Cika Electrónica

Nota de Aplicación: CAN-024
Título: **Control de Personal monitoreable Ethernet con Rabbit**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	08/09/04	

Nos encontramos esta vez para desarrollar una herramienta de control de personal, destinada a registrar el horario de ingreso y egreso mediante la identificación por elementos RFID, es decir, las conocidas tarjetas de proximidad y los modernos llaveros. El empleado recibe confirmación visual y auditiva, oyendo una chicharra y observando en un display alfanumérico su nombre y la hora y fecha en la cual se registra su tarjeta. El interesado en controlar al empleado, puede observar los horarios en orden decreciente en una página web, recibir un email con un log de accesos, y recibir un archivo en un servidor FTP, en formato CSV (Comma Separated Values), el cual puede procesar y filtrar automáticamente con cualquier herramienta orientada a procesar registros (*awk*, por ejemplo en ambiente Unix), o incluso manualmente con cualquier planilla de cálculo.

Aprovechamos el desarrollo de la nota de aplicación CAN-002 para conectar nuestro display; como lector utilizaremos el GP8F-R2. Para un control de acceso de muy bajo costo basado en este mismo lector RFID, el lector interesado puede consultar la CAN-017.

Descripción del GP8F-R2

El módulo GP8F-R2 lee tarjetas o tags RFID read-only de 64-bits, código Manchester a 125KHz. Posee una salida para conectar un LED, que permanece encendido y aumenta su intensidad al aproximar un RFID. El ID correspondiente se entrega por un terminal en formato 8 bits serie asincrónico, a 9600 bps, sin paridad, a nivel lógico TTL, lo cual lo hace excelente para ser leído desde un micro. El formato lógico responde al siguiente esquema, en ASCII:

```
<STX> <DATA (10 bytes)> <CR> <LF> <ETX>
```

STX: ASCII Start of Text (0x02)
ETX: ASCII End of Text (0x03)
CR: ASCII Carriage Return (0x0D)
LF: ASCII Line Feed (0x0A)

El campo de DATA es una representación en ASCII del ID del RFID, representando 5 bytes binarios (40 bits) de la siguiente forma:

byte ASCII
C1 43 31

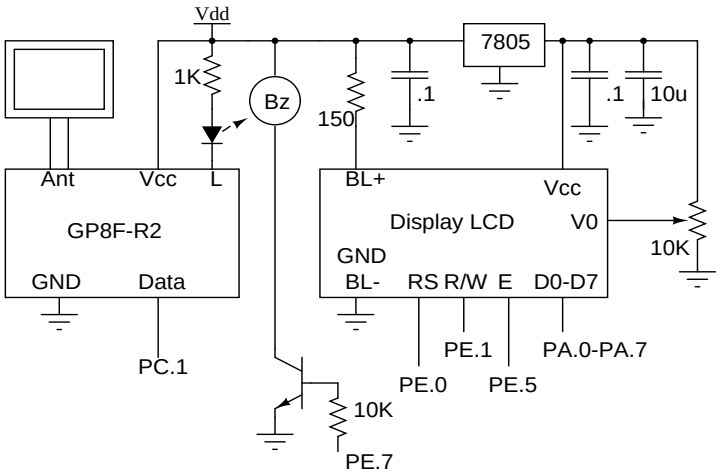
Por ejemplo, el ID `60 22 57 C0 31,` se transmite de la siguiente forma:

```
02 36 30 32 32 35 37 43 30 33 31 0D 0A 03
```

Recordemos que este tipo de RFID tags de 64 bits 125 Khz utiliza 9 bits para header, 40 para datos, 14 bits de paridad y uno de stop.

Hardware

Conectamos el módulo GP8F-R2 y un display LCD al módulo RCM2200 como indica el diagrama a continuación, utilizamos el port serie D para recibir la información del GP8F-R2, el port paralelo E para las señales de control del display y el port paralelo A para los datos:



A fin de preservar el contenido de la información ante un corte de alimentación, conectaremos una pila de litio de 3V entre el terminal *VBAT* y masa. La tensión de alimentación, *Vdd*, es de 12V.

Software

La primera vez que arranca el sistema, no tendrá datos válidos en las variables que se almacenan en RAM con battery-backup. Para corregir este problema sin complicar el código, el administrador dispone de la opción de purgar la base de IDs y la de registros, lo cual coloca ambos contadores en cero. Algo similar ocurre con la información relativa al envío de reportes, en este caso se verá información sin sentido en donde debería haber información de a dónde y cuándo enviar los reportes. Para los alcances de una nota de aplicación, esto no es demasiado problema, si el lector planea hacer un producto comercial, deberá planear algún sistema de detección de inconsistencias o "factory defaults" y obrar en consecuencia.

Cada empleado tendrá un dispositivo RFID que lo identifica (llavero, tarjeta, etc). Para memorizar dichos IDs, el administrador ingresa al sistema y solicita el aprendizaje de un ID, ingresando el nombre del empleado. En este momento, el equipo le indica que acerque al lector el RFID, el cual será memorizado en una tabla.

A fin de minimizar los requerimientos de memoria y poder almacenar mayor cantidad de registros, cuando se produce un evento (lectura de un ID válido), se almacena un registro que contiene la información de fecha y el número de orden en la tabla de IDs del empleado cuyo ID se acaba de reconocer. Esto requiere que cada vez que se solicita un listado, el mismo deba ser generado tomando la información de los registros y buscando el nombre del empleado en la tabla de IDs, pero minimiza la cantidad de memoria requerida sin complicar demasiado el proyecto. De igual modo, la información de fecha y hora se guarda en formato compacto, siendo reprocesada cada vez que se genera un listado.

Todas las funciones relacionadas con el manejo del display LCD, puede obtenerlas de la CAN-002 o en el archivo fuente, aquí nos concentraremos en los demás temas.

Primero incluímos algunas definiciones:

```
#define HOLD_TIME      5
#define BEEP_TIME      100

#define NUM_ENTRIES    60
#define NUM_RECORDS    256

#define DINBUFSIZE     15
#define DOUTBUFSIZE    15
```

Las mismas corresponden al tiempo de persistencia de la información en el display en segundos, duración del beep en milisegundos, cantidad máxima de IDs de empleados, cantidad máxima de registros en memoria, y tamaños de buffer para la interfaz serie.

