6 Diseño del Software

Según se desprende de secciones anteriores, disponemos de tres dispositivos susceptibles de ser programados según se muestra en la Figura 6.1, a saber: el módem GSM Xircom Eagle II, el PLC Siemens S7-200 y el microcontrolador ATmega128L de Atmel. De ellos, el que presenta una programación más simple es el módem, ya que requiere una configuración mínima y posteriormente es tratado como un periférico de la tarjeta que alberga al microcontrolador que simplemente envía y recibe caracteres según la norma RS-232 factibles de ser interpretados como comandos AT+ según se describe en el epígrafe 4.3. La programación de los dispositivos es tal que independientemente del orden de inicialización de los distintos componentes, el sistema siempre alcanza el punto adecuado para comenzar su funcionamiento cíclico.

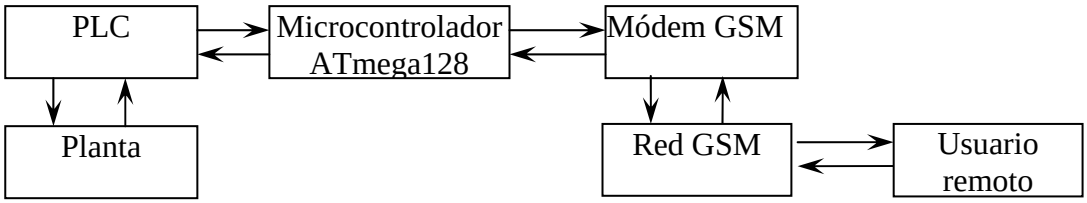

Figura 6.1 ***Esquema del sistema***

6.1 Configuración y programación del módem

Previa a su inclusión definitiva en el diseño final, se utilizó el módem conectado a un PC para probar sus funcionalidades, comportamiento e interacciones con la red GSM. De dicho estudio se obtuvieron las siguientes conclusiones:

- Es preciso indicar en algún momento al módem la dirección del centro servidor de mensajes, esta dirección corresponde con un número de telefono que variará de una red a otra. Su configuración se hará utilizando la orden `"AT+CSCA=\"XXX … X\"\r"`, donde el carácter '\"' es la comilla doble en código ASCII, '\r' es el carácter ASCII nueva línea (LF – *Line Feed*) y XX … X representa la dirección de dicho centro servidor provisto por el operador de red al que el módulo SIM esta abonado. Dicha dirección queda almadenada en la memoria interna del módem de tal modo que sólo es necesario indicarlo una vez (salvo cambio de operador), es por ello, que pues le fue al módem en la fase de prueba, dicho comando no aparece en el código final del microcontrolador por no ser necesaria su inclusión.

- Tras su puesta en marcha, el módem intenta encontrar una red válida, devolviendo el texto "`+CREG: 3`" cuando lo hace.

- Este es el momento de introducir el código PIN correspondiente a la SIM que alberga el módem (no incluida) con el comando AT+ "`AT+CPIN=\"****\"\r`", donde el asterisco representa un dígito decimal para formar una secuencia de cuatro dígitos típica para suscribirse a la red GSM. La respuesta del módem será: "`+CREG: 1<LF>OK<LF>`" dónde <LF>

representa al carácter nueva línea, o “ERROR” en caso de no poder completarse el proceso.

- A continuación indicamos al módem cómo gestionar los mensajes entrantes con la orden “AT+CNMI=1,0,0,0,0\r”, lo cual indicará al módem que no debe hacer ninguna indicación cuando reciba un nuevo mensaje entrante. El porque de esta configuración radica en que la aleatoriedad de la llegada de los mensajes puede provocar que dicha indicación coincida con un instante en que el microcontrolador espere una repuesta del módem y le llegue esta indicación en lugar de la respuesta con el prácticamente seguro error consecuente.

- El uso estándar del Terminal consistirá en la lectura de mensajes recibidos “AT+CMGL”, envío de nuevos mensajes “AT+CMGS” y borrado de los mismos “AT+CMGD” que se consideran suficientemente documentados en la sección 4.3 por lo que no se repetirán aquí, al igual que “AT+CPWROFF”, detallado en la sección 5.2.

- Se consideraron pasos críticos la búsqueda de red e inicialización del módem en general y el envío de un SMS, pues son casos en los que el servicio es Short Message Mobile Originated (SM MO) y la respuesta depende del estado de la red. En estos casos, en el supuesto de no recibir la señal con suficiente potencia (falla la cobertura en lenguaje coloquial), la respuesta será “ERROR” y el sistema debe ser capaz de gestionar dicho error. La solución más simple y la que se ha implementado es insistir en la petición hasta obtener respuesta afirmativa. Existen otras causas de error como un mal deletreo de una petición, pero las consideraremos irrelevantes en nuestro sistema.

6.2 Programación del PLC

6.2.1 Descripción general del programa

El PLC es el sistema a monitorizar por lo que su funcionamiento básico será describir su estado y enviarlo regularmente al microcontrolador para que, llegado el caso, éste pueda responder una eventual petición del usuario remoto. Así pues, dentro del bloque general OB1 instalamos un temporizador que cada 10 segundos realiza la llamada a una función que envía por el puerto serie del que dispone el autómata un número fijo de caracteres que codifican en ASCII el estado de las entradas y salidas del autómata. Para ello, las entradas y salidas digitales se agrupan en bloques de ocho para formar un Byte y ser enviadas como un carácter, al disponer en el momento de las pruebas de 14 entradas y 10 salidas, necesitaremos dos caracteres para las entradas y dos para las salidas. Los módulos de E/S analógicos disponen de 4 entradas o 2 salidas de 12 bits por lo que necesitaremos dos caracteres por entrada o salida a controlar, 8 caracteres en total por módulo de entrada o 4 por módulo de salida. Se dispone de 1 módulo de cada tipo por lo que en total se necesitaran 16 caracteres.

El bloque del autómata que encargado de las comunicaciones permanece por defecto en un estado de Standby y, sólo sale de ese estado cuando recibe un primer

carácter a enviar. Según se observa en la sección 2.6.1 podemos configurar al autómata para que cualquier carácter cumpla dicha misión o sea uno en concreto el que active al transmisor, obviándose a los precedentes, en este caso en particular se optó por la primera opción.

Con las premisas anteriores se creó un formato de trama que fuese lo más simple posible, sin ambigüedades y que albergase toda la información necesaria, el resultado es el siguiente:

Inic	Flag	Tam	Data	Data	…	Data

Donde Inic representa al carácter de inicialización del transmisor, en la codificación se escogió ‘z’ pero podría haber sido cualquier otro. Como Flag de inicialización de trama se escogió el carácter ‘I’, que indicará al receptor el comienzo de la misma. El tercer carácter será aquel cuyo código ASCII coincida con el número de caracteres útiles, es decir, aquellos que representan a entradas o salidas y, a continuación aparecerán dichos caracteres. El búfer de comunicaciones es capaz de albergar un máximo de 256 caracteres, coincidiendo con el número máximo de marcas internas de que dispone el PLC. En cualquier caso, una CPU 224 como la utilizada tiene capacidad para gestionar un máximo de 64 entradas y salidas digitales y 16 entradas y salidas analógicas. Ello suma un total de 80 caracteres, a los que sumando los tres caracteres de cabecera de la trama creada, alcanza los 83 caracteres, asumibles en todo momento tanto por el búfer del transmisor como por el formato de la trama SMS.

