrucción. Como se explicó anteriormente, el nombre está compuesto de una serie de caracteres de un *byte* cada uno, seguida del carácter de terminación ('\0).

Luego se deben enviar dos *bytes*. El primero indica el tipo de instrucción, cero (0) para una instrucción de lectura, uno (1) para una instrucción de escritura. El segundo indica la cantidad de datos a leer o escribir. Si la instrucción es de escritura, se deben enviar los *bytes* que se desean hacer llegar al microcontrolador.

Ahora el Programa Cliente debe recibir un *byte* que indica si el Unificador pudo recibir la instrucción o si hubo algún problema. Si el *byte* recibido es cero (0), el Unificador recibió la instrucción sin ningún problema. Si es uno (1), hubo un error no determinado en la recepción de la instrucción. Si es dos (2), los datos recibidos por el Unificador son menos de los que se indicó en la instrucción. Si es tres (3), el nombre del microcontrolador no se pudo recibir o no se pudo traducir. Si se recibe uno de los últimos tres códigos, la conexión se invalida, pues al ocurrir un error el Unificador cierra la conexión.

Si no ocurrió un error se debe recibir otro *byte* que indica si la instrucción pudo ejecutarse o si hubo algún problema. Si este byte es cero (0), la instrucción se ejecutó exitosamente. Si es uno (1), la cantidad de datos solicitados o enviados, sobrepasa la capacidad del microcontrolador. Si es dos (2), la dirección obtenida a partir del nombre no es válida. Si es tres (3), el Unificador no pudo establecer comunicación con el microcontrolador. Si se recibe uno de los últimos tres códigos, la conexión se invalida, al igual que en el caso anterior.

Si el Programa Cliente envió una instrucción de lectura, ahora debe recibir la cantidad de *bytes* que solicitó. La interacción entre el Programa Cliente y el Unificador ha concluido y se le debe pedir al medio de comunicación que cierre la conexión.

B. Ejemplos de programas cliente

Ahora se mostrarán dos ejemplos de Programas Cliente para ilustrar la interacción con el Unificador antes explicada. Estos programas están escritos en lenguaje C para ser ejecutados en una computadora personal con sistema operativo Linux.

Esta arquitectura puede ser utilizada junto con Internet para lograr control de procesos de forma remota. Para mostrar esta capacidad, los Programas Cliente de ejemplo son programas tipo Common Gateway Interface (CGI). Estos programas son utilizados para interactuar con un servidor web: reciben solicitudes del usuario desde un navegador y generan páginas que se ajusten a esta solicitud. Por ejemplo, una página de resultados de una búsqueda no existe antes de que el usuario realice la búsqueda, es generada a partir del criterio dado por el usuario, pues cada usuario solicita una búsqueda diferente. Estos programas se almacenan y ejecutan en la misma computadora donde está el servidor web.

En la figura siguiente se puede apreciar la forma en que interactúan el navegador, el servidor web y el programa CGI. El navegador envía al servidor web la solicitud de una página junto con una cadena de caracteres que son los argumentos que se le entregarán al CGI para que realice su tarea. El servidor web invoca al programa CGI y le entrega la cadena de caracteres que envió el navegador. El programa CGI realiza su tarea y envía de vuelta al servidor web el resultado de esta tarea y el servidor web a su vez la envía al navegador. El resultado puede ser una página web u otro tipo de archivo que el navegador pueda interpretar (December, 1996:483).

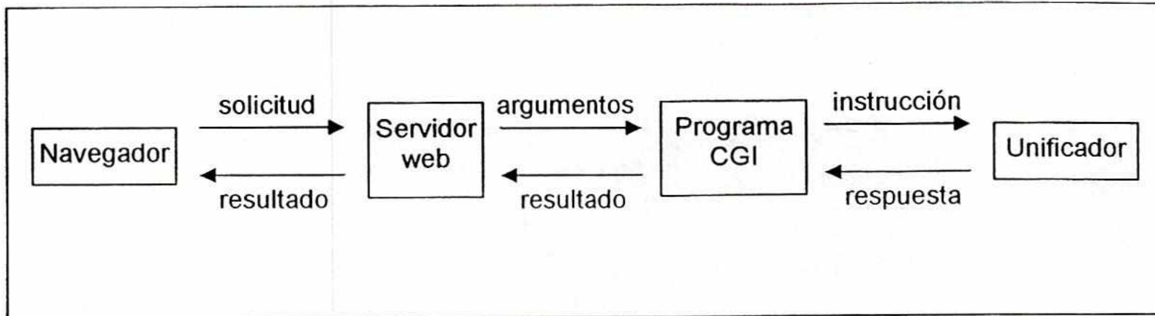


Figura 6 Interacción entre el navegador, el servidor web y un CGI

Los programas de ejemplo realizan una conexión hacia el Unificador para realizar su tarea y enviar el resultado en forma de una página web que el usuario puede observar en su navegador. A pesar de que el programa CGI debe estar en la misma computadora donde está el servidor web, no es necesario que en esa misma esté en ejecución el Unificador.

1. Programa Cliente: Monitor de ventilador

Este programa permite visualizar en un navegador la temperatura cercana a un ventilador y el estado del mismo. La temperatura está dada en grados Celsius y el estado del ventilador puede ser: apagado, velocidad 1, velocidad 2 o velocidad 3. Además permite cambiar el estado del ventilador (*vid* Apéndice E.1).

Al ser invocado, este programa verifica si el usuario solicitó el cambio de estado del ventilador[1]. Si se solicitó un cambio de estado, se intenta realizar una conexión al Unificador. Si el intento falla, se informa al usuario desplegando un mensaje de texto en el navegador[2].

Si se logra establecer la conexión, se envía el nombre del microcontrolador que maneja el ventilador, en este caso el nombre es "ventilador", seguido del carácter de terminación ('\0'). Luego se envía el tipo de instrucción que en este caso es de escritura, pues se le debe enviar al microcontrolador cuál será el

[1] Apéndice E.1, línea 84.
[2] Apéndice E.1, líneas 86 a 99.

nuevo estado del ventilador. Se envía la cantidad de datos que es un *byte* y luego se envía el *byte* que indica el nuevo estado del ventilador indicado por el usuario (0 = apagado, 1 = velocidad 1, 2 = velocidad 2, 3 = velocidad 3)[1].

Ahora el Programa Cliente debe verificar si la instrucción fue recibida por el Unificador. Para ello lee un *byte* de la conexión, si es 0 continúa, si no envía al navegador el mensaje de error correspondiente al error que ha ocurrido. Luego verifica que la instrucción haya sido ejecutada. En este caso también lee un *byte* de la conexión, si es 0 continúa, si no envía al navegador el mensaje correspondiente[2].

Si no ocurrió ningún error, la interacción con el Unificador ha concluido y la conexión se cierra. Se espera un segundo a que el cambio solicitado ocurra y se continúa con la ejecución[3].

Ahora se solicita al microcontrolador la temperatura y el nuevo estado del ventilador para desplegarlos en el navegador del usuario. Este paso se realiza si el usuario solicitó un cambio de estado o si se está invocando al Programa Cliente por primera vez por el mismo navegador. Se debe establecer una nueva conexión con el Unificador. Si la conexión no se pudo realizar, se informa al usuario desplegando en el navegador un mensaje de texto[4]. Si la conexión se pudo abrir, se envía el nombre del microcontrolador junto con el carácter de terminación. Se envía el tipo de instrucción, que en este caso será de lectura, pues se desea saber el estado del ventilador; y se envía la cantidad de *bytes* que se necesitan, dos en este caso[5].

Luego se verifica que la instrucción haya sido recibida y que haya sido exitosamente ejecutada[6]. A continuación se leen de la conexión los dos *bytes* solicitados. El primero indica la temperatura en grados Celsius alrededor del

[1] Apéndice E.1, líneas 104 a 121.
[2] Apéndice E.1, líneas 126 a 154.
[3] Apéndice E.1, líneas 160 a 163.
[4] Apéndice E.1, líneas 172 a 184.
[5] Apéndice E.1, líneas 189 a 199.
[6] Apéndice E.1, líneas 204 a 231.

ventilador. El segundo indica el estado actual del ventilador (0 = apagado, 1 = velocidad 1, 2 = velocidad 2, 3 = velocidad 3)[1]. Con estos datos se construye la página web que desplegará el navegador. La conexión se cierra y el Programa Cliente termina su ejecución.

2. Programa Cliente: Monitor de osciloscopio

Este programa permite visualizar en el navegador la misma gráfica que se ve en la pantalla de un osciloscopio digital. Esta gráfica se compone de aproximadamente 9,000 puntos, cada punto está representado por un *byte*. Puesto que existe un límite no mayor de 100 *bytes* para solicitar *bytes* al microcontrolador, se deben realizar múltiples solicitudes (*vid* Apéndice E.2).

El Programa Cliente es básicamente un ciclo que envía una instrucción de lectura un número determinado de veces. Es posible que el Programa Cliente solicite datos más rápido de lo que el microcontrolador los puede generar. Para evitar esto, primero se envía una instrucción de lectura de un *byte* para determinar si el microcontrolador tiene datos disponibles. Si el *byte* es el carácter 'R', se solicita el paquete de *bytes*, si el carácter es 'N' se espera y se vuelve a intentar.

El ciclo inicia intentando realizar una conexión con el Unificador. Si la conexión no se puede realizar se le envía al usuario un mensaje de texto por el navegador[2].

Luego se envía el nombre del microcontrolador seguido por el carácter de terminación ('\0). Se envía el tipo de instrucción, lectura en este caso, y se envía la cantidad de datos a leer[3]. Luego se verifica si la instrucción fue recibida correctamente y si fue procesada exitosamente[4]. Si hubiera ocurrido algún error,

---

[1] Apéndice E.1, línea 236.
[2] Apéndice E.2, líneas 90 a 105.
[3] Apéndice E.2, líneas 110 a 120.
[4] Apéndice E.2, líneas 125 a 157.

este se le indica al usuario enviando un mensaje de texto al navegador y la ejecución se interrumpe.

Se recibe el *byte* solicitado y se verifica que sea el carácter 'R'. Si es así, se abre una nueva conexión, se envía el nombre del microcontrolador, el tipo de instrucción y la cantidad de datos a solicitar[1]. Se verifica que el Unificador haya recibido la instrucción y que haya sido ejecutada exitosamente[2].

Luego se reciben los datos solicitados y se acumulan hasta recibir todos los puntos de la gráfica[3]. Al terminar la solicitud de datos se genera la gráfica y se envía al navegador para ser desplegada[4].

[1] Apéndice E.2, líneas 175 a 206.
[2] Apéndice E.2, líneas 212 a 244.
[3] Apéndice E.2, líneas 250 a 258.
[4] Apéndice E.2, líneas 275 a 277.

CONCLUSIONES

- El uso de esta arquitectura permite independencia de plataforma para el desarrollo de Programas Cliente que necesiten acceder a la información en microcontroladores. Es decir que el Programa Cliente puede estar escrito en cualquier lenguaje, ser ejecutado en cualquier tipo de computadora, y sobre cualquier sistema operativo que provea medios de comunicación utilizando los protocolos TCP/IP.

- Por la modularidad empleada en el diseño y construcción del Unificador, cada elemento del Unificador puede ser fácilmente modificado sin afectar a los demás elementos. El funcionamiento de los Programas Cliente tampoco se vería afectado.

- Esta arquitectura facilita y acelera el desarrollo de aplicaciones de control de procesos. Una operación que sea difícil de programar en el microcontrolador, podrá trasladarse a un programa en un sistema de cómputo de mayor nivel. Este sistema de cómputo provee mayores recursos para el desarrollo, como depuradores, lenguajes de alto nivel, facilidad de acceso a otras fuentes de información y librerías de software.

