TESI
IT20
B5
V.2

UNIVERSIDAD CATÓLICA ANDRÉS BELLO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES

DISEÑO DE UN TELÉFONO IP

APENDICE

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar al título de

INGENIERO EN TELECOMUNICACIONES

REALIZADO POR *Blanco Ascencion, Luis Fernando*
Suárez Rodríguez, Victor Eduardo

PROFESOR GUIA *Alvarado Robles, Luis*

FECHA Caracas, 21 de Febrero de 2007

INDICE

APÉNDICE 2. INSTRUCCIONES PARA LA EJECUCIÓN Y COMPILACIÓN DE LOS PROGRAMAS 2

EJECUCIÓN DEL PROGRAMA 3
INSTRUCCIONES PARA LA COMPILACIÓN 4

APÉNDICE 3. TELÉFONO IP, PRIMER PROTOTIPO 5

RTP.H 6
RTP.C 6
SIP.H 12
SIP.C 15
TELEF.C 25

APÉNDICE 4. TELÉFONO IP, VERSIÓN 2 35

VOIPV2.ASM 36
TIMERS.H 47
TIMERS.C 48
RTP2.H 53
RTP2.C 54
SIP2.H 70
SIP2.C 73

APÉNDICE 2. INSTRUCCIONES PARA LA EJECUCIÓN Y COMPILACIÓN DE LOS PROGRAMAS

Ejecución del Programa

Para poner a funcionar el dispositivo se requieren de diversos programas como son los manejadores de paquetes de la tarjeta de red, el programa Voip que representa el programa residente encargado de interrumpir cada 125 microsegundos para obtener y enviar bytes del circuito conversor y por ultimo se ejecuta el programa principal llamado Telef el cual maneja toda la lógica necesaria para crear y cerrar conversaciones en tiempo real. Todos estos ellos se ejecutan bajo el sistema operativo DOS.

Es importante recordar que cada tarjeta de red tiene su propio manejador de paquetes y es necesario ya que WATTCP lo utiliza para poder establecer la comunicación, normalmente se encuentran distribuidos en la página web de su fabricante.

Para el funcionamiento del programa en el procesador 486 se tiene incluido los siguientes comandos:

LSL.com
LAN9000.com
ODIPKT.com
Voip.exe
Telef.exe

Para un computador personal de prueba 486 se utilizo los siguientes comandos

rtspkt.com 0x7e –m auto
Voip.exe
Telef.exe

Como se puede observar para cada tarjeta de red se requiere algunos comandos diferentes para el manejador de paquetes, pero para los otros programas se mantiene igual.

Instrucciones para la Compilación

Para compilar los programas desarrollados se utilizó una herramienta llamada Borlandc 3.1 (se puede hallar dentro del Cd adjunto al tomo). Se recomienda que Borlandc debe copiarse la carpeta del programa a el directorio raíz (disco C del computador) para mantener el ambiente de trabajo, es decir que las librerías y funciones se mantengan con el mismo árbol de directorio.

Para compilar el primer prototipo del teléfono se utiliza comp.bat el cual tiene incluida las siguientes lineas de comando

Bcc -O2 telef.c sip.c rtp.c wattcpsm.lib

Recordando que tlef.c se encuentra la función principal, y utiliza funciones que se encuentran definidas en sip.c y rtp.c, así como también wattcpsm.lib que es la librería de WATTCP.

Bcc es la llamada al compilado de borlandc, el –O2 habilita la optimización de usos de registros y genera el codigo mas rapido posible

Para la compilación del segundo prototipo se utiliza compv2.bat que contiene las siguientes líneas

Bcc –O2 telefv2.c sp2.c rtp2.c timers.c wattcpsm.lib

APÉNDICE 3. TELÉFONO IP, PRIMER PROTOTIPO

En las siguientes líneas se presenta el código del primer prototipo del teléfono IP, se explica de forma precisa las principales funciones y aplicaciones que lo conforman RTP.c, SIP.c, SIP.h, RTP.h.

RTP.h

Definiciones de las funciones para el código RTP.C; declara las funciones de RTP.C que pueden ser utilizadas fuera del código fuente.

A saber, las funciones de interfaz para RTP son las siguientes

- int *crear_pack_rtp*(u_int8_t* buffer);
- void *recibir_pack_rtp*(u_int8_t * buffer_in);
- void *rtp_init*(void);
- void *rtp_fin*(void);
- int *rtp_estado*(void);

Código RTP.H

```
1
2#ifndef _RTP_H_
3 #define _RTP_H_
4
5 #define ACTIVO 1
6 #define INACTIVO 0
7
8 typedef unsigned char u_int8_t;
9
10 int crear_pack_rtp( u_int8_t* buffer);
11 void recibir_pack_rtp(u_int8_t * buffer_in);
12 void rtp_init(void);
13 void rtp_fin(void);
14 int rtp_estado(void);
15
16 #endif
```

RTP.c

RTp.c ofrece la posibilidad de crear, recibir, y finalizar las sesiones. Utiliza las librerías estándar string.h y la librería personalizada rtp.h.

Se utilizaron las siguientes definiciones

- #define *TRUE* -1

- #define *FALSE* 0
- #define *OBuf_Size* 8192: tamaño del buffer de salida para la carga RTP
- #define *IBuf_Size* 8192: tamaño del buffer de recepción de los paquetes RTP
- #define *RTP_PACKET_SIZE* 172: tamaño en bytes de un paquete RTP
- #define *RTP_PACKET_HEADER_SIZE* 12: tamaño en bytes del header
- #define *RTP_PACKET_PAYLOAD_SIZE* 160: cantidad de bytes de carga de un paquete RTP.
- unsigned char far **tempo_rtp_out*: Puntero controlado por la interrupción 82h, que me indica de cual buffer se pueden obtener los bytes para armar los paquetes RTP y puedan ser enviados.
- unsigned char far **rtp_out*: está siempre asignado como tempo_rtp_out para el manejo del programa y se usa para saber que el buffer ha cambiado
- unsigned char far **tempo_rtp_in*: Puntero controlado por la interrupción 81h, que me indica en cual buffer puede ser guardado la carga de un paquete rtp.
- unsigned char far **rtp_in*: Representa lo mismo que tempo_rtp_in, y es actualizada cuando rtp_in es distinto de tempo_rtp_in.
- int *primer_paquete_rtp* = TRUE: indica que es el primer mensaje rtp
- unsigned long *timestamp*;
- unsigned short *Nsecuen_t* = 11212; el numero de secuencia es aleatorio con un máximo de 65535.
- unsigned short * *psecuen_t*= &Nsecuen_t;
- unsigned short *Nsecuen_r* = 0;
- unsigned short * *psecuen_r*=&Nsecuen_r;
- unsigned short *Nsecuen_prueb*: cada vez que se reciba un paquete se tomará el número de secuencia de ese paquete para luego compararlo con la secuencia que se esta esperando.
- unsigned short * *psecuen_prueb*=&Nsecuen_prueb;
- unsigned char *pack_rtp_t*[172]={0x80,0x00,'o','o',0x00,0x1f,0xb3,0xa0, 0x00,0x00,0x58,0x9a,'\0'}: Se define pack_rtp_t con su cabecera en donde después puede ser modificada su número de secuencia
- unsigned char *pack_rtp_r*[172]: Es el buffer donde se va a guardar un paquete Rtp mientras es analizado su destino dependiendo del numero de secuencia.
- int estado_rtp: indica si el programa RTP se encuentra activo o desactivo.

A continuación se presenta la explicación detallada de cada función que conforma RTP.c

int *crear_pack_rtp*(u_int8_t* buffer)

Recibe:
Recibe la dirección del buffer donde se va crear el paquete rtp

Retorna:
EL tamaño completo del paquete RTP

Funcionamiento:
Se definen 2 punteros lejanos, el primero temp_ptr que indica el byte que debe ser sacado y se va a estar moviendo en el Ibuffer; segundo final_ptr que marca el final del Ibuffer, luego utilizando la función memcpy() se le asigna al buffer su respectiva cabecera y su numero de secuencia correspondiente, se incrementa el numero de secuencia para el próximo paquete. Se verifica que el puntero del buffer A/D no ha cambiado, en el caso en que haya cambiado, se actualiza el puntero con la nueva dirección y se inicializa los punteros temp_ptr y final_ptr. Si la cantidad de datos que se desean enviar es menor a un payload RTP, se envían los disponibles hasta el final del buffer y se completa con los que falten usando los del principio del mismo buffer.

void *recibir_pack_rtp*(u_int8_t * buffer)
Esta función se encarga de recibir un paquete RTP y lo almacena en el buffer D/A (Obuffer) utilizando el puntero tempo_rtp_in

Recibe:
El buffer donde se encuentra almacenado el paquete rtp recibido

Funcionamiento:
Al recibir un paquete RTP se verifica si es el primer paquete, utilizando la bandera primer_paquete_rtp, en el caso de que sea el primero se guarda el numero de secuencia que es recibido en Nsecuen_r para luego mantener el control de los paquetes que se esperan. En el caso en que no, se le asigna a temp_secuencia el numero de secuencia que es recibido en el paquete RTP, luego se verifica si el puntero que indica el buffer de llenado no ha cambiado (temp_rtp_in), esto se

hace comparando el rtp_in con temp_rtp_in, si ha cambiado se igualan y se inicializa temp_ptr. Luego se compara el numero de secuencia, si el paquete que se recibe es mayor que el esperado, se recalcula la nueva posición dentro del buffer donde este deberá ser guardado manteniendo un rango por encima de 10 números y se actualiza el numero de secuencia; si esta posición es mayor que el final del buffer se coloca en el inicio del mismo, y se actualiza el siguiente numero de secuencia a esperar. Si el paquete recibido tiene menor número de secuencia y con un temp_secuencias mayor que 10, el paquete es descartado|,. En el caso que el paquete es el que se está esperando se almacena y se incrementa en el número de secuencia. Si el espacio en el buffer disponible no es suficiente para guardar el paquete, se asumen las perdidas de esos bytes para evitar que el sistema se desincronoce.

Declaraciones de variables:

- static unsigned char far * *temp_ptr*: indica el lugar en donde se van a ir almacenando los bytes de carga del paquete RTP
- static unsigned char far * *aux_ptr*: señala el fin del buffer (obuffer)
- int length;
- unsigned int *temp_secuencia*: representa el numero de secuencia del paquete que se acaba de recibir
- unsigned int *Nsecuen_r_rango*: es el numero de secuencia máximo que se pudiera recibir a partir del que se está esperando

void *rtp_init*(void) :
Inicia el estado de RTP a activo, con su número de secuencia inicial que puede ser cualquiera.

void *rtp_fin*(void)
Reestablece la bandera del primer paquete y la bandera de estado_rtp la coloca en inactivo.

***rtp_estado*(void)**
Solo retorna el estado actual que se encuentra RTP

Código RTP.C

```
3 #include <string.h>
4 #include "rtp.h"
5
6 #define TRUE -1
7 #define FALSE 0
8
9  #define OBuf_Size 8192
10 #define IBuf_Size 8192
11
12 #define RTP_PACKET_SIZE 172
13 #define RTP_PACKET_HEADER_SIZE 12
14 #define RTP_PACKET_PAYLOAD_SIZE 160
15 unsigned char far *tempo_rtp_in;
16 unsigned char far *rtp_in;
17 unsigned char far *tempo_rtp_out;
18 unsigned char far *rtp_out;
19
20 int primer_paquete_rtp = TRUE; //indica que es el primer mns
rtp
21 int i;
22
23 unsigned long timestamp;
24 unsigned short Nsecuen_t = 11212;
25 unsigned short * psecuen_t= &Nsecuen_t;
26 unsigned short Nsecuen_r = 0;
27 unsigned short * psecuen_r=&Nsecuen_r;
28
29 //RTP header
30 unsigned char
pack_rtp_t[172]={0x80,0x00,'o','o',0x00,0x1f,0xb3,0xa0,0x00,0x0
0,0x58,0x9a,'\0'};
31
32 int estado_rtp;
33
34 int crear_pack_rtp( u_int8_t* buffer )
35 {
36 static unsigned char far * temp_ptr;
37 static unsigned char far * final_ptr;
38 int length;
39 memcpy(buffer, pack_rtp_t, RTP_PACKET_HEADER_SIZE);
40 buffer[2] = (unsigned char) ((*(psecuen_t) & 0xFF00) >> 8);
41 buffer[3] = (unsigned char) *(psecuen_t);
42 Nsecuen_t++;
43
44 if( rtp_out != tempo_rtp_out ){ //temp_rtp_out es modificado
por la interrupcion 82h
45 rtp_out = tempo_rtp_out; //si no coinciden los
punteros se igualan
46 temp_ptr = tempo_rtp_out; //y se inicializa el
siguiente byte de salida en RTP
47 final_ptr = (unsigned char far *) (rtp_out + OBuf_Size);
48 }
49
50 length = ( (rtp_out + OBuf_Size) - temp_ptr );
51 if( length > RTP_PACKET_PAYLOAD_SIZE ){
52 memcpy( (void*)(buffer+RTP_PACKET_HEADER_SIZE),
(void*)temp_ptr, RTP_PACKET_PAYLOAD_SIZE );
```

```
53 temp_ptr = (unsigned char far *) (temp_ptr +
RTP_PACKET_PAYLOAD_SIZE);
54 if( temp_ptr > final_ptr )
55 temp_ptr = rtp_out;
56 }
57 else{
58 memcpy( (void*)(buffer+RTP_PACKET_HEADER_SIZE),
(void*)temp_ptr, length );
59 temp_ptr = rtp_out;
60 memcpy( (void*)(buffer+RTP_PACKET_HEADER_SIZE),
(void*)temp_ptr, (RTP_PACKET_PAYLOAD_SIZE-length) );
61 temp_ptr = (unsigned char far *) (temp_ptr +
RTP_PACKET_PAYLOAD_SIZE - length);
62 }
63
64 return RTP_PACKET_SIZE;
65 }
66
67 void recibir_pack_rtp(u_int8_t * buffer)
68 {
69 static unsigned char far * temp_ptr;
70 static unsigned char far * aux_ptr;
71 int length;
72 unsigned int temp_secuencia;
73 unsigned int Nsecuen_r_rango;
74
75 //obteniendo la secuencia del paquete
76 if( primer_paquete_rtp == TRUE ){
77 Nsecuen_r = (unsigned int)( (buffer[2] << 8) | buffer[3]
);
78 Nsecuen_r_rango = Nsecuen_r + 10;
79 temp_secuencia = Nsecuen_r;
80 primer_paquete_rtp = FALSE;
81 }
82 else
83 temp_secuencia = (unsigned int)( (buffer[2] << 8) |
buffer[3] );
84
85 if( rtp_in != tempo_rtp_in ){ //temp_rtp_in es modificado
por la interrupcion 81h
86 rtp_in = tempo_rtp_in; //si no coinciden los punteros
se igualan
87 temp_ptr = tempo_rtp_in; //y se inicializa el
siguiente byte de salida en RTP
88 }
89
90 if ( (temp_secuencia > Nsecuen_r) || ( (temp_secuencia <=
Nsecuen_r_rango) && (temp_secuencia < 10) ) ){
91 temp_ptr = (unsigned char far *)( temp_ptr +
((temp_secuencia - Nsecuen_r) * RTP_PACKET_PAYLOAD_SIZE) );
92 aux_ptr = (unsigned char far *) (rtp_in + IBuf_Size);
93 while( temp_ptr > aux_ptr )
94 temp_ptr = (unsigned char far *) rtp_in;
95 Nsecuen_r = temp_secuencia + 1;
96 Nsecuen_r_rango = Nsecuen_r + 10;
97 }
98 else if(temp_secuencia < Nsecuen_r ){
99 return;
100 }
```

```
else{
Nsecuen_r++;
Nsecuen_r_rango++;
}

aux_ptr = (unsigned char far *)
(buffer+RTP_PACKET_HEADER_SIZE);
length = (unsigned int) ((rtp_in + IBuf_Size) - temp_ptr);

if( length > RTP_PACKET_PAYLOAD_SIZE ){
memcpy( (void*)temp_ptr, (void*)aux_ptr,
RTP_PACKET_PAYLOAD_SIZE );
temp_ptr = (unsigned char far*) (temp_ptr +
RTP_PACKET_PAYLOAD_SIZE);
}
else{
memcpy( (void*)temp_ptr, (void*)aux_ptr, length );
temp_ptr = rtp_in;
memcpy( (void*)temp_ptr, (void*)aux_ptr,
(RTP_PACKET_PAYLOAD_SIZE-length) );
temp_ptr = (unsigned char far*) (temp_ptr +
RTP_PACKET_PAYLOAD_SIZE - length);
}

return;

}

void rtp_init(void){
Nsecuen_t = 11212;
estado_rtp = ACTIVO;
}

void rtp_fin(void){
primer_paquete_rtp = FALSE;
estado_rtp = INACTIVO;
}

int rtp_estado(void){
return estado_rtp;
}
```

SIP.h

Definiciones de las funciones para el código SIP.C. Declara las funciones del SIP.C que pueden ser utilizadas fuera del código fuente y como también unos macros. Utiliza la librería dos.h

Las funciones de la interfaz para SIP son las siguientes

- void *sip_init*(void);

- int *estadoSIP*(void);
- void *check_sip*(void);
- int *recibir_pack_sip*(u_int8_t * buffer , u_int8_t * buffer_out);
- int *generar_invite*(u_int8_t * buffer);
- int *generar_200*(u_int8_t* buffer, u_int8_t* buffer_out);
- int *generar_bye*(u_int8_t * buffer);

También existen las siguientes definiciones o macros:

- #define *FALSE* 0
- #define *TRUE* -1
- typedef unsigned *char u_int8_t*;

Macros utilizadas para repicar en el teléfono y aceptar llamadas

- #define *LPT_STATUS* 0x379
- #define *PIN12* (1 << 5) // LED
- #define *PIN13* (1 << 4) // boton
- #define *repicar()* outportb(LPT_STATUS, (inport(LPT_STATUS) | PIN12))
- #define *fin_repicar()* outportb(LPT_STATUS, (inport(LPT_STATUS) & ~PIN12))
- #define *leerBOTON()* (inport(LPT_STATUS) & PIN13)

Macro para el cambio de estado

- #define *cambio_estado*(x,y) (x = y)

Macros para el manejo de los timers

- #define *BIOS_TIME_OFF* 0x046C
- #define *BIOS_TIME_SEG* 0x0000
- #define *get_bios_time()* (unsigned int) peek(BIOS_TIME_SEG, BIOS_TIME_OFF)
- #define *set_bios_time_end(var,seg)* (var += seg*18)
- #define *check_time_end(var)* ((unsigned int)(var <= get_bios_time()) ? TRUE : FALSE)

Representan los estados disponibles del teléfono

- #define *REGISTRANDO* 0
- #define *ESPERA* 1
- #define *LLAMANDO* 2

- #define *PROCESANDO* 3
- #define *REPICANDO* 4
- #define *LLAMADA* 5

Código SIP.H

```
1
2 #ifndef _SIP_H_
3 #define _SIP_H_
4
5
6#include <dos.h>
7 #define FALSE 0
8 #define TRUE -1
9
10 #define REGISTRANDO 0
11 #define ESPERA 1
12 #define LLAMANDO 2
13 #define PROCESANDO 3
14 #define REPICANDO 4
15 #define LLAMADA 5
16
17 #define LPT_STATUS 0x379
18 #define PIN12 (1 << 5) // LED
19 #define PIN13 (1 << 4) // boton
20 #define repicar() outportb(LPT_STATUS, (inport(LPT_STATUS) |
PIN12) )
21 #define fin_repicar() outportb(LPT_STATUS, (inport(LPT_STATUS)
& ~PIN12) )
22
23 #define leerBOTON() (inport(LPT_STATUS) & PIN13)
24
25 #define cambio_estado(x,y) (x = y)
26
27 #define BIOS_TIME_OFF 0x046C
28 #define BIOS_TIME_SEG 0x0000
29 #define get_bios_time() (unsigned int) peek(BIOS_TIME_SEG,
BIOS_TIME_OFF)
30 #define set_bios_time_end(var,seg) (var += seg*18)
31 #define check_time_end(var) ( (unsigned int)(var <=
get_bios_time()) ? TRUE : FALSE )
32
33 // #define u_int8_t unsigned char
34 typedef unsigned char u_int8_t;
35
36 void sip_init(void);
37 int estadoSIP(void);
38 void check_sip(void);
39 int recibir_pack_sip( u_int8_t * buffer , u_int8_t *
buffer_out);
40 int generar_invite( u_int8_t * buffer );
41 int generar_200(u_int8_t* buffer, u_int8_t* buffer_out);
42 int generar_bye(u_int8_t * buffer);
43
44
45 #endif
```

SIP.C

Se encuentra las funciones y aplicaciones necesarias para establecer y terminar una sesión de llamada. Trabaja junto a las librerías stdlib.h, stdio.h, string.h y sip.h.

