Cika Electrónica

Nota de Aplicación: CAN-043
Título: **Pizarra remota via HTTP con Rabbit 3000 y LCD gráfico (HD61202)**
Autor: Sergio R. Caprile, Senior Engineer

| Revisiones | Fecha | Comentarios |
|---|---|---|
| 0 | 12/10/05 | |
| | | |
| | | |

Presentamos en esta oportunidad una nueva encarnación de la conocida "pizarra remota", una simple aplicación de un módulo LCD gráfico inteligente y un módulo Rabbit conectado a una red Ethernet. Esta vez, agregamos la posibilidad de escribirla tanto vía HTTP como por email o interfaz serie/USB. El módulo Rabbit funciona como un servidor web y con cualquier navegador podemos escribir un simple mensaje en la pizarra; además, periódicamente revisa el correo en un servidor y si ve un mensaje muestra la primera línea en la pizarra; y por último, al recibir un retorno de carro (Enter) por un puerto serie, presenta un prompt que permite al usuario ingresar una línea de texto a ser mostrada en la pizarra. Empleando el conversor serie-USB descripto en la CAN-025, es posible además acceder a la pizarra vía USB.

Más allá de su posible utilidad en la vida real, esta aplicación nos permite ejemplificar el uso de las capacidades de TCP/IP de Dynamic C. No se darán demasiados detalles acerca del display y su software de control, dado que este tema se ha desarrollado ya en varias notas de aplicación. El lector interesado puede remitirse a dichas notas de aplicación para mayor información.

**Hardware**

El hardware de display puede ser cualquiera de las notas de aplicación que permiten controlar un display basado en HD61202. En nuestro caso (software adjunto) utilizamos el desarrollo de la CAN-042, pero reemplazando los drivers es posible hacerlo funcionar con el hardware de la CAN-004, por ejemplo.
Dado que elegimos realizar este proyecto en un kit RCM3720, para la interfaz USB empleamos el port serie E y el desarrollo de la CAN-025. Si bien el Rabbit 3000 funciona a 3,3V; es 5V-tolerant y el kit posee alimentación de 5V. Para portar este proyecto a R2000, simplemente utilizar cualquiera de los ports serie del R2000 en vez del port E que utilizamos aquí.

**Descripción del funcionamiento**

Nuestra pizarra será un display gráfico de 128x64, que mostrará texto en 8 filas de 20 caracteres. La fila superior mostrará la fecha y hora, y los mensajes ingresarán por la fila inferior, desplazando el texto anterior hacia arriba.

Para ingresar un mensaje, el usuario puede:

- Conectarse con su navegador preferido a nuestra dirección IP, donde verá una planilla en la que ingresará el texto a mostrar. El usuario puede ver además la fecha y hora y la cantidad de mensajes enviados. Esto se realiza mediante "server side includes" (SHTML). La planilla es un HTML FORM, en modo POST. Al remitirla (submit), el usuario hace que el módulo Rabbit ejecute una función (CGI), cuyos parámetros son provistos por el navegador y su resultado es la actualización del display; al mismo tiempo que se le mostrará en el navegador una segunda pantalla, confirmando la acción.
- Enviar un email a una dirección de mail válida. El módulo Rabbit verificará periódicamente la existencia de correo en esa dirección, y de encontrarlo lo traerá e imprimirá los primeros 20 caracteres de la primera línea del mensaje.
- Conectar el puerto serie o USB de una laptop o equivalente, correr algún programa de emulación de terminal asincrónica, enviar un retorno de carro, y al observar el prompt ingresar el mensaje. El usuario puede ver además la fecha y hora y la cantidad de mensajes enviados. Para conectar un puerto serie, se deberán conectar conversores de nivel a RS-232 como el HIN232, comercializado por Cika. Para conectar un puerto USB, puede utilizarse el conversor desarrollado en la CAN-025.

En todos los casos, la información de fecha y hora se toma directamente del timer provisto por Dynamic C, se asume que el usuario ya hizo la "puesta en hora" del sistema mediante alguno de los ejemplos provistos por

Dynamic C. No es necesario reajustar fecha y hora a menos que se apague el módulo y no se disponga de battery backup.