  Estas características permitirán construir soluciones económicas para empresas cuyo presupuesto no soportaría el utilizar soluciones propietarias que actualmente se ofrecen.

- Esta arquitectura combinada con el uso de programas tipo CGI permite el acceso a microcontroladores desde navegadores de Internet. Estas aplicaciones tienen la ventaja de que pueden ser ejecutadas desde computadoras donde esos programas no están instalados. En un viaje imprevisto se daría esta situación.

- Aunque la versión del Unificador que se presenta en este trabajo es una versión estable y funcional, aún hay aspectos que necesitan ser revisados y mejorados.

RECOMENDACIONES

- El protocolo de comunicación empleado para que el Unificador se comunique con el Intermediario debe ser mejorado, pues en el caso de enviar una instrucción de escritura, el Unificador no sabe si el Intermediario la recibió.

- Es necesario agregar una capa que provea secretividad en la comunicación entre el Unificador y el Programa Cliente. Es decir que los mensajes que se intercambian no puedan ser interpretados por algún intruso en la comunicación.

- Es necesario agregar un medio que permita al Unificador identificar y autenticar al Programa Cliente para que el acceso a los microcontroladores sea restringido y brindar mayor seguridad al sistema.

- Es necesario desarrollar un nuevo medio de comunicación entre el Unificador y el Intermediario, pues el puerto serial cada vez es menos utilizado. Además el puerto serial no provee la velocidad de comunicación necesaria para transmitir gran cantidad de datos.

- Los microcontroladores actualmente deben estar conectados a la misma fuente de poder para que la comunicación por el bus I^2C pueda realizarse pues se necesita una referencia común para interpretar las señales que viajan por el alambre de cobre. Se recomienda que esta referencia común viaje junto con el bus o que se cambie de alambre de cobre a fibra óptica para evitar la necesidad de una referencia para interpretar la señal.

- Puede desarrollarse un objeto que se acople a la arquitectura Common Object Request Broker Architecture (CORBA) para acceder a los microcontroladores de una forma más transparente y que este objeto pueda ser utilizado por cualquier programa que tenga la capacidad de solicitar objetos CORBA.

BIBLIOGRAFÍA

December, John; M. Ginsburg. 1996. *HTML 3.2 & CGI.* Indianápolis, Sams.net Publishing. 1321 págs.

Jones, Clarence. 1996. *Programable Logic Controllers.* Atlanta, Patrick-Turner. 457 págs

Mandado, Enrique, *et al.* 1999. *Controladores lógicos y autómatas programables.* 2ª ed. Barcelona, Alfaomega. 393 págs.

Robbins, Kay; S. Robbins. 1996. *Practical UNIX programming.* New Jersey. Prentice Hall. 658 págs.

Silverschatz, A. P. Galvin. 1999. *Sistemas operativos.* 5ª ed. Massachussets, Addison Wesley Longman. 889 págs.

Stevens, Richard. 1998. *Unix Network Programming.* 2ª ed. New Jersey, Prentice-Hall. 1009 págs.

Wilhelm, Robert. 1985. *Programmable Controller Handbook.* Indianápolis. Hayden Books. 718 págs.

APÉNDICE A

TRADUCCIÓN DE LA ESPECIFICACIÓN DE GENERAL MOTORS PARA UN CONTROLADOR LÓGICO PROGRAMABLE (1968)

TRADUCCIÓN DE LA ESPECIFICACIÓN DE GENERAL MOTORS PARA UN CONTROLADOR LÓGICO PROGRAMABLE (1968)

Los requerimientos que General Motors exigía de un controlador lógico programable en su especificación son los siguientes:

1. Debe ser fácil y rápidamente programable y reprogramable con un mínimo de interrupción del servicio.

2. Todos sus componentes deben ser capaces de operar en plantas industriales sin necesidad de equipo de soporte, herramientas o ambientes especiales.

3. Debe ser de fácil mantenimiento y reparación. Su diseño debe incluir indicadores de estado y el uso de modularidad para facilitar la reparación y la resolución de problemas en el mínimo de tiempo posible.

4. Debe ocupar menos espacio en la planta que el sistema de control por relevadores que reemplace, puesto que el espacio implica un costo. Además debe consumir menos energía eléctrica para operar que los actuales sistemas de control por relevadores.

5. Debe ser capaz de comunicarse con sistemas centrales de recolección de información para control de sus operaciones.

6. Debe ser capaz de aceptar señales de 120V AC provenientes de botones e interruptores estándar de sistemas de control existentes.

7. Las señales de salida deben ser capaces de manejar estárter de motores y válvulas de solenoides que operan a 120V AC. Cada salida debe ser diseñada para operar continuamente un dispositivo de 2 amperios.

8. Debe poder expandirse desde su configuración mínima hasta su configuración máxima con el mínimo de alteración en el sistema y en el mínimo de tiempo.

9. Debe ser competitiva en precio y costo de instalación en comparación a los sistemas de relevadores y de lógica de estado sólido que se utilizan actualmente.

10. La estructura de memoria empleada debe poder expandirse a un mínimo de 4000 palabras o elementos.

(Wilhelm, 1985:6)

APÉNDICE B

Página 1 del documento Microchip PIC16F87X Data Sheet

PIC16F87X

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

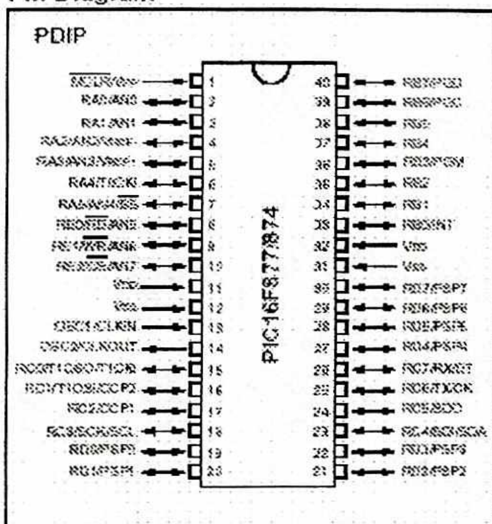
Devices Included in this Data Sheet:

- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram



Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I²C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

© 2001 Microchip Technology Inc.

APÉNDICE C

Página 13 del documento Microchip PIC16F87X Data Sheet

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP

Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3	File Address
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved(2)	18Fh
T1CON	10h		90h	General Purpose Register 16 Bytes	110h	General Purpose Register 16 Bytes	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h
			EFh		16Fh		1EFh
		accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
	7Fh		FFh		17Fh		1FFh

(shaded) Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

APÉNDICE D

Código fuente del Unificador

APÉNDICE D.1

Archivo: unificador.cc

```
 1//**********************************************************
 2//
 3//  Nombre del archivo:    unificador.cc
 4//  Proyecto:  Unificador
 5//  Autor:  Fredy Munoz
 6//  Lugar y fecha: Guatemala, febrero 2003.
 7//  Descripcion: Contiene el codigo principal del Unificador
 8//
 9//**********************************************************
10
11
12
13#include "Enlace/TEnlace.h"
14#include "Antena/TAntena.h"
15#include "Directorio/TDirectorio.h"
16
17
18#include <iostream.h>
19#include <fstream.h>
20#include <sys/types.h>
21#include <sys/stat.h>
22#include <sys/signal.h>
23#include <fcntl.h>
24#include <time.h>
25
26
27
28#define UNIFICADOR_INSTRUCCION_OK     0   //La instruccion fue recibida correctamente
29#define UNIFICADOR_INSTRUCCION_ERROR  1   //La instruccion no se recibio correctamente
30#define UNIFICADOR_INSTRUCCION_DATOS  2   //Los datos recibidos estan incompletos
31#define UNIFICADOR_INSTRUCCION_DIR    3   //La direccion no se recibio correctamente
32
33
34
35
36
37TEnlace* Enlace;         //Puntero al objeto Enlace
38int estado_enlace;       //Almacena el estado de inicializacion del Enlace
39
40TAntena* Antena;         //Puntero al objeto Antena
41int estado_antena;       //Almacena el estado de inicializacion de la Antena
42
43TDirectorio* Directorio;  //Puntero al objeto Directorio
44int estado_directorio;    //Almacena el estado de inicializacion del Directorio
45
46TRespuesta respuesta;     //Estructura TRespuesta que almacena la respuesta de un
47                          //Microcontrolador a una instruccion de lectura
48
49TInstruccion instruccion; //Objeto TInstruccion que almacena la instruccion a enviar
50                          //al Microcontrolador
51
52char nombre[200];         //Nombre del Microcontrolador al que se enviara la
53                          //instruccion
54int direccion;            //Direccion del Microcontrolador al que se enviara la
55                          //instruccion
56
57int conexion;      //Almacena el resultado de abrir una nueva conexion
58
59int en_ejecucion;  //Indica si el Unificador esta en ejecucion, 1 = si, 0 = no
60
61
62
63//*********************************************************
64//  Funcion que se ejecuta al vencer el time-out establecido
65//  al solicitar informacion de la Antena. Si no hay informacion
66//  disponible, se espera hasta que se caduque el time-out,
67//  se envia un mensaje de error y se cierra la conexion.
68//*********************************************************
69
70void alarm_antena( int status )
71{
```

```
72      Antena->EnviarEstado( UNIFICADOR_INSTRUCCION_ERROR );
73      Antena->CerrarConexion();
74 }
75
76
77
78 //********************************************************
79 //  Funcion que se ejecuta al recibir la senal de terminar
80 //  la ejecucion. Cierra la Antena para no recibir mas conexiones
81 //  y detiene la ejecucion del Unificador
82 //********************************************************
83
84 void terminar( int status )
85 {
86      delete Antena;
87      en_ejecucion = 0;
88 }
89
90
91
92 //********************************************************
93 //  Funcion principal que inicia la ejecucion del Unificador
94 //********************************************************
95
96 int main()
97 {
98      time_t tiempo;
99      ofstream logfile( "logfile", ios::app ); //Creacion del logfile
100
101     if ( !logfile )
102     {
103         cout << "No se pudo abrir el archivo 'logfile'" << endl;
104         return 1;
105     }
106
107
108
109
110     //Inicializacion del medio de comunicacion hacia el Intermediario (Enlace)
111     Enlace = new TEnlace( &estado_enlace );
112
113     //Chequeo del resultado de la inicializacion
114     if ( estado_enlace != ENLACE_OK )
115     {
116         cout << "Error al inicializar el Enlace" << endl;
117         return 1;
118     }
119     else
120         cout << "Enlace inicializado" << endl;
121
122
123
124
125
126     //Inicializacion del medio de comunicacion al exterior (Antena)
127     Antena = new TAntena( &estado_antena );
128
129     //Chequeo del resultado de la inicializacion
130     if ( estado_antena != ANTENA_OK )
131     {
132         cout << "Error al inicializar la Antena" << endl;
133         return 1;
134     }
135     else
136         cout << "Antena inicializada" << endl;
137
138
139
140
141
142     //Inicializacion del Directorio
```

```
143     Directorio = new TDirectorio( &estado_directorio );
144
145     //Chequeo del resultado de la inicializacion
146     if ( estado_directorio == DIRECTORIO_NO_ARCHIVO )
147     {
148         cout << "No se encontro el archivo 'directorio'" << endl;
149         return 1;
150     }
151     if ( estado_directorio == DIRECTORIO_ERROR_ARCHIVO )
152     {
153         cout << "Se encontro un error en el archivo 'directorio'" << endl;
154         return 1;
155     }
156     cout << "Directorio inicializado" << endl;
157
158
159
160     //Se indica la funcion a llamar cuando se ordene detener el Unificador
161     signal( SIGTERM, terminar );
162
163
164
165     cout << "Inicializacion completa" << endl;
166     cout << "Unificador listo y esperando conexiones" << endl;
167     tiempo = time( NULL );
168     logfile << "++++++++++++++++++++++++++++++++++" << endl;
169     logfile << ctime( &tiempo ) << "Unificador iniciado" << endl;
170
171
172     //Ciclo principal
173     en_ejecucion = 1;
174     conexion = CONEXION_NOK;
175
176     while ( en_ejecucion )
177     {
178         //Si al iniciar la siguiente vuelta del ciclo existe una conexion cerrarla
179         if ( conexion == CONEXION_OK )
180             Antena->CerrarConexion();
181
182
183         //Espera una conexion del mundo exterior
184         conexion = Antena->EsperarConexion();
185
186         //Si el Unificador ya no esta en ejecucion, salirse del ciclo y del programa
187         if ( !en_ejecucion )
188             return 0;
189
190
191         //Indicar la recepcion de una conexion
192         tiempo = time( NULL );
193         logfile << endl << ctime( &tiempo ) << "Conexion: ";
194
195         if ( conexion != CONEXION_OK )
196         {
197             logfile << "No se pudo aceptar la conexion" << endl;
198             continue;
199         }
200         else
201             logfile << "Conexion establecida" << endl;
202
203
204
205         //Establecer la funcion que se ejecutara al vencer el time-out de la antena
206         signal( SIGALRM, alarm_antena );
207
208
209         //Recepcion del nombre del Microcontrolador al que va dirigida la instruccion
210         alarm( 2 ); //Establecer el time-out a dos segundos
211         if ( Antena->RecibirNombre( nombre ) != NOMBRE_OK )
212         {
213             Antena->EnviarEstado( UNIFICADOR_INSTRUCCION_DIR );
```

```
214             logfile << "No se recibio correctamente el nombre" << endl;
215             alarm( 0 ); //Anular el time-out
216             continue;
217         }
218         alarm( 0 ); //Anular el time-out
219
220
221         //Traducir el nombre a la direccion numerica
222         direccion = Directorio->Traducir( nombre );
223         if ( direccion == -1 )
224         {
225             Antena->EnviarEstado( UNIFICADOR_INSTRUCCION_DIR );
226             logfile << "Nombre no registrado en el directorio" << endl;
227             continue;
228         }
229
230
231         //Recepcion de la instruccion a procesar
232         alarm( 2 ); //Establecer el time-out a dos segundos
233         if ( Antena->RecibirInstruccion( &instruccion ) != INSTRUCCION_OK )
234         {
235             Antena->EnviarEstado( UNIFICADOR_INSTRUCCION_ERROR );
236             logfile << "No se recibio correctamente la instruccion" << endl;
237             alarm( 0 ); //Anular el time-out
238             continue;
239         }
240         alarm( 0 ); //Anular el time-out
241
242
243         //Indicar que la instruccion se recibio de forma correcta
244         Antena->EnviarEstado( UNIFICADOR_INSTRUCCION_OK );
245
246
247         //Enviar la instruccion al Intermediario
248         respuesta = Enlace->EnviarInstruccion( direccion, instruccion );
249
250
251         //Enviar el resultado del envio de la instruccion hacia el Intermediario
252         Antena->EnviarEstado( respuesta.Resultado );
253
254
255         //Verificar que la instruccion se proceso con exito
256         if ( respuesta.Resultado != RESPUESTA_OK )
257         {
258             logfile << "Instruccion NO procesada" << endl;
259             continue;
260         }
261         else
262             logfile << "Instruccion procesada exitosamente" << endl;
263
264
265         //Si es una instruccion de lectura enviar los datos a la conexion
266         if ( instruccion.Tipo == INSTRUCCION_LECTURA )
267             Antena->EnviarRespuesta( respuesta );
268
269     }
270
271     Antena->CerrarConexion();
272
273 }
274
275 //Fin del archivo "unificador.cc"
```