Finalmente será necesario desconectar el transmisor, esta tarea se realiza automáticamente por el PLC cuando se detecta una condición de fin de trama. Estas condiciones son múltiples (consultar la Tabla 2.4), pero dada la estructura de nuestra trama, resulta particularmente útil detectar el fin de trama por haber alcanzado un número determinado de caracteres enviados al búfer, que coincidirá con Tam + 2.

Asociada al evento de transmisión finalizada, habilitamos una rutina de interrupción que esperará hasta recibir un carácter, si dicho carácter es ‘a’ indicará que el usuario desea iniciar un proceso de parada de emergencia. La condición de fin de trama en este caso se alcanza por agotar un temporizador de 10 ms lanzado al recibir el primer carácter, al recibirse sólo uno, tras estos 10 ms, el receptor se desconecta.

Se detectó no obstante una discrepancia, el juego extendido de caracteres ASCII va desde 1 hasta 256, lo que impediría, por ejemplo, enviar ocho entradas o salidas digitales consecutivas a cero. La solución es clara, es necesario enviar la codificación ASCII resultante de las E/S consideradas más uno. Se dejará al receptor la labor de restar uno para volver al sistema original.

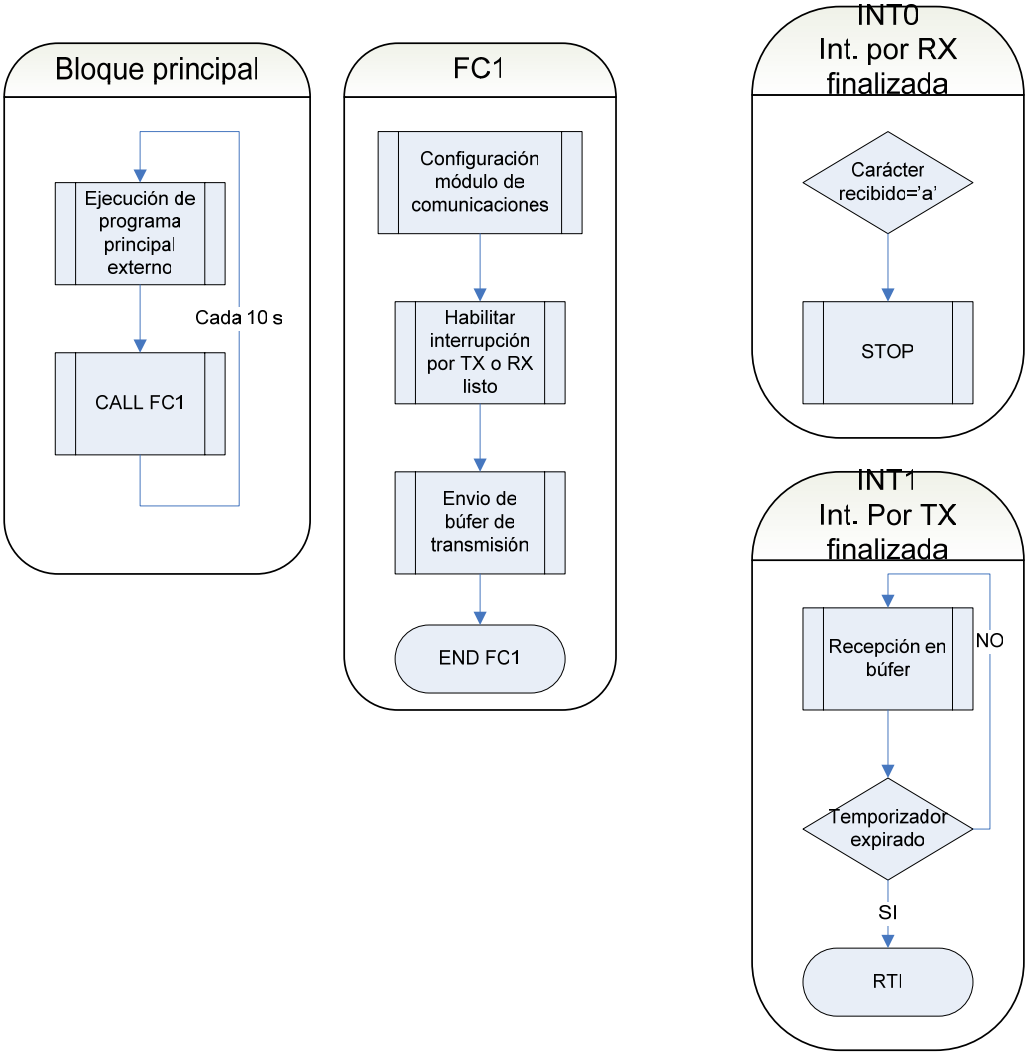

Figura 6.2 ***Diagrama de fujo del programa cargado en el autómata***

6.2.2 Código fuente del PLC

Se muestra a continuación el código fuente desarrollado para el PLC comentado, se mostrará en el lenguaje AWL por ser el lenguaje en que se desarrollo el código original. El lenguaje de contactos KOP no es adecuado para elaborar un código de esta complejidad pues no cuenta con instrucciones adecuadas para algunas de las operaciones.

Bloque OB1

```
1
2 NETWORK 1
3 LDN T39                  //Mientras el temporizador T39 = 0
4 = M0.1                   //Activa la marca M0.1
5
6 NETWORK 2
7 LD M0.1                  //Mientras la marca M0.1 esté a 1
8 TON T39, +100            //Lanza el temporizador T39 con
```

```
9                       //valor de cuenta de 10 segundos
10 NETWORK 3
11 LD T39               //Si T39 alcanza el valor final
12 S M0.2, 1            //Activa la marca M0.2
13
14 NETWORK 4
15 LD M0.2              //Mientras la marca M0.2 esté a 1
16 CALL SBR_0           //Llama a la subrutina 0
```