Luego incluímos el direccionamiento IP, podemos utilizar *tcpconfig.lib* o definir directamente los datos aquí:

```
#define MY_IP_ADDRESS   "192.168.1.52"
#define MY_NETMASK      "255.255.255.0"
```

```
#define MY_GATEWAY    "192.168.1.1"  // sólo si debe rutear
```

Como al transmitir el reporte via FTP nos interesa identificarlo con un nombre alegórico y la fecha del reporte, usaremos un nombre de archivo extenso. Para tal fin, modificamos la macro que define la cantidad máxima de caracteres en el nombre, para FTP:

```
#define FTP_MAX_FNLEN 32
```

Luego definimos el uso de memoria extendida e incluimos todas las bibliotecas de funciones que utilizamos:

```
#memmap xmem
#use "dcrtcp.lib"
#use "http.lib"
#use "smtp.lib"
#use "ftp_client.lib"
```

Importamos las imágenes y páginas que usará el servidor web para mostar la interfaz de monitoreo y configuración via HTTP. Para almacenarlas, utilizaremos la directiva *#ximport*, que nos permite leer un archivo al momento de compilar y guardarlo en memoria (XMEM), pudiendo referenciarse mediante un puntero:

```
#ximport "index.shtml"    index_html
#ximport "list.shtml"     list_html
#ximport "learn.shtml"    learn_html
#ximport "admin.shtml"    admin_html
#ximport "admail.shtml"   admail_html
#ximport "admftp.shtml"   admftp_html
#ximport "purgeid.shtml"  purgeid_html
#ximport "purgedb.shtml"  purgedb_html
#ximport "setdate.shtml"  setdate_html
#ximport "error.html"     error_html
#ximport "noroom.html"    noroom_html
#ximport "ok.html"         ok_html
#ximport "ok2.html"        ok2_html
#ximport "ok3.html"        ok3_html
#ximport "rabbit1.gif"    rabbit1_gif
```

Así, *index_html* es un puntero a la posición de memoria física donde está guardado el archivo *index.shtml*, tal cual figuraba en el directorio de nuestra máquina al momento de compilar el proyecto. El path especificado es relativo al archivo fuente de donde se lo llama.

A continuación, debemos definir los tipos MIME y su correspondiente handler. Esto se debe a que no tenemos un sistema operativo que nos resuelva estas tareas, y debemos decirle al servidor qué es cada cosa, es decir, algo así como la función desempeñada por el archivo *mime.types* en Linux.
Cuando el URL no especifica archivo, como por ejemplo http://alguna_dirección/ , el servidor generalmente entrega el archivo que se le especifica en su configuración, por lo general *index.html*. En este caso, debemos primero indicar el tipo MIME para ese caso en particular, por lo que la primera entrada en la estructura corresponde al acceso al URL sin especificar un archivo:

```
const HttpType http_types[] =
{
   { ".shtml", "text/html", shtml_handler}, // ssi
   { ".html", "text/html", NULL},           // html
   { ".cgi", "", NULL},                     // cgi
   { ".gif", "image/gif", NULL}
};
```

Así, definimos que un acceso a http://MY_IP_ADDRESS/ es un acceso a un archivo de tipo SHTML, que los archivos terminados en *.shtml* y *.html* serán reportados por el servidor como de tipo (MIME type) *text/html*, mientras que los terminados en *.gif* se reportarán como *image/gif*. Asimismo, definimos que los archivos de tipo SHTML serán procesados por un handler llamado *shtml_handler*, que es el engine para los server side includes que provee Dynamic C; el resto de los otros tipos definidos utilizará el handler por defecto. La entrada correspondiente a archivos de tipo CGI, *.cgi*, simplemente define la extensión, definiremos cómo se procesa cada función CGI más adelante.

Definimos a continuación las estructuras para guardar los IDs y los registros, más algunas variables globales que utilizaremos más adelante. Aquellas variables que contienen la configuración y los registros, las hemos hecho de tipo *protected*, es decir, el compilador genera código que crea una copia de seguridad de cada variable antes de modificarla, de modo que si por alguna eventualidad llega a ocurrir un reset en medio de una modificación de alguna de estas variables, es posible recuperar el valor original.

```
typedef struct {
       char name[21];
       unsigned char rfid[6];
} rfid_record;

typedef struct {
       int id;
       long time;
} time_record;


protected rfid_record rfid_base[NUM_ENTRIES];
protected time_record time_base[NUM_RECORDS];

protected int entries,records;

protected char from[21],to[21],subject[21],smtp_server[21],mail_time[9];
protected char user[21],password[21],filename[21],ftp_server[21],ftp_time[9];

const static char *months[]={
"ENE","FEB","MAR","ABR","MAY","JUN","JUL","AGO","SEP","OCT","NOV","DIC"
};

char date[12],time[9];

/* Program states */
#define ESCRACHATE    1
#define LEARN                 2
static int sstate;
```

Estas funciones se encargan de realizar las acciones solicitadas al modificar la configuración. Son llamadas cada vez que el usuario ingresa una forma a través de la página, como veremos más adelante. Debemos declararlas ya que serán utilizadas al definir el directorio del servidor web, en el párrafo siguiente.

```
int learn(HttpState*);
int list(HttpState*);
int show(HttpState*);
int setdate(HttpState*);
int setmail(HttpState*);
int setftp(HttpState*);
int purgeid(HttpState*);
int purgedb(HttpState*);
int delete(HttpState*);
```