**Software**

Desarrollaremos a continuación el software de la pizarra remota. Dado que, como dijéramos, se trata de una nueva encarnación de un viejo proyecto, comentaremos sólo aquellas partes que revistan alguna novedad o modificación. El lector interesado puede remitirse a la CAN-004, que contiene la descripción del código original.

La primera modificación a observar es en la función que se ejecuta al enviar el formulario, *submit()*, como recordarán, la página principal tiene un form que al ser ingresado envía un HTTP request de tipo POST a una función CGI, que al ejecutarse extrae los datos que se ingresaron al llenar la forma (el string a mostrar en el display). Dado que ahora tenemos varias formas de ingresar un mensaje, la función *submit()* se limitará a realizar el procesamiento que le corresponde, y llamará a otra función para presentar el mensaje en pantalla. Las otras alternativas de ingreso de mensajes harán uso de esta misma función: *putmsg()* para presentar los suyos.

```
int msg;                        // cuenta número de mensajes

void putmsg(char *msgptr)
{
        msg++;                                          // incrementa contador de msgs
        LCD_scroll();                                   // desplaza msgs anteriores hacia arriba
        LCD_printat(7,0,msgptr);                        // muestra nuevo mensaje
}

#define MAX_SENTENCE   20
#define MAX_FORMSIZE   64
#define REDIRECTHOST   MY_IP_ADDRESS
#define REDIRECTTO     "http://" REDIRECTHOST "/ok.html"


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

        FORMSpec[0].value[0] = 0;
        // init form data
        if(parse_post(state)) {
                if(*(http_urldecode(buffer,FORMSpec[0].value,MAX_FORMSIZE))) {
                        buffer[MAX_SENTENCE]=0;         // recorta a tamaño del display
                        putmsg(buffer);
                }
                cgi_redirectto(state,REDIRECTTO);       // muestra OK
        } else {
                // manejo de errores si se considera conveniente o necesario
        }
        return(0);
}
```

Dado que debemos entregar un prompt por el port serie/USB, es conveniente escribir una pequeña cofunction que lo haga, de modo de permitir el desarrollo normal de las otras tareas, la cual será llamada desde el costate que atienda al port serie/USB:

```
cofunc void putprompt(char *prompt)
{
        sprintf(prompt,"\r\n%s %s\tIngrese MSG%d\r\n>",date,time,msg);
        wfd cof_serEwrite(prompt, strlen(prompt));
}
```

Para recibir emails, dado que se trata de una cuenta fija, podemos precompilar los datos correspondientes al servidor, nombre de usuario y password. Debemos además indicar un DNS para que pueda resolver el nombre del servidor, tarea la cual se realiza mediante un llamado a la función *resolve()*. Esto se realiza definiendo algunas macros antes de incluir las bibliotecas de funciones de TCP/IP y POP:

```
#define MY_NAMESERVER  "192.168.1.1"
```

```
#define POP_HOST       "maihost.maidomein"
#define POP_USER       "maiiuser"
#define POP_PASS       "maipasguord"

#define POP_PARSE_EXTRA

#memmap xmem
#use "dcrtcp.lib"
#use "http.lib"
#use "pop3.lib"
```

La macro *POP_PARSE_EXTRA* hace que el data handler que procesa el email recibido sea llamado con más parámetros, entre los cuales está desglosado el encabezado del email. De esta forma, cuando la biblioteca de funciones de POP detecta la existencia de un mensaje en el servidor, llama a nuestra función para que lo procese. La función que escribimos descarta el encabezado del mail y toma solamente los primeros caracteres del mensaje (*MAX_SENTENCE*)

```
int n;                        // indicador para detectar la primera línea del mensaje
char mmsg[MAX_SENTENCE+2];    // almacena el cuerpo útil del email antes de publicarlo

int mailmsg(int num, char *to, char *from, char *subject, char *body, int len)
{
static int i;

#GLOBAL_INIT {
               n = -1;
}
                                              // toma sólo la primera línea de cada mensaje
       if(n!=num){
               n=num;
               i=(len< MAX_SENTENCE)? len:MAX_SENTENCE; // recorta al tamaño del display
               strncpy(mmsg,body,i);
               mmsg[i]=0;
               putmsg(mmsg);
       }
       return 0;
}
```