Bloque SBR0

```
1 // SUBRUTINA DE INICIALIZACIÓN DEL PUERTO DE
2 // COMUNICACIONES Y DE TRANSMISIÓN DEL ESTADO DEL PLC
3 //
4 //
5
6 NETWORK 1
7
8 NETWORK 2
9 LDN T39
10 MOVB 16#05, SMB30 //19.2kbps, 8 bits, sin paridad, freeport
11 MOVB 16#94, SMB87     //RCV on
12 MOVW +5, SMW90        //inactividad tras 5 ms
13 MOVW +1000, SMW92     //RX OFF si 50ms sin actividad
14 MOVB 18, SMB94        //máx de 6 caracteres por mensaje
15 ATCH INT_0, 23        //interrupción por RX finalizada
16 ATCH INT_1, 9         //interrupción por TX finalizada
17 ENI                   //habilitación global de interrup.
18
19 S Q0.2, 1             //Activa las salidas Q0.2-Q0.7
20 S Q0.3, 1
21 S Q0.4, 1
22 S Q0.5, 1
23 S Q0.6, 1
24 S Q0.7, 1
25 S Q1.0, 1             //Activa las salidas Q1.0 y Q1.1
26 S Q1.1, 1
27
28
29 MOVB 'z', VB99        //Carácter para inicializar el TX
30 MOVB 'I', VB100       //Flag de inicialización de trama
31 MOVB 16, VB101        //Tam = 16
40 MOVW IW0, VW102       //Carga las entradas digitales
41 MOVW QW0, VW104       //Carga las salidas digitales
42 //MOVB 'A', VB102
43 //MOVB 'B', VB103     //Sólo para la simulación
44 //MOVB 'C', VB104     //de entradas y salidas digitales
45 //MOVB 'D', VB105
46 INCB VB102
47 INCB VB103            //Incrementar las E/S digitales
```

```
48 INCB VB104           //Para evitar enviar cero
49 INCB VB105
50 //MOVB '1', VB106
51 //MOVB '2', VB107
52 //MOVB '3', VB108     //Sólo para la simulación de
53 //MOVB '4', VB109     //un bloque de entradas analógicas
54 //MOVB '5', VB110
55 //MOVB '6', VB111
56 //MOVB '7', VB112
57 //MOVB '8', VB113
58 MOVW AIW0, VW106
59 MOVW AIW2, VW108      //Carga de un bloque de entradas
60 MOVW AIW4, VW110      //analógicas
61 MOVW AIW6, VW112
62 //MOVB 'A', VB114
63 //MOVB 'B', VB115     //Sólo para la simulación de un
64 //MOVB 'C', VB116     //bloque de salidas analógicas
65 //MOVB 'D', VB117
66 //Introducir las salidas como marcas en el bloque Princ.
67 XMT VB99, 0           //TX habilitado con búfer en VB99
68
69 NETWORK 3
70 LD T40               //Si el temporizador T40 = 1
71 R M0.2, 1            //desactiva la marca M0.2
72
73 NETWORK 4
74 LDN T40              //Mientras el temporizador T40 = 0
75 = M0.3               //activa la marca M0.3
76
77 NETWORK 5
78 LD M0.3              //Mientras la marca M0.3 esté a 1
79 TON T40, +1          //lanza el temporizador T40 con
80                      //valor de cuenta 10 ms.
```

INT0

```
1 //
2 //RUTINA DE INTERRUPCIÓN POR RECEPCIÓN FINALIZADA
3 //
4 //
5
6 NETWORK 1
7 //
8 //
9 //
10 LDB= 'a', VB51       //Si el primer carácter recibido
11                      //es el carácter ‘a’
11 STOP                 //Pasa al PLC a modo STOP
```

INT1

```
1 //
2 //RUTINA DE INTERRUPCIÓN POR TRANSMISIÓN FINALIZADA
3 //
4 //
5
6
NETWORK 1
7 //
8 //
9 //
10 LDN T39                //Si el temporizador T39 = 0
11 RCV VB50, 0            //RX habilitado con búfer en VB50
```

6.3 Programación del microcontrolador

6.3.1 Descripción general del código

A grandes rasgos, y si nos centramos en la función main(), el código del programa cargado en el microcontrolador inicializa vía software el módem y el microcontrolador con los parámetros adecuados para este sistema en concreto para entrar finalmente en el bucle principal del cual no se saldrá si no se provoca un reset hardware.

Dicha inicialización consiste en el caso del microcontrolador en la asignación de valores iniciales para los puertos de entrada/salida (poner los pines 0 y 1 del puerto A a cero lógico forzando que se iluminen los LEDs de la placa para indicar que el microcontrolador funciona correctamente, y poner el pin 7 del puerto D a cero lógico para permitir la alimentación del módem GSM). Además de la tarea antes citada, destaca la inicialización de las UARTs con los valores adecuados, es decir, comunicación asíncrona, con 8 bits de datos, uno de stop y sin paridad a una velocidad de 9,600 bps para la UART0 y la misma configuración pero a 19,200 bps para la UART1, se puede comprobar que los valores introducidos en los registros de configuración de velocidad son los anticipados en el último párrafo de la sección 3.3.3.7. El resto de parámetros resultan menos interesantes para la aplicación a desarrollar.

Hemos de indicar que dado que la librería C estándar stdio.h sólo ofrece soporte para una UART, fue necesario diseñar sendas rutinas *getchar1* y *putchar1* que respectivamente leyesen o escribiesen un carácter en el búfer asociado a la UART1 similares a las funciones *getchar* y *putchar* contempladas en la citada librería estándar. Indiquemos del mismo modo que la UART0 se utilizará para las comunicaciones entre el microcontrolador y el módem y la UART1 para comunicar al microcontrolador con el PLC.

Si nos detenemos a observar la función que inicializa el módem veremos que comparte el espíritu del texto de la sección 6.1 en la página 116, si bien la función

comienza forzando el apagado del módem para estar seguros de este modo de cual es su estado en el momento de solicitar su inicialización. Dicho apagado consiste en el paso vía software del módem a Stand by para posteriormente proceder al corte del suministro eléctrico utilizando el interruptor controlado por el pin PORTD.7 según se ha descrito ya profusamente en apartados anteriores.

Si se observa el bucle principal, se aprecia que su armazón consta de cuatro pasos, el primero de los cuales consiste en el envío del carácter 'z' al PLC, con ello se habilita la rutina de interrupción por fin de recepción en el PLC, permitiéndose que se pueda lanzar el transmisor de nuevo, ya que si el puerto de comunicaciones del PLC esta siendo utilizado por el receptor, el transmisor queda en stand by por no disponer de funcionamiento full-duplex.

Continúan dos funciones, la primera lee el estado del PLC y, a continuación, la segunda comprueba si hay alguna petición del usuario remoto en el búfer del módem y la gestiona caso de ser cierto utilizando la codificación base64 en caso de ser necesario el envío de datos al usuario final.

Además de lo indicado, se comprueba si nos encontramos ante la primera lectura del estado del PLC tras un reinicio del sistema, pues en dicho caso tendremos que descartar la trama leída para asegurar la robustez del sistema. Puesto que puede darse el caso de que el comienzo de la lectura por parte del microcontrolador coincida con un instante de tiempo en que el PLC ha comenzado la transmisión de su estado y que, además, el carácter 'I' utilizado como flag de inicio forme parte del campo de datos. En este caso, el carácter siguiente, caso de existir, indicaría una errónea longitud del campo de datos y se produciría una inconsistencia que se evita descartando por completo la primera trama.