Ahora, debemos decirle al servidor HTTP (el cual Dynamic C provee listo para nuestro uso) de qué archivos dispone para trabajar, es decir, asociamos los URLs con los punteros a la información que importamos antes con *#ximport*. En este caso utilizaremos la forma más fácil, que consiste en aprovechar una estructura definida en HTTP.LIB, la biblioteca de funciones del web server, llamada *http_flashspec*. Como algunas de las páginas están protegidas por password, definiremos primero el *realm* al que dicho password está asociado:

```
const static HttpRealm myrealm[]={"usu","pass","Escrachator"};
const static HttpSpec http_flashspec[] =
{
   { HTTPSPEC_FILE,  "/",              index_html,    NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE,  "/index.shtml",   index_html,    NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE,  "/learn.shtml",   learn_html,    NULL, 0, NULL, myrealm},
   { HTTPSPEC_FILE,  "/admin.shtml",   admin_html,    NULL, 0, NULL, myrealm},
   { HTTPSPEC_FILE,  "/admail.shtml",  admail_html,   NULL, 0, NULL, myrealm},
   { HTTPSPEC_FILE,  "/admftp.shtml",  admftp_html,   NULL, 0, NULL, myrealm},
   { HTTPSPEC_FILE,  "/purgeid.shtml", purgeid_html,  NULL, 0, NULL, myrealm},
   { HTTPSPEC_FILE,  "/purgedb.shtml", purgedb_html,  NULL, 0, NULL, myrealm},
   { HTTPSPEC_FILE,  "/list.shtml",    list_html,     NULL, 0, NULL, myrealm},
```

```
   { HTTPSPEC_FILE,  "/setdate.shtml", setdate_html,  NULL, 0, NULL, myrealm},
   { HTTPSPEC_FILE,  "/ok.html",        ok_html,      NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE,  "/ok2.html",       ok2_html,     NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE,  "/ok3.html",       ok3_html,     NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE,  "/error.html",    error_html,    NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE,  "/noroom.html",   noroom_html,   NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE,  "/rabbit1.gif",   rabbit1_gif,   NULL, 0, NULL, NULL},

   { HTTPSPEC_VARIABLE, "date", 0,    date,         PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "time", 0,    time,         PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "records", 0, &records,     INT16,  "%d", NULL},
   { HTTPSPEC_VARIABLE, "entries", 0, &entries,     INT16,  "%d", NULL},
   { HTTPSPEC_VARIABLE, "server",  0, smtp_server,  PTR16, "%s", NULL},
   { HTTPSPEC_VARIABLE, "from",         0, from,    PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "to",           0, to,      PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "subject", 0, subject,      PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "rtime",   0, mail_time,    PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "fserver", 0, ftp_server,   PTR16,  "%s", NULL},
   { HTTPSPEC_VARIABLE, "user",         0, user,    PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "password",0, password,     PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "filename",0, filename,     PTR16,   "%s", NULL},
   { HTTPSPEC_VARIABLE, "ftime",   0, ftp_time,     PTR16,   "%s", NULL},

   { HTTPSPEC_FUNCTION,       "/learn.cgi",          0,  learn,    0, NULL, myrealm},
   { HTTPSPEC_FUNCTION,       "/purgeid.cgi",        0,  purgeid,  0, NULL, myrealm},
   { HTTPSPEC_FUNCTION,       "/purgedb.cgi",        0,  purgedb,  0, NULL, myrealm},
   { HTTPSPEC_FUNCTION,       "/delete.cgi",         0,  delete,   0, NULL, myrealm},
   { HTTPSPEC_FUNCTION,       "/setdate.cgi",        0,  setdate,  0, NULL, myrealm},
   { HTTPSPEC_FUNCTION,       "/setmail.cgi",        0,  setmail,  0, NULL, myrealm},
   { HTTPSPEC_FUNCTION,       "/setftp.cgi",         0,  setftp,   0, NULL, myrealm},
   { HTTPSPEC_FUNCTION,       "show",                0,  show,     0, NULL, NULL},
   { HTTPSPEC_FUNCTION,       "list",                0,  list,     0, NULL, myrealm}
};
```

Las entradas `HTTPSPEC_FILE` asocian los siguientes URLs con los punteros y por ende archivos que figuran a continuación (ver *#ximport* más arriba):

http://MY_IP_ADDRESS/ ó http://MY_IP_ADDRESS/index.shtml -> index_html, puntero a index.shtml
http://MY_IP_ADDRESS/rabbit1.gif -> rabbit1_gif, puntero a rabbit1.gif. Lo mismo ocurre para las demás imágenes en formato GIF.

Cabe recordar que el contenido de dichos archivos fue copiado y asociado al puntero al utilizar *#ximport*. A su vez, le dijimos al servidor cómo debía manejar cada tipo de archivo al definir *http_types* (ver más arriba).

Las líneas `HTTPSPEC_VARIABLE` definen variables a ser procesadas por el servidor al recibir una petición de una página SHTML (HTTP GET). Definimos el nombre como se la refiere en el código fuente SHTML de la página, la variable en el programa (que debe ser global), el tipo, y la forma de tratarla para mostrar su valor.

Las líneas `HTTPSPEC_FUNCTION` asocian los URLs con las funciones a ejecutar cuando se las solicita. Las mismas serán ejecutadas cuando el usuario ingrese el form o solicite via SSI la función, como veremos al analizar el código SHTML. La función recibe un puntero a la estructura donde el servidor HTTP contiene toda la información necesaria como para que la función pueda procesar la información, de ser necesario.

Por comodidad, al ejecutar el CGI preferimos realizar un HTTP REDIRECT en vez de manejar la entrega de la respuesta dentro de la función, lo cual implicaría escribir en el handler del port TCP. Al usar el redirect, simplemente le decimos al navegador que vaya a buscar otra página y allí se le presentará la nueva información.

```
#define REDIRECTHOST          MY_IP_ADDRESS
#define REDIRECTTOK     "http://" REDIRECTHOST "/ok.html"
#define REDIRECTTOK2    "http://" REDIRECTHOST "/ok2.html"
#define REDIRECTTOK3    "http://" REDIRECTHOST "/ok3.html"
#define REDIRECTTOERR   "http://" REDIRECTHOST "/error.html"
#define REDIRECTTOERR2  "http://" REDIRECTHOST "/noroom.html"
```

Escribimos ahora dos simples funciones que nos permitan manejar comodamente la información de fecha y hora, y generar un nombre de archivo con información de fecha:

```
void get_datetime()
{
```

```
auto struct tm thetm;

        mktm(&thetm, SEC_TIMER);
        sprintf(date,"%02d %s %04d",thetm.tm_mday,months[thetm.tm_mon-1],1900+thetm.tm_year);
        sprintf(time,"%02d:%02d:%02d",thetm.tm_hour, thetm.tm_min, thetm.tm_sec);
}

void gen_filename(char *dest,char *source)
{
auto struct tm thetm;

        mktm(&thetm, SEC_TIMER);
        sprintf(dest,"%s_%02d%02d%02d.csv",source,thetm.tm_mday,thetm.tm_mon,thetm.tm_year-
100);
}
```