Se tienen los siguientes arreglos que están inicializados con sus respectivos datos dependiendo del mensaje que se trate.

- unsigned char**REGISTER_primera_linea*={"REGISTER sip:192.168.10.7 SIP/2.0"};
- unsigned char **INVITE_primera_linea* = {"INVITE sip:192.168.10.7 SIP/2.0"};
- unsigned char **ACK_primera_linea*={"ACK sip:192.168.10.7 SIP/2.0"};
- unsigned char **BYE_primera_linea*={"BYE sip:192.168.10.7 SIP/2.0"};
- unsigned char **primera_linea180*={"SIP/2.0 180 Repicando"};
- unsigned char **primera_linea200*={"SIP/2.0 200 OK"};
- unsigned char **primera_linea486*={"SIP/2.0 486 Ocupado"};
- unsigned char **array*[26]: donde se van a colocar el orden de los punteros que indican las partes de cualquiera de los mensajes posibles a generar.

Estos arreglos contienen la información necesaria para crear un paquete sip, cada uno de ellos representan partes importantes a la hora de definir un mensaje SIP.

- unsigned char **via*={"Via: SIP/2.0/UDP 192.168.10.7;branch=z9hg4bk80d5fe006aeddd 11a7950008a1641510"};
- unsigned char **from*={"From: luis fblanco<sip:luis_blanco_1@1ucab.edu.ve>;tag=701 0"};
- unsigned char **to*={"To:<sip:victor_suarez_1@ucab.edu.ve>"};
- unsigned char **to_tag*={"To:<sip:victor_suarez_1@ucab.edu.ve>;tag=92874"};
- unsigned char **call_id*={"Call-id: 80d5fe00-6aed-dd11@ucab.edu.ve"};
- unsigned char **cseq_bye*={"Cseq: 6 BYE"};
- unsigned char **cseq_register*={"Cseq: 6 REGISTER"};
- unsigned char **cseq_ack*={"Cseq: 6 ACK"};
- unsigned char **cseq_invite*={"Cseq: 6 INVITE"};
- unsigned char **contact*={"Contact: <sip:luis_blanco_1@ucab.edu.ve>"};

- unsigned char **max_forwards*={"Max-forwards: 70"};
- unsigned char **user_agent*={"User-agent: Ucab phone 1"};
- unsigned char **content_type*={"Content-type: application/SDP"};
- unsigned char **content_length*={"Content-length: 143"};

Arreglos utilizados para definir el mensaje SDP

- unsigned char **version*={"v=0"};
- unsigned char **owner*={"o=- 3442329143 3442329143 IN IP4 192.168.10.5"};
- unsigned char **session*={"s= Ucab IPphone"};
- unsigned char **info_conex*={"c=IN IP4 192.168.10.5"};
- unsigned char **time_description*={"t=0 0"};
- unsigned char **media_description*={"m=audio 5004 rtp/avp 0"};
- unsigned char **media_atribute*={"a=rtpmap:0 pcmu/8000"};

- unsigned char *fin_string*[3]={0xd,0xa,0x00}: esta declaración es usada al crear mensajes SIP ya que con cada parámetro que se necesite, se le agrega estos caracteres que son: retorno del carro, salto de línea y final de cadena de caracteres.
- unsigned int *estado* = 0: maneja el estado que se encuentra el teléfono
- unsigned int *tiempo*;

Funciones que se encuentran dentro de SIP.c

Int *generar_invite*(u_int8_t *buffer)

Recibe:

La dirección del buffer donde se va formar el paquete

Retorna:

Un entero indicando la longitud del mensaje generado

Funcionamiento:

Se le asigna a cada valor del arreglo las direcciones de cada tipo de mensaje definido anteriormente, para luego ser agregado uno a uno al buffer y formar un

paquete completo Invite. Una ves cargado el paquete SIP se cambia de estado de ESPERA a estado LLamando, además se activa un temporizador usando el macro set_bios_time_end.

int *generar_ack*(u_int8_t * buffer)

Recibe:
Un puntero a un buffer donde se va a crear el mensaje ACK

Retorna:
El tamaño del mensaje generado

Funcionamiento:
Se asigna al arreglo de punteros los parámetros necesarios para formar un mensaje ACK, colocando siempre fin_string después de cada asignación. Además se tiene el tamaño en byte del mensaje que fue generado.

Int *generar_byte*(u_int8_t * buffer)

Recibe:
Un puntero a un buffer donde se va a crear el mensaje byte

Retorna:
El tamaño del mensaje generado

Funcionamiento:
Se asigna al arreglo de punteros los parámetros necesarios para formar un mensaje BYE, colocando siempre fin_string después de cada asignación. Además se tiene el tamaño en byte del mensaje que fue generado.

Int *generar_register*(u_int8_t * buffer)

Recibe:
Un puntero a un buffer donde se va a crear el mensaje byte

Retorna:
El tamaño del mensaje generado

Funcionamiento:

Se asigna al arreglo de punteros los parámetros necesarios para formar un mensaje register, colocando siempre fin_string después de cada asignación. Además se tiene el tamaño en byte del mensaje que fue generado.

int *generar_respuesta*(int respuesta, u_int8_t * buffer, u_int8_t * buffer_out)

Recibe:
Un entero que le indica que tipo de respuesta debe generar, la dirección del buffer donde se va a crear el mensaje buffer_out.

Retorna:
El tamaño en bytes del mensaje generado

Funcionamiento:
Esta función es llamada para generar un mensaje 200, 486 o 180 dependiendo del parámetro respuesta que se le pase a la función, al arreglo de punteros se le asigna los parámetros del mensaje que sean necesarios según sea el caso.

int *generar_200*(u_int8_t* buffer, u_int8_t* buffer_out)

Esta función hace un llamado a la función generar_respuesta, el cual genera el mensaje 200.

void *sip_init*(void)

Esta función es llamada en la iniciación de una sesión Sip, colocando el estado a registrando, y activando un temporizador de 3 segundos.

int *estadoSIP*(void)

Función que retorna el estado actual en que se encuentra el teléfono.

void *check_sip*(void)

Esta función verifica si el tiempo del temporizador no se ha agotado, si el tiempo es agotado entonces el estado regresa a espera y se llama a la función fin_repicar para que finalice sus acciones.

int *recibir_pack_sip*(u_int8_t * buffer, u_int8_t * buffer_out)

Recibe:
Un buffer donde se encuentra el mensaje recibido

Retorna:
El tamaño del mensaje generado.

Funcionamiento:
Esta función está hecha para que en función de su estado y el tipo de mensaje que reciba, genere una respuesta o llame a la función correspondiente. Cambia el estado en que se encuentra y genera un temporizador según sea el caso.

Código SIP.C

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include "sip.h"
5
6 unsigned char *REGISTER_primera_linea={"REGISTER
sip:192.168.10.7 SIP/2.0"};
7 unsigned char *INVITE_primera_linea = {"INVITE sip:192.168.10.7
SIP/2.0"};
8 unsigned char *ACK_primera_linea={"ACK sip:192.168.10.7
SIP/2.0"};
9 unsigned char *BYE_primera_linea={"BYE sip:192.168.10.7
SIP/2.0"};
10
11 unsigned char *primera_linea180={"SIP/2.0 180 Repicando"};
12 unsigned char *primera_linea200={"SIP/2.0 200 OK"};
13 unsigned char *primera_linea486={"SIP/2.0 486 Ocupado"};
14
15 unsigned char *array[26]; //un arreglo de punteros
16
17 //***************************Mensaje
header***************************
18 unsigned char *via={"Via: SIP/2.0/UDP
192.168.10.7;branch=z9hg4bk80d5fe006aeddd11a7950008a1641510"};
19 unsigned char *from={"From: luis
blanco<sip:luis_blanco_1@1ucab.edu.ve>;tag=7010"};
20 unsigned char *to={"To:<sip:victor_suarez_1@ucab.edu.ve>"};
21 unsigned char
*to_tag={"To:<sip:victor_suarez_1@ucab.edu.ve>;tag=92874"};
22 unsigned char *call_id={"Call-id:
80d5fe00-6aed-dd11@ucab.edu.ve"};
23 unsigned char *cseq_bye={"Cseq: 6 BYE"};
24 unsigned char *cseq_register={"Cseq: 6 REGISTER"};
25 unsigned char *cseq_ack={"Cseq: 6 ACK"};
26 unsigned char *cseq_invite={"Cseq: 6 INVITE"};
27 unsigned char *contact={"Contact:
<sip:luis_blanco_1@ucab.edu.ve>"};
```

```
28 unsigned char *max_forwards={"Max-forwards: 70"};
29 unsigned char *user_agent={"User-agent: Ucab phone 1"};
30 //unsigned char *allow={"INVITE, ACK, BYE"};
31 unsigned char *content_type={"Content-type: application/SDP"};
32 unsigned char *content_length={"Content-length: 143"};
33 //*************************Mensaje body--
SDP***********************
34 unsigned char *version={"v=0"};
35 unsigned char *owner={"o=- 3442329143 3442329143 IN IP4
192.168.10.5"};
36 unsigned char *session={"s= Ucab IPphone"};
37 unsigned char *info_conex={"c=IN IP4 192.168.10.5"};
38 unsigned char *time_description={"t=0 0"};
39 unsigned char *media_description={"m=audio 5004 rtp/avp 0"};
40 unsigned char *media_atribute={"a=rtpmap:0 pcmu/8000"};
41
42 unsigned char fin_string[3]={0xd,0xa,0x00};
43
44 unsigned int estado = 0;
45 unsigned int tiempo;
46
47 #if 0
48 char matriz[20][20] = {"Call-ID", "i", \
49 "Contact", "m", \
50 "Cseq", \
51 "From", "f", \
52 "To", "t", \
53 "Via", "v", \
54 "Max-Forwards", \
55 "Allow", \
56 "Content-Length", "l", \
57 "Content-Type", "c" };
58
59 void (*matrizFunc[20])(char*)= { \
60 Call_ID_Func, \
61 Call_ID_Func, \
62 Contact_Func, \
63 Contact_Func, \
64 Cseq_Func, \
65 From_Func, \
66 From_Func, \
67 To_Func, \
68 To_Func, \
69 Via_Func, \
70 Via_Func, \
71 Max_Forwards_Func, \
72 Allow_Func, \
73 Content_Length_Func, \
74 Content_Length_Func, \
75 Content_Type_Func, \
76 Content_Type_Func, };
77 #endif
78
79 int generar_invite( u_int8_t * buffer ){
80 int i;
81 int longitud_msg = 0;
82
83 //primera linea
84 strcpy((void*)buffer, (void*)INVITE_primera_linea);
```

```
85 strcat((void*)buffer,(void*)fin_string);
86 longitud_msg = strlen(INVITE_primera_linea) +
strlen(fin_string);
87
88 //****header*******
89 array[1]=via;
90 array[2]=from;
91 array[3]=to;
92 array[4]=call_id;
93 array[5]=cseq_invite;
94 array[6]=contact;
95 array[7]=max_forwards;
96 array[8]=content_type;
97 array[9]=content_length;
98 //*****body SDP*********
99 array[10]=version;
100 array[11]=owner;
101 array[12]=session;
102 array[13]=info_conex;
103 array[14]=time_description;
104 array[15]=media_description;
105 array[16]=media_atribute;
106
107 for(i = 1; i <= 16; i++)
108 {
109 strcat((void*)buffer,*(array+i));
110 strcat((void*)buffer,(void*)fin_string);
111 longitud_msg += strlen(*(array+i)) + strlen(fin_string);
112 }
113
114 cambio_estado(estado,LLAMANDO);
115 tiempo = get_bios_time();
116 set_bios_time_end(tiempo,30);
117 return longitud_msg;
118 }
119
120 int generar_ack( u_int8_t * buffer ){
121 int i;
122 int longitud_msg = 0;
123
124 //primera linea
125 strcpy((void*)buffer, (void*)ACK_primera_linea);
126 strcat((void*)buffer,(void*)fin_string);
127 longitud_msg = strlen(ACK_primera_linea) +
strlen(fin_string);
128
129 //****header*******
130 array[1]=via;
131 array[2]=from;
132 array[3]=to_tag;
133 array[4]=call_id;
134 array[5]=cseq_ack;
135 array[6]=contact;
136 array[7]=max_forwards;
137
138 for(i = 1; i <= 7; i++)
139 {
140 strcat((void*)buffer,*(array+i));
141 strcat((void*)buffer,(void*)fin_string);
```

```
142 longitud_msg += strlen(*(array+i)) + strlen(fin_string);
143 }
144 return longitud_msg;
145 }
146
147 int generar_bye( u_int8_t * buffer ){
148 int i;
149 int longitud_msg;
150
151 //primera linea
152 strcpy((void*)buffer, (void*)BYE_primera_linea);
153 strcat((void*)buffer,(void*)fin_string);
154 longitud_msg = strlen(BYE_primera_linea) +
strlen(fin_string);
155
156 //****header*******
157 array[1]=via;
158 array[2]=from;
159 array[3]=to_tag;
160 array[4]=call_id;
161 array[5]=cseq_bye;
162 array[6]=max_forwards;
163
164 for(i = 1; i <= 6; i++)
165 {
166 strcat((void*)buffer,*(array+i));
167 strcat((void*)buffer,(void*)fin_string);
168 longitud_msg += strlen(*(array+i)) + strlen(fin_string);
169 }
170 return longitud_msg;
171 }
172
173 int generar_register( u_int8_t * buffer ){
174 int i;
175 int longitud_msg;
176 //primera linea
177 strcpy((void*)buffer, (void*)REGISTER_primera_linea);
178 strcat((void*)buffer,(void*)fin_string);
179 longitud_msg = strlen(REGISTER_primera_linea) +
strlen(fin_string);
180
181 //****header*******
182 array[1]=via;
183 array[2]=from;
184 array[3]=to_tag;
185 array[4]=call_id;
186 array[5]=cseq_register;
187 array[6]=contact;
188 array[7]=max_forwards;
189 array[8]=user_agent;
190
191 for(i=1; i <= 8; i++)
192 {
193 strcat((void*)buffer,*(array+i));
194 strcat((void*)buffer,(void*)fin_string);
195 longitud_msg += strlen(*(array+i)) + strlen(fin_string);
196 }
197 return longitud_msg;
198 }
```

```
199
200
201 int generar_respuesta(int respuesta, u_int8_t * buffer,
u_int8_t * buffer_out){
202 int i;
203 int longitud_msg;
204
205 switch(respuesta){
206 case 200:
207 strcpy((void*)buffer_out, (void*)primera_linea200);
208 strcat((void*)buffer_out,(void*)fin_string);
209 longitud_msg = strlen(primera_linea200) +
strlen(fin_string);
210 array[9] = NULL;
211 break;
212 case 486:
213 strcpy((void*)buffer_out, (void*)primera_linea486);
214 strcat((void*)buffer_out,(void*)fin_string);
215 longitud_msg = strlen(primera_linea486) +
strlen(fin_string);
216 array[9] = NULL;
217 break;
218 case 180:
219 strcpy((void*)buffer_out, (void*)primera_linea180);
220 strcat((void*)buffer_out,(void*)fin_string);
221 longitud_msg = strlen(primera_linea180) +
strlen(fin_string);
222
223 array[9]=content_type;
224 array[10]=content_length;
225 //*****body SDP********
226 array[11]=version;
227 array[12]=owner;
228 array[13]=session;
229 array[14]=info_conex;
230 array[15]=time_description;
231 array[16]=media_description;
232 array[17]=media_atribute;
233 array[18]=NULL;
234
235 }
236 array[1]=via;
237 array[2]=from;
238 array[3]=to_tag;
239 array[4]=call_id;
240 array[5]=cseq_ack;
241 array[6]=contact;
242 array[7]=max_forwards;
243 array[8]=user_agent;
244
245
246 for( i=1; array[i] != NULL; i++)
247 {
248 strcat((void*)buffer_out,*(array+i));
249 strcat((void*)buffer_out,(void*)fin_string);
250 longitud_msg += strlen(*(array+i)) + strlen(fin_string);
251 }
252
253 return longitud_msg;
```

```
254 }
255
256 int generar_200(u_int8_t* buffer, u_int8_t* buffer_out){
257 return (generar_respuesta(200,buffer,buffer_out));
258 }
259
260 void sip_init(void){
261 estado = REGISTRANDO;
262 tiempo = get_bios_time();
263 set_bios_time_end(tiempo,3);
264 }
265
266 int estadoSIP(void){
267 return estado;
268 }
269
270 void check_sip(void){
271 if( check_time_end(tiempo) && (estado != LLAMADA) ){
272 estado = ESPERA;
273 fin_repicar();
274 }
275 }
276
277 int recibir_pack_sip( u_int8_t * buffer, u_int8_t *
buffer_out){
278
279 if( (memcmp((void*)buffer,"INVITE",6) == 0) && (estadoSIP()
== ESPERA) ){
280 cambio_estado(estado,REPICANDO);
281 tiempo = get_bios_time();
282 set_bios_time_end(tiempo,30);
283 repicar();
284 return( generar_respuesta(180, buffer, buffer_out) );
285 }
286 else if( (memcmp((void*)buffer,"BYE",3) == 0) &&
(estadoSIP() == LLAMADA) ){
287 cambio_estado(estado,ESPERA);
288 return ( generar_respuesta(200, buffer, buffer_out) );
289 }
290 else if( (memcmp((void*)buffer,"ACK",3) == 0) &&
(estadoSIP() == LLAMADA) ){
291 return 0;
292 }
293 else if( (memcmp((void*)(buffer+8),"180",3) == 0) &&
(estadoSIP() == LLAMANDO) ){
294 cambio_estado(estado,PROCESANDO);
295 tiempo = get_bios_time();
296 set_bios_time_end(tiempo,30);
297 return 0;
298 }
299 else if( (memcmp((void*)(buffer+8),"200",3) == 0) && \
300 ( (estadoSIP() == LLAMANDO) || (estadoSIP() ==
PROCESANDO)) ){
301 cambio_estado(estado,LLAMADA);
302 fin_repicar();
303 return ( generar_ack( buffer_out ) );
304 }
305 else if( (memcmp((void*)(buffer+8),"487",3) == 0) &&
(estadoSIP() == LLAMANDO) ){
```