Antes hemos aludido a la codificación base64, ideada para homogeneizar los mensajes de correo electrónico enviados a través de SMTP. Dicha codificación consiste en una modificación sobre el código ASCII que toma únicamente 64 caracteres de dicho código: caracteres alfanuméricos además de '+' y '/', evitándose de este modo el envío de caracteres ASCII no imprimibles o que no sean ASCII estándar (recordemos que enviamos caracteres de 8 bits mientras que ASCII estándar cuenta sólo con 7 bits). De no utilizarse esta codificación muchos de los caracteres enviados no tendrían significado para el Terminal GSM del usuario remoto. Comprendida la situación anterior, es claro que si de los 8 bits enviados por carácter, sólo 6 (obsérvese que $2^6 = 64$) contienen datos útiles, no se producirán errores en la recepción del mensaje.

La firma de trabajar con la codificación base64 consistió en fragmentar el mensaje original en otros mensajes de tres caracteres como máximo con lo que tenemos un total de 3x8 = 24 bits. Si ahora este mensaje se codifica en cuatro caracteres base64, tenemos 4x6 = 24 bits, queda justificada por tanto la fragmentación en bloques de tres bytes. Si la longitud total del mensaje no es múltiplo de tres, se codificarán los caracteres posibles del último bloque y se rellenará con el carácter '=' según indica la norma hasta completar el bloque de cuatro. Para realizar la codificación de cada carácter, fragmentamos los caracteres de cada bloque en bits y se reensamblan tomados de tres en tres, el número que resulte se utiliza como índice de la tabla que se muestra a continuación:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

Así, el texto `ABC` se codifica como `QUJD` y el texto `ABCD` como `QUJDRA==`. Para este último caso:

```
ASCII     A (65)     B (66)     C (67)     D (68)   (Vacío) (Vacío)
         01000001   01000010   01000011   01000100

BASE64    16 (Q)  20 (U)  9 (J)   3 (D)    17 (R) 0 (A)  NA (=) NA (=)
          010000  010100  001001  000011   010001 000000 000000 000000
```

La función que lee el estado del PLC realiza una tarea simple, almacena en una cadena de caracteres definida globalmente el campo de datos de la trama enviada por parte del PLC y discierne si es la primera vez que ha sido ejecutada tras reset por los motivos antes expuestos.

La segunda y última de las funciones del bucle principal se encarga de dialogar con el cliente a través del módem, para ello, analiza los mensajes entrantes y segrega su contenido en dos cadenas, una que contendrá la cabecera del mensaje (remitente, fecha, hora y posición de almacenamiento dentro del módem) y otra que almacenará el cuerpo del mensaje. El remitente contenido en la primera cadena servirá para identificar a los usuarios válidos. Para ello, se comparará dicho remitente con sendos números de teléfono almacenados en la memoria EEPROM integrada en el microcontrolador.

Si bien la gestión de la memoria EEPROM no resultó una labor fácil, en el plano teórico resultaba muy interesante. Los datos almacenados en el microcontrolador pueden hacerse en memoria volátil o estática, si se utiliza el primero de estos almacenamientos para los teléfonos de los abonados autorizados y alguno de ellos decide cambiar de número, es posible modificar el código almacenado, pero al producirse un reset se pierde esta información. Si utilizamos memoria flash, si el usuario desea cambiar de número de teléfono es necesario reprogramar el microcontrolador para almacenar el nuevo número. En cambio, empleando memoria EEPROM, el número almacenado en origen puede ser modificado sin problemas un número casi ilimitado de veces con la seguridad de que los nuevos datos no se perderán tras un reinicio.

Una vez leído el nuevo mensaje entrante, se borra para evitar que se sature la memoria del módem y se comprueba si el remitente es un usuario válido según el método antes descrito.

En este momento, el microcontrolador está en disposición de analizar que petición le ha llegado por parte del cliente. Es obvio que un sistema de esta naturaleza ofrece un amplio abanico de posibilidades en cuanto a servicios que pueden ser ofrecidos. Para limitar las opciones e implementar un número más reducido de ellas, se tomaron tres que se consideraron a la par de máxima utilidad y significativas de las posibilidades globales ofrecidas. Para discernir el tipo de petición, el primer carácter de la misma será ‘1’, ‘2’ ó ‘3’ respectivamente. Indicaremos que las peticiones primera y tercera requieren de un campo de datos a continuación, mientras que la segunda no lo necesita. A continuación se detallará el significado de dichas peticiones.

La primera de ellas consiste en interrogar al microcontrolador sobre el estado del autómata. Puesto que en cada iteración del bucle principal el PLC comunica al microcontrolador el estado de sus entradas y salidas, sólo resta reenviarle dichos datos al usuario remoto dentro de un mensaje codificado en base64. Ya se demostró en la sección 6.2.1, página 117 que para el autómata empleado bastan 80 caracteres para codificar todas sus entradas y salidas. Esta cantidad, incluso aumentada por la utilización del la codificación base64 es fácilmente gestionable con un único mensaje. No obstante, es nuestra vocación obtener un sistema que con mínimos cambios pueda ser trasladable a cualquier autómata con capacidad para comunicarse sobre RS-232 y, puede darse el caso de que se instale el sistema sobre un PLC distinto con un número de entradas y salidas muy superior. Para conseguirlo se ha ideado un método que consiste en enviar el estado de todas las entradas y salidas digitales y sólo las entradas o salidas analógicas que solicite el usuario. Para ello, en el mensaje de petición, el usuario deberá incluir los números de entradas o salidas analógicas que desea consultar, de este modo, si el autómata tuviese 20 entradas analógicas y 20 salidas y el usuario sólo estuviese interesado en las entradas 5 y 17 y las salidas 1 y 8, enviará un mensaje que contiene el tipo de petición (consultar estado de planta) seguido de 5172128, es decir, las entradas y a continuación las salidas más 20 para que no se solapen con las entradas.

El segundo tipo de petición consiste en solicitar al autómata que pase a modo STOP. Se consideró demasiado arriesgado que el usuario pudiese modificar las consignas del autómata remotamente y por ello lo más que se le permite es detener el proceso bajo su responsabilidad en caso de que observe alguna anomalía. Para conseguirlo, lo que se hace es enviar el carácter ‘a’ en lugar del carácter ‘z’ para la reactivación del transmisor, provocando la detención del PLC según se detalló en el apartado 6.2.1. Finalmente se envía al usuario un mensaje de confirmación.