APÉNDICE D.2

Archivo: red.h

```
 1//**********************************************************
 2//
 3//  Nombre del archivo:    red.h
 4//  Proyecto:  Unificador
 5//  Autor:  Fredy Munoz
 6//  Lugar y fecha: Guatemala, febrero 2003.
 7//  Descripcion: Contiene las librerias del sistema necesarias
 8//          para la comunicacion por una red TCP/IP
 9//
10//**********************************************************
11
12
13
14 #ifndef _red_h_
15 #define _red_h_
16
17 #include <sys/types.h>
18 #include <sys/socket.h>
19 #include <netinet/in.h>
20 #include <arpa/inet.h>
21 #include <errno.h>
22 #include <unistd.h>
23
24 #endif
25
26 //Fin del archivo "red.h"
```

APÉNDICE D.3

Archivo: TAntena.h

```
1//*****************************************************
2//
3//  Nombre del archivo:    TAntena.h
4//  Proyecto:  Unificador
5//  Autor:  Fredy Munoz
6//  Lugar y fecha: Guatemala, febrero 2003.
7//  Descripcion: Contiene la definicion del objeto TAntena
8//       que permite al Unificador recibir comunicaciones
9//       provenientes de otros programas
10//
11//*****************************************************
12
13
14
15 #ifndef _antena_h_
16 #define _antena_h_
17
18 #include "red.h"
19 #include "../Tipos/TInstruccion.h"
20 #include "../Tipos/TRespuesta.h"
21
22
23 #define ANTENA_OK          0   //La antena pudo ser inicializada correctamente
24 #define ANTENA_NOK         1   //Surgio algun problema al inicializar la antena
25
26 #define CONEXION_OK        0   //La conexion pudo realizarse correctamente
27 #define CONEXION_NOK       1   //El intento de conexion no pudo concretarse
28
29 #define NOMBRE_OK          0   //El nombre del Microcontrolador fue recibido
30                                //correctamente
31 #define NOMBRE_NOK         1   //El nombre del Microcontrolador no fue recibido
32
33 #define INSTRUCCION_OK      0   //La instruccion fue recibida correctamente
34 #define INSTRUCCION_NOK     1   //La instruccion no fue recibida
35
36
37
38 class TAntena
39 {
40     private:
41         int SocketEscuchar;                 //Socket utilizado para recibir conexiones
42                                             //provenientes de otros programas
43         int Conexion;                       //Socket utilizado para atender a una conexión
44                                             //recibida
45         sockaddr_in DireccionServidor;       //Estructura que almacena la direccion del
46                                              //Socket utilizado para escuchar
47
48     public:
49         TAntena(int*);                   //Constructor del objeto TAntena que devuelve su
50                                          //estado
51         ~TAntena();                      //Destructor del objeto TAntena
52
53         int EsperarConexion();     //Funcion que espera alguna conexion del exterior
54         void CerrarConexion();     //Funcion que cierra la conexion actual
55
56         int RecibirNombre( char* );        //Funcion que recibe el nombre del
57                                                // Microcontrolador que se desea acceder
58         int RecibirInstruccion( TInstruccion* ); //Funcion que recibe la Instrucción
59                                                  //que se desea enviar al
60                                                  //Microcontrolador
61
62         void EnviarEstado( char );           //Funcion que envia el resultado de la
63                                              //Instruccion y mensajes de error
64         void EnviarRespuesta( TRespuesta );      //Funcion que envia la respuesta del
65                                                  //Microcontrolador al programa que lo
66                                                  //solicito
67 };
68
69 #endif
70
71 //Fin del archivo "TAntena.h"
```

APÉNDICE D.4

Archivo: TAntena.cc

```
1 //*************************************************************
2 //
3 //  Nombre del archivo:    TAntena.cc
4 //  Proyecto:  Unificador
5 //  Autor:  Fredy Munoz
6 //  Lugar y fecha: Guatemala, febrero 2003.
7 //  Descripcion: Contiene la implementacion del objeto TAntena
8 //      que permite al Unificador recibir comunicaciones
9 //      provenientes de otros programas
10 //
11 //*************************************************************
12
13
14
15 #include "TAntena.h"
16
17 #include <string.h>
18
19
20 #define SERVIDOR_PUERTO    3232    //puerto TCP por el que la Antena recibe las
21                                    //conexiones
22
23
24
25 //************************************************************************
26 //  Constructor del objeto TAntena que inicializa el socket utilizado para
27 //  recibir conexiones de otros programas
28 //  ->estado: puntero a un entero que indica el resultado de la inicializacion
29 //************************************************************************
30
31 TAntena::TAntena( int* estado )
32 {
33     //Descripcion del socket por el que se escucharan las instrucciones
34     memset( &DireccionServidor, 0, sizeof(DireccionServidor) );
35     DireccionServidor.sin_family = AF_INET;
36     DireccionServidor.sin_addr.s_addr = htonl ( INADDR_ANY );
37     DireccionServidor.sin_port = htons( SERVIDOR_PUERTO );
38
39
40     //Apertura del socket por el que se esperaran las instrucciones
41     SocketEscuchar = socket( AF_INET, SOCK_STREAM, 0 );
42     if ( SocketEscuchar  < 0 )
43     {
44         *estado = ANTENA_NOK;
45         return;
46     }
47
48
49     //Vinculacion del socket con su descripcion
50     if( bind( SocketEscuchar, (sockaddr*) &DireccionServidor,
51 sizeof(DireccionServidor) ) < 0 )
52     {
53         *estado = ANTENA_NOK;
54         return;
55     }
56
57
58     //Se configura el socket para que escuche
59     if( listen( SocketEscuchar, 1024 ) < 0 )
60     {
61         *estado = ANTENA_NOK;
62         return;
63     }
64
65
66     *estado = ANTENA_OK;
67 }
68
69
70
71 //************************************************************************
```

```
72//  Destructor del objeto TAntena que cierra el socket utilizado para
73//  recibir conexiones de programas externos
74//**********************************************************************
75
76TAntena::~TAntena()
77{
78     close( SocketEscuchar );
79}
80
81
82
83//**********************************************************************
84//  Funcion que Espera a que un programa externo solicite una conexion
85//  <-Devuelve CONEXION_OK si la conexion se pudo realizar y CONEXION_NOK
86//    si el intento de conexion no se concreto
87//**********************************************************************
88
89int TAntena::EsperarConexion()
90{
91     Conexion = accept( SocketEscuchar, NULL, NULL );
92
93     if ( Conexion < 0 )
94         return CONEXION_NOK;
95
96     return CONEXION_OK;
97}
98
99
100
101//**********************************************************************
102//  Funcion que cierra la conexion que actualmente se encuetra abierta
103//**********************************************************************
104
105void TAntena::CerrarConexion()
106{
107     close( Conexion );
108}
109
110
111
112//**********************************************************************
113//  Funcion que recibe el nombre del Microcontrolador al que se quiere
114//  acceder
115//  ->nombre: puntero al arreglo de caracteres donde se colocara el nombre
116//  <-Devuelve DIRECCION_OK si se pudo recibir la direccion correctamente, de lo
117//    contrario devuelve DIRECCION_NOK
118//**********************************************************************
119
120int TAntena::RecibirNombre( char* nombre )
121{
122     char caracter;
123
124     //Inicializar el buffer donde se almacenara el nombre
125     nombre[0] = 0;
126
127     do
128     {
129         //Leer un caracter de la conexion, si no se pudo leer, devolver NOMBRE_NOK
130         if ( read( Conexion, &caracter, 1 ) < 1 )
131             return NOMBRE_NOK;
132
133         //Agregarlo al buffer del nombre
134         strncat( nombre, &caracter, 1 );
135     }while( caracter != 0 );  //Repetir hasta encontrar el fin de cadena
136
137     return NOMBRE_OK;
138}
139
140
141
142//**********************************************************************
```

```
143//  Funcion que recibe la Instruccion a enviar al Microcontrolador
144//  ->instruccion: puntero a un objeto de tipo TInstruccion donde se almacenara
145//    la instruccion que se desea enviar al Microcontrolador
146//  <-Devuelve INSTRUCCION_OK si se pudo recibir la instruccion correctamente,
147//    de lo contrario devuelve INSTRUCCION_NOK
148//*******************************************************************
149
150int TAntena::RecibirInstruccion( TInstruccion* instruccion )
151{
152     char cantidad;
153
154     //Se recibe el tipo de la instruccion, puede ser de lectura o escritura
155     if ( read( Conexion, &(instruccion->Tipo), 1 ) < 1 )
156         return INSTRUCCION_NOK;
157
158     //Se recibe la cantidad de datos a leer o escribir
159     if ( read( Conexion, &cantidad, 1 ) < 1 )
160         return INSTRUCCION_NOK;
161
162     instruccion->Cantidad = cantidad;
163
164     //Si la instruccion es de escritura, recibir los datos a enviar al
165     //Microcontrolador
166     if ( instruccion->Tipo == INSTRUCCION_ESCRITURA )
167     {
168         char datos[instruccion->Cantidad];
169         if ( read( Conexion, datos, instruccion->Cantidad ) < instruccion->Cantidad )
170             return INSTRUCCION_NOK;
171
172         instruccion->setDatos( datos, instruccion->Cantidad );
173     }
174
175     return INSTRUCCION_OK;
176}
177
178
179
180//*******************************************************************
181//  Funcion que envia el resultado de la Instruccion y mensajes de error
182//  ->estado: caracter de estado o codigo de error que se envia al programa
183//*******************************************************************
184
185void TAntena::EnviarEstado( char estado )
186{
187     write( Conexion, &estado, 1);
188}
189
190
191
192//*******************************************************************
193//  Funcion que envia la respuesta del Microcontrolador al programa que lo solicito
194//  ->respuesta: estructura de tipo TRespuesta que contiene la respuesta del
195//    Microcontrolador si se envio una instruccion de lectura
196//*******************************************************************
197
198void TAntena::EnviarRespuesta( TRespuesta respuesta )
199{
200     write( Conexion, respuesta.Datos, respuesta.Cantidad );
201}
202
203//Fin del archivo "TAntena.cc"
```