Comenzamos ahora a desarrollar las funciones CGI que manejan la recepción de variables para actualizar la configuración. Las dos primeras funciones han sido provistas por Rabbit y son las que resuelven el problema de parsear el contenido del formulario, haciendo nuestra vida más fácil

```
#define MAX_FORMSIZE   21
typedef struct {
        char *name;
        char value[MAX_FORMSIZE];
} FORMType;
FORMType FORMSpec[5];

/*
 * Parse one token 'foo=bar', matching 'foo' to the name field in
 * the struct, and storing 'bar' into the value
 */
void parse_token(HttpState* state)
{
        auto int i, len;

        for(i=0; i<HTTP_MAXBUFFER; i++) {
                if(state->buffer[i] == '=')
                        state->buffer[i] = '\0';
        }
        state->p = state->buffer + strlen(state->buffer) + 1;

        for(i=0; i<(sizeof(FORMSpec)/sizeof(FORMType)); i++) {
                if(!strcmp(FORMSpec[i].name,state->buffer)) {
                        len = (strlen(state->p)>MAX_FORMSIZE) ? MAX_FORMSIZE - 1: strlen(state-
>p);
                        strncpy(FORMSpec[i].value,state->p,1+len);
                        FORMSpec[i].value[MAX_FORMSIZE - 1] = '\0';
                }
        }
}

/*
 * parse the url-encoded POST data into the FORMSpec struct
 * (ie: parse 'foo=bar&baz=qux' into the struct
 */
int parse_post(HttpState* state)
{
        auto int retval;

        while(1) {
                retval = sock_fastread((sock_type *)&state->s, state->p, 1);
                if(0 == retval) {
                        *state->p = '\0';
                        parse_token(state);

                        return 1;
                }

                /* should this only be '&'? (allow the eoln as valid text?) */
                if((*state->p == '&') || (*state->p == '\r') || (*state->p == '\n')) {
                        /* found one token */
                        *state->p = '\0';
                        parse_token(state);
```

```
                state->p = state->buffer;
            } else {
                state->p++;
            }

            if((state->p - state->buffer) > HTTP_MAXBUFFER) {
                /* input too long */
                return 1;
            }
        }

        return 0; /* end of data - loop again to give it time to write more */
}
```

La función siguiente se encarga de aprender un ID. Recibe el nombre del usuario via HTTP y setea al programa principal en el estado de aprendizaje, de modo que la primera RFID que sea leída será ingresada en la base. Todo lo que hacemos es obtener el nombre e informarle al usuario que debe acercarse al lector. Es importante que ningún otro usuario ejecuta una form mientras esto ocurre, dado que sobreescribiría el lugar donde guaradamos el nombre del empleado. Esto resulta simple si sólo existe un administrador, quien disponde del password para evitar el acceso a las páginas de configuración, que son las que utilizan las forms. Si el lector así lo desea, puede definir una variable extra y almacenar allí el nombre a ingresar en la base cuando se reciba el RFID.

```
int learn(HttpState* state)
{
char buffer[MAX_FORMSIZE];

        if(entries<NUM_ENTRIES){
           FORMSpec[0].name = "nombre";
           FORMSpec[0].value[0] = 0;                        // inicializa datos form
                if(parse_post(state)) {                     // obtiene datos
                   if(*(http_urldecode(buffer,FORMSpec[0].value,MAX_FORMSIZE))) { //decodifica
                        strcpy(FORMSpec[0].value,buffer);
                        sstate=LEARN;                       // se pone en estado “aprender”
                        cgi_redirectto(state,REDIRECTTOK);  // Muestra pantalla “acerque”
                        }
                   else cgi_redirectto(state,REDIRECTTOERR); // Muestra error
                } else {
                   cgi_redirectto(state,REDIRECTTOERR);     // Muestra error
                               // Manejo más sofisticado de errores, si fuera necesario
                }
        }
        else cgi_redirectto(state,REDIRECTTOERR2);          // Muestra error
        return(0);
}
```

La siguiente función es llamada via SSI al pedirse la página de listado de IDs, como veremos al analizar el código SHTML. Esta función se encarga de generar el listado de IDs, pero la tarea se realiza de forma sumamente fácil, dado que el servidor HTTP ya de por sí es una máquina de estados, y nos permite tener sub-estados con los cuales podemos ir generando el listado; es decir, el mismo servidor nos irá llamando a esta función hasta que retornemos con el valor 1, lo cual haremos cuando hayamos terminado de imprimir todas las entradas:

```
int list(HttpState* state)
{
auto int printline;

        if(state->substate >= entries)              // si terminó indica salida
                return 1;

        printline = state->substate;                // obtiene línea a imprimir = subestado
        sprintf(state->buffer, "<tr><td>%2d<td>%s\r\n",printline,rfid_base[printline].name);
        state->headerlen = strlen(state->buffer);   // prepara datos, indica longitud
        state->headeroff = 0;
        state->substate++;                          // nueva línea, nuevo estado
        return(0);                                  // sigue trabajando
}
```

De igual modo, la siguiente función muestra todos los registros en memoria, basándose en el mismo principio

```
int show(HttpState* state)
{
       auto int printline;
       auto struct tm thetm;

       if(state->substate >= records)              // si terminó, indica salida
              return 1;

       printline = records-(state->substate)-1;            // los registros son en orden
                                                           // cronológico inverso
       mktm(&thetm, time_base[printline].time);            // traduce fecha y hora
       sprintf(state->buffer, "<tr><td>%02d %s %04d<td>%02d:%02d:%02d<td>%
         s\r\n",thetm.tm_mday,months[thetm.tm_mon-1],1900+thetm.tm_year,
         thetm.tm_hour, thetm.tm_min, thetm.tm_sec, rfid_base[time_base[printline].id].name);
       state->headerlen = strlen(state->buffer);           // prepara data y longitud
       state->headeroff = 0;
       state->substate++;                                  // nuevo registro, nuevo estado
       return 0;                                           // indica sigo trabajando
}
```