La última petición considerada permite al usuario modificar su clave de acceso. Para ello, tras el indicativo del tipo de petición se envía el nuevo teléfono que se desea dar de alta en sustitución del que actualmente se tiene. Supóngase que el usuario se equivoca al enviar el mensaje y da de alta un teléfono desconocido. Este caso justifica que no se envíe nunca mensaje de confirmación para evitar que el usuario de este teléfono anónimo sea consciente de que puede acceder al sistema. Para retomar el control por parte del usuario confundido, es necesario reprogramar el microcontrolador.

En la Figura 6.3 y la Figura 6.4 se muestran diagramas de flujo que pueden resultar interesantes para comprender las tareas que realiza el microcontrolador.

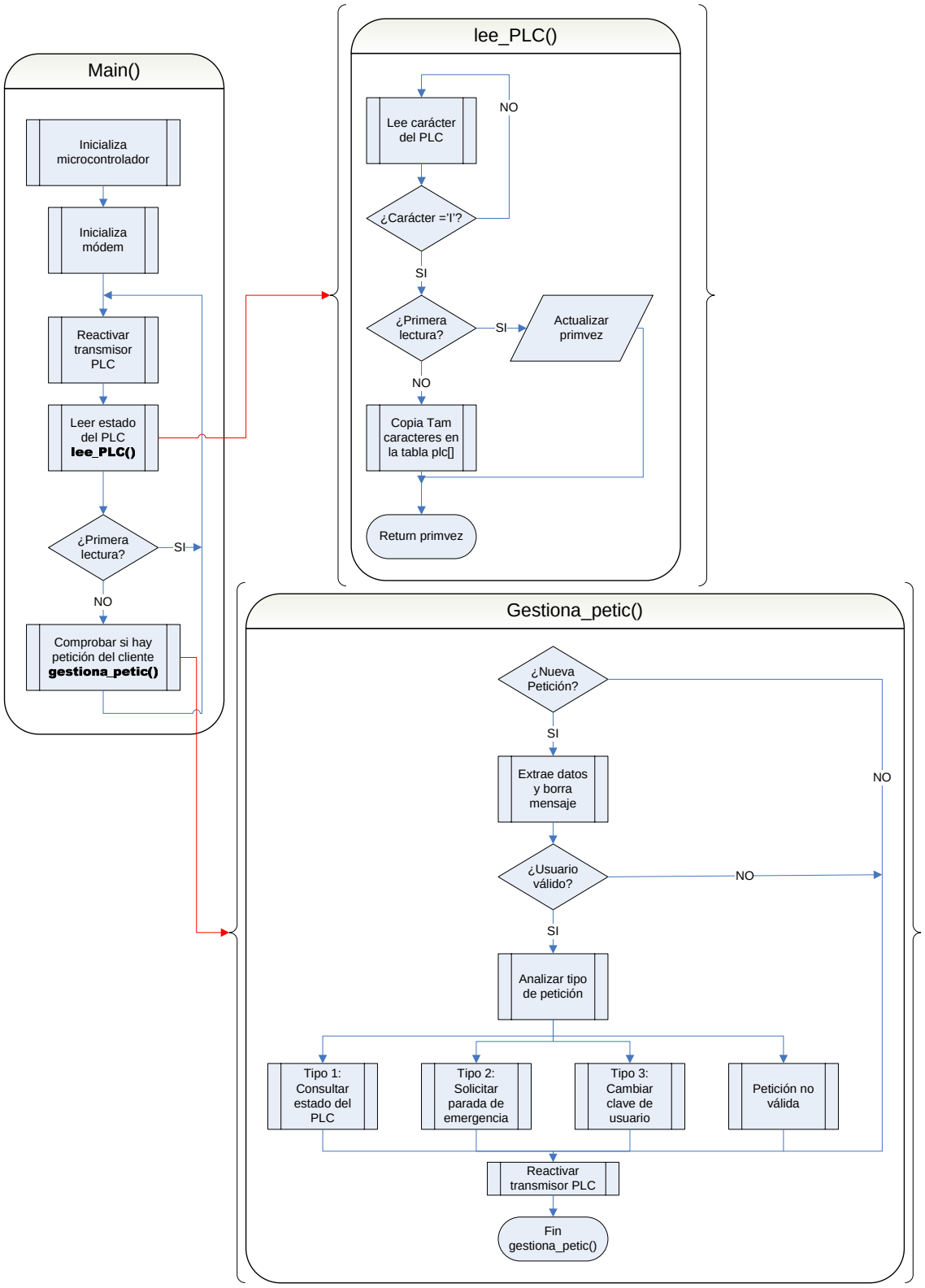

Figura 6.3 ***Diagrama de flujo del programa del microcontrolador (1/2)***

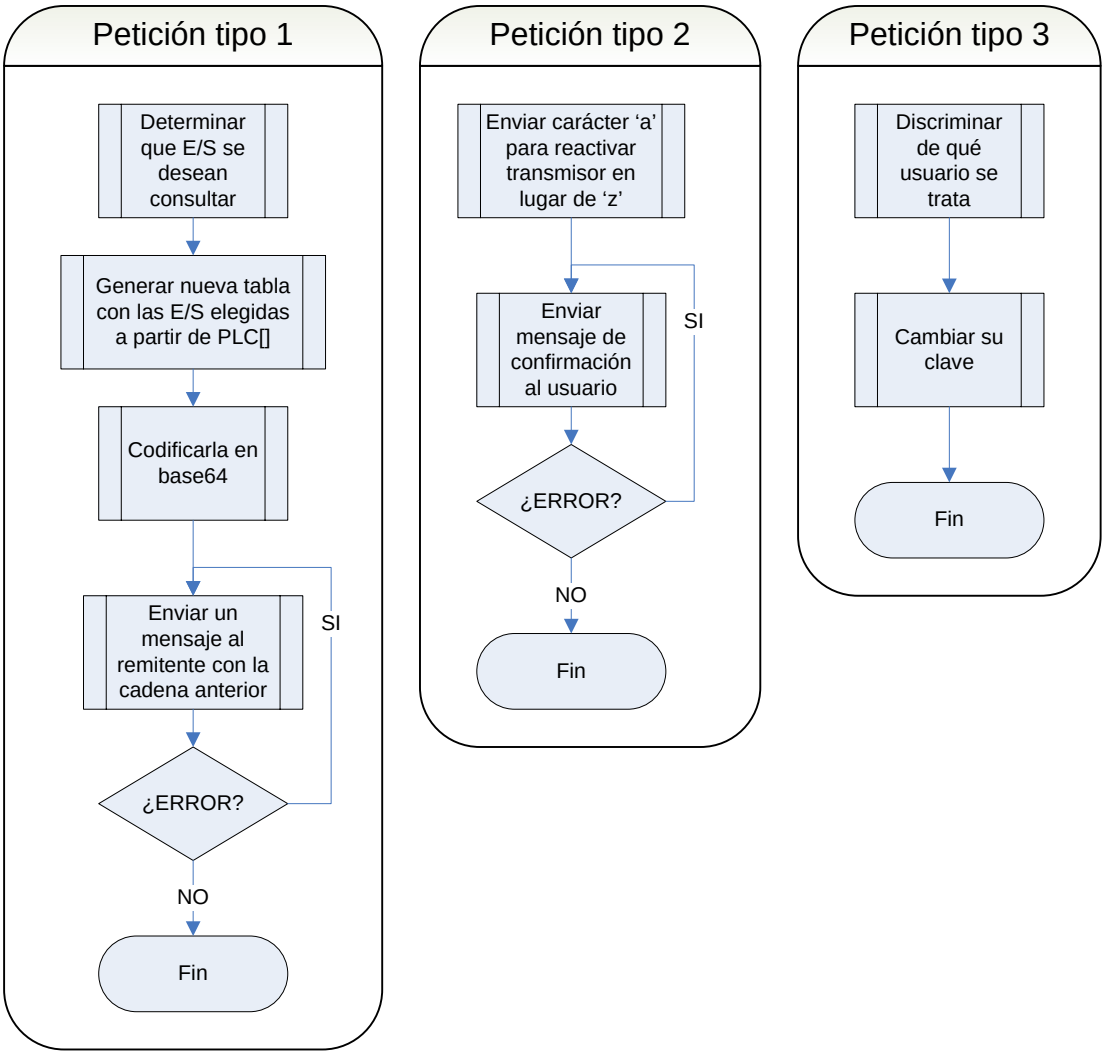

Figura 6.4 ***Diagrama de flujo del programa del microcontrolador (2/2)***

6.3.2 Código fuente del microcontrolador

```
/*****************************************************
Proyecto:
Versión:
Fecha: 17/10/2005

Autor: Nicolás Velasco Martos
Para: Dept. Ing. de Sistemas y Automática, Escuela Superior de Ingenieros de
      Sevilla


Chip: ATmega128L
Tipo de programa: Aplicación
Frecuencia de reloj: 8,000000 MHz

Modelo de memoria: Pequeño
Tamaño de la SRAM externa: 0
```

```
Tamaño de la pila de datos: 1024
*****************************************************/

#include <mega128.h>
#include <stdio.h>
#include <delay.h>
#include <string.h>

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

//PREDECLARACIONES
char getchar1(void);
void putchar1(char);
void escribe(char *);
void reset(void);
void inicia_micro(void);
int lee_PLC(void);
void gestiona_petic(void);
void encode( unsigned char in[3], unsigned char out[5], int len );
int lee_petic(char cad1[250], char cad2[250]);
void consulta_planta(char cad2[250], char telefono[12]);

//Almacenamiento en memoria EEPROM

//Necesario para no usar el primer byte de EEPROM
#pragma warn-
eeprom int nousar;
#pragma warn+

//Almacena el telefono de usuario valido Nº 1
eeprom char clave1[12]="629743715\r";

//Almacena el telefono de usuario valido Nº 2
eeprom char clave2[12]="620573083\r";


//Variables Globales
//Almacena el estado de la planta trasvasado desde el PLC
```

```
char plc[250];
//Almacena el estado de la planta en código base64
char plc_out[250];
//Almacena las variables de la planta requeridas por el usuario
char enviar[250];
//Flag que indica la primera recepción desde el PLC tras reset
int primvez=1;

//Cadena empleada como índice de conversión ASCCII <==> base64
char cb64[]=
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
/";


//Función Principal
void main(void)
{
 //Flag que indica si nos encontramos ante la primera cadena leída desde el PLC
 int k;

 /*Inicialización de los dispositivos desde software, en concreto se
   inicializarán el módem y el microcontrolador con los parámetros adecuados
   para este sistema en concreto*/

 inicia_micro();
 reset();

 /*Bucle principal que estará continuamente iterando, consta de dos funciones,
   la primera lee el estado del PLC. A continuación, la segunda comprueba si
   hay alguna petición del usuario remoto en el búfer del módem y la gestiona
   caso de ser cierto utilizando la codificación base64*/

 while (1)
   {
    //Reactivar el transmisor del PLC
    putchar1('z');
    //Se lee el estado y se indica si es el primer paso por la función
    k=lee_PLC();

    //Si no es el primer paso, se examina el búfer del módem para averiguar si
    //hay alguna petición pendiente
    if(k)
      gestiona_petic();
   };
}                          //FIN DE LA FUNCIÓN MAIN


//Inicializa los registros del microcontrolador con los valores adecuados
void inicia_micro(void)
{
```

```
// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=Out Func0=Out
// State7=T State6=T State5=T State4=T State3=T State2=T State1=0 State0=0
PORTA=0x00;
DDRA=0x03;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=Out Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x80;

// Port E initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTE=0x00;
DDRE=0x00;

// Port F initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTF=0x00;
DDRF=0x00;

// Port G initialization
// Func4=In Func3=In Func2=In Func1=In Func0=In
// State4=T State3=T State2=T State1=T State0=T
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
ASSR=0x00;
TCCR0=0x00;
```

```
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: Timer 3 Stopped
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
// Timer 3 Overflow Interrupt: Off
// Input Capture Interrupt: Off
```

```
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
ETIMSK=0x00;

// USART0 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART0 Receiver: On
// USART0 Transmitter: On
// USART0 Mode: Asynchronous
// USART0 Baud rate: 9600
UCSR0A=0x00;
UCSR0B=0x18;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x33;

// USART1 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART1 Receiver: On
// USART1 Transmitter: On
// USART1 Mode: Asynchronous
```

```
 // USART1 Baud rate: 19200
 UCSR1A=0x00;
 UCSR1B=0x18;
 UCSR1C=0x06;
 UBRR1H=0x00;