```
cambio_estado(estado,ESPERA);
return 0;
}
return 0;
}
```

Telef.c

Programa principal del teléfono donde se presentan las principales llamadas a las funciones, establecimiento de sesiones udp, recepción de paquetes entre otros. Utiliza las librerias estandar stdlib.h, dtdio.h, conio.h, dos.h, string.h, la librería de WATTCP tcp.h y las personales rtp.h y sip.h.

Macros y definiciones:

Tamaño de los buffer muestreo de voz (IBuf_Size) colocación del payload de RTP provenientes de la red (OBuf_Size). Estos buffer son apuntados por un FAR * cada uno, y son utilizados por el programa residente voip.exe para conversiones A/D y D/A

- #define *IBuf_Size* 8192
- #define *OBuf_Size* 8192
- typedef unsigned char *u_int8_t*

- extern unsigned char far * *tempo_rtp_in* : apunta al inicio del Obuffer sea cual sea que se este manejando en el momento, la interrupción cambia automáticamente la localidad de destino o del buffer D/A, este puntero indica de cual buffer se pueden sacar los bytes que van a ser enviados al usuario.
- extern unsigned char far * *tempo_rtp_out*: funciona de la misma forma que el anterior pero apunta al Ibuffer el cual tiene los bytes que van a ser utilizados para generar los paquetes RTP.

Punteros a interrupciones originales, se utilizan para que cuando se termine el programa sean cargadas después de finalizar el programa.

- void interrupt (*old_int81) (void);
- void interrupt (*old_int82) (void);

Prototipo de interrupciones nuevas, utilizadas para medir ciertos tiempos.

- void interrupt *my_int81*(void);
- void interrupt *my_int82*(void);

Funciones definidas en este programa:

void interrupt my_int81(void)
Se ejecuta una vez cada 8192 * 125 uSeg = 1,024 seg, cambia el puntero que indica el inicio del buffer de conversion Analogico/Digital (IBuf)

void interrupt my_int82(void)
Se ejecuta una vez cada 8192 * 125 uSeg = 1,024 seg, cambia el puntero que indica el inicio del buffer de conversión Digital/Analogico (OBuf)

void unload(void)
Función de salida del programa, recompone la interrupción 81h y 82h como se encontraban al inicio del programa.

Función principal main()

- unsigned char far **in_audio_buf_ptr*;
- unsigned char far **out_audio_buf_ptr*;

Temporizador para envio de paquetes RTP cada 20 mSeg

- unsigned int far **timer20mseg*

Buffer para sockets WATTCP, WATTCP recibe los paquetes y los almacena hasta su lectura por la funcion sock_recv()

- u_int8_t *in_buf_sip* [8192];
- u_int8_t *in_buf_rtp* [8192];

La funcion *sock_recv() de* WATTCP lee los paquetes de entrada (RTP,SIP)y los almacena en el buffer_in

- u_int8_t *buffer_in* [1024];

Los paquetes de salida, tanto de RTP como de SIP, son almacenados temporalmente en buffer_out, hasta ejecutar la funcion sock-fastwrite

- u_int8_t *buffer_out* [1024];

Se definen los sockets tanto para SIP como para RTP

- udp_Socket *rtp_stream_data*;
- udp_Socket *sip_socket*;

Locación remota de destino (IP)

- longword *remote_address* = 0;

Puertos utilizados por las sesiones SIP como RTP

- word *rtp_local_port* = 5004;
- word *rtp_remote_port* = 5004;
- word *sip_local_port* = 5060;
- word *sip_remote_port* = 5060;

Variables temporales

- word *temp_length*;
- int *temp*;

Esta función le indica al programa cuales son los últimos comandos que tiene que realizar antes de salir del programa

- atexit(unload);

La función de WATTCP que transforma una IP puntiada a un valor binario

- remote_address = inet_addr("255.255.255.255");

Cambio de los punteros de las interrupciones 81h y 82h

- old_int81 = getvect(0x81);
- old_int82 = getvect(0x82);
- setvect(0x81, my_int81);
- setvect(0x82, my_int82);

Inicio de WATTCP

- sock_init();

El puntero a la interrupcion 80h contiene el inicio del buffer A/D

- in_audio_buf_ptr = (unsigned char far *) getvect(0x80);

Se obtiene el puntero al inicio del buffer D/A

- out_audio_buf_ptr = (unsigned char far *)(in_audio_buf_ptr + 2*IBuf_Size);

El puntero al timer de 20 mSeg

- timer20mseg = (unsigned int far *)(out_audio_buf_ptr + 2*OBuf_Size);

Abriendo el socket de UDP para SIP

- If(!udp_open(&sip_socket, sip_local_port, remote_address, sip_remote_port, NULL))

Función de WATTCP que prepara el buffer de recepción

- sock_recv_init(&sip_socket, in_buf_sip, sizeof(in_buf_sip))
- Sock_mode(&rtp_stream_data, UDP_MODE_NOCHK);

Función que se encuentra en SIP.c e inicia los parámetros de una sesión SIP

- sip_init()

Ciclo infinito de recepción y envío de paquetes, esta parte del programa se encarga de verificar los sockets abiertos usando la función de WATTCP tcp_tick(NULL), y en la recepción de los paquetes usando sock_recv tanto para SIP como para RTP, si en algunos de los dos recibe un paquete este llama a su funciones correspondientes ya sea recibir_pack_rtp(buffer_in) o en el caso de SIP utilizando la función recibir_paquete_sip, este mensaje puede generar genera alguna respuesta, la cual es enviada de inmediato utilizando la función de WATTCP sock_fastwrite(&sip_socket, buffer_out, temp).

Se está verificando constantemente si el temporizador es de 20 mlseg es mayor o igual a uno, si es mayor indica que se debe enviar un mensaje RTP, una vez enviado el paquete se reduce este contador

Se lee el estado del botón para SIP leerBOTON(); si se presiona el botón y el estado de SIP es ESPERA, se genera y envía un mensaje INVITE, además se abre un socket para RTP. Si se presiona el botón y el estado de SIP es llamada, se genera un BYE y se cierra el socket y regresa al estado de espera. Si el botón no se presiona pero el estado es REGISTRANDO se chequea el timer de SIP y si se presiona el botón, en el caso en que el estado de SIP es REPICANDO, se genera un 200 y se abre el socket RTP

Código TELEF.C

```
3 /* 14/02/07 TELEF.C
4 * Vsuarez
5 * LBlanco
6 * Telefono IP
9 * Session Initiaton Protocol (SIP)
10 * Señalizacion:
11 * REGISTER
12 * INVITE
13 * BYE
14 * ACK
15 * mensajes de respuesta
16 * 180 repicando
17 * 200 ok
18 * 486 ocupado
19 *
20 * Real Time Protocol(RTP)
21 * Envio de paquetes cada 20 mseg
22 * Recepción de paquetes
23 *
24 */
25
26 #define debug 1
27
28 // Librerias estandar
29 #include <stdlib.h>
30 #include <stdio.h>
31 #include <conio.h>
32 #include <dos.h>
33 #include <string.h>
34 // Libreria de WATTCP
35 #include "tcp.h"
36 // Librerías personales
37 #include "rtp.h"
38 #include "sip.h"
39
40 /* tamaño de los buffer
41 muestreo de voz (OBuf_Size)
42 colocación del payload de RTP (IBuf_Size)
43 Estos buffer son apuntados por un FAR * cada uno,
44 y son utilizados por el programa residente voip0X.exe
45 para conversiones A/D y D/A
46 */
47 #define IBuf_Size 8192
48 #define OBuf_Size 8192
49
50 typedef unsigned char u_int8_t;
51
52 //RTP
53 /* punteros a IBuf y a OBuf (utilizados para conversiones D/A y
A/D, respectivamente)
54 estan declarados en rtp.c
55 */
56 extern unsigned char far * tempo_rtp_in;
57 extern unsigned char far * tempo_rtp_out;
58
59 //punteros a interrupciones antiguas
```

```
60 void interrupt (*old_int81) (void);
61 void interrupt (*old_int82) (void);
62
63 //prototipo de interrupciones nuevas
64 void interrupt my_int81(void);
65 void interrupt my_int82(void);
66
67 /*
68 se ejecuta una vez cada 8192 * 125 uSeg = 1,024 seg
69 cambia el puntero que indica el inicio del buffer de
70 conversion Analogico/Digital IBuf
71 */
72 void interrupt my_int81(void)
73 {
74 tempo_rtp_out = MK_FP(_DX,_AX);
75 asm {
76 mov ax,0xB800
77 mov es,ax
78 mov bx,0
79 inc word ptr es:[bx]
80 }
81 };
82
83 /*
84 se ejecuta una vez cada 8192 * 125 uSeg = 1,024 seg
85 cambia el puntero que indica el inicio del buffer de
86 conversion Digital/Analogico OBuf
87 */
88 void interrupt my_int82(void)
89 {
90 tempo_rtp_in = MK_FP(_DX,_AX);
91 asm {
92 mov ax,0xB800
93 mov es,ax
94 mov bx,8
95 inc word ptr es:[bx]
96 }
97
98 };
99
100 /*
101 funcion de salida del programa
102 recompone la interrupción 81h y 82h como
103 se encontraban al inicio del programa
104 */
105 void unload(void){
106 setvect( 0x081, old_int81 );
107 setvect( 0x082, old_int82 );
108 }
109
110 int main(void){
111
112 unsigned char far *in_audio_buf_ptr;
113 unsigned char far *out_audio_buf_ptr;
114 //timer para envio de paquetes RTP cada 20 mSeg
115 unsigned int far *timer20mseg;
116 //buffer para sockets WATTCP
117 //WATTCP recibe los paquetes y los almacena
118 //hasta su lectura por la funcion sock_recv()
```

```
119 u_int8_t in_buf_sip [8192];
120 u_int8_t in_buf_rtp [8192];
121
122 //la funcion sock_recv() lee los paquetes de entrada
(RTP,SIP)
123 //y los almacena en el buffer_in
124 u_int8_t buffer_in [1024];
125 //los paquetes de salida, tanto de RTP como de SIP
126 //son almacenados temporalmente en buffer_out
127 //hasta ejecutar la funcion sock-fastwrite
128 u_int8_t buffer_out [1024];
129
130 //sockets
131 udp_Socket rtp_stream_data;
132 udp_Socket sip_socket;
133 //locación remota de destino
134 longword remote_address = 0;
135 //puertos
136 word rtp_local_port = 5004;
137 word rtp_remote_port = 5004;
138 word sip_local_port = 5060;
139 word sip_remote_port = 5060;
140
141 //variables temporales
142 word temp_length;
143 int temp;
144
145 atexit(unload);
146
147 // remote_address = inet_addr("192.168.10.3");
148 remote_address = inet_addr("255.255.255.255");
149
150 //cambio de los punteros de las interrupciones 81h y 82h
151 old_int81 = getvect(0x81);
152 old_int82 = getvect(0x82);
153 setvect(0x81, my_int81);
154 setvect(0x82, my_int82);
155
156 //inicio de WATTCP
157 sock_init();
158
159 //el puntero a la interrupcion 80h contiene el inicio del
buffer A/D
160 in_audio_buf_ptr = (unsigned char far *) getvect(0x80);
161 printf("buffer entrada %x
%x\n",FP_SEG(in_audio_buf_ptr),FP_OFF(in_audio_buf_ptr) );
162
163 //se obtiene el puntero al inicio del buffer D/A
164 out_audio_buf_ptr = (unsigned char far *)(in_audio_buf_ptr +
2*IBuf_Size);
165 printf("buffer salida %x
%x\n",FP_SEG(out_audio_buf_ptr),FP_OFF(out_audio_buf_ptr) );
166
167 //y el puntero al timer de 20 mSeg
168 timer20mseg = (unsigned int far *)(out_audio_buf_ptr +
2*OBuf_Size);
169 printf("timer ptr %x
%x\n",FP_SEG(timer20mseg),FP_OFF(timer20mseg) );
170 printf("valor timer rtp %x H %u d\n",
```

```
*timer20mseg,*timer20mseg);
171
172 //inicialización de socket SIP
173 if( !udp_open( &sip_socket, \
174 sip_local_port, \
175 remote_address, \
176 sip_remote_port, \
177 NULL ) ) {
178 puts("Could not open sip socket");
179 return EXIT_FAILURE;
180 }//end if
181
182 if( sock_recv_init( &sip_socket, \
183 in_buf_sip, \
184 sizeof(in_buf_sip) ) ) {
185 puts("Could not enable large buffer");
186 return EXIT_FAILURE;
187 }//end if
188
189 sock_mode( &rtp_stream_data, UDP_MODE_NOCHK );
190
191 (*timer20mseg) = 0;
192 sip_init();// no envia el mensaje de registro, aún
193
194 //ciclo infinito de recepción y envio de paquetes
195 do{
196 //funcion de WATTCP, para revisar los sockets abiertos
197 tcp_tick(NULL);
198
199 //chequeo de recepcion, SIP y RTP
200 if( temp_length = sock_recv( &rtp_stream_data, \
201 buffer_in, \
202 sizeof(buffer_in), \
203 0 ) && rtp_estado() ==
ACTIVO) {
204 //si temp_length != 0 se recibe un paquete RTP
205 //además rtp_estado debe ser igual a ACTIVO
206 recibir_pack_rtp(buffer_in);
207 printf("Got %u byte packet rtp\n", temp_length);
208 }
209
210 if( temp_length = sock_recv( &sip_socket, \
211 buffer_in, \
212 sizeof(buffer_in), \
213 0 ) ) {
214 //si temp_length != 0 se recibe un paquete SIP
215 temp = recibir_pack_sip(buffer_in, buffer_out);
216 //si temp != 0 la recepción del paquete generó una
respuesta
217 //que se envia de inmediato
218 if( temp != 0 )
219 sock_fastwrite( &sip_socket, buffer_out, temp );
220 }
221
222 //transmision de RTP
223 //si el timer de 20 mSeg es mayor o igual a uno
224 //se debe enviar un paquete RTP
225 if( (*timer20mseg) >= 1 ){
226 temp = crear_pack_rtp( buffer_out );
```

```
227 sock_fastwrite( &rtp_stream_data, buffer_out, temp );
//corregir aqui
228 asm {
229 mov ax,0xB800
230 mov es,ax
231 mov bx,12
232 inc word ptr es:[bx]
233 }
234 //luego del envio se resta 1 al timer para esperar
hasta la siguiente
235 //cuenta de 20 mSeg
236 (*timer20mseg)--;
237 }
238
239 //se lee el estado del boton, para SIP
240 if( leerBOTON() ){
241 /* si se presiona el boton y el estado de SIP es
ESPERA
242 * se genera y envía un mensaje INVITE
243 * además se abre un socket para RTP
244 */
245 if(estadoSIP() == ESPERA){
246 temp = generar_invite(buffer_out);
247 sock_fastwrite( &rtp_stream_data, buffer_out, temp
);
248 //socket
249 if( !udp_open( &rtp_stream_data, \
250 rtp_local_port, \
251 remote_address, \
252 rtp_remote_port, \
253 NULL ) ) {
254 puts("Could not open rtp socket");
255 return EXIT_FAILURE;
256 }//end if
257
258 if( sock_recv_init( &rtp_stream_data, \
259 in_buf_rtp, \
260 sizeof(in_buf_rtp) ) ) {
261 puts("Could not enable large buffer");
262 return EXIT_FAILURE;
263 }//end if
264 rtp_init();
265 (*timer20mseg) = 0;
266 }
267 /* si se presiona el boton y el estado de SIP es
LLAMADA
268 * se genera un BYE
269 * y se cierra el socket RTP
270 */
271 else if( estadoSIP() == LLAMADA ){
272 temp = generar_bye(buffer_out);
273 sock_fastwrite( &sip_socket, buffer_out, temp );
274 sock_close( &rtp_stream_data );
275 rtp_fin(); // el estado de RTP se coloca en
DESACTIVO
276 }
277 /* si se presiona el boton y el estado de SIP es
REPICANDO
278 * se genera un 200
```