APÉNDICE D.5

Archivo: TNodo.h

```
1 //**********************************************************
2 //
3 //  Nombre del archivo:    TNodo.h
4 //  Proyecto:  Unificador
5 //  Autor:  Fredy Munoz
6 //  Lugar y fecha: Guatemala, febrero 2003.
7 //  Descripcion: Contiene el objeto TNodo utilizado para
8 //       almacenar datos en el objeto TLista
9 //
10 //**********************************************************
11
12
13
14 #ifndef _TNodo_h_
15 #define _TNodo_h_
16
17
18
19 template < class T >
20 class TNodo
21 {
22     public:
23         T Dato;             //Dato almacenado
24         TNodo *Siguiente;  //Puntero al siguiente nodo
25
26         TNodo();        //Constructor del objeto TNodo
27 };
28
29
30
31
32 //************************************************************
33 //  Constructor del objeto TNodo
34 //************************************************************
35
36 template < class T >
37 TNodo<T>::TNodo()
38 {
39     Siguiente = 0;
40 }
41
42
43
44 #endif
45
46 //Fin del archivo "TNodo.h"
```

APÉNDICE D.6

Archivo: TLista.h

```
1//*******************************************************
2//
3//  Nombre del archivo:    TLista.h
4//  Proyecto:  Unificador
5//  Autor:  Fredy Munoz
6//  Lugar y fecha: Guatemala, febrero 2003.
7//  Descripcion: Contiene el objeto TLista que implementa una
8//      lista encadenada que almacena cualquier tipo de dato
9//
10//*******************************************************
11
12
13
14#ifndef _TLista_h_
15#define _TLista_h_
16
17#include "TNodo.h"
18
19
20
21template < class T >
22class TLista
23{
24    private:
25        TNodo<T> *Primero;      //Primer nodo de la lista
26        int Cantidad;           //Cantidad de nodos en la lista
27
28    public:
29        TLista();                   //Constructor del objeto TLista
30        ~TLista();                  //Destructor del objeto TLista
31
32        void Vaciar();              //Funcion que permite vaciar la lista
33
34        void Agregar( const T );              //Funcion para agregar un dato a la lista
35        T* TLista<T>::Obtener( int posicion );   //Funcion para obtener un dato de la
36                                                  //lista dada su posicion
37        int ObtenerCantidad();                //Funcion para obtener la cantidad de
38                                              //datos que contiene la lista
39};
40
41
42
43
44//*********************************************************
45//  Constructor del objeto TLista, inicializa la cantidad de la
46//  lista a cero y el puntero al primer nodo de la lista
47//*********************************************************
48
49template < class T >
50TLista<T>::TLista()
51{
52    Primero = 0;   //Se inicializa el puntero al primer nodo
53    Cantidad = 0;  //Se inicializa la cantidad de datos en la lista a cero
54}
55
56
57
58//*********************************************************
59//  Destructor del objeto TLista que vacia el contenido de la misma
60//*********************************************************
61
62template < class T >
63TLista<T>::~TLista()
64{
65    Vaciar();  //Vacia el contenido de la lista
66}
67
68
69
70//*********************************************************
71//  Funcion que permite vaciar la lista
```

```
72 //*************************************************************
73
74 template < class T >
75 void TLista<T>::Vaciar()
76 {
77     TNodo<T> *siguiente;
78
79     //Mientras no se llegue al final de la lista
80     while(Primero != 0)
81     {
82         //Tomar el nodo siguiente y borrarlo
83         siguiente = Primero->Siguiente;
84         delete Primero;
85         Primero = siguiente;
86     }
87
88     //Reinicializar la Cantidad
89     Cantidad = 0;
90 }
91
92
93
94 //*************************************************************
95 //  Funcion que permite agregar un dato a la lista
96 //  ->dato: dato de tipo T que se almacenara en la lista
97 //*************************************************************
98
99 template < class T >
100 void TLista<T>::Agregar( const T dato )
101 {
102     TNodo<T> *actual, *nodo;
103
104     nodo = new TNodo<T>;   //Crear un nuevo nodo
105     nodo->Dato = dato;     //Almacenar el dato dentro del nuevo nodo
106
107
108     actual = Primero;
109
110     //Si la lista esta vacia, colocar el nodo como el primero de la lista
111     if(Primero == 0)
112         Primero = nodo;
113     else
114     {
115         //Si no, recorrer la lista hasta el final y colocar alli el nodo
116         while(actual->Siguiente != 0)
117             actual = actual->Siguiente;
118
119         actual->Siguiente = nodo;
120     }
121
122     //Incrementar en uno la cantidad
123     Cantidad++;
124 }
125
126
127
128 //*************************************************************
129 //  Funcion que permite obtener un dato de la lista
130 //  ->posicion: indica la posicion dentro de la lista del dato que
131 //     se desea obtener, el primer dato esta en la posicion 0
132 //  <-Devuelve un puntero al dato dentro de la lista, si la posicion
133 //     esta fuera del rango o si la lista esta vacia se devuelve 0
134 //*************************************************************
135
136 template < class T >
137 T* TLista<T>::Obtener( int posicion )
138 {
139     int idx;
140     TNodo<T> *actual;
141
142
```

```
143     if ( posicion >= Cantidad )         //Si la posicion solicitada es mayor o igual que
144                                         //la cantidad...
145         return 0;                       //...devolver 0
146     else
147     {
148         actual = Primero;
149
150         //Recorrer la lista hasta llegar a la posicion solicitada
151         for ( idx = 0; idx < posicion; idx++ )
152             actual = actual->Siguiente;
153
154         return &(actual->Dato);         //Devolver un puntero al dato dentro de la lista
155     }
156 }
157
158
159
160 //**************************************************************
161 //  Funcion que permite obtener la cantidad de datos almacenados
162 //  en la lista
163 //  <-Devuelve la cantidad de datos almacenados en la lista, si esta
164 //    vacia se devuelve 0
165 //**************************************************************
166
167 template < class T >
168 int TLista<T>::ObtenerCantidad()
169 {
170     return Cantidad;
171 }
172
173
174
175 #endif
176
177 //Fin del archivo "TLista.h"
```

APÉNDICE D.7

Archivo: TDirectorio.h

```
1//********************************************************
2//
3//  Nombre del archivo:   TDirectorio.h
4//  Proyecto:  Unificador
5//  Autor:  Fredy Munoz
6//  Lugar y fecha: Guatemala, febrero 2003.
7//  Descripcion: Contiene la definicion del objeto TDirectorio
8//      que se encarga de traducir los nombres de los
9//      Microcontroladores a direcciones numericas
10//
11//********************************************************
12
13
14
15#ifndef _TDirectorio_h_
16#define _TDirectorio_h_
17
18#include "TLista.h"
19
20
21#define DIRECTORIO_OK              1
22#define DIRECTORIO_NO_ARCHIVO      2
23#define DIRECTORIO_ERROR_ARCHIVO   3
24
25
26
27//********************************************************
28//  El objeto TNombre almacena la correlacion entre el nombre
29//  de un Microcontrolador y su direccion numerica
30//********************************************************
31
32class TNombre
33{
34    public:
35        char* nombre;       //Cadena de caracteres que constituyen el nombre del
36                            //Microcontrolador
37        int direccion;      //Direccion numerica del Microcontrolador
38};
39
40
41
42class TDirectorio
43{
44    private:
45        TLista<TNombre> TablaNombres; //Almacena las correspondencias de nombres con
46                                      //las direcciones numericas
47
48    public:
49        TDirectorio(int*);      //Constructor del objeto TDirectorio que devuelve el
50                                //resultado de la inicializacion
51        ~TDirectorio();             //Destructor del objeto TDirectorio que libera la
52                                    //memoria utilizada por la Tabla
53
54        int Traducir(char*);    //Funcion que permite obtener la direccion numerica de
55                                //un Microcontrolador a partir de su nombre
56};
57
58
59#endif
60
61//Fin del archivo "TDirectorio.h"
```