 UBRR1L=0x19;

 // Analog Comparator initialization
 // Analog Comparator: Off
 // Analog Comparator Input Capture by Timer/Counter 1: Off
 ACSR=0x80;
 SFIOR=0x00;
}


/*Toma un carácter del receptor de la USART1. Es necesaria esta función pues en
  stdio.h sólo se encuentra soporte para la USART0 a través de getchar()*/
char getchar1(void)
{
 char status,data;
 while (1)
   {
    //Se espera hasta que en UCSR1A no se active el bit de RX completo
    while (((status=UCSR1A) & RX_COMPLETE)==0);
    data=UDR1;
    //Si el reg de estado no contiene errores devuelve el reg de datos UDR1
    if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
        return data;
   };
}


/* Pasa un carácter al transmisor de la USART1. Es necesaria esta función pues
   en stdio.h sólo se encuentra soporte para la USART0 a través de getchar()*/
void putchar1(char c)
{
 //Se espera hasta que en UCSR1A se indica que hay algo en el reg de datos
 while ((UCSR1A & DATA_REGISTER_EMPTY)==0);
 //y lo enviamos a UDR1 para su transmisión
 UDR1=c;
}


//Adapta una cadena para su tx como comando AT comprensible por el módem
void escribe (char *linea)
{
 int i;

 //Escribe AT+
```

```
  putchar('A');
  putchar('T');
  putchar('+');

  //Escribe la cadena pasada como parámetro
  for(i=0;linea[i]!='\r';i++)
    putchar(linea[i]);

  //Termina con el carácter '/r' requerido por el módem como fin de línea
  putchar('\r');
}


//Rutina de inicialización del módem
void reset()
{
  do{
    //Se pasa por software a STAND BY
    printf("AT+CPWROFF\r");
    delay_ms(2000);
    /*Se corta la alimentación por hardware durante dos segundos para que se apague*/
    PORTD.7=1;
    delay_ms(2000);
    //y se vuelve a conectar
    PORTD.7=0;

    /*Espera que el módem indique que la red está accesible con el mensage
      "+CREG: 3"*/
    while(getchar()!='3');
    delay_ms(10);

    //Se pasa a estado activo introduciéndole el PIN
    printf("AT+CPIN=\"2849\"\r");
    //Esperamos la respuesta de la red y sin importar cual sea
    while(getchar()!='O');
    //Configuramos el módem
    printf("AT+CNMI=1,0,0,0,0\r");
    //Esperamos la respuesta de la red
    while(getchar()!='O');
    //Se repite el bucle mientras el módem devuelva ERROR
  }while(getchar()=='R');
}


//Rutina que lee el estado de la planta trasvasado desde el PLC
int lee_PLC()
{
  int i=0, j=0;
  int k=0;
```

```
  //Esperamos el carácter de comienzo de trama
  while(getchar1()!='I');

  //Si no es la primera vez que se recibe 'I'
  if(!primvez)
    {
     /*Tomamos el siguiente carácter cuyo código ASCII indica la longitud de la
          trama y se promociona de char a int para emplearlo como fin de cuenta*/

     for(i=0,j=(int)(getchar1());j>0;j--,i++)
          //Copiamos los caracteres recibidos en la tabla plc[]
          plc[i]=getchar1();
     //E incluimos el carácter '\r'
     plc[i]='\r';

     //Sólo para depuración
     delay_ms(1000);
     escribe(plc);
     delay_ms(1000);

     //Indica que se ha realizado al menos una lectura del estado del PLC
     k=1;
    }

  /*Si es la primera recepción de 'I', actualizamos primvez y esperamos un
    segundo de modo que aseguramos que se ha concluido la recepción de
    caracteres. Se hace para evitar que coincida la primera lectura desde el
    microcontrolador con un instante de tiempo en que el autómata esté enviando
    su estado y el carácter 'I' forme parte del campo de datos*/

  if(primvez)
    {
     primvez=0;
     //Sólo para depuración
     printf("AT+de paso\r");
     delay_ms(1000);
    }

  return (k);
}


//Rutina que lee del módem si hay una nueva petición por parte del cliente y la gestiona
void gestiona_petic(void)
{
  char c='a';
  char c2;
  //Flag que indica si hay nueva petición
  int recibido=0;
  int i=0, j=0;
```

```
//Cadena que almacena la cabecera del mensaje: Nº mensaje, teléfono del remitente, ...
char cad1[250];
//Cadena que almacena el cuerpo del mensaje
char cad2[250];
//Cadena que almacena el tfno del remitente que aparece en cad1[]
char telefono[12];
//Flags que indican si el remitente es un usuario válido
int distinto=0;
int distinto_aux=0;

/*Llama a la función lee_petic que solicita el listado de los mensajes
  existentes en la memoria del módem, guardando los datos más destacados en
  las cadenas cad1[] y cad2[]*/

recibido=lee_petic(cad1, cad2);

//Si hay nuevo SMS
if(recibido)
  {
    //Sólo para depuración
    //escribe(cad1);
    //delay_ms(1000);

    //printf("AT+numero %c\r",cad1[1]);
    //delay_ms(1000);

    /*Copia de cad1 los datos del teléfono remitente, el mensaje podría ser
        leído o no leído, (READ o UNREAD en inglés) la distinta longitud de
        estas palabras hace que el campo teléfono comience en la posición 18 ó
        20*/
    if(cad1[8]=='U')
        i=20;
    else
        i=18;

    for(j=0;cad1[i]!='\"';i++,j++)
        telefono[j]=cad1[i];
    telefono[j]='\r';

    escribe(telefono);
    delay_ms(1000);

    //Sólo para depuración
    //escribe(cad2);
    //delay_ms(1000);
    //Borra el mensaje de la memoria de la SIM
    printf("AT+CMGD=%c\r",cad1[1]);

    //Leemos la respuesta para evitar escribir mientras se reciben datos
    while(getchar()!='O');
```

```
getchar();

//Bucle de comprobación para averiguar si el teléfono es valido