Como nadie es perfecto, agregamos una función que nos permita borrar un ID, compactando la base de IDs moviendo los restantes de forma de ocupar el espacio vacío. Recibimos el número de ID a borrar, el cual el usuario ingresó al observar el listado en la página web, decodificando la forma:

```
int delete(HttpState* state)
{
char buffer[MAX_FORMSIZE];
int id,err,i;

       err=0;
       FORMSpec[0].name = "id";
       FORMSpec[0].value[0] = 0;            // inicializa datos
       if(parse_post(state)) {
          if(*(http_urldecode(buffer,FORMSpec[0].value,MAX_FORMSIZE))) { // decodifica
               id=atoi(buffer);             // obtiene #ID en base de IDs
               if((id>=0)&&(id<entries)&&entries){   // si corresponde
                  for(i=id+1;i<entries;i++){         // copia toda el resto un lugar abajo
                       strcpy(rfid_base[i-1].name,rfid_base[i].name);
                       memcpy(rfid_base[i-1].rfid,rfid_base[i].rfid,5);
                  }
                  entries--;                                  // existe un ID menos
                  cgi_redirectto(state,REDIRECTTOK3);         // Muestra OK
               }
               else err=1;
          }
          else err=1;
       }
       else err=1;
       if(err)
          cgi_redirectto(state,REDIRECTTOERR);      // Muestra error
       return(0);
}
```

Para inicializar correctamente el sistema, proveemos la opción de borrar la base de IDs y los registros. Estas funciones son llamadas por el servidor como CGIs al obrar sobre un link que aparece en una página web

```
int purgeid(HttpState* state)
{
       entries=0;                                  // No hay IDs
       cgi_redirectto(state,REDIRECTTOK3);         // Muestra OK
       return 0;
}

int purgedb(HttpState* state)
{
       records=0;                                  // No hay registros
       cgi_redirectto(state,REDIRECTTOK3);         // Muestra OK
       return 0;
}
```

Las siguientes funciones configuran los parámetros para enviar el mail, el reporte FTP y la fecha y hora del equipo. Obtenemos todos los datos necesarios de un formulario que el administrador llena en una página web:

```
int setmail(HttpState* state)
{
char buffer[MAX_FORMSIZE];
unsigned int err;

        FORMSpec[0].name = "server";
        FORMSpec[1].name = "from";
        FORMSpec[2].name = "to";
        FORMSpec[3].name = "subject";
        FORMSpec[4].name = "time";
        FORMSpec[0].value[0] = 0;            // inicializa los datos
        FORMSpec[1].value[0] = 0;
        FORMSpec[2].value[0] = 0;
        FORMSpec[3].value[0] = 0;
        FORMSpec[4].value[0] = 0;
        err=0;
        if(parse_post(state)) {
           if(*(http_urldecode(buffer,FORMSpec[0].value,MAX_FORMSIZE))) { // decodifica
                strcpy(smtp_server,buffer);
                if(*(http_urldecode(buffer,FORMSpec[1].value,MAX_FORMSIZE))) {
                    strcpy(from,buffer);
                    if(*(http_urldecode(buffer,FORMSpec[2].value,MAX_FORMSIZE))) {
                        strcpy(to,buffer);
                        if(*(http_urldecode(buffer,FORMSpec[3].value,MAX_FORMSIZE))) {
                            strcpy(subject,buffer);
                            if(*(http_urldecode(buffer,FORMSpec[4].value,MAX_FORMSIZE))) {
                                strcpy(mail_time,buffer);
                                cgi_redirectto(state,REDIRECTTOK3);   // Muestra OK
                            }
                            else err=1;
                        }
                        else err=1;
                    }
                    else err=1;
                }
                else err=1;
           }
           else err=1;
        }
        else err=1;
        if(err)
           cgi_redirectto(state,REDIRECTTOERR);             // Muestra error
        return(0);
}

int setftp(HttpState* state)
{
char buffer[MAX_FORMSIZE];
unsigned int err;

        FORMSpec[0].name = "server";
        FORMSpec[1].name = "user";
        FORMSpec[2].name = "password";
        FORMSpec[3].name = "filename";
        FORMSpec[4].name = "time";
        FORMSpec[0].value[0] = 0;            // inicializa datos
        FORMSpec[1].value[0] = 0;
        FORMSpec[2].value[0] = 0;
        FORMSpec[3].value[0] = 0;
        FORMSpec[4].value[0] = 0;
        err=0;
        if(parse_post(state)) {
            if(*(http_urldecode(buffer,FORMSpec[0].value,MAX_FORMSIZE))) { // decodifica
                strcpy(ftp_server,buffer);
                if(*(http_urldecode(buffer,FORMSpec[1].value,MAX_FORMSIZE))) {
                    strcpy(user,buffer);
                    if(*(http_urldecode(buffer,FORMSpec[2].value,MAX_FORMSIZE))) {
                        strcpy(password,buffer);
                        if(*(http_urldecode(buffer,FORMSpec[3].value,MAX_FORMSIZE))) {
                            strcpy(filename,buffer);
                            if(*(http_urldecode(buffer,FORMSpec[4].value,MAX_FORMSIZE))) {
```

```
                        strcpy(ftp_time,buffer);
                        cgi_redirectto(state,REDIRECTTOK3);   // Muestra OK
                    }
                    else err=1;
                }
                else err=1;
            }
            else err=1;
        }
        else err=1;
    }
    else err=1;
    if(err)
        cgi_redirectto(state,REDIRECTTOERR);     // Muestra error
    return(0);
}


int setdate(HttpState* state)
{
char buffer[MAX_FORMSIZE],*p,*endptr;
unsigned int month, day, year, hour, minute, done;
struct tm rtc;                              // para fecha y hora


    FORMSpec[0].name = "date";
    FORMSpec[0].value[0] = 0;
    if(parse_post(state)) {
        if(*(http_urldecode(buffer,FORMSpec[0].value,MAX_FORMSIZE))) {
            p=buffer;
            done=0;
            day = (unsigned int)strtod(p, &endptr);
            if (endptr != p && day > 0 && day < 32) {
              p = endptr + 1;
              month = (unsigned int)strtod(p, &endptr);
              if (endptr != p && month > 0 && month < 32) {
                p = endptr + 1;
                year = (unsigned int)strtod(p, &endptr);
                if (endptr != p && (year < 48 || year > 79)) {
                  if (year < 48)
                    year += 100;
                  p = endptr + 1;
                  hour = (unsigned int)strtod(p, &endptr);
                  if (endptr != p && hour < 24) {
                    p = endptr + 1;
                    minute = (unsigned int)strtod(p, &endptr);
                    if (minute < 60) {
                        rtc.tm_sec  = 0;                // 0-59
                        rtc.tm_min  = minute;           // 0-59
                        rtc.tm_hour = hour;             // 0-23
                        rtc.tm_mday = day;              // 1-31
                        rtc.tm_mon  = month;            // 1-12
                        rtc.tm_year = year;             // 80-147, sumar 1900 para año
                        tm_wr(&rtc);                    // pone RTC en hora,
                        done=1;                         // efectivo al reset
                        cgi_redirectto(state,REDIRECTTOK2);  // Muestra OK
                    }
                  }
                }
              }
            }
            if (!done) {
                cgi_redirectto(state,REDIRECTTOERR);     // Muestra error
            }
        }
        else cgi_redirectto(state,REDIRECTTOERR);
    } else {
        cgi_redirectto(state,REDIRECTTOERR);
    }
    return(0);
}
```