```
279 * y se abre el socket RTP
280 */
281 else if( estadoSIP() == REPICANDO ){
282 temp = generar_200(buffer_in,buffer_out);
283 //socket
284 if( !udp_open( &rtp_stream_data, \
285 rtp_local_port, \
286 remote_address, \
287 rtp_remote_port, \
288 NULL ) ) {
289 puts("Could not open rtp socket");
290 return EXIT_FAILURE;
291 }//end if
292
293 if( sock_recv_init( &rtp_stream_data, \
294 in_buf_rtp, \
295 sizeof(in_buf_rtp) ) ) {
296 puts("Could not enable large buffer");
297 return EXIT_FAILURE;
298 }//end if
299 rtp_init();
300 (*timer20mseg) = 0;
301 }
302 }
303 //si el boton no se presiona pero el estado es
REGISTRANDO
304 //se chequea el timer de SIP
305 else if( estadoSIP() == REGISTRANDO ){
306 check_sip();
307 }
308 //si el boton no se presiona pero el estado es ESTADO
309 //se chequea el timer de SIP
310 else if( estadoSIP() != ESPERA ){
311 check_sip();
312 }
313 }
314 while( !kbhit() );
315
316 getch();
317 return EXIT_SUCCESS;
318 }
```

APÉNDICE 4. TELÉFONO IP, VERSIÓN 2

VOIPV2.ASM

El código VOIPV2.ASM contiene la interrupción que se ejecutará cada 125 μseg. Y las funciones que esta utilizará para el muestreo analógico a digital y digital a analógico, así como el almacenamiento de las muestras de voz codificadas y la obtención de muestras codificadas para su envío al conversor digital a analógico.

El código asume que el hardware de conversión utiliza el puerto paralelo.

Posee además un programa inicial, que se encarga de dejar residente la interrupción y sus funciones, cambiar la velocidad de la interrupción 8 para que se ejecuta cada 125 μseg y modificar el vector de interrupciones de las interrupciones $80_{(16)}$, $81_{(16)}$ y $82_{(16)}$. El vector de interrupción $80_{(16)}$ se hace apuntar al principio de los datos de la interrupción, más específicamente al inicio del vector de almacenamiento de las muestras codificadas del conversor analógico a digital. Las interrupciones $81_{(16)}$ y $82_{(16)}$ se modifican y se hacen apuntar a una interrupción vacía, pero pueden ser utilizadas para fines de corrección.

Las rutinas del código se esquematizan a continuación

- Código Inicial
- DA_Sample
- AD_Sample
- TestOBuffer
- TestIBuffer
- OBuffer2DA
- AD2IBuffer
- DummyISR
- New_ISR08

Las definiciones hechas en este código son las siguientes.

- IB_Size = 1600 : Tamaño del vector de almacenamiento de muestras para el conversor Analógico a Digital (IBuffer)
- OB_Size = 8000 : Tamaño para el vector de almacenamiento de muestras para el conversor Digital a Analógico (OBuffer)
- PSP = 16 :

- Xtal = 3579545 : Frecuencia aproximada de reloj de hardware, 3.579.353 Hz
- T_Sample = 125 : Periodo de la interrupción, 125 µseg
- T0_Reload = (T_Sample * Xtal)/(3 * 1000000) : Cálculo para adicionar a variable, y mantener la interrupción 8 aproximadamente a periodo constante y aproximadamente igual al periodo que mantenia antes de colocar como nueva interrupción 8 a la rutina New_ISR8.
- LPT0 = 378h : Dirección base del Puerto paralelo
- LPT_DATA = LPT0 + 0 : Dirección puerto paralelo, pines del 2 al 9
- LPT_STATUS = LPT0 + 1 : Dirección de puerto paralelo, pines 10, 11, 12, 13 y 15.
- LPT_CONTROL = LPT0 + 2 : Dirección base de puerto paralelo, pines 1, 14, 16 y 17.
- IB_Trigger = 81h : Definición para el llamado a la Interrupción 81h
- OB_Trigger = 82h : Definición para el llamado a la Interrupción 82h

Las variables utilizadas en VOIPV2.ASM son las siguientes:

- IBuffer
- Vector de bytes de tamaño igual a IB_SIZE, inicialmente de valores iguales a cero
- OBuffer
- Vector de bytes de tamaño igual a OB_SIZE, inicialmente de valores iguales a cero
- Timer1mseg dw 0
- Variable de 16 bits que se incrementa una vez cada ocho ejecuciones de la rutina New_ISR8, lo que se traduce en un incremento cada milisegundo.
- switch_AD
- Variable de 8 bits, si su valor es cero (valor inicial es cero) la rutina New_ISR8 no intentará realizar conversiones Analógico a Digital. Con cual quier otro valor distinto de cero, la rutina realizará conversiones Analógico a Digital.
- Ibuffer_fin
- Variable que almacena el desplazamiento de memoria del final del vector IBuffer.
- Obuffer_fin
- Variable que almacena el desplazamiento de memoria del final del vector OBuffer.
- RTP_Deq_IB
- Variable que almacena el desplazamiento de memoria de donde se pueden obtener nuevos datos del vector OBuffer. La rutina

New_ISR8 no modifica esta variable, y esta pensada para ser utilizada por un programa externo.

- RTP_Enq_OB
- Variable que almacena el desplazamiento de memoria donde se pueden guardar nuevos datos en el vector OBuffer. La rutina New_ISR8 no modifica esta variable, y esta pensada para ser utilizada por un programa externo.
- IB_Enq dw IBuffer
- Variable que almacena el desplazamiento de memoria donde se pueden guardar nuevos datos en el vector IBuffer.

- OB_Deq dw OBuffer
- Variable que almacena el desplazamiento de memoria de donde se pueden obtener nuevos datos del vector OBuffer.
- MuestrasAD
- Variable que almacena la cantidad de muestras codificadas que mantiene el vector IBuffer.
- Old_ISR08 dd 0
- Variable de 32 bits que almacena el puntero lejano de la interrupción 8 anterior. Se utiliza para el encadenamiento de las interrupciones.
- AuxCounter y AuxCounter2
- Contadores auxiliares de 16 bits.

Explicación de rutinas

DummyISR

Funcionamiento:

Rutina vacía, cuya única instrucción es una retorno de rutina. Se utiliza como rutina por defecto para las interrupciones $81_{(16)}$ y $82_{(16)}$.

AD_Sample

Funcionamiento:

Rutina que obtiene el valor codificado de una muestra analógica. Trabaja por aproximaciones sucesivas; a cada aproximación realizada revisa por el resultado de la comparación para continuar con la siguiente aproximación.

La primera comparación siempre es realizada con el bit más significativo en alto, si el resultado de la comparación indica que el valor de la aproximación es

mayor al valor comparado, el bit que está actualmente en uso para la comparación se coloca en cero, en caso contrario –la aproximación es menor que el valor comparado- el bit no se altera; en cualquier de los resultados de la comparación se coloca el siguiente bit menos significativo en alto para la siguiente comparación. Cuando todos los bits del byte sean utilizados para la comparación el ciclo termina y el valor obtenido se considera como la codificación de la muestra analógica.

```
AD_Sample:
    mov   bx,8000h          ;init value, MSB set, BL = 0
AD_loop:
    or    bl,bh             ;set bit
    mov   dx,LPT_DATA       ;LPT data port
    mov   al,bl
    out   dx,al             ;to D/A
    inc   dx                ;LPT input port (get comparator's output)

    mov   cx,100
    loop  $

    in    al,dx
    xor   al,80h            ;complement D7, on pin LPT.11
    and   al,80h            ;isolate D7
    jnz   @f                ;skip if comparator high, D/A < Vin
    xor   bl,bh             ;too high D/A value, clear bit
@@:
    shr   bh,1              ;next lower bit
    jnz   AD_loop           ;until no more bits left

    mov   al,bl
    ret
```

TestIBuffer , TestOBuffer

Recibe:

En el registro BX el desplazamiento de la posición actual dentro de IBuffer u OBuffer.

Retorna:

El valor en BX inalterado o el inicio de IBuffer u OBuffer (dependiendo de la función) en el registro BX.

Funcionamiento:

Las rutinas TestIBuffer y TestOBuffer comparan el valor en el registro BX contra el valor del fin de IBuffer u OBuffer respectivamente. Si el valor es idéntico, se asigna al registro BX el valor de inicio de IBuffer u OBuffer, en caso contrario el valor en BX es retornado sin modificaciones

DA_Sample

Recibe:

En el registro AL el valor de muestra codificada.

Funcionamiento:

La rutina obtiene el valor en DX del puerto paralelo LPT0 y enviar por ese puerto el valor de la muestra codificada. Luego, por el pin 1 del puerto paralelo se envía un valor en alto y luego un valor en bajo para recrear un borde de bajada que active un integrado de muestreo y retención externo.

AD2IBuffer

Funcionamiento:

La rutina AD2IBuffer (Analógico a digital hacia el vector IBuffer) se encarga principalmente de guardar la muestra digital obtenida en el vector IBuffer.

Primero se llama a la rutina AD_Sample, que retorna el valor codificado de la muestra Analógica en el registro AL, luego se obtiene en el registro BX el valor de la variable IB_enq, que es el valor de desplazamiento en memoria donde se guardará la codificación de la muestra; se guarda el valor del registro AL en la posición de memoria apuntada por el registro BX y se incrementa el valor de BX para seguidamente ejecutar la rutina TestIBuffer que retorna el valor de BX actualizado y este se guarda en la variable IB_enq. Por último, se incrementa la variable MuestrasAD, indicando que hay una nueva muestra en el vector IBuffer.

OBuffer2DA

Funcionamiento:

La rutina principalmente realiza la obtención del siguiente valor que será argumento de la rutina DA_Sample. Esto lo realiza obteniendo la variable OB_Deq en el registro BX, y luego obteniendo el valor apuntado por BX en el registro AL. Este valor el el registro AL es el argumento de la función DA_Sample.

Luego de la llamada a DA_Sample se incrementa el valor en el registro BX y se ejecuta la rutina TestOBuffer para actualizar el valor en BX de la siguiente posición de memoria de donde se obtendrá la siguiente muestra para el conversor D/A. El valor de BX es luego guardado en la variable OB_enq.

Rutina de interrupción, New_ISR8

Funcionamiento:

La rutina de interrupción se encarga de cada 125 μseg de llamar a las rutinas AD2IBuffer y OBuffer2DA, de incrementar un contador cada milisegundo y de llamar a la rutina de interrupción 8 cada 55 mseg. aproximadamente.

Primero la rutina revisa el valor de la variable swtich_AD, si este es cero la rutina AD2IBuffer no es ejecutada, en caso contrario se llama a la rutina AD2IBuffer. Luego se revisa el valor de las variables OB_Deq y RTP_Enq_OB, si sus valores son iguales la rutina OBuffer2DA no es ejecutada, puesto que en ese caso el vector OBuffer visto como una estructura de datos de tipo cola circular se encontraría vacia, con su apuntador de encolamiento igual a su apuntador de desencolado.

Luego se incrementa el contador auxiliar auxcounter2, y si su valor es igual a 8 se le asigna el valor cero y además se incrementa la variable Timer1mseg.

Finalmente se realiza el fin de la interrupción enviando el valor 20 hexadecimal al puerto 20 hexadecimal, y se le suma a la variable AuxCounter2 una cantidad definida como T0_Reload. Cuando el contador AuxCounter2 se desborde indica que se un periodo de aproximadamente 55 mseg se cumplió y que

la vieja rutina de interrupción 8 debe ser ejecutada, por lo que se realiza un brinco lejano utilizando el puntero almacenado en Old_ISR08.

Rutina de inicio, Start

Funcionamiento:

La rutina de carga modifica el vector de interrupciones de la las interrupciones 8 y $80_{(16)}$, así como inicia las variables de la rutina de interrupción y finaliza dejando residente las instrucciones de la rutina de interrupción New_ISR8 y sus funciones asociadas, así como las variables de datos que esta necesita para su funcionamiento.

Al inicio de la rutina de carga el apuntador en el vector de interrupciones a la interrupción $80_{(16)}$ es modificado para que apunte al inicio de los datos de la rutina de interrupción (Inicio de IBuffer). Luego se inician los datos de las variables Ibuffer_fin, Obuffer_fin, RTP_Deq_IB, RTP_Enq_OB y es obtenido el puntero a la rutina 8 actual.

Luego el puntero en el vector de interrupción a la rutina 8 es modificado para que apunte a la nueva rutina 8 - New_ISR8 -. Finalmente él periodo al que es ejecutado la rutina de la interrupción 8 es modificado a 125 µseg. y la rutina finaliza dejando residente a todo el código y los datos de la nueva rutina de interrupción 8.

Código VOIPV2.ASM

```
1 .model small
2 .386
3 .code
45
IB_Size equ 1600 ;tamaño IBuffer
6 OB_Size equ 8000 ;tamaño OBuffer
78
PSP equ 16
9 Xtal equ 3579545
10 T_Sample equ 125 ;125 uSec por muestra
11 T0_Reload equ (T_Sample * Xtal)/(3 * 1000000)
12
13 LPT0 equ 378h
14 LPT_DATA equ LPT0 + 0
15 LPT_STATUS equ LPT0 + 1
16 LPT_CONTROL equ LPT0 + 2
```

```
17
18 ;RUTINA INTERRUPCION New_ISR08-------------------------
19 New_ISR08:
20 pusha
21 push ds
22 ;los segmentos CS y DS se igualan
23 push cs
24 pop ds
25
26 ;se revisa el valor de switch_AD
27 mov al,[switch_AD]
28 cmp al,0
29 je @f
30
31 ;si el valor de SwitchAD != 0
32 ;se llama a la rutina AD2IBuffer
33 call AD2IBuffer
34 @@:
35 ;los valores de OB_Deq y RTP_Enq_OB
36 ;son comparados
37 mov ax,[OB_Deq]
38 cmp ax,word ptr [RTP_Enq_OB]
39 je @f
40
41 ;si OB_Deq != RTP_Enq_OB
42 call OBuffer2DA
43 @@:
44 ;temporizador de 1 mseg
45 inc word ptr [auxcounter2]
46 cmp word ptr [auxcounter2],8
47
48 ;si han sucedido 8 interrupciones
49 ;se incrementa el contador Timer1mseg
50 jne @f
51 inc [Timer1mseg]
52 mov word ptr [auxcounter2],0
53 @@:
54 ;fin de la interrupcion
55 mov al,20h
56 out 20h,al
57 pop ds
58 popa
59
60 ;se agraga el valor T0_Reload(149)
61 ;al contador auxiliar
62 ;si el contador se desborda han transcurrido
63 ;aproximadamente 55 mseg
64 add word ptr cs:[AuxCounter],T0_Reload
65 jc @f
66 iret
67
68 @@:
69 ;si han transcurrido 55 mseg
70 ;se brinca a la vieja rutina de interrupcion 8
71 jmp dword ptr cs:[Old_ISR08]
72
73 ;DATOS-------------------------------------------------
74
75 ;Un puntero lejano en el vector
```

```
76 ;de interrupciones de la
77 ;interrupcion 80h (0:200h)
78 ;apunta hacia el inicio de estos datos
79
80 IBuffer db IB_Size dup (0) ;vector para almacenar
81 ;las muestras
82 ;del conversor A/D
83 OBuffer db OB_Size dup (0) ;vector para almacenar
84 ;las muestras para el
85 ;conversor D/A
86 dummy dw 0
87 Timer1mseg dw 0 ;variable que se incrementa cada mili segundo
88 switch_AD db 0 ;bandera para elegir si la rutina realiza o no
89 ;conversiones A/D
90 ;inicialmente su valor es cero por lo
91 ;que no se realizaran conversiones
92
93 Ibuffer_fin dw 0 ;puntero al fin de IBuffer
94 Obuffer_fin dw 0 ;puntero al fin de OBuffer
95
96 RTP_Deq_IB dw 0 ;puntero que indica donde se
97 ;pueden obtener muestras
98 ;del conversor A/D
99 RTP_Enq_OB dw 0 ;puntero que indica donde se
100 ;pueden guardar muestras
101 ;para el conversor A/D
102
103 IB_Enq dw IBuffer ;puntero hacia IBuffer
104 OB_Deq dw OBuffer ;puntero hacia OBuffer
105
106 MuestrasAD dw 0 ;cantidad de muestras en IBuffer que aun
107 ;no han sido leidas
108
109 Old_ISR08 dd 0 ;puntero a la rutina de la interrupcion 8
anterior
110 AuxCounter dw 0 ;contador auxiliar
111 AuxCounter2 dw 0 ;contador auxiliar
112
113 ;AD2IBuffer-----------------------------------------------------
114 AD2IBuffer:
115 call AD_Sample
116 mov bx,[IB_Enq]
117 mov byte ptr [bx],al
118 inc bx
119 call TestIBuffer
120 mov [IB_Enq],bx
121 inc word ptr [MuestrasAD]
122 ret
123
124 ;OBuffer2DA----------------------------------------------------
125 OBuffer2DA:
126 mov bx,[OB_Deq]
127 mov al,byte ptr [bx]
128 call DA_Sample
129 inc bx
130 call TestOBuffer
131 mov [OB_Deq],bx
132 ret
133
```

```
134 ;TestIBuffer-----------------------------------------------------
135 TestIBuffer:
136 cmp bx,word ptr [Ibuffer_fin]
137 jne @f
138 mov bx,offset IBuffer
139 int 81h
140 ; jmp TestIB001
141 @@:
142 TestIB001:
143 ret
144
145 ;TestOBuffer---------------------------------------------------
146 TestOBuffer:
147 cmp bx,word ptr [Obuffer_fin]
148 jne @f
149 mov bx,offset OBuffer
150 int 82h
151 ; jmp TestOB001
152 @@:
153 TestOB001:
154 ret
155 ;-------------------------------------------------------------
156 AD_Sample:
157 mov bx,8000h ;init value, MSB set,
BL = 0
158 AD_loop:
159 or bl,bh ;set bit
160 mov dx,LPT_DATA ;LPT data port
161 mov al,bl
162 out dx,al ;to D/A
163 inc dx ;LPT input port (get
comparator's output)
164
165 mov cx,100
166 loop $
167
168 in al,dx
169 xor al,80h ;complement D7, on pin
LPT.11
170 and al,80h ;isolate D7
171 jnz @f ;skip if comparator
high, D/A < Vin
172 xor bl,bh ;too high D/A value,
clear bit
173 @@:
174 shr bh,1 ;next lower bit
175 jnz AD_loop ;until no more bits
left
176
177 mov al,bl
178 ret
179
180 ;DA_Sample----------------------------------------------------
181 PIN1 EQU 00000001b
182 DA_Sample:
183
184 ;put data
185 mov dx,LPT0
186 out dx,al
```

```
187
188 ;clock
189 add dx,02 ; port = 037Ah
190 mov al, not PIN1
191 out dx,al
192
193 mov cx,100
194 loop $
195
196 mov al,PIN1
197 out dx,al
198
199 ret
200
201 ;START-----------------------------------------------------
202 Start:
203 cli
204 ;los segmentos CS y DS se igualan
205 push cs
206 pop ds ;DS == CS
207
208 mov ax,2580h ;0:200, INT80, global
ptr to IBuffer0
209 mov dx,offset IBuffer
210 int 21h
211
212 ;init data
213 mov word ptr [Ibuffer_fin],offset IBuffer+IB_Size
214 mov word ptr [Obuffer_fin],offset OBuffer+OB_Size
215
216 mov word ptr [RTP_Deq_IB],offset IBuffer
217 mov word ptr [RTP_Enq_OB],offset OBuffer
218
219 mov ax,3508h
220 int 21h
221 mov word ptr [Old_ISR08],bx
222 mov word ptr [Old_ISR08 + 2],es
223
224 mov ax,2508h
225 mov dx,offset New_ISR08
226 int 21h
227
228 mov al,00110100b
229 out 43h,al ;TIMER0, mode 2, binary
230
231 mov al,low T0_Reload ;TIMER0, change overflow rate
232 out 40h,al
233 mov al,high T0_Reload
234 out 40h,al
235 sti
236
237 mov ax,3100h ;keep resident
238 mov dx,PSP + (offset Start - offset New_ISR08 +
15)/16
239 int 21h
240
241 end Start
```

TIMERS.H

Prototipos de las funciones para el código TIMERS.C.

Los prototipos de las funciones declarados en TIMERS.H son los siguientes

void *timer_init*(unsigned int far *)
unsigned int *set_timer*(timer_struct*, unsigned int mseg)
int *check_timer*(timer_struct*)
unsigned int *wait_timer*(timer_struct*, unsigned int mseg)
unsigned int *set_timer_seg*(timer_struct*, unsigned int mseg)
int *check_timer_seg*(timer_struct*)
unsigned int *wait_timer_seg*(timer_struct*, unsigned int mseg)

El archivo TIMERS.H además define la estructura de datos siguiente

```
typedef struct
{
  int bandera;
  unsigned valor_final;
}timer_struct;
```

Código TIMERS.H