for(i=0;telefono[i]!='\r';i++)
     {
      if(telefono[i]!=clave1[i])
        distinto=1;
      if(telefono[i]!=clave2[i])
        distinto_aux=1;

     }
if(!(distinto*distinto_aux))
     /*Si es un teléfono válido miramos el primer carácter de cad2[] que nos
      indicará cual es la petición del cliente. Tenemos tres opciones:
      1, 2 y 3 que se corresponden respectivamente con las peticiones
      consultar el estado de la planta, solicitar parada de emergencia o
      cambiar la clave de usuario, lo que supone cambiar el número de móvil
      llamante. En caso de error en el mensaje y que no se reciba lo
      esperado, no se hará nada*/

     {
      //Bucle de selección
      switch (cad2[0]) {
      case '1':
        //Consulta el estado de la planta
        consulta_planta(cad2, telefono);
        break;

      case '2':
        //Parada de emergencia
        //Sólo para depuración
        printf("AT+tipo 2\r");
        delay_ms(1000);

        //Enviamos el caracter 'a' para detener el PLC
        putchar1('a');

        /*Y generamos un mensaje para notificarlo al usuario se incluye en
          un bucle do-while para repetir el proceso mientras no se obtenga
          respuesta afirmativa de la red*/
        do
          {
              printf("AT+CMGS=\"");
              for(i=0;telefono[i]!='\r';i++)
                putchar(telefono[i]);
              printf("\"\r");
              delay_ms(5000);
              printf("Modo STOP");
              delay_ms(5000);
              putchar(26);
```

```
        while(getchar()!='O');
    }while(getchar()=='R');

  break;

case '3':

  //Cambio de clave de usuario
  //Sólo para depuración
  printf("AT+tipo 3\r");
  delay_ms(1000);

  //Se emplean los flags distinto y distinto_aux para saber que
  //usuario es el activo y cambiar su clave
  if(!distinto)
    {
        for(i=1;cad2[i]!='\r';i++)
          clave1[i-1]=cad2[i];
    }

  if(!distinto_aux)
    {
        for(i=1;cad2[i]!='\r';i++)
          clave2[i-1]=cad2[i];
    }

  //Sólo para depuración
  /*putchar('A');
    putchar('T');
    putchar('+');
    for(i=0;clave1[i]!='\r';i++)
    putchar(clave1[i]);
    putchar('*');
    for(i=0;clave2[i]!='\r';i++)
    putchar(clave2[i]);
    putchar('\r');    */

  break;

default:

  //En caso de mensaje erroneo
  //Sólo para depuración
  printf("AT+tipo nulo\r");
  delay_ms(1000);

  break;

};
//Sólo para depuración
```

```
            //escribe(telefono);
            //delay_ms(1000);
            //escribe(cad2);
           }

    //Se reinicia el flag para sucesivos pasos por la función
    recibido=0;

    //Reactivamos el transmisor del PLC
    delay_ms(1000);
    putchar1('z');

    //Sólo para depuración
    if(distinto*distinto_aux)
        {
         delay_ms(1000);
         printf("AT+numero no valido\r");
         delay_ms(1000);
        }
   }
}
```

```
//Rutina que codifica tres caracteres ASCII en cuatro caracteres base64
void encode(unsigned char in[3], unsigned char out[5], int len)
{
 /*out[0] contiene el carácter indexado en la tabla cb64[] por los seis bits
 más significativos de in[0]*/
 out[0] = cb64[ in[0] >> 2 ];

 /*out[1] contiene el carácter indexado en la tabla cb64[] por los dos bits
 menos significativos de in[0] seguidos de los cuatro bits más significativos
 de in[1]*/
 out[1] = cb64[ ((in[0] & 0x03) << 4) | ((in[1] & 0xf0) >> 4) ];

 /*out[2] contiene el carácter indexado en la tabla cb64[] por los cuatro bits
   menos significativos de in[1] seguidos de los dos más significativos de
   in[2] si len es mayor que uno, siendo '=' en caso contrario*/
 out[2] = (unsigned char) (len > 1 ? cb64[ ((in[1] & 0x0f) << 2) | ((in[2] & 0xc0) >> 6)
] : '=');

 /*out[3] contiene el carácter indexado en la tabla cb64[] por los seis bits
   menos significativos de in[2] si len es mayor que dos, siendo '=' en caso
   contrario*/
 out[3] = (unsigned char) (len > 2 ? cb64[ in[2] & 0x3f ] : '=');
}
```

```
//Rutina que captura en dos cadenas los datos de cada nuevo mensaje recibido
int lee_petic(char cad1[250], char cad2[250])
```

```
{
 char c='a';
 int mal=0;
 int i=0;
 int recibido=0;

 //Solicita el listado de los mensajes almacenados en la memoria del módem
 printf("AT+CMGL=\"ALL\"\r");

 /*Lee caracteres hasta que encuentra el carácter ':', indicativo de que se ha
   recibido un mensaje o hasta encontrar el carácter 'K', indicativo de que no
   hay nueva recepción*/

 while (c!=':' && mal==0)
   {
    c=getchar();
    if(c=='K')
        mal=1;
   }

 //Si el carácter no es 'K' hay una nueva recepción y debemos capturar el
 //mensaje entrante

 if(!mal)
   {
    do{
        //Guardamos en cad1[] la cabecera del mensaje
        c=getchar();
        cad1[i]=c;
        i++;
    }while(c!='\r');

    getchar();
    getchar();
    getchar();
    i=0;

    do{
        //Guardamos en cad2[] el cuerpo del mensaje
        c=getchar();
        cad2[i]=c;
        i++;
    }while(c!='\r');

    //Indicamos finalmente con un flag que hemos capturado un mensaje
    recibido=1;

    while(getchar()!='K');
   }
```

```
  //Se habilita de nuevo al transmisor del PLC en caso de no haber peticiones
  if(mal)
    {
     delay_ms(1000);
     putchar1('z');
    }

  return (recibido);
}
```