APÉNDICE D.8

Archivo: TDirectorio.cc

```
1//*******************************************************
2//
3//  Nombre del archivo:    TDirectorio.cc
4//  Proyecto:  Unificador
5//  Autor:  Fredy Munoz
6//  Lugar y fecha: Guatemala, febrero 2003.
7//  Descripcion: Contiene la implementacion del objeto TDirectorio
8//      que se encarga de traducir los nombres de los
9//      Microcontroladores a direcciones numericas
10//
11//*******************************************************
12
13
14
15#include "TDirectorio.h"
16
17#include <fstream.h>
18#include <iostream.h>
19
20
21
22//*******************************************************
23//  Constructor del objeto TDirectorio, lee el archivo 'directorio'
24//  que contiene las correlaciones entre nombres y direcciones
25//  ->estado: puntero a un entero donde se almacena el
26//    resultado de la inicializacion del Directorio
27//*******************************************************
28
29TDirectorio::TDirectorio( int* estado )
30{
31     TNombre entrada;
32     int basura;
33     ifstream archivo("directorio");
34
35     //Si no se pudo abrir el archivo 'directorio' indicarlo
36     if ( !archivo )
37     {
38         *estado = DIRECTORIO_NO_ARCHIVO;
39         return;
40     }
41
42
43     //Se extrae del archivo la direccion numerica
44     archivo >> entrada.direccion;
45
46     //Mientras no se llegue al final del archivo...
47     while ( !archivo.eof() )
48     {
49         //Se verifica que la direccion sea un entero positivo
50         if ( entrada.direccion < 0 )
51         {
52             *estado = DIRECTORIO_ERROR_ARCHIVO;
53             return;
54         }
55
56
57         //Se reserva memoria para almacenar el nombre del Microcontrolador
58         entrada.nombre = new char[200];
59
60         //Se extrae del archivo el nombre del Microcontrolador
61         archivo >> entrada.nombre;
62
63         //Se verifica que el nombre no haya sido asignado a otra direccion
64         if ( Traducir( entrada.nombre ) != -1 )
65         {
66             *estado = DIRECTORIO_ERROR_ARCHIVO;
67             return;
68         }
69
70         //Se agrega la entrada a la Tabla de Nombres
71         TablaNombres.Agregar( entrada );
```

```
72
73          //Se extrae del archivo la direccion para el siguiente ciclo
74          archivo >> entrada.direccion;
75      }
76
77      *estado = DIRECTORIO_OK;
78 }
79
80
81
82 //*******************************************************
83 //  Destructor del objeto TDirectorio que libera la memoria
84 //  utilizada por la Tabla de nombres
85 //*******************************************************
86
87 TDirectorio::~TDirectorio()
88 {
89      int idx;
90      TNombre entrada;
91
92      //Por cada entrada en la tabla...
93      for ( idx = 0; idx < TablaNombres.ObtenerCantidad(); idx++ )
94      {
95          entrada = *(TablaNombres.Obtener( idx ));
96
97          //Liberar la memoria utilizada para el nombre
98          delete [] entrada.nombre;
99      }
100 }
101
102
103
104 //*******************************************************
105 //  Funcion que traduce un nombre de Microcontrolador a su
106 //  direccion numerica
107 //  ->nombre: puntero a la cadena de caracteres con el nombre
108 //    del Microcontrolador a traducir
109 //  <-Devuelve la direccion numerica asociada a este nombre o
110 //    -1 si no se encontro la direccion en la tabla
111 //*******************************************************
112
113 int TDirectorio::Traducir( char* nombre )
114 {
115     int idx;
116     TNombre entrada;
117
118     //Buscar en todas las entradas de la tabla...
119     for ( idx = 0; idx < TablaNombres.ObtenerCantidad(); idx++ )
120     {
121         entrada = *(TablaNombres.Obtener( idx ));
122
123         //...la que tenga el nombre buscado
124         if ( strcmp( entrada.nombre, nombre ) == 0 )
125             return entrada.direccion;
126     }
127
128     //Si no se encontro devolver -1
129     return -1;
130 }
131
132
133 //Fin del archivo "TDirectorio.cc"
```

APÉNDICE D.9

Archivo: puerto_serial.h

```
 1//******************************************************
 2//
 3//  Nombre del archivo:    puerto_serial.h
 4//  Proyecto:  Unificador
 5//  Autor:  Fredy Munoz
 6//  Lugar y fecha: Guatemala, febrero 2003.
 7//  Descripcion: Contiene la interface para el objeto
 8//      PuertoSerial que permite la comunicacion entre
 9//      el objeto TIntermediario y el Intermediario
10//
11//******************************************************
12
13
14
15#ifndef _puerto_serial_h_
16#define _puerto_serial_h_
17
18#include <termios.h>
19
20
21#define PUERTO_SERIAL_OK        0       //El puerto serial fue inicializado
22                                        //correctamente
23#define PUERTO_SERIAL_NO_CONECTADO    1   //Surgio un problema al inicializar el
24                                          //puerto serial
25
26
27class PuertoSerial
28{
29
30    private:
31
32        termios configuracion_actual;      //Configuracion que se le dara al puerto
33                                           //serial al inicializarlo
34        termios configuracion_anterior;      //Configuracion del puerto serial antes
35                                             //de inicializar el puerto
36
37        int estado_puerto;              //Indica el resultado de la inicializacion del
38                                        //puerto
39
40
41    public:
42
43        PuertoSerial(int* estado);      //Construcctor del objeto PuertoSerial, recibe
44                                        //un puntero a un entero donde colocara el
45                                        //estado del puerto
46        ~PuertoSerial();                //Destructor del objeto PuertoSerial
47
48        int enviar(char caracter);      //Envia un caracter por el puerto serial
49        int recibir(char* caracter);    //Devuelve un caracter de los recibidos por el
50                                        //puerto serial
51        void flush(void);               //Vacia el contenido del buffer del puerto
52                                        //serial
53};
54
55
56#endif
57
58
59//Fin del archivo "puerto_serial.h"
```

APÉNDICE D.10

Archivo: puerto_serial.cc

```
1 //*****************************************************
2 //
3 //  Nombre del archivo:    puerto_serial.cc
4 //  Proyecto:  Unificador
5 //  Autor:  Fredy Munoz
6 //  Lugar y fecha: Guatemala, febrero 2003.
7 //  Descripcion: Contiene la implementacion del objeto
8 //      PuertoSerial que permite la comunicacion entre
9 //      el objeto TIntermediario y el Intermediario
10 //
11 //*****************************************************
12
13
14
15 #include "puerto_serial.h"
16
17 #include <termios.h>
18 #include <unistd.h>
19 #include <fcntl.h>
20 #include <sys/signal.h>
21 #include <sys/types.h>
22 #include <pthread.h>
23
24
25 #define BAUDRATE   B115200     //Velocidad a la que operara el puerto serial
26 #define BUFF_SIZE  256         //Cantidad maxima de caracteres que puede almacenar el
27                                //buffer del puerto serial
28
29
30
31 char buff[BUFF_SIZE];  //Buffer ciclico donde se almacenaran los caracteres que se
32                        //reciban por el puerto serial
33 int cabeza;            //Puntero a la cabeza del buffer
34 int cola;              //Puntero a la cola del buffer
35 int tamano;            //Cantidad de caracteres almacenados en el buffer
36
37
38 int serial;     //file descriptor para el puerto serial
39
40 pthread_mutex_t mutex; //semaforo utilizado para proteger el acceso al buffer
41
42
43
44 //*****************************************************************
45 //  Rutina invocada al recibir la senal del sistema que se ha recibido
46 //  un caracter por el puerto serial
47 //*****************************************************************
48
49 void handler( int status )
50 {
51     int recibidos;
52
53
54     //Se pide un caracter al puerto serial y se almacena en el buffer
55     recibidos = read( serial, &buff[cabeza], 1 );
56
57     //Mientras existan mas caracteres disponibles pedirlos al puerto serial y
58     //almacenarlos en el buffer
59     while ( recibidos > 0 )
60     {
61         pthread_mutex_lock(&mutex);
62         tamano++;
63         cabeza = (cabeza + 1) % BUFF_SIZE;
64         pthread_mutex_unlock(&mutex);
65
66         recibidos = read( serial, &buff[cabeza], 1 );
67     };
68 }
69
70
71
```

```
72//*************************************************************************
73//  Constructor del objeto PuertoSerial, este inicializa el puerto serial
74//  de la computadora, establece la rutina que respondera a la senal de
75//  ingreso de un caracter por el puerto serial e inicializa el semaforo
76//  ->estado: puntero a un entero que indicara el resultado de la inicializacion
77//*************************************************************************
78
79PuertoSerial::PuertoSerial( int* estado )
80{
81
82     //Se abre el puerto serial
83     serial = open( "/dev/ttyS0", O_RDWR | O_NOCTTY );
84
85     if ( serial < 0 )
86         *estado = estado_puerto = PUERTO_SERIAL_NO_CONECTADO;
87     else
88         *estado = estado_puerto = PUERTO_SERIAL_OK;
89
90     fcntl( serial, F_SETOWN, getpid() );
91     fcntl( serial, F_SETFL, FASYNC );
92
93
94
95     //Se almacena la configuracion actual
96     tcgetattr( serial, &configuracion_anterior );
97
98     configuracion_actual.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
99     configuracion_actual.c_iflag = IGNPAR;
100    configuracion_actual.c_oflag = 0;
101    configuracion_actual.c_lflag = 0;
102    configuracion_actual.c_cc[VMIN] = 0;
103    configuracion_actual.c_cc[VTIME] = 0;
104
105    //Se establece la nueva configuracion del puerto serial
106    tcsetattr( serial, TCSAFLUSH, &configuracion_actual );
107
108
109    //Se iniciliza el buffer
110    cabeza = cola = tamano = 0;
111
112
113    //Se establece la rutina que respondera a la senal de ingreso de un caracter por
114    //el puerto serial
115    signal( SIGIO, handler );
116
117
118    //Se inicializa el semaforo
119    pthread_mutex_init( &mutex, NULL );
120}
121
122
123
124//*************************************************************************
125//  Destructor del objeto PuertoSerial que devuelve al puerto serial de
126//  la computadora su configuracion anterior
127//*************************************************************************
128
129PuertoSerial::~PuertoSerial()
130{
131    if ( estado_puerto == PUERTO_SERIAL_OK )
132    {
133        tcsetattr( serial, TCSANOW, &configuracion_anterior );
134        close( serial );
135    }
136}
137
138
139
140//*************************************************************************
141//  Funcion que envia un caracter por el puerto serial
142//  ->caracter: el caracter a enviar
```

```
143//  <-Devuelve -1 si el puerto no ha sido inicializado, 0 si no se pudo
144//    enviar el caracter y 1 si se pudo enviar el caracter
145//*********************************************************************
146
147int PuertoSerial::enviar( char caracter )
148{
149    //Si el puerto no ha sido inicializado devolver -1
150    if ( estado_puerto != PUERTO_SERIAL_OK )
151        return -1;
152
153    //Tratar de enviar el caracter y devolver el resultado
154    return write( serial, &caracter, 1 );
155}
156
157
158
159//*********************************************************************
160//  Funcion que devuelve un caracter de los almacenados en el buffer
161//  ->caracter: puntero donde se almacenara el caracter
162//  <-Devuelve -1 si el puerto no ha sido inicializado, 0 si no hay ningun
163//    caracter disponible en el buffer y 1 si se pudo extraer un caracter
164//    del buffer
165//*********************************************************************
166
167int PuertoSerial::recibir( char* caracter )
168{
169    //Si el puerto no ha sido inicializado devolver -1
170    if ( estado_puerto != PUERTO_SERIAL_OK )
171        return -1;
172
173    //Bloquear el acceso al buffer
174    pthread_mutex_lock( &mutex );
175
176    //Si no hay caracteres en el buffer, devolver 0
177    if ( tamano == 0 )
178    {
179        pthread_mutex_unlock( &mutex );
180        return 0;
181    }
182
183    //Si hay caracteres disponibles, extraer uno del buffer
184    *caracter = buff[cola];
185    cola = (cola + 1) % BUFF_SIZE;
186    tamano--;
187
188    //Desbloquear el acceso al buffer
189    pthread_mutex_unlock( &mutex );
190
191    return 1;
192}
193
194
195
196//*********************************************************************
197//  Funcion que reinicializa el buffer de ingreso de caracteres
198//*********************************************************************
199
200void PuertoSerial::flush(void)
201{
202    //Bloquear el acceso al buffer
203    pthread_mutex_lock( &mutex );
204
205    //Reinicializar el buffer
206    cabeza = cola = tamano = 0;
207
208    //Desbloquear el acceso al buffer
209    pthread_mutex_unlock( &mutex );
210}
211
212
213//Fin del archivo "puerto_serial.cc"
```