```
1
2//prototipos de funciones para timers.c
3 #ifndef __TIMERS_H__
4 #define __TIMERS_H__
56
typedef struct
7 {
8 int bandera;
9 unsigned valor_final;
10 }timer_struct;
11
12 //funcion de inicialización de variables
13 void timer_init(unsigned int far *);
14
15 //funciones para temporizadores
16 //con precision de milisegundos
17 unsigned int set_timer(timer_struct*, unsigned int mseg);
18 int check_timer(timer_struct*);
19 unsigned int wait_timer(timer_struct*, unsigned int mseg);
20
21 //funciones para temporizadores
22 //con precision de segundos
23 unsigned int set_timer_seg(timer_struct*, unsigned int sseg);
24 int check_timer_seg(timer_struct*);
25 unsigned int wait_timer_seg(timer_struct*, unsigned int seg);
26
27 #endif
```

TIMERS.C

Utiliza las librerías estándar stdlib.h y dos.h, y la librería personalizada timers.h.

Se encuentran definidas 7 funciones encargadas de iniciación general de los temporizadores, iniciación de un temporizador, revisión de expiración de un temporizador y espera activa por la finalización de un temporizador.

Las funciones de iniciación de un temporizador, revisión de expiración de un temporizador y espera activa por la finalización de un temporizador se encuentran en dos versiones, una para temporizadores que utilicen esperas de segundos y otra versión para temporizadores que utilicen esperas de milisegundos. La lista de las funciones en TIMERS.C es la siguiente:

void *timer_init*(unsigned int far *)
unsigned int *set_timer*(timer_struct*, unsigned int mseg)
int *check_timer*(timer_struct*)
unsigned int *wait_timer*(timer_struct*, unsigned int mseg)
unsigned int *set_timer_seg*(timer_struct*, unsigned int mseg)
int *check_timer_seg*(timer_struct*)
unsigned int *wait_timer_seg*(timer_struct*, unsigned int mseg)

Las definiciones de este modulo son las siguientes.

ACTIVO = -1 : estado de temporizador, si se encuentra en este estado el temporizador ha sido asignado para agotarse en un tiempo futuro.
INACTIVO = 0 : estado de temporizador, si se encuentra en este estado el temporizador ya se ha agotado y ha sido revisado previamente.
TRUE = -1
FALSE = 0

Las variables globales son:

unsigned far * *clock55mseg* : puntero lejano a una posición de memoria que se incremente cada 55 mili segundos.
unsigned far * *clock1mseg* = NULL : puntero lejano a una posición de memoria que se incremente cada milisegundo.

Explicación de funciones

void timer_init(unsigned int far * temporizador1mseg)

Recibe:

unsigned int far * temporizador1mseg: puntero lejano a un posición de memoria que se incremente cada mili segundo.

Funcionamiento:

La función timer_init() debe ser llamada antes de utilizar cualquier otra función del módulo TIMERS. Esta función recibe como parámetro un puntero lejano a una posición de memoria que se incremente cada mili segundo. Si el puntero es nulo, no se podrán utilizar las versiones de las funciones en TIMERS.C que trabajen con mili segundos (wait_timer(), set_timer(), check_timer()). Si el puntero es válido, su valor se asigna al puntero global clock1mseg.

La función inicializa el valor del puntero lejano clock55mseg, utilizando la posición de memoria 0x46C, donde se encuentra el valor del temporizador del BIOS en la arquitectura Pentium.

unsigned int set_timer(timer_struct* timer, unsigned int mseg)

unsigned int set_timer_seg(timer_struct* timer, unsigned int seg)

Recibe:

timer_struct* timer: la dirección de memoria de la estructura que almacena el valor de expiración del temporizador

unsigned int mseg ó unsigned int seg : cantidad de milisegundos o de segundos para la finalización del temporizador

Retorna:

El valor final del reloj, en el cual el temporizador expirará.

Funcionamiento:

La función asigna a la variable bandera de la estructura del temporizador el valor de ACTIVO, y a la variable valor final le asigna el valor actual del reloj, mediante el uso del puntero lejano clock1mseg o clock55mseg según sea el caso, más la cantidad de milisegundos o segundos de espera.

En el caso de la función set_timer_seg, el valor de parámetros de seg es multiplicado por 18 y el resultado es sumado al valor en lo apuntado por clock55mseg. Esto debido a que el resultado de la multiplicación de 55 mseg * 18 es aproximadamente un segundo.

int check_timer(timer_struct* timer)

int check_timer_seg(timer_struct* timer)

Recibe:

timer_struct* timer: la dirección de memoria de la estructura que almacena el valor de expiración del temporizador.

Retorna:

El valor TRUE si el temporizador ha expirado y es la primera vez que es revisado.

El valor FALSE si el temporizador aún no ha expirado, o si ya habiendo expirado es revisado por segunda vez.

Funcionamiento:

Las funciones check_timer() y check_timer_seg(), revisan primero por la bandera del temporizador a revisar, y si se encuentra en estado de INACTIVO retorna el valor de FALSE. Si por el contrario, el temporizador se encuentra en estado de ACTIVO se compara el valor de la variable valor_final en la estructura apuntada por el parámetro timer contra el valor apuntado por la variable clock1mseg para la versión de milisegundos (check_timer()) o el valor de clock55mseg para la versión de segundos (checl_timer_seg()). Si el valor de la varible valor_final es mayor que el valor de la variable apuntada por clockXmseg, se considera que el temporizador aún no ha expirado y se retorna FALSE, en caso contrario se retorna TRUE.

unsigned int wait_timer(timer_struct* timer, unsigned int mseg)

unsigned int wait_timer_seg(timer_struct* timer, unsigned int seg)

Recibe:

timer_struct* timer: la dirección de memoria de la estructura que almacena el valor de expiración del temporizador y el estado del temporizador.

unsigned int mseg ó unsigned int seg : cantidad de milisegundos o de segundos para la espera activa.

Retorna:

El valor final del reloj, en el cual el temporizador expirará.

Funcionamiento:

Las funciones wait_timer() y wait_timer_seg(), integran a las dos funciones explicadas anteriormente (set_timer y check_timer), para recrear una espera activa a la finalización del temporizador.

La función asigna el valor de finalización del timer mediante la llamada la función set_timer(), y luego realiza un ciclo de espera activa, que finaliza cuando el temporizador ha expirado; una vez ha expirado el temporizador la función retorna.

Código TIMERS.C

```
1 #include <stdlib.h>
2 #include <dos.h>
3 #include "timers.h"
45
//definiciones
6 #define TRUE -1
7 #define FALSE 0
8 #define ACTIVO 1
9 #define INACTIVO 2
10
11 //deficiones para reloj de BIOS
12 #define SEGMENTO_BIOS 0x40
13 #define RELOJ_BIOS 0x6c
14
15 //variables para punteros lejanos a
16 //variables que se incrementen
17 //cada 55 mseg
18 unsigned far * clock55mseg;
19 //y cada mili segundo
20 unsigned far * clock1mseg = NULL;
21
22 /*void timer_init(unsigned int far * temporizador1mseg)
```

```
23 funcion que inicaliza las variables
24 clock55mseg y clock1mseg
25 */
26 void timer_init(unsigned int far * temporizador1mseg){
27 if (temporizador1mseg != NULL)
28 clock1mseg = temporizador1mseg;
29 clock55mseg = MK_FP(SEGMENTO_BIOS,RELOJ_BIOS);
30 }
31
32 /*unsigned int set_timer(timer_struct* timer, unsigned int
mseg)
33 funcion para iniciar cuenta de un temporizador
34 */
35 unsigned int set_timer(timer_struct* timer, unsigned int mseg){
36 (*timer).bandera = ACTIVO;
37 return ( (*timer).valor_final = (unsigned) ( (*clock1mseg) +
mseg ) );
38 }
39
40 /*int check_timer_seg(timer_struct* timer)
41 funcion para revisar la cuenta de un temporizador
42 si el temporizador ha expirado se retorna TRUE
43 si el temporizador aun no ha expirado, o expiro pero
44 fue revisado anteriormente se retorna FALSE
45 */
46 int check_timer(timer_struct* timer){
47 if( (*timer).bandera == INACTIVO )
48 return FALSE;
49 if ( (*clock1mseg) < (*timer).valor_final )
50 return FALSE;
51 (*timer).bandera = INACTIVO;
52 return (TRUE);
53 }
54
55 /*unsigned int wait_timer(timer_struct* timer, unsigned int
mseg)
56 funcio para espera activa por un temporizador
57 */
58 unsigned int wait_timer(timer_struct* timer, unsigned int
mseg){
59 set_timer(timer,mseg);
60 while ( !check_timer(timer) )
61 ;
62 return (*timer).valor_final;
63 }
64
65 /*unsigned int set_timer(timer_struct* timer, unsigned int
mseg)
66 funcion para iniciar cuenta de un temporizador
67 */
68 unsigned int set_timer_seg(timer_struct* timer, unsigned int
seg){
69 (*timer).bandera = ACTIVO;
70 return ( (*timer).valor_final = (unsigned) ( (*clock55mseg)
+ (seg*18) ) );
71 }
72 /*int check_timer_seg(timer_struct* timer)
73 funcion para revisar la cuenta de un temporizador
74 si el temporizador ha expirado se retorna TRUE
```

```
si el temporizador aun no ha expirado, o expiro pero
fue revisado anteriormente se retorna FALSE
*/
int check_timer_seg(timer_struct* timer){
if( (*timer).bandera == INACTIVO )
return FALSE;
if ( (*clock55mseg) < (*timer).valor_final )
return FALSE;
(*timer).bandera = INACTIVO;
return (TRUE);
}

/*unsigned int wait_timer(timer_struct* timer, unsigned int mseg)
funcion para espera activa por un temporizador
*/
unsigned int wait_timer_seg(timer_struct* timer, unsigned int seg){
set_timer_seg(timer,seg);
while ( !check_timer_seg(timer) )
;
return (*timer).valor_final;
}
```

Módulo RTP

RTP2.H

Definiciones de las funciones para el código RTP2.C. Declara las funciones de RTP2.C que pueden ser utilizadas fuera del código fuente RTP2.C.

A saber, las funciones de interfaz para RTP2 son las siguientes

- int *rtp_ini*(longword remote_ip, word remote_port)
- void *rtp_fin*(void);
- void *rtp_tick*(void);
- int *rtpestado*(void);

Las definiciones en este archivo son las siguientes

- *ACTIVO* = -1 : Estado RTP
- *INACTIVO* = 0 : Estado RTP

RTP2.C

Utiliza las librerías estándar stdlib.h, stdio.h, string.h, dos.h, y las librerías personalizadas type.h, tcp.h de WATTCP, rtp2.h, timers.h para el manejo de los temporizadores.

Las funciones utilizadas por RTP para el envío y recepción de paquetes RTP, generación y decodificación de paquetes RTP, ordenación de paquetes recibidos, revisión de paquetes esperados, actualización y modificación de los mismos, revisión de temporizadores de espera de paquetes y de temporizador para el envío en tiempo constante de paquetes, iniciación de variables, reinicialización de estados para recepción de paquetes, terminación de estado para envío y recepción de paquetes, y las funciones de integración de todas estas funciones para ser utilizada desde otro código fuente se encuentra declaradas y codificadas en este archivo. La lista de las funciones es la siguiente:

- int *rtp_ini*(longword remote_ip, word remote_port)
- void *rtp_fin*(void)
- void *rtp_reini*(void)
- int *rtpestado*(void)
- void *rtp_recv*(void)
- void *inc_OB_enq*(int incremento)
- void *inc_IB_deq*(int incremento)
- void *copy2OB*(u_int_t * ptr, int bytes, int desplazamiento)
- void *getFromIB*(u_int_t * dest, int bytes_to_get)
- void *guardar_carga_rtp*(void)
- void *init_seq_vector*(void)
- void *actualizar_seq_vector*(void)
- void *mover_seq_vector*(int ii)
- int *modificar_seq_vector*(void)
- void *crear_paquete_rtp*(void)
- void *enviar_paquete_rtp*(void)
- void *duplicar_paquete_rtp*(void)
- void *rtp_tick*(void)

Las definiciones creadas en este archivo son las siguientes:

- *IB_SIZE* = 1600 : define el tamaño del vector IBuffer de la interrupción.
- *OB_SIZE* = 8000 : define el tamaño del vector OBuffer del a interrupción.
- *PCMU* = 0 : Indica el tipo de carga de los paquetes RTP que se envían y se reciben.

- NUMERO_SEQ_MAX = 65536 : El número de secuencia máximo posible, utilizado para la inicialización del número de secuencia local.
- *SSCR_MAX* = 0xFFFFFFFF : El número de identificación de fuente máximo posible, utilizado para la inicialización del SSCR local.
- *TIMESTAMP_MAX* = 0xFFFFFFFF : El número de Timestamp máximo posible en unpaquete RTP, utilizado para la inicialización del Timestamp local.
- TRUE = -1
- FALSE = 0
- *TAMANYO_PAQUETE_RTP* = 172 : el tamaño de los paquetes RTP a enviar siempre son fijos
- *PAQUETES_BUFFER* 45 : tamaño del vector seq_vector
- *RTP_BUFFER_SIZE* = 8192 : el tamaño del vector de recepción ára WATTCP
- *RTP_OUT_BUFFER_SIZE* = 1500 : el tamaño del vector de salida de los paquetes RTP
- *RTP_RECV_SIZE* = 1500 : define el tamaño del vector de recepción de los paquetes RTP
- *RTP_ESTANDAR_HEADER_SIZE* = 12 : define el tamaño de la cabecera de los paquetes RTP a enviar, pues estas siempre son fijas.
- *Version_bits* = (3 << 6) : Valor para enmascaramiento de byte de cabecera
- *Pading_bit* = (1 << 5) : Valor para enmascaramiento de byte de cabecera
- *Extension_bit* = (1 << 4) : Valor para enmascaramiento de byte de cabecera
- *CC_bits* = (0xF) : Valor para enmascaramiento de byte de cabecera
- *Marca_bit* = (1 << 7) : Valor para enmascaramiento de byte de cabecera
- *Tipo_carga_bits* = (0x7F) : Valor para enmascaramiento de byte de cabecera

Las variables utilizadas en RTP son las siguientes:

Vectores para envío y recepción de paquetes

- u_int8_t *in_buffer*[RTP_BUFFER_SIZE] : vector para que las funciones de WATTCP reciban los paquetes UDP temporalmente
- u_int8_t *recv_buffer*[RTP_RECV_SIZE] : vector donde se reciben los paquetes RTP, al utilizar la función sock_recv()
- u_int8_t *out_buffer*[RTP_OUT_BUFFER_SIZE] : vector donde se almacenan los paquetes RTP a ser enviados.

Banderas

- Int *rtp_estado* : mantiene el estado de RTP.
- Int *bandera_primer_paquete* : bandera que indica si el siguiente paquete RTP recibido será considerado como el primer paquete RTP recibido en la sesión

- Int *bandera_delay* : bandera que indica si se el módulo RTP está esperando por varios paquetes iniciales antes de indicarle al módulo de la interrupción si ya puede sacar los valores codificados de voz ya recibidos
- Int *bandera_paquete_perdido* : bandera que indica si el último paquete esperado fue recibido o no exitosamente. Si el paquete fue recibido posee un valor de FALSE, por el contrario si el paquete no fue recibido posee un valor de TRUE.
- U_int8_t far * *bandera_muestreo* : apuntador lejano a variable switch_AD en el módulo de la interrupción.

Temporizadores

- timer_struct *timer_silencio* : temporizador para espera de siguiente paquete RTP.
- timer_struct *timer_espera* : temporizador para espera de algún paquete RTP.

Apuntadores lejanos

- u_int8_t far * *IB_ini* : apuntador al inicio del vector de la interrupción IBuffer.
- u_int16_t far * *IB_enq* : apuntador a la variable IB_enq de la interrupción.
- u_int16_t far * *IB_deq* : apuntador a la variable RTP_Deq_IB de la interrupción.
- u_int16_t far * *IB_fin* : apuntador a la variable Ibuffer_fin de la interrupción.
- u_int8_t far * *OB_ini* : apuntador al inicio del vector de la interrupción OBuffer.
- u_int16_t far * *OB_enq* : apuntador a la variable RTP_enq_OB de la interrupción.
- u_int16_t far * *OB_deq* : apuntador a la variable OB_deq de la interrupción.
- u_int16_t far * *OB_fin* : apuntador a la variable Obuffer_fin de la interrupción.
- u_int16_t far * *ptr_muestras* : apuntador a la variable MuestrasAD de la interrupción.

Definición de estructuras

```
typedef struct{
  udp_Socket rtp_socket;
  word rtp_puerto_local;
  unsigned numero_seq_local;
  unsigned tipo_payload;
  unsigned long SSCR_local;
```

```
  unsigned long Timestamp;
}id_local_struct;

typedef struct{
  word rtp_puerto_remoto;
  longword rtp_ip_remoto;
  unsigned long SSRC_remoto;
  unsigned numero_seq_remoto;
}id_remoto_struct;

typedef struct{
  id_local_struct id_local;
  id_remoto_struct id_remoto;
}id_sesion_struct;

typedef struct{
  int bandera_guardado;
  unsigned nro_seq;
}seq_struct;

typedef struct{
  char indice_menor;
  char indice_mayor;
}control_struct;