/*Rutina que realiza las operaciones oportunas para enviar al usuario el estado del PLC*/

void consulta_planta(char cad2[250], char telefono[12])

```
{
  int i=0, j=0, k=0;
  int ii=0;
  char in[3];
  char out[4];
  int len=0;

  /*En el mensaje de consulta del estado de la planta, que comienza con el carácter '1', se indican a continuación las entradas y salidas analógicas que se desean consultar, para ello, se indica con caracteres numéricos el índice de entradas y/o salidas analógicas de la cadena plc[] que se desean consultar. Se muestran todas las E/S digitales.*/

  //Incluimos todas las entradas y salidas digitales incluidas en la CPU 224
  enviar[0]=plc[0];
  enviar[1]=plc[1];
  enviar[2]=plc[2];
  enviar[3]=plc[3];

  /*Se toman todos los caracteres restantes del mensaje entrante, se convierten
    en su valor entero y se utilizan como índice para la tabla plc[]*/
  for(i=1;cad2[i]!='\r';i++)
    {
     enviar[(i+1)*2]=plc[((int)(cad2[i])-48+1)*2];
     enviar[(i+1)*2+1]=plc[((int)(cad2[i])-48+1)*2+1];
    }
  enviar[(i+1)*2]='\r';

  //k almacena el número total de caracteres
  k=(i+1)*2;

  //Sólo para depuración
  escribe(enviar);
  delay_ms(1000);
  //printf("AT+%d\r",k);
  //delay_ms(1000);
```

```
j=0;

//Para todos los caracteres de la tabla enviar[]
while(j<k)
  {
    len=0;

    //Copiamos en in[] los caracteres contenidos en enviar[] de tres en tres
    for(i=0;i<3;i++,j++)
         {
           in[i]=enviar[j];
           if(j<k)
             len++;
           else
             in[i]=0;
         }

    //Si in[] no está vacía, se codifica en base64 su contenido
    if(len)
         encode(in,out,len);

    /*Finalmente, se concatenan las sucesivas cadenas out[] para generar el
         mensaje de salida*/
    for(i=0;i<4;i++,ii++)
         plc_out[ii]=out[i];
  }

plc_out[ii]='\r';

//Sólo para depuración
escribe(plc_out);
delay_ms(500);

//Sólo para depuración
printf("AT+tipo 1\r");
delay_ms(1000);

/*Finalmente, enviamos un mensaje al teléfono del usuario con el mensaje
  codificado en base64 contenido en plc_out[]. Se incluye en un bucle do-while
  para repetir el proceso mientras no se obtenga respuesta afirmativa de la red*/
do
  {
    printf("AT+CMGS=\"");
    for(i=0;telefono[i]!='\r';i++)
         putchar(telefono[i]);
    printf("\"\r");
    delay_ms(5000);
    for(i=0;plc_out[i]!='\r';i++)
         putchar(plc_out[i]);
    delay_ms(5000);
```

```
      putchar(26);
      while(getchar()!='O');
    }while(getchar()=='R');
}
```