APÉNDICE D.11

Archivo: TEnlace.h

```
 1//*******************************************************
 2//
 3//  Nombre del archivo:    TEnlace.h
 4//  Proyecto:  Unificador
 5//  Autor:  Fredy Munoz
 6//  Lugar y fecha: Guatemala, febrero 2003.
 7//  Descripcion: Contiene la interface para el objeto
 8//      TEnlace que permite al Unificador comunicarse
 9//      con el Intermediario
10//
11//*******************************************************
12
13
14
15#ifndef _TEnlace_h_
16#define _TEnlace_h_
17
18#include "../Tipos/TInstruccion.h"
19#include "../Tipos/TRespuesta.h"
20
21#include "puerto_serial/puerto_serial.h"
22
23
24
25#define ENLACE_OK  0   //El Enlace se inicializo correctamente
26#define ENLACE_NOK 1   //Surgio un problema al tratar de iniciarlizar el Enlace
27
28
29
30class TEnlace
31{
32    private:
33        PuertoSerial* puerto; //Puntero al puerto serial que se utilizara en...
34                              //...la comunicacion hacia el Intermediario
35
36    public:
37        TEnlace(int*); //Construcctor del objeto TEnlace que devuelve el estado del
38                       //mismo
39        ~TEnlace();    //Destructor del objeto TEnlace
40
41        TRespuesta EnviarInstruccion(int, TInstruccion); //Funcion que envia al
42                                                         //Intermediario una
43                                                         //instrucción dirigida a un
44                                                         //Microcontrolador especifico
45};
46
47
48#endif
49
50
51//Fin del archivo "TEnlace.h"
```

APÉNDICE D.12

Archivo: TEnlace.cc

```
1 //*******************************************************
2 //
3 //  Nombre del archivo:    TEnlace.cc
4 //  Proyecto:  Unificador
5 //  Autor:  Fredy Munoz
6 //  Lugar y fecha: Guatemala, febrero 2003.
7 //  Descripcion: Contiene la implementacion para el objeto
8 //      TEnlace que permite al Unificador comunicarse
9 //      con el Intermediario
10 //
11 //*******************************************************
12
13
14
15 #include "TEnlace.h"
16
17 #include <string.h>
18 #include <unistd.h>
19 #include <sys/signal.h>
20
21
22 #define STX 2
23 #define ETX 3
24
25
26
27 int time_out;  //Su valor es cero si no se ha vencido el time-out
28                //para esperar datos por el puerto serial
29                //Su valor es uno, si el time-out ha vencido
30
31 int time_out_establecido; //Indica si ya se establecio el time-out
32
33
34
35 //*******************************************************************
36 //  Funcion invocada al concluir el time-out para la recepcion de datos
37 //  por el puerto serial
38 //*******************************************************************
39
40 void alarm_serial( int status )
41 {
42     time_out = 1;
43 }
44
45
46
47 //*******************************************************************
48 //Constructor del objeto TEnlace, este inicializa el puerto serial
49 //que servira para la comunicacion con el Intermediario
50 //->estado: puntero a un entero que indicara el resultado de la inicializacion
51 //*******************************************************************
52
53 TEnlace::TEnlace( int* estado )
54 {
55     int estado_puerto;
56
57     puerto = new PuertoSerial( &estado_puerto );
58
59     if ( estado_puerto == PUERTO_SERIAL_OK )
60         *estado = ENLACE_OK;
61     else
62         *estado = ENLACE_NOK;
63 }
64
65
66
67 //*******************************************************************
68 //Destructor del objeto TEnlace, este cierra apropiadamente
69 //el puerto serial
70 //*******************************************************************
71
```

```
72 TEnlace::~TEnlace()
73 {
74     delete puerto;
75 }
76
77
78
79 //*********************************************************************
80 //Funcion que envia la instruccion al Intermediario para que este
81 //a su vez la envie al Microcontrolador especificado
82 //->direccion: la direccion del Microcontrolador al que se enviara la instruccion
83 //->instruccion: la instruccion que sera enviada
84 //<-Devuelve el resultado de la instruccion o el codigo de error si fallo el envio
85 //*********************************************************************
86
87 TRespuesta TEnlace::EnviarInstruccion( int direccion, TInstruccion instruccion )
88 {
89     TRespuesta respuesta;
90
91     char trama[MAX_BYTES_ESCRITURA+5];
92     int tamano_trama;
93     int idx;
94
95
96     //Si es una instruccion de lectura se verifica que la cantidad de bytes a pedir
97     //no exceda la capacidad del Microcontrolador
98     if ( instruccion.Tipo == INSTRUCCION_LECTURA
99         && instruccion.Cantidad > MAX_BYTES_LECTURA )
100     {
101         respuesta.Resultado = RESPUESTA_LEN;
102         return respuesta;
103     }
104
105
106     //Si es una instruccion de escritura se verifica que la cantidad de bytes a enviar
107     //no exceda la capacidad del Microcontrolador
108     if ( instruccion.Tipo == INSTRUCCION_ESCRITURA
109         && instruccion.Cantidad > MAX_BYTES_ESCRITURA )
110     {
111         respuesta.Resultado = RESPUESTA_LEN;
112         return respuesta;
113     }
114
115
116     //Se verifica que la direccion sea valida
117     if ( direccion < 1 || direccion > 126 )
118     {
119         respuesta.Resultado = RESPUESTA_DIR;
120         return respuesta;
121     }
122
123     trama[0] = STX;
124     trama[1] = direccion;
125     trama[2] = instruccion.Tipo;
126     trama[3] = instruccion.Cantidad;
127
128
129     //Si es una instruccion de escritura agregar a la trama los datos
130     if ( instruccion.Tipo == INSTRUCCION_ESCRITURA )
131     {
132         char datos[instruccion.Cantidad];
133
134         instruccion.getDatos( datos, instruccion.Cantidad );
135         memcpy( &trama[4], datos, instruccion.Cantidad );
136         trama[instruccion.Cantidad + 4] = ETX;
137         tamano_trama = instruccion.Cantidad + 5;
138     }
139     else
140     {
141         tamano_trama = 5;
142         trama[4] = ETX;
```

```
143     }
144
145
146     //Descartar cualquier residuo que exista en el puerto serial
147     puerto->flush();
148
149
150     //Enviar la trama al puerto serial
151     for ( idx = 0; idx < tamano_trama; idx++ )
152         puerto->enviar(trama[idx]);
153
154
155     //Si es una instruccion de lectura esperar los datos del puerto serial
156     if ( instruccion.Tipo == INSTRUCCION_LECTURA )
157     {
158         int idx;
159         int cantidad_recibido_serial;
160         char caracter_recibido_serial;
161
162         idx = 0;
163
164
165         //Establecer el timer para que de la senal de time-out
166         signal( SIGALRM, alarm_serial );
167         time_out = 0;
168         time_out_establecido = 0;
169
170
171         //Leer datos hasta que se leean todos los que se pidieron o hasta que se haya
172         //vencido el time-out
173         while ( idx < instruccion.Cantidad && !time_out )
174         {
175             //Leer un caracter del puerto serial
176             cantidad_recibido_serial = puerto->recibir( &caracter_recibido_serial );
177
178             //Si se logro leer, agregar al arreglo datos
179             if ( cantidad_recibido_serial != 0 )
180             {
181                 alarm( 0 ); //Anular el time-out
182                 time_out_establecido = 0;
183                 respuesta.Datos[idx] = caracter_recibido_serial;
184                 idx++;
185             }
186             //Si no, establecer el time-out a un segundo si no se ha establecido antes
187             else
188             {
189                 if ( !time_out_establecido )
190                 {
191                     time_out_establecido = 1;
192                     alarm( 1 );
193                 }
194             }
195         }
196
197         if ( time_out )
198         {
199             respuesta.Resultado = RESPUESTA_INT;
200             return respuesta;
201         }
202
203         respuesta.Cantidad = instruccion.Cantidad;
204     }
205
206     respuesta.Resultado = RESPUESTA_OK;
207     return respuesta;
208 }
209
210
211 //Fin del archivo "TEnlace.cc"
```

APÉNDICE D.13

Archivo: TRespuesta.h

```
 1//*****************************************************
 2//
 3//  Nombre del archivo:    TRespuesta.h
 4//  Proyecto:  Unificador
 5//  Autor:  Fredy Munoz
 6//  Lugar y fecha: Guatemala, febrero 2003.
 7//  Descripcion: Contiene la definicion de la estructura
 8//      TRespuesta que representa a la respuesta que puede
 9//      emitir un Microcontrolador
10//
11//*****************************************************
12
13
14
15#ifndef _TRespuesta_h_
16#define _TRespuesta_h_
17
18
19#define MAX_BYTES_LECTURA 75
20
21
22#define RESPUESTA_OK   0   //La instruccion se ejecuto con exito
23#define RESPUESTA_LEN  1   //Error: La cantidad de datos solicitados o enviados...
24                         //        ...sobrepasa la capacidad del Microcontrolador
25#define RESPUESTA_DIR  2   //Error: Direccion no valida o no existe
26#define RESPUESTA_INT  3   //Error: No se pudo establecer la comunicacion con el
27                         //Intermediario
28
29
30
31struct TRespuesta
32{
33    char Resultado;     //Indica el resultado de la operacion de lectura o escritura
34
35    int Cantidad;       //Indica la cantidad de datos devueltos de una operacion de
36                        //lectura si la operacion tuvo exito, de lo contrario su valor
37                        //es indefinido
38                        //Si es una operacion de escritura este numero es cero (0)
39
40    char Datos[MAX_BYTES_LECTURA];     //Arreglo de bytes que contienen la información
41                                       //devuelta de una operacion de lectura si tuvo
42                                       //exito, de lo contrario su contenido es
43                                       //indefinido
44};
45
46
47#endif
48
49
50//Fin del archivo "TRespuesta.h"
```

APÉNDICE D.14

Archivo: TInstruccion.h

```
 1//*********************************************************
 2//
 3//  Nombre del archivo:    TInstruccion.h
 4//  Proyecto:  Unificador
 5//  Autor:  Fredy Munoz
 6//  Lugar y fecha: Guatemala, febrero 2003.
 7//  Descripcion: Contiene la interface para el objeto
 8//      TInstruccion que representa a la instruccion
 9//      que puede ser enviada a un Microcontrolador
10//
11//*********************************************************
12
13
14
15#ifndef _TInstruccion_h_
16#define _TInstruccion_h_
17
18
19#define MAX_BYTES_ESCRITURA   80      //Cantidad maxima de bytes que pueden ser
20                                      //enviados a un Microcontrolador
21
22
23#define INSTRUCCION_LECTURA       0
24#define INSTRUCCION_ESCRITURA     1
25
26
27
28class TInstruccion
29{
30    private:
31        char Datos[MAX_BYTES_ESCRITURA];  //Puntero al arreglo de bytes que contienen
32                                          //la informacion a escribir si es una
33                                          //instruccion de escritura
34
35    public:
36        char Tipo;     //El tipo de instruccion, que puede ser lectura o escritura
37
38        int Cantidad;  //Cantidad de datos que se desean leer o escribir
39
40        int setDatos( char*, int );   //Funcion para colocar bytes en el arreglo datos
41        int getDatos( char*, int );   //Funcion para obtener bytes del arreglo datos
42};
43
44
45#endif
46
47
48//Fin del archivo "TInstruccion.h"
```