Ahora sí, esta función obtiene el RFID en binario extrayéndolo del paquete ASCII que recibimos del GP8F-R2:

```
void decode(unsigned char *rfid)
{
int i;
       for(i=0;i<5;i++){             // 5 bytes (10 caracteres)
           rfid[2*i+1]-=0x30;        // resta '0' primer caracter
           if(rfid[2*i+1]>=10)       // compensa A-F
               rfid[2*i+1]-=7;
           rfid[2*i+2]-=0x30;        // segundo caracter
           if(rfid[2*i+2]>=10)
               rfid[2*i+2]-=7;
           rfid[i]=(rfid[2*i+1]<<4) | rfid[2*i+2]; // compone y repite para cada byte
       }
}
```

La siguiente función busca un ID en la base de IDs, para ver si corresponde a uno conocido. Devuelve el número de ID ó -1 si no lo encuentra

```
int lookup(unsigned char *id)
{
int i;
       for(i=0;i<entries;i++){
           if(!memcmp(id,rfid_base[i].rfid,5))
               return(i);
       }
       return(-1);
}
```

Cuando enviemos el mail, lo que haremos es generar un listado de registros línea a línea en tiempo real, a medida que mandamos el mail. Es decir, nunca tendremos el cuerpo del mail en memoria, sino que lo generamos a pedido, esto reduce considerablemente los requerimientos de memoria y nos muestra una forma de trabajar diferente a la que usáramos en otras notas de aplicación como la CAN-018, por ejemplo. Observen la simpleza de esta función, nuevamente sacamos provecho de la máquina de estados contenida en las bibliotecas de funciones que Dynamic C nos provee.

```
int mail_generator(char *buf, int len, longword offset, int flags, void *dhnd_data)
{
auto int *printline;
auto struct tm thetm;

       if (flags == SMTPDH_OUT) {
           printline = (int *)dhnd_data;     // apunta al número de línea a enviar

           if((*printline) >= records)       // si terminó el listado, lo indica
               return 0;

           mktm(&thetm, time_base[(*printline)].time);      // traduce fecha y hora
           sprintf(buf, "%02d %s %04d\t%02d:%02d:%02d\t%s\t\t%d\t%d\t%
               d\r\n",thetm.tm_mday,months[thetm.tm_mon-1],1900+thetm.tm_year,
               thetm.tm_hour, thetm.tm_min, thetm.tm_sec,
               rfid_base[time_base[(*printline)].id].name,
               thetm.tm_mday,thetm.tm_mon,1900+thetm.tm_year);      // arma registro
           (*printline)++;                   // nueva línea
           return(strlen(buf));              // longitud, indica sigue generando mail
       }
       return -1;                            // indica “no interesa procesar esta condición”
}
```

Para generar el reporte FTP, realizamos la misma tarea. Otra vez, también disponemos de una máquina de estados para sacarle provecho.

```
int ftp_datahandler(char *data, int len, longword offset, int flags, void *dhnd_data)
{
static printline;
auto struct tm thetm;

#GLOBAL_INIT {
       printline=0;          // inicializa variable estática cuando arranca
```

```
}

        switch (flags) {
            case FTPDH_IN:            // sólo enviamos
                return 0;
            case FTPDH_OUT:           // estado de envío
                if(printline >= records)      // si terminó, lo indica
                    return 0;
                mktm(&thetm, time_base[printline].time);     // traduce fecha y hora
                sprintf(data, "%02d %s %04d,%02d:%02d:%02d,%s,%d,%d,%
                       d\r\n",thetm.tm_mday,months[thetm.tm_mon-1],1900+thetm.tm_year,
                       thetm.tm_hour, thetm.tm_min, thetm.tm_sec,
                       rfid_base[time_base[printline].id].name,
                       thetm.tm_mday,thetm.tm_mon,1900+thetm.tm_year);
                printline++;                  // Nuevo registro
                return(strlen(data));         // longitud, indica que sigue enviando

            case FTPDH_END:
            case FTPDH_ABORT:         // al cerrarse la conexión
                printline=0;          // inicializa variable para próxima vez
                return 0;
        }
        return -1;                    // cualquier otro caso, “no lo procesamos”
}
```

Llegamos finalmente al cuerpo del programa principal. Luego de la inicialización, procesaremos cada una de las diversas tareas mediante *costates*, aprovechando las funciones de multitarea cooperativo de Dynamic C.

```
main()
{
int i,sending,linenum,beep,ftpret,fsending;
unsigned char RFID[15];
char my_filename[32];
const static char wrong[]="El RFID leido";
static unsigned long timer;


#GLOBAL_INIT {
        sstate=ESCRACHATE;    // el estado inicial es de operación
}

        _sysIsSoftReset();    // detecta reset y recupera variables protegidas
        sending=0;            // no estamos enviando ningún mail
        fsending=0;           // no estamos enviando ningún reporte
        beep=0;               // no estamos haciendo beep
        LCD_init();           // inicializa display LCD
        LCD_clear();          // lo borra
        LCD_show("Time Logger v1.0   READY");        // muestra pantalla de inicio
        LCD_at(1,0);
        LCD_show("(C)2004 Cika Electronica");        // copyright
        timer=SEC_TIMER;      // mantiene texto un cierto tiempo si no hay eventos
```

Luego inicializamos la red, el servidor HTTP, y la interfaz serie

```
        sock_init();          // inicializa Ethernet y TCP/IP
        http_init();          // inicializa servidor web
        serDopen(9600);       // inicializa interfaz serie 8N1
```

Y finalmente el loop principal. Las tareas definidas son:

- ➔ Buzzer, tiempo de beep sin molestar otras tareas y sin interrupciones
- ➔ TCP/IP: procesamiento de HTTP, manejo de clientes SMTP y FTP cuando se deben enviar reportes
- ➔ Display: usamos una tarea aparte para mostrar la hora mientras no hay nada que hacer
- ➔ Control de personal: toma control del display y realiza las tareas del control en sí
- ➔ Reporte Mail: espera hasta que sea el momento requerido y genera el reporte
- ➔ Reporte FTP: espera hasta que sea el momento requerido y genera el reporte

Cada tarea entrega el control a las demás cuando no tiene nada que hacer.