typedef struct{
  u_int8_t bandera_recibido;
  int tamanyo;
  u_int8_t Version;
  u_int8_t Pading;
  u_int8_t Extension;
  u_int8_t CC;
  u_int8_t Marca;
  u_int8_t Tipo_carga;
  u_int16_t Nro_seq;
  u_int32_t Timestamp;
  u_int32_t SSRC;
}rtp_header_struct;
```

Estructuras y vectores

- id_sesion_struct *id_sesion* : estructura que almacena la información de la sesión SIP.
- seq_struct *seq_vector*[PAQUETES_BUFFER] : vector que almacena los números de secuencia de los siguientes paquetes RTP esperados, así como también una bandera que indica si estos fueron almacenados.
- control_struct *seq_control* : estructura que almacena los índices de seq_vector donde se encuentra el valor del próximo número de secuencia esperado (indice_menor) y el último valor del número de secuencia esperado (indice_mayor).

- rtp_header_struct *rtp_header* : esta estructura almacena los valores de la cabecera del último paquete RTP recibido, y un valor de bandera que indica si existe un nuevo paquete RTP recibido, o los datos ya fueron utilizados.

Explicación de funciones

int rtp_ini(longword remote_ip, word remote_port)

Recibe:

longword remote_ip: dirección IP remota a la cual estarán destinados los paquetes RTP

word remote_port: puerto remoto al cual estarán dirigidos los paquetes RTP

Retorna:

El estado de la inicialización. Si la iniciación fue exitosa se retorna el valor de EXIT_SUCCESS, en caso de algún fallo en la inicialización del socket utilizado para RTP se retorna EXIT_FAILURE.

Funcionamiento:

Inicializa la variable de estado rtp_estado en el valor ACTIVO, y las banderas bandera_primer_paquete en TRUE, bandera_delay en TRUE, bandera_paquete_perdido en FALSE. Dentro de la estructura id_sesion se inicializan las variables de ip_remoto con el parámetro remote_ip de la función, el numero de secuencia local con un valor aleatorio, el numero de puerto local, identificador SSCR con un valor aleatorio, el valor de Timestamp con un valor aleatorio y el tipo de carga a ser enviada en los paquetes RTP con el valor de PCMU para carga de tipo PCM con ley μ.

Se inicia el socket RTP utilizando los parámetros de función remote_ip y remote_port para indicar el ip y puerto destino respectivamente. Luego se inicializan los punteros a los datos de la interrupción. Las variables inicializadas en este punto son bandera_muestreo, IB_fin, OB_fin, IB_deq, OB_enq, IB_enq, OB_deq, ptr_muestras. Luego las variables de inicio de los vectores OBuffer e IBuffer son inicializadas; estas inicializaciones de punteros se hacen mediante la

obtención del puntero a la interrupción 0x80, que fue previamente modificada por la carga de la interrupción y apunta al inicio del vector IBuffer.

Por último se inicializa el temporizador timer_silencio en 500 mseg, y se le indica a la interrupción que empiece el muestreo analógico digital, al hacer la variable apuntada por bandera_muestreo igual a TRUE.

void rtp_fin(void)

Funcionamiento:

Esta función indica a la interrupción que ya no debe seguir codificando valores analógicos y almacenandolos en IBuffer, al colocar el valor apuntado por bandera_muestreo en FALSE, además coloca el valor de rtp_estado en INACTIVO y cierra el socket utilizado por RTP.

void rtp_reini(void)

Funcionamiento:

Esta función se encarga de colocar las banderas bandera_primer_paquete, bandera_delay y bandera_paquete_perdido en los valores asignados en rtp_ini(); estos son bandera_primer_paquete en TRUE, bandera_delay en TRUE, bandera_paquete_perdido en FALSE.

Esta asignación modifica el flujo de los paquetes de recepción, y permite que se vuelvan a recibir paquetes aún cuando se hallan desincronizado los números de secuencia de los paquetes RTP recibidos actualmente y los esperados.

int rtpestado(void)

Funcionamiento:

Esta función retorna el valor de la variable rtp_estado; esta variable debe sólo contener los valores definidos por ACTIVO e INACTIVO.

void rtp_recv(void)

Funcionamiento:

La función se encarga de revisar si ha sido recibido algún nuevo paquete RTP. Si este paquete fue recibido se decodifican los valores dentro del paquete y se coloca la bandera rtp_header.bandera_recibido en TRUE, indicando que hay nuevos valores en el vector recv_buffer esperando ser colocados en el vector de la interrupción OBuffer.

void inc_OB_enq(int incremento)

Recibe:

Int incremento: valor que será sumado al valor apuntado por OB_enq.

Funcionamiento:

Esta función incrementa el valor apuntado por OB_enq en una cantidad igual al valor que contenga el parámetro de la función; si la suma resulta un valor mayor o igual al valor apuntado por OB_fin, entonces se le asigna el valor del principio del vector IBuffer al valor apuntado por OB_enq.

void inc_IB_deq(int incremento)

Recibe:

Int incremento: valor que será sumado al valor apuntado por IB_deq.

Funcionamiento:

Esta función incrementa el valor apuntado por IB_deq en una cantidad igual al valor que contenga el parámetro de la función; si la suma resulta un valor mayor o igual al valor apuntado por IB_fin, entonces se le asigna el valor del principio del vector IBuffer al valor apuntado por IB_deq.

void copy2OB(u_int_t * ptr, int bytes, int desplazamiento)

Recibe:

U_int8_t * ptr: puntero hacia los valores a copiar en OBuffer.

Int bytes: cantidad de bytes a copiar en OBuffer.

Int desplazamiento: incremento con respecto al puntero donde se esperan copiar los siguientes valores en OBuffer.

Funcionamiento:

Copy2OB obtiene el puntero lejano donde deben ser copiados los datos dentro de OBuffer.

Primero utiliza la macro MK_FP incluida en dos.h para obtener el puntero f_ptr donde se esperan copiar los siguientes datos en OBuffer, si existe un desplazamiento con respecto a este puntero se le suma y el resultado es comparado contra el final de OBuffer; si el puntero f_ptr es mayor que el final del buffer, f_ptr se decrementa en una cantidad igual al tamaño de OBuffer.

Utilizando la función _fmemcpy, se copian los valores apuntados por el parámetro ptr en el puntero encontrado f_ptr, y se copia una cantidad indicada por el parámetro bytes.

void getFromIB(u_int_t * dest, int bytes_to_get)

Recibe:

U_int8_t * dest: puntero que indica donde se almacenaran los valores obtenidos de IBuffer.

Int bytes_to_get: cantidad de bytes a obtener desde IBuffer.

Funcionamiento:

getFromIB obtiene el puntero lejano f_ptr, desde donde se obtendrán los valores del vector IBuffer.

Utiliza la macro MK_FP incluida en dos.h para obtener el puntero f_ptr desde el cual se obtendran los datos de IBuffer, luego utilizando la función _fmemcpy, se copia una cantidad indicada por el parámetro bytes_to_get, los valores apuntados por el puntero f_ptr dentro del espacio de memoria apuntado por el puntero parámetro dest.

void guardar_carga_rtp(void)

Funcionamiento:

Utilizando los valores en la estructura rtp_header, la función obtiene la cantidad de valores que deben ser copiados desde recv_buffer y hacia OBuffer, en

la variable rtp_header_tamanyo, y el puntero ptr que indica el inicio de los valores de carga del paquete RTP dentro del vetor recv_buffer.

Si el numero de secuencia siguiente esperado, no coincide con el número de secuencia del paquete, se realiza un ciclo en busca de si es alguno de los paquetes esperados, en caso de serlo, se obtiene el desplazamiento con respecto al valor del paquete esperado y se le indica en el parámetro desplazamiento de la función copy2OB.

void init_seq_vector(void)

Funcionamiento:

La función init_seq_vector, utiliza el valor en Nro_seq en la estructura rtp_header para obtener el valor del número de secuencia del paquete RTP recibido, e inicializa la estructura el arreglo de estructuras seq_vector y la estructura seq_control en base a este valor.

El valor indice_menor dentro de la estructura seq_control, indica el índice dentro del vector seq_vector donde se encuentra el número de secuencia esperado para el siguiente paquete RTP, y el valor indice_mayor dentro de la estructura seq_control indica el último valor esperado para el número de secuencia del siguiente paquete RTP. Estos valores, indice_menor e indice_mayor, son inicializados en 0 y el tamaño del vector seq_vector menos uno (PAQUETES_BUFFER-1) respectivamente.

Luego el valor del número de secuencia recibido actualmente es asignado al valor de secuencia del siguiente paquete RTP esperado dentro de la estructura seq_vector[0], y su bandera de si ya fue o no recibido es inicializada en FALSE. Sucesivamente los valores de números de secuencia esperados dentro de seq_vector son iniciados con los valores siguientes al ya obtenido de rtp_header.Nro_seq y sus banderas de si fueron o no recibidos son asignadas en FALSE, de manera que se espere por ellos en las siguientes recepciones de paquete RTP.

void actualizar_seq_vector(void)

Funcionamiento:

La función actualizar_seq_vector, utiliza el valor en Nro_seq en la estructura rtp_header para obtener el valor del número de secuencia del paquete RTP recibido, y busca dentro del vector seq_vector si corresponde con alguno de los paquetes RTP esperados, en caso de hallar una coincidencia se asigna a su bandera de si fue o no recibido (bandera_recibido) un valor de TRUE.

void mover_seq_vector(int ii)

Recibe:

Int ii: indica la cantidad de números de secuencia a remover del vector seq_vector

Funcionamiento:

La función mover_seq_vector, no verifica el valor de Nro_seq en la estructura rtp_header. Esta función remueve los 'ii' (parámetro de la función) valores de números de secuencia esperados para el siguiente paquete RTP, hayan sido estos recibidos o no.

El índice del próximo paquete esperado, indice_menor, es aumentado en una cantidad igual al parámetro ii, igualmente ocurre con el valor de indice_mayor; si algunos de estos valores se hiciese igual o mayor que el valor definido por PAQUETES_BUFFER, se les asigna un valor de cero.

Los números de secuencia dentro de seq_vector son también actualizados, y sus banderas, bandera_guardado, colocadas en valor FALSE.

Int modificar_seq_vector(void)

Retorna:

La cantidad de números de secuencia actualizados.

Funcionamiento:

La función actualizar_seq_vector, revisa por las banderas, bandera_ guardado, dentro de las estructuras de seq_vector.

La primera bandera revisada es la del próximo paquete esperado indicada por indice_menor dentro de seq_control, seq_vector[seq_control.indice_menor]. bandera_guardado, si esta bandera posee un valor de TRUE se modifica a valor de

FALSE y su número de secuencia asociado es aumentado en un valor igual al definido por PAQUETES_BUFFER para subsiguientes esperas de números de secuencia.

Se repite el ciclo de revisión del valor en bandera_guardado hasta que su valor sea igual a FALSE. Esto actualiza los números contiguos de secuencia recibidos.

Por cada actualización de número de secuencia se incrementa el contador entero 'temp', y se retorna su valor al final de la función.

void crear_paquete_rtp(void)

Funcionamiento:

La función crea un paquete RTP y lo almacena en el vector out_buffer.

Utiliza un puntero, prt, para asignar byte a byte el valor de la cabecera dentro de out_buffer, y actualiza los valores de cabecera numero_seq_local y Timestamp en la estructura id_sesion. Luego obtiene 160 bytes mediante el llamado a la función getFromIB(), y termina incrementando el valor apuntado por *IB_deq llamando a la función inc_IB_deq()

void enviar_paquete_rtp(void)

Funcionamiento:

Esta función solo se encarga de enviar el paquete RTP almacenado en out_buffer. Si el envío falla se reintenta una ves más luego de dos milisegundos. Se considera que el envío falla si la llamada a la función de envío sock_fastwrite() devuelve un valor menor a la cantidad de bytes que se desea enviar, y que para el caso de esta función será siempre el definido por TAMANYO_PAQUETE_RTP

void duplicar_paquete_rtp(void)

Funcionamiento:

Duplicar_paquete_rtp duplica el último paquete rtp almacenado justo antes de donde apunta el valor *OB_enq; si este valor se encuentra apuntando al inicio del vector OBuffer, se obtiene el valor del puntero temporal, temp_ptr, como el

final del OBuffer menos ciento sesenta (*OB_fin – 160), sino se obtiene como el valor en OB_enq menos ciento sesenta (*OB_enq – 160). Este puntero, temp_ptr, indicará desde donde copiar a un vector temporal temp_vector ciento sesenta bytes que serán luego copiadas a OBuffer, pero a partir del valor indicado por OB_enq. Esto resultará en 160 bytes duplicados dentro de OBuffer, y la función aplica cuando algún paquete RTP no haya sido recibido.

void rtp_tick(void)

Funcionamiento:

Esta función integra el llamado a las funciones explicadas anteriormente.

Revisa por la recepción de paquetes y en caso de obtener un paquete ejecuta las funciones para decodificarlo y guardar los valores de PCM contenidos en el.

Luego revisa los valores de los temporizadores, si el temporizador timer_espera se agotó se asume que se perdió el próximo paquete RTP esperado, y se modifica el vector seq_vector para actualizar a nuevos valores de secuencia esperados y en caso de que el paquete RTP anterior no haya sido perdido se llama a la función duplicar_paquete_rtp().

Se revisa luego por el temporizador timer_silencio y en caso de que este se haya agotado se ejecuta la función rtp_reini() para reiniciar los números de secuencia esperados.

Por ultimo la función revisa si se puede enviar un paquete nuevo mediante la revisión del valor apuntado por ptr_muestras; si este posee un valor igual o mayor a 160, significa que ya hay al menos 160 muestras de voz codificadas que pueden ser enviadas por RTP. Se supone esto sucederá una vez cada 20 mili segundos.

Los siguientes diagramas de flujo esquematizan el funcionamiento de la función

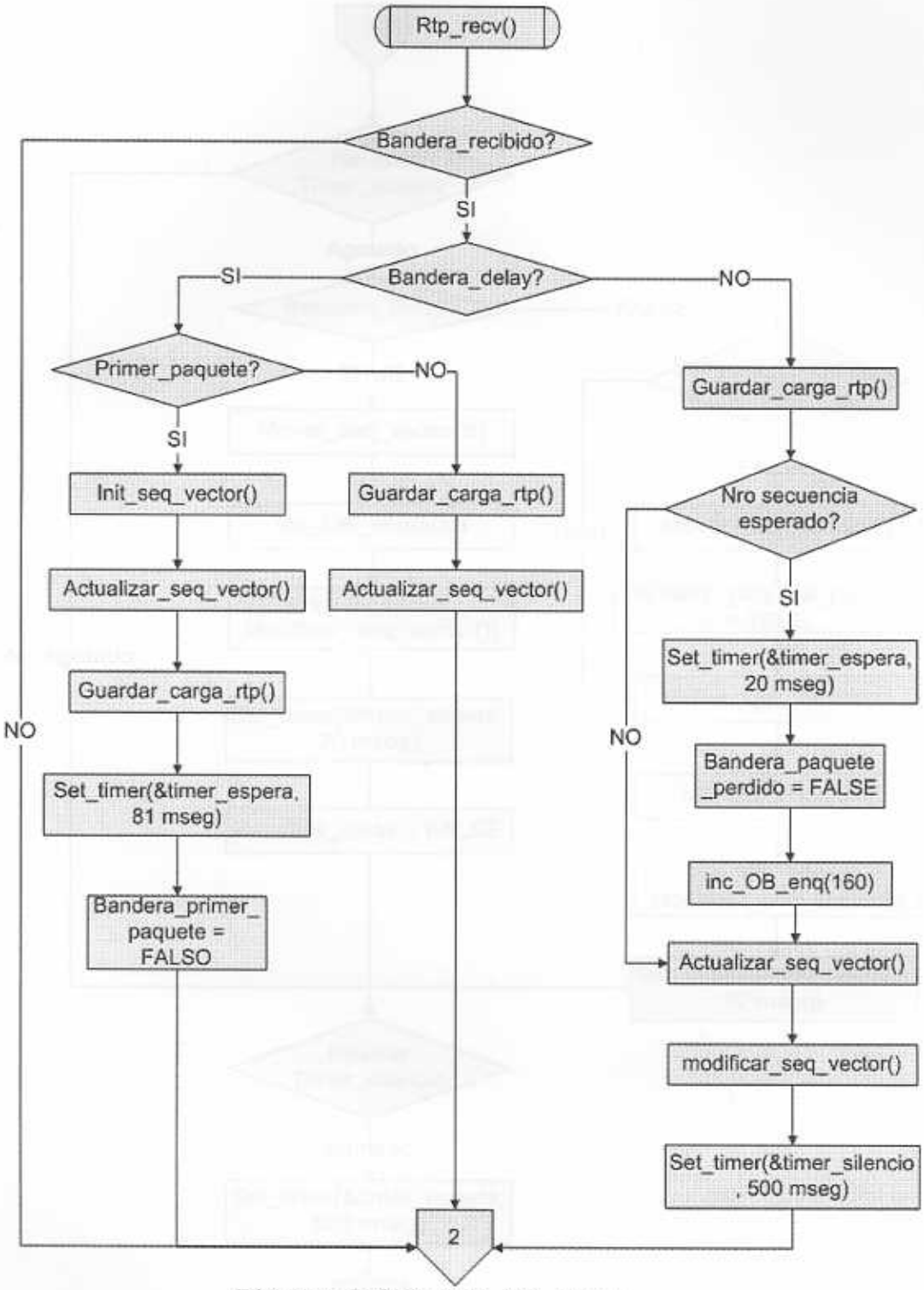

Diagrama de flujo de rtp_tick, parte 1

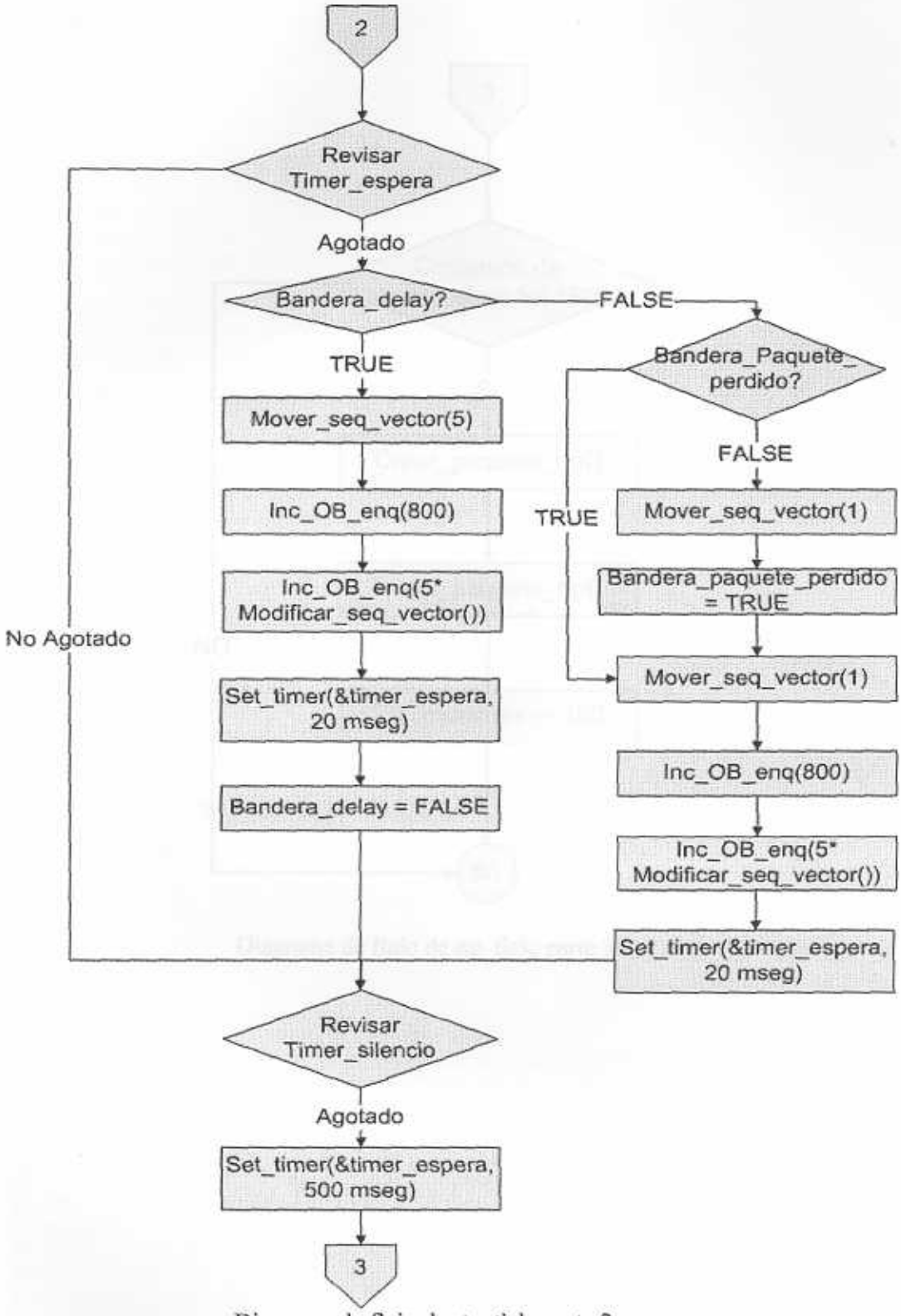

Diagrama de flujo de rtp_tick, parte 2

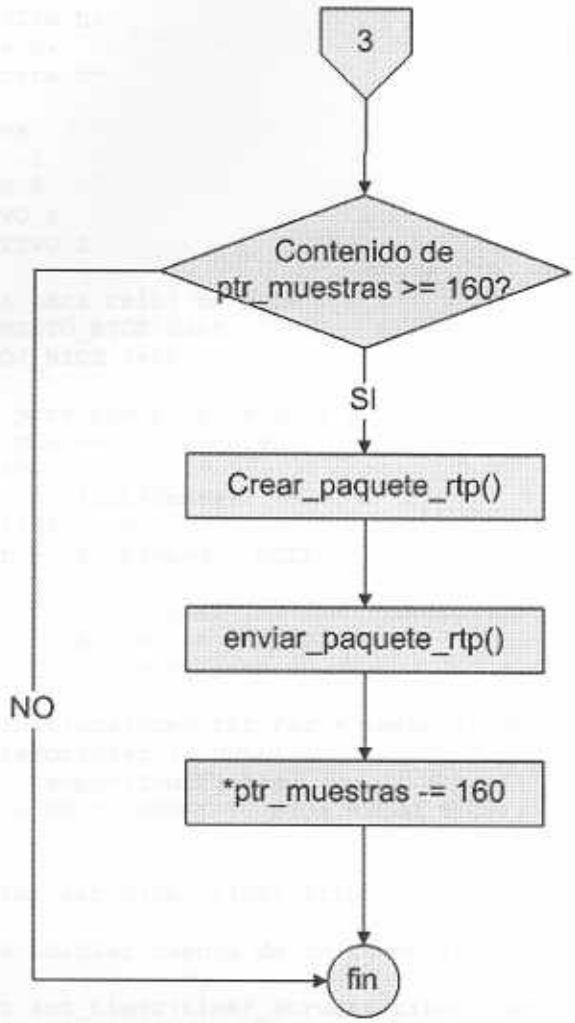

Diagrama de flujo de rtp_tick, parte 3

Código RTP2.C