APÉNDICE D.15

Archivo: TInstruccion.cc

```
 1//*******************************************************
 2//
 3//  Nombre del archivo:    TInstruccion.cc
 4//  Proyecto:  Unificador
 5//  Autor:  Fredy Munoz
 6//  Lugar y fecha: Guatemala, febrero 2003.
 7//  Descripcion: Contiene la implementacion del objeto
 8//      TInstruccion que representa a la instruccion
 9//      que puede ser enviada a un Microcontrolador
10//
11//*******************************************************
12
13
14
15#include "TInstruccion.h"
16
17#include <string.h>
18
19
20
21//**************************************************************************
22//Funcion que permite colocar bytes en el arreglo 'Datos' verificando que no se
23//exeda la capacidad maxima del arreglo
24//->fuente: puntero al arreglo de datos de donde se extraeran los bytes
25//->cantidad: numero de bytes a ser copiados de 'fuente' a 'Datos'
26//<-Devuelve la cantidad de bytes que fueron copiados
27//**********************************************************************
28
29int TInstruccion::setDatos( char* fuente, int cantidad )
30{
31    if ( cantidad > MAX_BYTES_ESCRITURA )
32    {
33        memcpy( Datos, fuente, MAX_BYTES_ESCRITURA );
34        return MAX_BYTES_ESCRITURA;
35    }
36    else
37    {
38        memcpy( Datos, fuente, cantidad );
39        return cantidad;
40    }
41}
42
43
44
45//**************************************************************************
46//Funcion que permite leer bytes del arreglo Datos verificando que no se
47//exeda la capacidad maxima del arreglo
48//->destino: puntero al arreglo donde se colocaran los bytes
49//->cantidad: numero de bytes a ser copiados de 'Datos' a 'destino'
50//<-Devuelve la cantidad de bytes que fueron copiados
51//**********************************************************************
52
53int TInstruccion::getDatos( char* destino, int cantidad )
54{
55    if ( cantidad > MAX_BYTES_ESCRITURA )
56    {
57        memcpy( destino, Datos, MAX_BYTES_ESCRITURA );
58        return MAX_BYTES_ESCRITURA;
59    }
60    else
61    {
62        memcpy( destino, Datos, cantidad );
63        return cantidad;
64    }
65}
66
67
68//Fin del archivo "TInstruccion.cc"
```

APÉNDICE E

Código fuente de los Programas Cliente de ejemplo

APÉNDICE E.1

Archivo: ventilador.c

```
1 //*****************************************************
2 //
3 //  Nombre del archivo:    ventilador.c
4 //  Proyecto:  Unificador
5 //  Autor:  Fredy Munoz
6 //  Lugar y fecha: Guatemala, febrero 2003.
7 //  Descripcion: Contiene el codigo para el CGI que maneja el
8 //      modulo del ventilador.
9 //
10 //*****************************************************
11
12
13
14 #include "../cgic/cgic.h"
15
16 #include "../red.h"
17 #include "../constantes.h"
18
19 #include <string.h>
20
21
22
23 #define NOMBRE_MICROCONTROLADOR        "ventilador"
24 #define ARCHIVO_CONF               "../unificador.conf"
25
26
27
28
29
30 //*****************************************************
31 //  Funcion principal ejecutada al invocar el CGI
32 //*****************************************************
33
34 int cgiMain()
35 {
36     sockaddr_in direccion_unificador; //Estructura de datos que almacena la direccion
37                                        //del Unificador
38
39     int conexion;       //Socket utilizado en la comunicacion hacia el Unificador
40
41     unsigned char buff[10];        //Buffer utilizado en la comunicacion con el
42                                    //Unificador
43
44
45     FILE* archivo_configuracion;  //Archivo de configuracion del CGI
46     char IP_Unificador[20];           //Direccion IP del Unificador
47
48     int velocidad;      //Indica la velocidad a la que esta trabajando el ventilador
49
50
51
52
53     //Enviar encabezado del documento HTML
54
55     cgiHeaderContentType( "text/html" );
56
57     fprintf( cgiOut, "<HTML><HEAD>\n" );
58     fprintf( cgiOut, "<TITLE>Ventilador</TITLE></HEAD>\n" );
59     fprintf( cgiOut, "<BODY><H1>Ventilador</H1>\n" );
60
61
62
63     //Abrir archivo de configuracion
64     archivo_configuracion = fopen( ARCHIVO_CONF, "r");
65     if ( archivo_configuracion  == NULL )
66     {
67         fprintf( cgiOut, "No se encontro el archivo de configuracion" );
68         return 0;
69     }
70
71     //Leer direccion IP del Unificador
```

```
72      fscanf( archivo_configuracion, "%s", IP_Unificador );
73
74
75      //Construir la direccion del Unificador
76      memset( &direccion_unificador, 0, sizeof( direccion_unificador ) );
77      direccion_unificador.sin_family = AF_INET;
78      direccion_unificador.sin_addr.s_addr = inet_addr ( IP_Unificador );
79      direccion_unificador.sin_port = htons( UNIFICADOR_PUERTO );
80
81
82
83      //Si el usuario pidio cambiar velocidad
84      if ( cgiFormSubmitClicked( "cambiar" ) == cgiFormSuccess )
85      {
86          //Abrir conexion hacia el Unificador
87          conexion = socket( AF_INET, SOCK_STREAM, 0 );
88          if ( conexion < 0 )
89          {
90              fprintf( cgiOut, "Error al abrir socket" );
91              return 0;
92          }
93
94          if ( connect( conexion, (sockaddr*) &direccion_unificador,
95 sizeof(direccion_unificador) ) < 0 )
96          {
97              fprintf( cgiOut, "Error al conectarse al Unificador" );
98              return 0;
99          }
100
101
102
103
104         //Enviar el nombre del Microcontrolador
105         write( conexion, NOMBRE_MICROCONTROLADOR, strlen(NOMBRE_MICROCONTROLADOR) );
106         write( conexion, "\0", 1 ); //Terminador de cadena
107
108         //Enviar el tipo de instruccion
109         buff[0] = INSTRUCCION_ESCRITURA;
110         write( conexion, buff, 1 );
111
112         //Enviar la cantidad de datos a escribir
113         buff[0] = 1;
114         write( conexion, buff, 1 );
115
116         //Determinar que velocidad eligio el usuario en la pagina web
117         cgiFormInteger( "velocidad", &velocidad, 0 );
118         buff[0] = velocidad;
119
120         //Enviar los datos a escribir al Microcontrolador
121         write( conexion, buff, 1 );
122
123
124
125
126         //Verificar que la instruccion haya sido recibida por el Unificador
127         read( conexion, buff, 1 );
128
129         if ( buff[0] == UNIFICADOR_INSTRUCCION_ERROR )
130         {
131             fprintf( cgiOut, "La instruccion no fue recibida correctamente por el
132 Unificador" );
133             return 0;
134         }
135         if ( buff[0] == UNIFICADOR_INSTRUCCION_DATOS )
136         {
137             fprintf( cgiOut, "El Unificador recibio datos incompletos" );
138             return 0;
139         }
140         if ( buff[0] == UNIFICADOR_INSTRUCCION_DIR )
141         {
142             fprintf( cgiOut, "El nombre del Microcontrolador no es correcto" );
```

```
143             return 0;
144         }
145
146
147         //Verificar que la instruccion haya sido procesada
148         read( conexion, buff, 1 );
149
150         if ( buff[0] != RESPUESTA_OK )
151         {
152             fprintf( cgiOut, "La instruccion no fue procesada" );
153             return 0;
154         }
155
156
157
158
159         //Cerrar la conexion
160         close( conexion );
161
162         //Esperar que el ventilador cambie de velocidad
163         sleep(1);
164     }
165
166
167
168
169     //Actualizar informacion en la pagina
170
171     //Abrir conexion al Unificador
172     conexion = socket( AF_INET, SOCK_STREAM, 0 );
173     if ( conexion < 0 )
174     {
175         fprintf( cgiOut, "Error al abrir socket" );
176         return 0;
177     }
178
179     if ( connect( conexion, (sockaddr*) &direccion_unificador,
180sizeof(direccion_unificador) ) < 0 )
181     {
182         fprintf( cgiOut, "Error al conectarse al Unificador" );
183         return 0;
184     }
185
186
187
188
189     //Enviar el nombre del Microcontrolador
190     write( conexion, NOMBRE_MICROCONTROLADOR, strlen(NOMBRE_MICROCONTROLADOR) );
191     write( conexion, "\0", 1 ); //Terminador de cadena
192
193     //Enviar el tipo de instruccion
194     buff[0] = INSTRUCCION_LECTURA;
195     write( conexion, buff, 1);
196
197     //Enviar la cantidad de datos a leer
198     buff[0] = 2;
199     write( conexion, buff, 1);
200
201
202
203
204     //Verificar que la instruccion haya sido recibida por el Unificador
205     read( conexion, buff, 1 );
206
207     if ( buff[0] == UNIFICADOR_INSTRUCCION_ERROR )
208     {
209         fprintf( cgiOut, "La instruccion no fue recibida correctamente por el
210Unificador" );
211         return 0;
212     }
213     if ( buff[0] == UNIFICADOR_INSTRUCCION_DATOS )
```

```
214     {
215         fprintf( cgiOut, "El Unificador recibio datos incompletos" );
216         return 0;
217     }
218     if ( buff[0] == UNIFICADOR_INSTRUCCION_DIR )
219     {
220         fprintf( cgiOut, "El nombre del Microcontrolador no es correcto" );
221         return 0;
222     }
223
224     //Verificar que la instruccion haya sido procesada
225     read( conexion, buff, 1 );
226
227     if ( buff[0] != RESPUESTA_OK )
228     {
229         fprintf( cgiOut, "La instruccion no fue procesada" );
230         return 0;
231     }
232
233
234
235     //Leer los datos provenientes del Microcontrolador
236     read( conexion, buff, 2 );
237
238
239     //Desplegar la temperatura en la pagina
240     fprintf( cgiOut, "Temperatura: " );
241     fprintf( cgiOut, "%d<BR>", ( buff[0] - 128 ) );
242
243
244     //Desplegar la velocidad actual en la pagina
245     fprintf( cgiOut, "Velocidad actual: " );
246
247     switch( buff[1] )
248     {
249         case 0: fprintf( cgiOut, "Detenido" );
250             break;
251         case 1: fprintf( cgiOut, "Velocidad 1" );
252             break;
253         case 2: fprintf( cgiOut, "Velocidad 2" );
254             break;
255         case 3: fprintf( cgiOut, "Velocidad 3" );
256             break;
257     }
258
259
260     //Desplegar la forma para modificar la velocidad del ventilador
261     fprintf( cgiOut, "<BR><BR><FORM action = temperatura.cgi method = post>\n" );
262     fprintf( cgiOut, "    <INPUT type = submit name = actualizar value = 'Actualizar
263 lectura'><BR><BR>\n" );
264     fprintf( cgiOut, "    <INPUT type = radio name = velocidad value =
265 0>Apagado<BR>\n" );
266     fprintf( cgiOut, "    <INPUT type = radio name = velocidad value = 1>Velocidad
267 1<BR>\n" );
268     fprintf( cgiOut, "    <INPUT type = radio name = velocidad value = 2>Velocidad
269 2<BR>\n" );
270     fprintf( cgiOut, "    <INPUT type = radio name = velocidad value = 3>Velocidad
271 3<BR>\n" );
272     fprintf( cgiOut, "    <INPUT type = submit name = cambiar value = 'Cambiar
273 Velocidad' >\n" );
274     fprintf( cgiOut, "</FORM></BODY></HTML>" );
275
276     return 0;
277 }
278
279
280 //Fin del archivo "ventilador.cgi"
```