Dado que el reporte via FTP borra los registros ya reportados, si interesa generarse un reporte via mail, éste debe ser anterior al reporte FTP, de otro modo no habrá registros que enviar y no se generará reporte; a menos, claro está, que alguien ingrese en el timepo intermedio.

```
while(1){
    costate {
        if(beep){
            BitWrPortI ( PEDR, &PEDRShadow, 1,7 );   // Habilita buzzer c/osc
            waitfor(DelayMs(BEEP_TIME));             // espera tiempo de beep
            BitWrPortI ( PEDR, &PEDRShadow, 0,7 );   // Inhabilita buzzer
               beep=0;
        }
    }
    costate{
        http_handler();                              // maneja TCP/IP
        if(sending)
            if (smtp_mailtick()!=SMTP_PENDING)       // manda mail, si ya terminó
                sending=0;                           // lo indica
        if(fsending)
            if((ftpret=ftp_client_tick())!= FTPC_AGAIN) { // FTP, si terminó
                fsending=0;                          // lo indica
                if(ftpret==FTPC_OK)                  // llegó bien ?
                records=0;                           // borra registros enviados
            }
        }
    costate {                                        // muestra hora en display
        do {
            yield;                                   // cede procesador
            waitfor(DelaySec(1));                    // espera 1 segundo
            get_datetime();                          // actualiza info fecha y hora
        } while((SEC_TIMER-timer)<HOLD_TIME);        // mientras hay algo que mostrar
        LCD_clear();                                 // sino, borra display
        LCD_at(0,4);                                 // muestra fecha y hora
        LCD_show("Cika Time Logger");
        LCD_at(1,1);
        LCD_show(date);
        LCD_at(1,15);
        LCD_show(time);
    }
    costate {                                        // procesa registro RFID
        wfd i=cof_serDread(RFID,14,100);     // cede procesador hasta que hay datos
        if(i==14){                                   // si recibió 14 bytes
            if((RFID[0]==2)&&(RFID[13]==3)){         // y el registro es válido
                decode(RFID);                        // lo decodifica
                beep=1;                              // emite beep
                LCD_clear();                         // borra display
                switch (sstate){
                    case ESCRACHATE:                 // en funcionamiento normal
                        if((i=lookup(RFID))!= -1){   // lo busca en la base, si existe
                            LCD_at(0,1);             // lo muestra
                            LCD_show(time);          // junto con fecha y hora
                            LCD_at(0,12);
                            LCD_show(date);
                            LCD_at(1,2);
                            LCD_show(rfid_base[i].name);
                            time_base[records].id=i;
                            time_base[records].time=SEC_TIMER; // mantiene un tiempo
                            if(records<NUM_RECORDS)  // sobreescribe último registro
                               records++;            // si no hay espacio
                        }
                        else {                       // si el registro NO es válido
                            LCD_at(0,5);             // muestra un mensaje
                            LCD_show(wrong);         // (RFID desconocido)
                            LCD_at(1,2);
                            LCD_show("no figura en la base");
                        }
                        break;
                    case LEARN:                      // en modo “aprender”
                        if((i=lookup(RFID))!= -1){   // si existe en la base
                            LCD_at(0,5);             // lo rechaza y lo indica
                            LCD_show(wrong);
                            LCD_at(1,2);
                            LCD_show("ya figura en la base");
                        }
                        else {                       // sino,
                            LCD_at(0,2);
                            LCD_show("Ingresado a nombre de"); // lo ingresa
```

```
                                LCD_at(1,2);              // y muestra en pantalla
                                strcpy(rfid_base[entries].name,FORMSpec[0].value);
                                memcpy(rfid_base[entries].rfid,RFID,5);
                                LCD_show(rfid_base[entries++].name);
                            }
                            sstate=ESCRACHATE;            // vuelve al estado normal
                            break;
                        default:
                            break;                        // ignora estados no válidos
                    }
                    timer=SEC_TIMER;              // mantiene info en display por un tiempo
                }
            }             // ignora info inválida por el port serie
        }
        costate {         // tiempos de reportes
            waitfor(!strncmp(time,mail_time,5));          // espera a que sea el momento
            if((!sending)&&records){                      // si hay algo que reportar
                linenum=0;                                // inicializa número de línea
                smtp_setserver(smtp_server);              // a quien mandar el mail
                smtp_sendmail(to, from, subject, NULL);  // mensaje dinámico
                smtp_data_handler(mail_generator, &linenum, 0);  // función que genera msg
                sending=1;                                // habilita envío
                waitfor(DelaySec(65));                    // evita repetición
            }
        }
        costate {
            waitfor(!strncmp(time,ftp_time,5));           // espera a que sea el momento
            if((!fsending)&&records){                     // si hay algo que enviar
                gen_filename(my_filename,filename);       // genera nombre del reporte
                if(!ftp_client_setup(resolve(ftp_server),0,user,password,
                    FTP_MODE_UPLOAD,my_filename,NULL,NULL,0)) {  // conecta con server
                    ftp_data_handler(ftp_datahandler, NULL, 0);  // función que genera rpt
                    fsending=1;                           // habilita envío
                    waitfor(DelaySec(65));                // evita repetición
                }
            }
        }
    }
    serDclose();

}
```

La función *costate* es la que nos define cada tarea. La función *waitfor* espera que su parámetro evalúe como cierto (true), caso contrario devuelve el control a las otras tareas. Si el parámetro es *DelaySec(n)*, el resultado concreto es que a cada paso por esa instrucción, se devuelve el control a las demás tareas hasta tanto hayan transcurrido *n* segundos, momento en el cual se ejecuta el bloque a continuación y se reevalúa el loop. La función *yield* cede el control a las demás tareas, y cuando el *costate* resume su ejecución, la misma continúa desde la sentencia siguiente.