```
1 #include <stdlib.h>
2 #include <dos.h>
3 #include "timers.h"
4
5 //definiciones
6 #define TRUE -1
7 #define FALSE 0
8 #define ACTIVO 1
9 #define INACTIVO 2
10
11 //deficiones para reloj de BIOS
12 #define SEGMENTO_BIOS 0x40
13 #define RELOJ_BIOS 0x6c
14
15 //variables para punteros lejanos a
16 //variables que se incrementen
17 //cada 55 mseg
18 unsigned far * clock55mseg;
19 //y cada mili segundo
20 unsigned far * clock1mseg = NULL;
21
22 /*void timer_init(unsigned int far * temporizador1mseg)
23 funcion que inicaliza las variables
24 clock55mseg y clock1mseg
25 */
26 void timer_init(unsigned int far * temporizador1mseg){
27 if (temporizador1mseg != NULL)
28 clock1mseg = temporizador1mseg;
29 clock55mseg = MK_FP(SEGMENTO_BIOS,RELOJ_BIOS);
30 }
31
32 /*unsigned int set_timer(timer_struct* timer, unsigned int
mseg)
33 funcion para iniciar cuenta de un temporizador
34 */
35 unsigned int set_timer(timer_struct* timer, unsigned int mseg){
36 (*timer).bandera = ACTIVO;
37 return ( (*timer).valor_final = (unsigned) ( (*clock1mseg) +
mseg ) );
38 }
39
40 /*int check_timer_seg(timer_struct* timer)
41 funcion para revisar la cuenta de un temporizador
42 si el temporizador ha expirado se retorna TRUE
43 si el temporizador aun no ha expirado, o expiro pero
44 fue revisado anteriormente se retorna FALSE
45 */
46 int check_timer(timer_struct* timer){
47 if( (*timer).bandera == INACTIVO )
48 return FALSE;
49 if ( (*clock1mseg) < (*timer).valor_final )
50 return FALSE;
51 (*timer).bandera = INACTIVO;
52 return (TRUE);
53 }
54
55 /*unsigned int wait_timer(timer_struct* timer, unsigned int
```

```
mseg)
56 funcio para espera activa por un temporizador
57 */
58 unsigned int wait_timer(timer_struct* timer, unsigned int
mseg){
59 set_timer(timer,mseg);
60 while ( !check_timer(timer) )
61 ;
62 return (*timer).valor_final;
63 }
64
65 /*unsigned int set_timer(timer_struct* timer, unsigned int
mseg)
66 funcion para iniciar cuenta de un temporizador
67 */
68 unsigned int set_timer_seg(timer_struct* timer, unsigned int
seg){
69 (*timer).bandera = ACTIVO;
70 return ( (*timer).valor_final = (unsigned) ( (*clock55mseg)
+ (seg*18) ) );
71 }
72 /*int check_timer_seg(timer_struct* timer)
73 funcion para revisar la cuenta de un temporizador
74 si el temporizador ha expirado se retorna TRUE
75 si el temporizador aun no ha expirado, o expiro pero
76 fue revisado anteriormente se retorna FALSE
77 */
78 int check_timer_seg(timer_struct* timer){
79 if( (*timer).bandera == INACTIVO )
80 return FALSE;
81 if ( (*clock55mseg) < (*timer).valor_final )
82 return FALSE;
83 (*timer).bandera = INACTIVO;
84 return (TRUE);
85 }
86
87 /*unsigned int wait_timer(timer_struct* timer, unsigned int
mseg)
88 funcion para espera activa por un temporizador
89 */
90 unsigned int wait_timer_seg(timer_struct* timer, unsigned int
seg){
91 set_timer_seg(timer,seg);
92 while ( !check_timer_seg(timer) )
93 ;
94 return (*timer).valor_final;
95 }
```

SIP2.H

El código SIP2.H contiene los prototipos de las funciones que deben ser utilizadas por el programa principal del teléfono para mantener el flujo de datos SIP, así como las definiciones utilizadas dentro de SIP2.C. Define también una estructura para la comunicación con el módulo RTP

Los prototipos de funciones son las siguientes

- int *sip_init*(void);
- void *sip_tick*(void);
- int *sip_estado*(host_remoto_struct*);
- void *sip_close*(void);

Las definiciones hechas en este código son las siguientes.

- TRUE = -1
- FALSE = 0

Para las definiciones de estados de los mensajes SIP y sus respuestas son las siguientes

- INVITE = 1
- ACK = 2
- BYE = 3
- REGISTER = 4
- R_REPICANDO = 180
- R_OK = 200
- R_OCUPADO = 486
- R_ERROR = 487

Para los estados de SIP, y los estados de las transacciones SIP se definieron las siguientes variables.

- ESPERA = 10
- LLAMANDO = 20
- PROCESANDO = 30
- COMPLETADO = 40
- COMFIRMADO = 50
- TERMINADO = 60
- INICIO_LLAMADA = 5
- EN_LLAMADA = 15
- REPICANDO = 25

Se define la macro max__(x,y) que retorna el mayor de los dos valores x ó y

max__(x,y) ((x > y)? x : y)

El siguiente tipo de datos es definido para ser utilizado en la comunicación entre SIP y RTP por parte del programa principal.

```
typedef struct{
 unsigned long IP;
 unsigned int puerto;
}host_remoto_struct;
```

Código SIP2.H

```
3 #ifndef _SIP2_H_
4 #define _SIP2_H_
5
6 #define TRUE -1
7 #define FALSE 0
8
9 #define INVITE 1
10 #define ACK 2
11 #define BYE 3
12 #define REGISTER 4
13
14 #define R_REPICANDO 180
15 #define R_OK 200
16 #define R_OCUPADO 486
17 #define R_ERROR 487
18
19 #define ESPERA 10
20 #define LLAMANDO 20
21 #define PROCESANDO 30
22 #define COMPLETADO 40
23 #define COMFIRMADO 50
24 #define TERMINADO 60
25
26 #define INICIO_LLAMADA 5
27 #define EN_LLAMADA 15
28 #define REPICANDO 25
29
30 #define max__(x,y) ((x > y)? x : y)
31
32 /*
33 punteros para funciones de interfaz
34 int (*init_call)(void) = init_call_default;
35 int (*pick_up)(void) = pick_up_default;
36 int (*put_down)(void) = put_down_default;
37 void (*get_uri)(void) = get_uri_default;
38 //fin punteros
39 */
40
41 typedef struct{
42 unsigned long IP;
43 unsigned int puerto;
44 }host_remoto_struct;
45
46 int sip_init(void);
47 void sip_tick(void);
```

```
48 int sip_estado(host_remoto_struct*);
49 void sip_close(void);
50
51 #endif
```

SIP2.C

El código de SIP2.C se encuentra las funciones de creación de paquetes SIP y de decodificación de los mismos para la obtención de los datos de importancia dentro de la sesión SIP, el archivo SIP2.C incluye un archivo sipprim.h, no explicado en el dodumento, pero que solo conserva los prototipos de las funciones básicas de SIP.

Las funciones de mayor relevancia en SIP.C son las siguientes

- void *sip_tick*(void)
- void *sip_check_timer*(void)
- void *sip_close*(void)
- int *sip_estado*(host_remoto_struct* host)
- void *sip_recv*(void)
- int *sip_init*(void)
- void *sip_send*(void)
- void *send*(int tamanyo)
- void *sip_reinit*(void)
- int *decode_first_line*(char* line)
- void *respuesta*(int mensaje_primera_linea)

Las variables de mayor relevancia son

- timer_struct *timerA y timerB* : son temporizadores para controlar la retransmicion de paquetes INVITE.
- u_int8_t *sip_in_buffer*[SIP_BUFFER_SIZE] : vector utilizado por WATTCP para almacenar temporalmente paquetes SIP hasta su lectura con la utilización de la funcion sock_recv()
- u_int8_t *sip_recv_buffer*[SIP_RECV_BUFFER_SIZE] : vector para almacenamiento temporal de paquetes de llegada de SIP
- u_int8_t *sip_out_buffer*[SIP_OUT_BUFFER_SIZE] : vector para almacenamiento temporal de paquetes de salida SIP
- int *bandera_respuesta* : variable que mantiene un número entero que indica el tipo de paquete que enviará SIP.
- char *smatriz*[MAXCOMP][21] : cadenas de caracteres a ser comparadas con el principio de las lineas de los paquetes SIP, para obtener cual es la cabecera de la linea en cuestion y llamar a la función que decodificará su significado.

- void (**smatrizFunc*[MAXCOMP])(char*) : vector de punteros a funciones; almacena los punteros a las funciones de tipo <_Func> (ver comentarios en codigo).

Funciones de SIP.C

int *decode_first_line*(char* line)

Recibe:

Un puntero a una cadena de caracteres que la funcion supone contiene la linea de inicio de un paquete SIP

Retorna:

EXIT_SUCCESS

Funcionamiento:

Realiza una serie de comparaciones mediante el uso de la función strncmp a los primeros caracteres de la cadena de caracteres 'linea' o compara el carácter en la posición 8 de la línea, buscando obtener el primer digito de la respuesta del mensaje, ya sea un mensaje de tipo informativo con un dígito de inicio 1, o un mensaje de error con dígito de inicio igual a 4

void *respuesta*(int mensaje_primera_linea)

Recibe:

Un valor numérico que indica el tipo de mensaje que se recibió, ya sea

Funcionamiento:

Coloca una respuesta provicional en la variable bandera_respuesta, basada en el estado del SIP y de la transacción. Esta resputesta provisional, puede ser luego modificada cuando se revisen una a una las lineas del mensaje SIP.

void *sip_check_timer*(void)

Funcionamiento:

Esta funcion revisa la expiración de los temporizadores TimerA y TimerB; si el temporizador TimerA expira, se genera un paquete INVITE y se envía por UDP, además se reinicia el temporizador en un valor igual al valor que contiene la variable T1a multiplicado por dos. Si el temporizador TimerB expira, se asume que el mensaje INVITE que se esta enviando no será respondido y se reinicia el módulo sip mediante la funcion sip_reinit()

void *sip_close*(void)

Funcionamiento:

Cierrra el Socket UDP que utiliza SIP

int *sip_estado*(host_remoto_struct* host)

Recibe:

Recibe un apuntador a una estructura que almacena una dirección IP y un puerto

Retorna:

Retorna el estado de la variable dialog.estado, o estado del módulo SIP; además en la estructura que recibe como parámetro retorna la IP del host remoto con el cual RTP debe comunicarse y el puerto del host remoto.

Funcionamiento:

sip_estado, asigna el valor de puerto a la variable puerto de la estructura que obtiene como argumento y la dirección de IP remoto a la variable IP de la estructura que obtiene como argumento. Retorna el valor del estado del módulo SIP (ESPERA, LLAMANDO, INICIO_LLAMADA, EN_LLAMADA).

void *sip_recv*(void)

Funcionamiento:

Revisa por un nuevo paquete SIP, en caso de haber uno, lo recibe y lo decodifica.

int *sip_init*(void)

Inicia el socket UDP utilizado por SIP revisa por la existencia de un archivo de configuracion para SIP, si este existe obtiene los datos de proxy y uri local sino se asignan dos valores por defecto inicia los valores de estado de SIP. Coloca el

valor de T1 para inicio de los temporizadores A y B en un valor de 800 milisegundos. Obtiene la direccion IP local y genera los valores cseq y to_tag aleatoriamente.

void *send*(int tamanyo)

Recibe:

El tamaño del paquete a mandarse por UDP.

Funcionamiento:

Envía por UDP el paquete SIP almacenado en sip_out_buffer

void *sip_send*(void)

Funcionamiento:

Revisa el valor de la variable bandera_respuesta y genera un mensaje acorde a su valor además revisa por si alguna de las funciones de interfaz de usuario ha sido activada y envía un mensaje acorde a la accion del usuario iniciar llamada , genera un mensaje INVITE colgar , genera un mensaje BYE levantar el telefono si esta repicando, genera un mensaje 200 ok.

void *sip_reinit*(void)

Funcionamiento:

Reinicia los estados de SIP destruye los valores en call_id, via, y from, from_tag resetea el temporizador timerA

void *sip_tick*(void)

Funcionamiento:

Ejecuta las funciones sip_recv(), sip_send() y sip_check_timer() en ese orden; esto hace que el módulo SIP revise por nuevos mensajes en su socket, envíe las respuestas correspondientes y además revise los temporizadores por si se necesita de alguna retransmición

Código SIP2.C

```
1
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <string.h>
5 #include <ctype.h>
6 #include <conio.h>
7 #include "tcp.h"
8 #include "timers.h"
9
10 //definiciones
11 #define TRUE -1
12 #define FALSE 0
13
14 //estructuras de datos utilizadas comunmente en este codigo
15 typedef unsigned char u_int8_t;
16 typedef unsigned int u_int16_t;
17 typedef unsigned long u_int32_t;
18
19 #include "sipprim.h"
20 #include "sip2.h"
21
22 //temporizadores
23 /*El temporizador A es utilizado para controlar el
24 tiempo de retransmicion de los mensajes INVITE
25 */
26 timer_struct timerA;
27 /*T1 temporizador B es utilizado para controlar
28 el fin de las retransmisiones de mensajes INVITE
29 */
30 timer_struct timerB;
31
32 //las variables T1 y T1a controlan los valores de
33 //inicio y de reinicio del temporizador A
34 int T1;
35 int T1a;
36
37 //vectores
38 /*el vector sip_in_buffer es utilizado por WATTCP para
39 almacenar temporalmente los paquetes UDP que reciba
40
41 el vector sip_recv_buffer es utilizado para almacenar
42 los paquetes recibidos mediante la instruccion sock_recv()
43 de WATTCP, el vector sip_out_buffer mantiene temporalmente
44 los paquetes que se van a transmitir por UDP
45 */
46 #define SIP_BUFFER_SIZE 8192
47 u_int8_t sip_in_buffer[SIP_BUFFER_SIZE];
48 #define SIP_RECV_BUFFER_SIZE 1500
49 u_int8_t sip_recv_buffer[SIP_RECV_BUFFER_SIZE];
```

```
50 #define SIP_OUT_BUFFER_SIZE 1500
51 u_int8_t sip_out_buffer[SIP_OUT_BUFFER_SIZE];
52
53 //definiciones de estructuras de datos que almacenan
54 //datos importantes de la sesion SIP
55 typedef struct{
56 u_int8_t call_id[100];
57 u_int8_t remote_tag[20];
58 u_int8_t local_tag[20];
59 }dialog_id_struct;
60
61 typedef struct{
62 u_int8_t local[100];
63 u_int8_t remote[100];
64 }uri_struct;
65
66 typedef struct{
67 longword local;
68 longword remote;
69 }address_struct;
70
71 typedef struct{
72 unsigned local;
73 char metodo_local[10];
74 unsigned remote;
75 char metodo_remoto[10];
76 }cseq_struct;
77
78 typedef struct{
79 int estado;
80 int estado_tran;
81 dialog_id_struct dialog_id;
82 uri_struct uri;
83 address_struct address;
84 cseq_struct cseq;
85 udp_Socket sip_socket;
86 char proxy_name[100];
87 int sdp_content;
88 int content_length;
89 char via_temp[200];
90 }dialog_struct;
91
92 //la estructura dialog es la estructura que almacenara
93 //la informacion de la sesion SIP
94 dialog_struct dialog;
95
96 //la estructura packet se utiliza para mantener la información
97 //de paquetes SIP y obtener linea a linea cada una
98 //de las cabeceras del mensaje
99 #define PACKET_TIMESTAMP_LEN 32
100 typedef struct packet
101 {
102 u_int8_t * contents;
103 u_int8_t * current;
104 u_int8_t * end;
105 u_int8_t * apparent_end; /* donde se piensa se termina el
paquete */
106 u_int8_t * mark;
107 u_int32_t media_length; /* tamaño reportado del paquete*/
```

```
108 char timestamp [PACKET_TIMESTAMP_LEN];
109 } packet_t;
110
111 /*int get_packet_line(u_int8_t *dst, u_int32_t max, packet_t
*p)
112 obtiene una linea (desde el comienzo hasta el
113 encentro del primer \n) de un paquete tipo packet_t
114 y coloca el resultado
115 */
116 int get_packet_line(u_int8_t *dst, u_int32_t max, packet_t *p)
117 {
118 u_int8_t * ptr;
119 int ret;
120
121 //se busca el siguiente \n
122 ptr = p->current;
123 while( (ptr < p->apparent_end) && ((ptr - p->current) < max -
1) )
124 {
125 if (*ptr == '\n')
126 break;
127 ptr++;
128 }
129
130 //si se llega al final, entonces no hay linea que leer
131 if (ptr >= p->apparent_end)
132 return 0;
133
134 //no se copia el \n
135 //se agrega el \0 al final de la cadena
136 memcpy(dst, p->current, ptr - p->current);
137 dst[ptr - p->current] = '\0';
138
139 //se incrementa el puntero 'current' para pasar el \n
140 ret = ptr - p->current;
141 p->current += ret + 1;
142
143 //se retorna el tamaño de la linea, sin incluir el \n
144 return ret;
145 }
146
147 //bandera para indicar el mensaje SIP de respuesta
148 int bandera_respuesta;
149
150 //cadenas de caracteres para comparar las
151 //cabeceras de los mensajes SIP
152 #define MAXCOMP 16
153 char smatriz[MAXCOMP][21] = {"Call-ID", "i", \
154 "Contact", "m", \
155 "Cseq", \
156 "From", "f", \
157 "To", "t", \
158 "Via", "v", \
159 "Max-Forwards", \
160 "Content-Length", "l", \
161 "Content-Type", "c"
162 };
163
164 //vector de punteros a las funciones que manejan
```

```
165 //las cabeceras SIP
166 void (*smatrizFunc[MAXCOMP])(char*)= {Call_ID_Func, \
167 Call_ID_Func, \
168 Contact_Func, \
169 Contact_Func, \
170 Cseq_Func, \
171 From_Func, \
172 From_Func, \
173
174 To_Func, \
175 To_Func, \
176 Via_Func, \
177 Via_Func, \
178
179 Max_Forwards_Func, \
180
181 Content_Length_Func, \
182 Content_Length_Func, \
183 Content_Type_Func, \
184 Content_Type_Func \
185 };
186
187 //funciones de interfaz con usuario
188 //estas funciones estan apuntadas por iniciacion por los
punteros
189 //correspondientes. Se encargan de la revision de funciones
190 //de inicio de llamada, levantar el telefono, colgar el
telefono
191 //repicar el telefono, finalizar el repique del telefono
192 //las funciones por defecto utilizan el teclado y monitor de
193 //un computador personal como interfaz, pero se pueden
modificar
194 //mediante los punteros presentados mas adelante
195 //para utilizar otra interfaz con el usurio
196 int init_call_default(){
197 int c;
198 if( kbhit() ){
199 c = getch();
200 if( c == 0x20 )
201 return TRUE;
202 }
203 return FALSE;
204 }
205
206 int pick_up_default(){
207 int c;
208 if( kbhit() ){
209 c = getch();
210 if( c == 0x20 )
211 return TRUE;
212 }
213 return FALSE;
214 }
215
216 int put_down_default(){
217 int c;
218 if( kbhit() ){
219 c = getch();
220 if( c == 0x20 )
```