APÉNDICE E.2

Archivo: tektronix.c

```
1 //*********************************************************
2 //
3 //  Nombre del archivo:    tektronix.c
4 //  Proyecto:  Unificador
5 //  Autor:  Fredy Munoz
6 //  Lugar y fecha: Guatemala, febrero 2003.
7 //  Descripcion: Contiene el codigo para el CGI que maneja el
8 //      modulo del osciloscopio Tektronix.
9 //
10 //*********************************************************
11
12
13
14 #include "../cgic/cgic.h"
15
16 #include <values.h>
17 #include "../gdchart/gdc.h"
18 #include "../gdchart/gdchart.h"
19
20 #include "../red.h"
21 #include "../constantes.h"
22
23 #include <string.h>
24
25
26
27 #define NOMBRE_MICROCONTROLADOR       "tektronix"
28 #define ARCHIVO_CONF                "../unificador.conf"
29
30 #define CANTIDAD   75         //Cantidad de datos a pedir por paquete
31 #define PAQUETES   125        //Cantidad de paquetes a recibir
32
33
34
35
36 //*********************************************************
37 //  Funcion principal ejecutada al invocar el CGI
38 //*********************************************************
39
40 int cgiMain()
41 {
42     struct sockaddr_in direccion_unificador; //Estructura de datos que almacena la
43                                              //direccion del Unificador
44
45     int conexion;       //Socket utilizado en la comunicacion hacia el Unificador
46
47     char buff[CANTIDAD];  //Buffer utilizado en la comunicacion con el Unificador
48     int cantidad;         //Cantidad de datos recibidos del Unificador
49
50     FILE* archivo_configuracion;       //Archivo de configuracion del CGI
51     char IP_Unificador[20];            //Direccion IP del Unificador
52
53
54     int paquete;   //Indica el numero de paquete solicitado
55     int dato;      //Indica el dato leido dentro del buffer
56
57
58     float puntos[9250] = {0.0};   //Arreglo donde se almacenan los puntos que
59                                   //conformaran la grafica
60     int punto;                 //Indica la posicion dentro de 'puntos' donde se debe
61                                //almacenar el punto recibido
62
63
64     //Abrir archivo de configuracion
65     archivo_configuracion = fopen( ARCHIVO_CONF, "r");
66     if ( archivo_configuracion  == NULL )
67     {
68         cgiHeaderContentType( "text/html" );
69         fprintf( cgiOut, "No se encontro el archivo de configuracion" );
70         return 0;
71     }
```

```
72
73     //Leer direccion IP del Unificador
74     fscanf( archivo_configuracion, "%s", IP_Unificador );
75
76
77     //Construir la direccion del Unificador
78     memset( &direccion_unificador, 0, sizeof(direccion_unificador) );
79     direccion_unificador.sin_family = AF_INET;
80     direccion_unificador.sin_addr.s_addr = inet_addr ( IP_Unificador );
81     direccion_unificador.sin_port = htons( UNIFICADOR_PUERTO );
82
83
84     //Iniciar la solicitud de los paquetes
85     paquete = 0;
86     punto = 0;
87
88     while ( paquete < PAQUETES )
89     {
90         //Abrir la conexion hacia el Unificador
91         conexion = socket( AF_INET, SOCK_STREAM, 0 );
92         if ( conexion < 0 )
93         {
94             cgiHeaderContentType( "text/html" );
95             fprintf( cgiOut, "Error al abrir socket" );
96             return 0;
97         }
98
99         if ( connect( conexion, (struct sockaddr*) &direccion_unificador,
100sizeof(direccion_unificador) ) < 0 )
101        {
102            cgiHeaderContentType( "text/html" );
103            fprintf( cgiOut, "Error al conectarse al servidor" );
104            return 0;
105        }
106
107
108
109
110        //Enviar el nombre del Microcontrolador
111        write( conexion, NOMBRE_MICROCONTROLADOR, strlen(NOMBRE_MICROCONTROLADOR) );
112        write( conexion, "\0", 1 ); //Terminador de cadena
113
114        //Enviar el tipo de instruccion
115        buff[0] = INSTRUCCION_LECTURA;
116        write( conexion, buff, 1);
117
118        //Enviar la cantidad de datos a leer
119        buff[0] = 1;
120        write( conexion, buff, 1);
121
122
123
124
125        //Verificar que la instruccion haya sido recibida por el Unificador
126        read( conexion, buff, 1 );
127
128        if ( buff[0] == UNIFICADOR_INSTRUCCION_ERROR )
129        {
130            cgiHeaderContentType( "text/html" );
131            fprintf( cgiOut, "La instruccion no fue recibida correctamente por el
132Unificador" );
133            return 0;
134        }
135        if ( buff[0] == UNIFICADOR_INSTRUCCION_DATOS )
136        {
137            cgiHeaderContentType( "text/html" );
138            fprintf( cgiOut, "El Unificador recibio datos incompletos" );
139            return 0;
140        }
141        if ( buff[0] == UNIFICADOR_INSTRUCCION_DIR )
142        {
```

```
143             cgiHeaderContentType( "text/html" );
144             fprintf( cgiOut, "El nombre del Microcontrolador no es correcto" );
145             return 0;
146         }
147
148
149         //Verificar que la instruccion haya sido procesada
150         read( conexion, buff, 1 );
151
152         if ( buff[0] != RESPUESTA_OK )
153         {
154             cgiHeaderContentType( "text/html" );
155             fprintf( cgiOut, "La instruccion no fue procesada" );
156             return 0;
157         }
158
159
160
161
162         //Recibir bandera de indicacion
163         read( conexion, buff, 1 );
164
165
166         //Cerrar conexion
167         close( conexion );
168
169
170
171
172         //Si la bandera indica que el paquete esta listo, solicitar el paquete.
173         if ( buff[0] == 'R' )
174         {
175             //Abrir la conexion al Unificador
176             conexion = socket( AF_INET, SOCK_STREAM, 0 );
177             if ( conexion < 0 )
178             {
179                 cgiHeaderContentType( "text/html" );
180                 fprintf( cgiOut, "Error al abrir socket" );
181                 return 0;
182             }
183
184             if ( connect( conexion, (struct sockaddr*) &direccion_unificador,
185sizeof(direccion_unificador) ) < 0 )
186             {
187                 cgiHeaderContentType( "text/html" );
188                 fprintf( cgiOut, "Error al conectarse al servirdor" );
189                 return 0;
190             }
191
192
193
194
195             //Enviar el nombre del Microcontrolador
196             write( conexion, NOMBRE_MICROCONTROLADOR, strlen(NOMBRE_MICROCONTROLADOR)
197);
198             write( conexion, "\0", 1 ); //Terminador de cadena
199
200             //Enviar el tipo de instruccion
201             buff[0] = INSTRUCCION_LECTURA;
202             write( conexion, buff, 1 );
203
204             //Enviar la cantidad de datos a leer
205             buff[0] = CANTIDAD;
206             write( conexion, buff, 1 );
207
208
209
210
211
212             //Verificar que la instruccion haya sido recibida por el Unificador
213             read( conexion, buff, 1 );
```

```
214
215            if ( buff[0] == UNIFICADOR_INSTRUCCION_ERROR )
216            {
217                cgiHeaderContentType( "text/html" );
218                fprintf( cgiOut, "La instruccion no fue recibida correctamente por el
219Unificador" );
220                return 0;
221            }
222            if ( buff[0] == UNIFICADOR_INSTRUCCION_DATOS )
223            {
224                cgiHeaderContentType( "text/html" );
225                fprintf( cgiOut, "El Unificador recibio datos incompletos" );
226                return 0;
227            }
228            if ( buff[0] == UNIFICADOR_INSTRUCCION_DIR )
229            {
230                cgiHeaderContentType( "text/html" );
231                fprintf( cgiOut, "El nombre del Microcontrolador no es correcto" );
232                return 0;
233            }
234
235
236            //Verificar que la instruccion haya sido procesada
237            read( conexion, buff, 1 );
238
239            if ( buff[0] != RESPUESTA_OK )
240            {
241                cgiHeaderContentType( "text/html" );
242                fprintf( cgiOut, "La instruccion no fue procesada" );
243                return 0;
244            }
245
246
247
248
249
250            //Leer datos
251            read( conexion, buff, CANTIDAD );
252
253            //Almacenar los puntos recibidos
254            for ( dato = 1; dato < CANTIDAD; dato++ )
255            {
256                puntos[punto] = (float) buff[dato];
257                punto++;
258            }
259
260
261
262            paquete++;
263
264            //Cerrar la conexion
265            close( conexion );
266
267
268            //Esperar a que el Microcontrolador tenga otro paquete listo
269            usleep( 2000 );
270        }
271
272    }
273
274
275    //Enviar la grafica en forma de imagen en formato GIF
276    cgiHeaderContentType( "image/gif" );
277    out_graph( 600,200, cgiOut, GDC_LINE, 9250, NULL, 1, puntos );
278
279
280    return 0;
281}
282
283
284//Fin del archivo "tektronix.c"
```

APÉNDICE F

Instalación y manejo del Unificador
Instalación de los Programas Cliente

Instalación y manejo del Unificador

Para ejecutar el Unificador se necesita una computadora personal que posea al menos un puerto serial. Si el Unificador recibiera conexiones de programas en otras computadoras, necesita también un adaptador para red, una tarjeta Ethernet por ejemplo.

La velocidad del procesador y la cantidad de memoria RAM no importa, este programa no necesita muchos recursos de procesamiento y la mayor parte de tiempo estará en modo de espera. Además se necesita 5MB libres en el disco duro. Esta computadora personal debe poseer un sistema operativo Linux cuyo *kernel* sea de la versión 2.4 en adelante.

Es aconsejable crear un usuario en el sistema para manejar el unificador. En su directorio se deberá colocará el software y este usuario será el encargado de manejar el Unificador. Esto para restringir la operación del Unificador y que no logre accesos que comprometan la seguridad del sistema.

Acompañando a este documento se encuentra un CD que contiene un archivo llamado `unificador.tar.gz`. El contenido de este archivo debe ser colocado en su totalidad en el directorio del usuario creado para el Unificador.

El archivo `unificador.tar.gz` contiene los siguientes archivos en la siguiente jerarquía:

- directorio
- Makefile
- start
- status
- stop
- unificador.cc
- Antena:
    - red.h
    - TAntena.h
    - TAntena.cc
- Directorio :
    - TNodo.h

    - TLista.h
    - TDirectorio.h
    - TDirectorio.cc
- Enlace:
    - TEnlace.h
    - TEnlace.cc
    - puerto_serial:
        - puerto_serial.h
        - puerto_serial.cc
- Tipos:
    - TRespuesta.h
    - TInstruccion.h
    - TInstruccion.cc

Ahora se debe ejecutar el comando `make` para construir el archivo ejecutable `unificador`. Aunque este archivo es en sí el Unificador no se ejecuta directamente. Para ello existe tres scripts: start, status y stop. Como sus nombres lo indican, el primero inicia la ejecución del Unificador, verificando primero si existe ya un Unificador en ejecución. El segundo indica si ya existe un Unificador en ejecución o no. El tercero detiene la ejecución del Unificador si lo hubiera.

Antes de iniciar el Unificador por primera ves se debe verificar el archivo `/dev/ttyS0`. Debe tener los permisos adecuados para que el usuario dedicado al Unificador pueda leer y escribir a ese archivo.

Ahora se debe colocar en el archivo `directorio` las correlaciones de direcciones a nombres de microcontroladores. Un ejemplo de una lista de correlación es:

```
12   puerta_principal
14   puerta_interna
20   ventilador1
21   ventilador2
23   ventilador3
```

Además el Unificador genera un archivo donde registra las operaciones realizadas. Este archivo se llama `logfile`. Puede ser empleado para detectar problemas en alguna aplicación.

Instalación de los Programas Cliente de ejemplo

Dentro del mismo CD se encuentra un archivo llamado `programas.tar.gz`. Este contiene el código fuente de los Programas Cliente de ejemplo, así como dos librerías. Una de ellas es utilizada para crear programas tipo CGI en lenguaje C. La segunda permite crear gráficas en formato GIF.

El archivo `programas.tar.gz` contiene los siguientes archivos:

- cgic — Librería para crear programas tipo CGI en lenguaje C.
- gdchart — Librería para crear gráficas en formato GIF.
- constantes.h
- red.h
- ventilador:
  - Makefile
  - ventilador.c
- tektronix:
  - Makefile
  - tektronix.c

El contenido de este archivo debe ser colocado en cualquier directorio. Se ejecuta el comando `make` para el directorio `ventilador` y el directorio `tektronix`. De esta forma son generados los archivos `ventilador.cgi` y `tektronix.cgi`. Ellos deben ser colocados en el directorio apropiado para que el servidor web pueda ejecutarlos. Generalmente es en `/var/www/cgi-bin/`.

F