Para leer el port serie, dado que necesitamos que el equipo siga funcionando, atendiendo el servidor web y otras tareas, empleamos una llamada a *cofunction*. Una *cofunction* es llamada dentro de un *costate* y se emplea para realizar tareas que tienen un principio y un fin, en este caso *wfd i=cof_serDread(RFID,14,100)* es equivalente a *waitfordone {i=cof_serDread(RFID,14,100)}* y llama a la *cofunction cof_serDread()*, la cual cederá el procesador hasta tanto termine, lo cual sucederá cuando reciba 14 caracteres, que almacenará en RFID, o hayan transcurrido 100ms sin que reciba ningún caracter luego de empezada la recepción de un paquete.

Hasta aquí todo lo relacionado con el código en sí. Veremos ahora la parte SHTML que será procesada por el servidor HTTP (provisto por Dynamic C), sin intervención de nuestra parte.

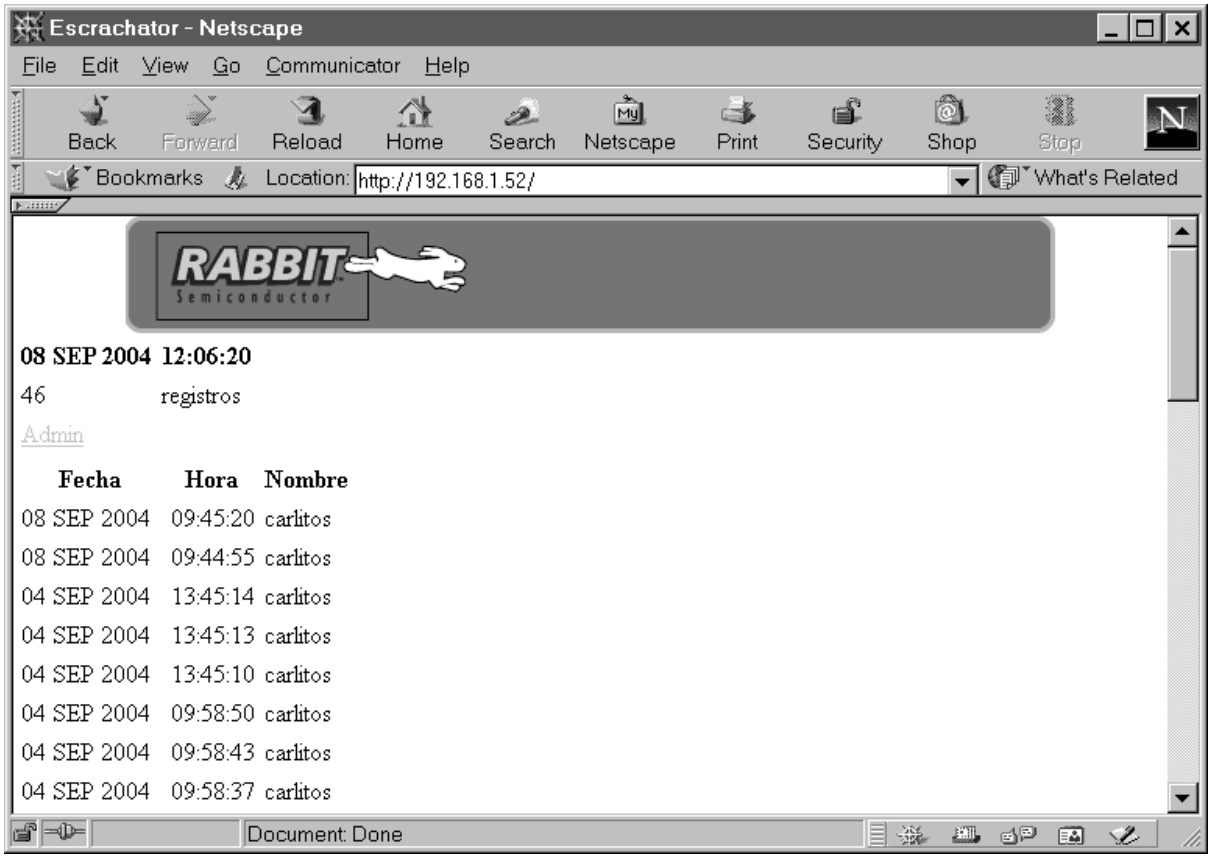
Veamos *index.shtml*, este archivo es SHTML e informa al servidor qué variables debe incluir para que la página se muestre actualizada cada vez que se la solicita

```
<html>
<head><title>Escrachator</title></head>
<body   bgcolor="#FFFFFF"   link="#009966"   vlink="#FFCC00"   alink="#006666"   topmargin="0"
leftmargin="0" marginwidth="0" marginheight="0">
<center><img SRC="rabbit1.gif" ></center>
<table>
```

```
<tr>
<th><!--#echo var="date"--><th><!--#echo var="time"-->
<tr>
<td><!--#echo var="records"--><td>registros
<tr><td>
<A HREF="admin.shtml">Admin</A><td>
</table>
<table>
<th>Fecha<th>Hora<th>Nombre
<!--#exec cmd="show"-->
</table>
</body>
</html>
```

El servidor detecta `<!--#echo var="date"-->` y lo reemplaza por el contenido de la variable *`date`*. De igual modo, detecta `<!--#exec cmd="show"-->` y ejecuta la función *`show`*, como dijéramos anteriormente.

Esto se ve así:



A continuación , veremos una de las páginas de administración: *learn.shtml*

```
<html>
<head><title>Learn ID</title></head>
<body bgcolor="#FFFFFF" link="#009966" vlink="#FFCC00" alink="#006666" topmargin="0"
leftmargin="0" marginwidth="0" marginheight="0">
<center><img SRC="rabbit1.gif" ></center>
<form ACTION="learn.cgi" METHOD="POST">
<table>
<tr>
<th><!--#echo var="date"--><th><!--#echo var="time"-->
<tr>
<td WIDTH="80%"><input TYPE="TEXT" NAME="nombre" SIZE=20>
</table>
<input TYPE="SUBMIT" VALUE="Ingresar"></form>
</body>
</html>
```

Cuando el usuario presiona el botón “Ingresar”, el web browser indica al servidor HTTP en el Rabbit que ejecute la función *learn.cgi*, tal como se indica en *form action=learn.cgi*.

Esto se ve así:

Por una cuestión de espacio, no desarrollaremos las demás páginas SHTML, que son más o menos equivalentes, excepto por el contenido de FORMs. El lector puede consultar el código fuente para analizarlas.