El manejo del puerto serie lo realizamos de forma similar a lo estudiado en la CAN-019: una tarea se ocupa de ver si se excede un tiempo máximo para ingreso del mensaje, más allá del cual se lo descarta (se purga el buffer). Mientras tanto, otra tarea atiende la interactividad de la interfaz de la siguiente forma. Como sabemos, es muy práctico escribir las tareas en forma de costates, dentro del programa principal:

```
unsigned long t;
int input_char, string_pos;
char replydata[128], sentence[MAX_SENTENCE+2];

       costate {
               t = MS_TIMER;
               wfd input_char = cof_serEgetc();
               if(input_char == '\r' || input_char == '\n') {
                       if(string_pos){                       // procesa el paquete
                               sentence[string_pos]=0;       // termina string
                               sprintf(replydata,"\r\nOK\r\nMSG%d: %s\r\n",msg,sentence);
                               wfd cof_serEwrite(replydata, strlen(replydata)); // contesta
                               putmsg(sentence);             // muestra msg
                               string_pos = 0;               // limpia buffer
                       }
                       else wfd putprompt(replydata);        // msg nulo, devuelve prompt
               }
               else {
                       sentence[string_pos] = input_char;           // almacena
                       wfd cof_serEwrite(&sentence[string_pos],1);  // eco
                       string_pos++;
                       if(string_pos == MAX_SENTENCE){ // acción si excede tamaño de buffer
                               sentence[string_pos]=0;       // termina string
                               string_pos=0;                 // descarta
                               sprintf(replydata,"\r\n>> Buffer excedido: %s\r\n",sentence);
                               wfd cof_serEwrite(replydata, strlen(replydata)); // mensaje
                               wfd putprompt(replydata);     // prompt
                     }
```

```
		}
	}
	costate {                                              // Rx timeout
		if (MS_TIMER > t + IDLE_TMOUT){
			t = MS_TIMER;
			string_pos=0;
			sentence[string_pos]=0;
		}
	}
```

Finalmente, una última tarea se encarga de revisar el correo cada 2 minutos, y recibirlo.:

```
	costate {
		waitfor(DelaySec(120));                              // espera 2 minutos
		pop3_getmail(POP_USER, POP_PASS, resolve(POP_HOST)); // inicia recepción
		while(pop3_tick()== POP_PENDING)                     // recibe correo (si hay)
			yield;
		if(n>=0)
			n=-1;
	}
```

El único inconveniente es que la función *resolve()* es bloqueante, y las demás tareas no se ejecutarán mientras se está resolviendo el nombre, lo cual generalmente no demora nada en condiciones normales. Si esto es un problema, el usuario puede precompilar la IP (numérica), resolver al inicio de la aplicación, o si es necesario resolver más a menudo deberá escribir código no bloqueante para resolver el nombre, llamando a funciones como *resolve_name_start()* y *resolve_name_check()*, lo cual, si bien no es demasiado complejo, escapa a los alcances de esta nota de aplicación.

Resumido y genérico, el programa principal, donde se integra todo, sería algo más o menos así:

```
main()
{
int i;
unsigned long t;
int input_char, string_pos;
char replydata[128], sentence[MAX_SENTENCE+2];

	msg=1;
	string_pos = 0;
	FORMSpec[0].name = "texto";

	pop3_init(mailmsg);
	brdInit();
	sock_init();
	http_init();
	tcp_reserveport(80);

	serEopen(9600);

	loopinit();

	LCD_init();
	LCD_clear();

	while(1){
		http_handler();                      // TCP/IP + página web
		loophead();
		costate{
			// port serie/USB
		}
		costate{
			// obtención de email
		}
		costate {
			waitfor(DelaySec(1));        // espera 1 segundo
			get_datetime();              // actualiza fecha y hora
			LCD_printat(0,0,date);       // lo muestra
			LCD_printat(0,12,time);
		}
	}
}
```