```
221 return TRUE;
222 }
223 return FALSE;
224 }
225
226 int ringing_default(){
227 printf("180 Ringing\n");
228 return EXIT_SUCCESS;
229 }
230
231 int end_ringing_default(){
232 printf("end ringing\n");
233 return EXIT_SUCCESS;
234 }
235
236 void get_uri_default(){
237 strcpy( dialog.uri.remote, "carlos@ucab" );
238 }
239
240 //punteros para funciones de interfaz
241 //estos punteros se utilizan para llamar a las funciones de
242 //interfaz con el usuario por defecto, pero pueden ser
243 //modificadas de manera externa para utilizar una forma
244 //de interfaz distinta a la del monitor y teclado
245 int (*init_call)(void) = init_call_default;
246 int (*pick_up)(void) = pick_up_default;
247 int (*put_down)(void) = put_down_default;
248 void (*get_uri)(void) = get_uri_default;
249 int (*ringing)(void) = ringing_default;
250 int (*end_ringing)(void) = end_ringing_default;
251 //fin punteros
252
253 //FUNCIONES HANDLE
254 /*Estas funciones son llamadas en los casos de recepcion de
255 paquetes SIP, su funcion es la de obtener los datos
256 de importancia de una cabecera SIP.
257 Cada tipo de cabecera posee su funcion handle.
258 Por ejemplo la funcion handle_call_id se encarga de
259 almacenar el valor del call_id de un mensaje SIP
260 */
261 void handle_call_id(char* line){
262 char* ptr;
263 if ( (ptr = strchr(line, ':')) == NULL ){
264 bandera_respuesta = 487;
265 return;
266 }
267 ptr++;
268 strcpy((char*)dialog.dialog_id.call_id, ptr );
269 }
270 //FUNCION HANDLE
271 void handle_via(char * line){
272 char* ptr;
273 if ( (ptr = strchr(line, ':')) == NULL ){
274 bandera_respuesta = 487;
275 return;
276 }
277 ptr++;
278 if ( dialog.via_temp[0] == '\0' )
279 strcpy( dialog.via_temp, ptr );
```

```
280 else{
281 strcat( dialog.via_temp, "\n" );
282 strcat( dialog.via_temp, line );
283 }
284 }
285 //FUNCION HANDLE
286 void handle_from(char* line){
287 char* ptr, *ptr2;
288 if ( (ptr = strchr(line, '<')) == NULL ){
289 bandera_respuesta = 487;
290 return;
291 }
292 if ( (ptr2 = strchr(line, '>')) == NULL ){
293 bandera_respuesta = 487;
294 return;
295 }
296 strncpy((char*)dialog.uri.remote, ptr+1, (int)(ptr2-ptr-1)
);
297 dialog.uri.remote[ptr2-ptr] = '\0';
298
299 if( (ptr = strchr(line, '=')) == NULL ){
300 bandera_respuesta = 487;
301 return;
302 }
303 ptr++;
304 strcpy((char*)dialog.dialog_id.remote_tag, ptr);
305
306 }
307 //FUNCION HANDLE
308 void handle_to(char* line){
309 char* ptr, *ptr2, temp[20];
310
311 strcpy( dialog.dialog_id.local_tag, itoa(rand(), temp, 10)
);
312
313 }
314 //FUNCION HANDLE
315 void handle_cseq(char* line){
316 char* ptr, *ptr2, temp[20];
317 if ( (ptr = strchr(line, ':')) == NULL ){
318 bandera_respuesta = 487;
319 return;
320 }
321 ptr++;
322 if ( (ptr2 = strchr(line, ' ')) == NULL ){
323 bandera_respuesta = 487;
324 return;
325 }
326 strncpy(temp, ptr, ptr2-ptr);
327 temp[ptr2-ptr] = '\0';
328 dialog.cseq.remote = atoi(temp);
329 if ( (ptr = strchr(line, '\0')) == NULL ){
330 bandera_respuesta = 487;
331 return;
332 }
333 ptr2++;
334 strncpy( dialog.cseq.metodo_remoto, ptr2, ptr-ptr2 );
335 dialog.cseq.metodo_remoto[ptr-ptr2+1] = '\0';
336
```

```
337 }
338 //FUNCION HANDLE
339 void handle_content_length(char* line){
340 char* ptr;
341 if ( (ptr = strchr(line, ':')) == NULL )
342 bandera_respuesta = 487;
343 ptr++;
344 dialog.content_length = atoi(ptr);
345 }
346
347 //FUNCIONES DE REVISION
348 /*las funciones de revision se encargan de comparar
349 un dato en una cabecera con otro almacenado anteriormente
350 si estos datos no coinciden se considera un error
351 y la peticion o mensaje son respondidso con un mensaje 487
352 Peticion terminada
353 */
354 void revisar_to(char* line){
355 char* ptr;
356 if ( (ptr = strchr(line, '=')) == NULL ){
357 bandera_respuesta = 487;
358 return;
359 }
360 ptr++;
361 if( (strcmp(dialog.dialog_id.remote_tag, ptr ) != 0) && \
362 (strcmp(dialog.dialog_id.local_tag, ptr ) != 0) )
363 bandera_respuesta = 487;
364 }
365 //FUNCION DE REVISION
366 void revisar_from(char* line){
367 char* ptr;
368 if ( (ptr = strchr(line, '=')) == NULL ){
369 bandera_respuesta = 487;
370 return;
371 }
372 ptr++;
373 if( (strcmp((char*)dialog.dialog_id.remote_tag, ptr ) != 0)
&& (strcmp((char*)dialog.dialog_id.local_tag, ptr ) != 0) )
374 bandera_respuesta = 487;
375 }
376 //FUNCION DE REVISION
377 void revisar_call_id(char* line){
378 char* ptr;
379 if ( (ptr = strchr(line, ':')) == NULL ){
380 bandera_respuesta = 487;
381 return;
382 }
383 ptr++;
384 if( strcmp(dialog.dialog_id.call_id, ptr ) != 0){
385 bandera_respuesta = 487;
386 return;
387 }
388 }
389
390 /*Funciones *cabecera*_Func
391 estas funciones se encargan de revisar el estado de SIP
392 y llamar a la funcion handle o de revision correspondiente
393 si la funcion _Func puede llamar a un rutina de revision
394 se revisa si la bandera de respuesta fue colocada
```

```
395 para responder con un mensaje de Ocupado
396 En caso de ser este el caso, la funcion de revision no es
397 ejecutada
398 */
399 void Call_ID_Func(char* line){
400 switch(dialog.estado){
401 case(ESPERA):
402 handle_call_id(line);
403 break;
404 case(INICIO_LLAMADA):
405 case(REPICANDO):
406 revisar_call_id(line);
407 break;
408 case(EN_LLAMADA):
409 if( bandera_respuesta != R_OCUPADO )
410 revisar_call_id(line);
411 break;
412 }
413 }
414 //Funcion *cabecera*_Func
415 void Via_Func(char* line){
416 switch(dialog.estado){
417 case(ESPERA):
418 case(INICIO_LLAMADA):
419 case(REPICANDO):
420 case(EN_LLAMADA):
421 handle_via(line);
422 break;
423 }
424 }
425 //Funcion *cabecera*_Func
426 void Contact_Func(char* line){
427 switch(dialog.estado){
428 case(ESPERA):
429 case(INICIO_LLAMADA):
430 case(REPICANDO):
431 case(EN_LLAMADA):
432 break;
433 }
434 }
435 //Funcion *cabecera*_Func
436 void From_Func(char* line){
437 switch(dialog.estado){
438 case(ESPERA):
439 handle_from(line);
440 break;
441 case(INICIO_LLAMADA):
442 case(REPICANDO):
443 revisar_from(line);
444 break;
445 case(EN_LLAMADA):
446 if( bandera_respuesta != R_OCUPADO )
447 revisar_from(line);
448 break;
449 }
450 }
451 //Funcion *cabecera*_Func
452 void To_Func(char* line){
453 switch(dialog.estado){
```

```
454 case(ESPERA):
455 handle_to(line);
456 break;
457 case(INICIO_LLAMADA):
458 case(REPICANDO):
459 revisar_to(line);
460 break;
461 case(EN_LLAMADA):
462 if( bandera_respuesta != R_OCUPADO )
463 revisar_to(line);
464 break;
465
466 }
467 }
468 //Funcion *cabecera*_Func
469 void Cseq_Func(char* line){
470 char* ptr;
471 if ( (ptr = strchr(line, ':')) == NULL )
472 bandera_respuesta = 487;
473 ptr++;
474
475 switch(dialog.estado){
476 case(ESPERA):
477 case(INICIO_LLAMADA):
478 case(REPICANDO):
479 case(EN_LLAMADA):
480 handle_cseq(line);
481 break;
482 }
483 }
484 //Funcion *cabecera*_Func
485 void Max_Forwards_Func(char* line){
486 switch(dialog.estado){
487 case(ESPERA):
488 case(INICIO_LLAMADA):
489 case(EN_LLAMADA):
490 case(REPICANDO):
491 break;
492 }
493 }
494 //Funcion *cabecera*_Func
495 void Content_Length_Func(char* line){
496 switch(dialog.estado){
497 case(ESPERA):
498 case(INICIO_LLAMADA):
499 case(EN_LLAMADA):
500 case(REPICANDO):
501 handle_content_length(line);
502 }
503 }
504 //Funcion *cabecera*_Func
505 void Content_Type_Func(char* line){
506 char* ptr, ptr2;
507 if ( (ptr = strchr(line, ':')) == NULL )
508 bandera_respuesta = 487;
509 ptr++;
510 if( strcmp(ptr, "application/sdp") == 0 )
511 dialog.sdp_content = TRUE;
512 else
```

```
513 dialog.sdp_content = FALSE;
514 }
515
516 /*FUNCIONES de generacion
517 Estas funciones se encargan de generar o buscar
518 un dato y colocarlo en el mensaje SIP
519 que se esta creando
520 */
521 void gen_first_line(u_int8_t* ptr, int metodo){
522 if( metodo == INVITE ){
523 strcpy( ptr, "INVITE " );
524 strcat( ptr, dialog.uri.remote );
525 strcat( ptr, " SIP/2.0" );
526 }
527 else if( metodo == BYE ){
528 strcpy( ptr, "BYE " );
529 strcat( ptr, dialog.uri.local );
530 strcat( ptr, " SIP/2.0" );
531 }
532 else if( metodo == ACK ){
533 strcpy( ptr, "ACK " );
534 strcat( ptr, dialog.uri.local );
535 strcat( ptr, " SIP/2.0" );
536 }
537 else if(metodo == REGISTER ){
538 strcpy( ptr, "REGISTER " );
539 strcat( ptr, dialog.uri.local );
540 strcat( ptr, " SIP/2.0" );
541 }
542 }
543 //FUNCION de generacion
544 void generate_call_id(u_int8_t* ptr){
545 char temp[20];
546 strcpy( dialog.dialog_id.call_id, itoa(rand(),temp,10) );
547 strcat( dialog.dialog_id.call_id, "@" );
548
549 strcat( dialog.dialog_id.call_id, inet_ntoa
(dialog.address.local) );
550 strcat( ptr, dialog.dialog_id.call_id );
551 }
552 //FUNCION de generacion
553 void generate_via(u_int8_t* ptr){
554 strcat( ptr, "SIP/2.0/UDP " );
555 strcat( ptr, dialog.proxy_name );
556 strcat( ptr, ";branch=z9hG4bK776asdhds" );
557
558 }
559 //FUNCION de generacion
560 void generate_from(u_int8_t* ptr){
561 char temp[20];
562 strcat( ptr, "<");
563 strcat( ptr, dialog.uri.local );
564 strcat( ptr, ">;tag=");
565 strcpy( dialog.dialog_id.local_tag, itoa(rand(), temp, 10)
);
566 strcat( ptr, dialog.dialog_id.local_tag );
567 }
568 //FUNCION de generacion
569 void generate_to(u_int8_t* ptr){
```

```
570 strcat( ptr, "<");
571 strcat( ptr, dialog.uri.remote );
572 strcat( ptr, ">");
573 }
574 //FUNCION de generacion
575 void generate_contact(u_int8_t* ptr){
576 strcat( ptr, dialog.uri.local );
577 }
578 //FUNCION de generacion
579 void generate_max_forwards(u_int8_t* ptr){
580 strcat( ptr, "70");
581 }
582 //FUNCION de generacion
583 void generate_cseq(u_int8_t* ptr, int metodo){
584 char temp[20];
585 if( metodo == INVITE ){
586 strcat( ptr, itoa(dialog.cseq.local, temp, 10) );
587 strcat( ptr, " INVITE" );
588 strcpy( dialog.cseq.metodo_local, " INVITE");
589 }
590 else if( metodo == BYE ){
591 strcat( ptr, itoa(dialog.cseq.local, temp, 10) );
592 strcat( ptr, " BYE" );
593 strcpy( dialog.cseq.metodo_local, " BYE");
594 }
595 else if( metodo == ACK ){
596 strcat( ptr, itoa(dialog.cseq.local, temp, 10) );
597 strcat( ptr, " ACK" );
598 strcpy( dialog.cseq.metodo_local, " ACK");
599 }
600 else if(metodo == REGISTER ){
601 strcat( ptr, itoa(dialog.cseq.local, temp, 10) );
602 strcat( ptr, " REGISTER" );
603 strcpy( dialog.cseq.metodo_local, " REGISTER");
604 }
605 dialog.cseq.local++;
606 }
607 //FUNCION de generacion
608 void generate_content_type(u_int8_t* ptr){
609 strcat( ptr, "aplication/sdp");
610 }
611 //FUNCION de generacion
612 void generate_content_length(u_int8_t* ptr){
613 strcat( ptr, "130");
614 }
615 //FUNCION de generacion
616 void generate_contact_response(u_int8_t * ptr){
617 char* temp_ptr;
618 strcat(ptr, "<");
619 temp_ptr = strchr( dialog.uri.local, '@');
620 strncat( ptr, dialog.uri.local , (int)(temp_ptr -
(int)dialog.uri.local)+1);
621 strcat( ptr, inet_ntoa (dialog.address.local) );
622 strcat( ptr, ">");
623 }
624 //FUNCION de generacion
625 //esta funcion esta hecha para la generacion de
626 //la cabecera TO en mensajes de respuesta
627 void generate_to_response(u_int8_t* ptr){
```

```
628 char temp[20];
629 strcat( ptr, "<" );
630 strcat( ptr, dialog.uri.local );
631 strcat( ptr, ">" );
632 strcat( ptr, ";tag=");
633 strcat( ptr, dialog.dialog_id.local_tag );
634 }
635
636 /*FUNCIONES de copiado
637 Estas funcionoes solo copian un valor ya almacenado en
638 en el mensaje SIP
639 */
640 void copy_call_id(u_int8_t* ptr){
641 strcat( ptr, dialog.dialog_id.call_id );
642 }
643 //FUNCION de copiado
644 void copy_from(u_int8_t* ptr){
645 strcat( ptr, "<");
646 strcat( ptr, dialog.uri.local);
647 strcat( ptr, ">;tag=");
648 strcat( ptr, dialog.dialog_id.local_tag );
649 }
650 //FUNCION de copiado
651 void copy_to(u_int8_t* ptr){
652 strcat( ptr, "<");
653 strcat( ptr, dialog.uri.remote );
654 strcat( ptr, ">;tag=");
655 strcat( ptr, dialog.dialog_id.remote_tag );
656 }
657 //FUNCION de copiado
658 void copy_cseq(u_int8_t* ptr){
659 char temp[10];
660 strcat( ptr, itoa(dialog.cseq.local, temp, 10) );
661 strcat( ptr, dialog.cseq.metodo_local );
662 }
663 //FUNCION de copiado
664 //Cuando el mensaje SIP genera un metodo ACK fuera
665 //del contexto de un metodo INVITE, debido a una respuesta 200
666 //el copiado de la cabecera cseq es ligeramente diferente
667 //pues el metodo enviado es un ACK en vez de un INVITE
668 //y el numero de secuencia es incrementado
669 void copy_cseq_ack(u_int8_t* ptr){
670 char temp[10];
671 strcat( ptr, itoa(dialog.cseq.local++, temp, 10) );
672 strcpy( dialog.cseq.metodo_local, " ACK" );
673 strcat( ptr, dialog.cseq.metodo_local );
674 }
675 //FUNCION de copiado
676 void copy_remote_cseq(u_int8_t* ptr){
677 char temp[10];
678 strcat( ptr, itoa(dialog.cseq.remote, temp, 10) );
679 strcat( ptr, " " );
680 strcat( ptr, dialog.cseq.metodo_remoto );
681 }
682 //FUNCION de copiado
683 void copy_via_response(u_int8_t* ptr){
684 strcat( ptr, dialog.via_temp );
685 }
686 //FUNCION de copiado
```

```
687 void copy_from_response(u_int8_t* ptr){
688 strcat( ptr, "<");
689 strcat( ptr, dialog.uri.remote );
690 strcat( ptr, ">");
691 strcat( ptr, ";tag=");
692 strcat( ptr, dialog.dialog_id.remote_tag );
693 }
694 //FUNCION de copiado
695 void copy_to_response(u_int8_t* ptr){
696 char temp[20];
697 strcat( ptr, dialog.uri.remote );
698 strcat( ptr, ";tag=");
699 strcat( ptr, dialog.dialog_id.local_tag );
700 }
701
702 /*FUNCIONES destroy
703 estas funciones borran los datos que identifican a la sesion
704 y se utilizan al terminar la misma para limpiarlos y obtener
705 nuevos datos para la siguiente sesion
706 */
707 void destroy_call_id(void){
708 strcpy(dialog.dialog_id.call_id,"");
709 }
710 //FUNCION destroy
711 void destroy_via(void){
712 strcpy(dialog.via_temp,"");
713 }
714 //FUNCION destroy
715 void destroy_from(void){
716 strcpy(dialog.dialog_id.remote_tag ,"");
717 }
718
719 /*FUNCIONES Gen
720 son las funciones pricipales encargadas de copiar los datos
721 para generar los mensajes SIP
722 existen tres funciones para generar métodos
723 gen_invite: para generar un mensaje INVITE
724 gen_ack : para generar un mensaje ACK
725 gen_bye : para generar un mensaje BYE
726
727 además existe una funcion para la generación de
728 los mensajes de respuesta
729 */
730 int gen_invite(void){
731 char* ptr;
732 sip_out_buffer[0] = '\0';
733 ptr = sip_out_buffer;
734 gen_first_line(ptr, INVITE);
735 strcat(ptr, "\n");
736
737 strcat(ptr, "call_id:");
738 generate_call_id(ptr);
739 strcat(ptr, "\n");
740
741 strcat(ptr,"via:");
742 generate_via(ptr);
743 strcat(ptr, "\n");
744
745 strcat(ptr,"max-forwards:");
```

```
746 generate_max_forwards(ptr);
747 strcat(ptr, "\n");
748
749 strcat(ptr,"contact:<");
750 generate_contact(ptr);
751 strcat(ptr,">");
752 strcat(ptr, "\n");
753
754 strcat(ptr,"from:");
755 generate_from(ptr);
756 strcat(ptr, "\n");
757
758 strcat(ptr,"to:");
759 generate_to(ptr);
760 strcat(ptr, "\n");
761
762 strcat(ptr,"cseq:");
763 generate_cseq(ptr, INVITE);
764 strcat(ptr, "\n");
765
766 strcat(ptr,"content-type:");
767 generate_content_type(ptr);
768 strcat(ptr, "\n");
769
770 strcat(ptr,"content-length:");
771 generate_content_length(ptr);
772 strcat(ptr, "\n");
773
774 // generate_sdp_packet(ptr);
775
776 return strlen(sip_out_buffer);
777
778 }
779 //FUNCION Gen
780 //esta funcion gen recibe una bandera {ACK, X}
781 //si elparametro es ACK se genera una cabecera cseq
782 //correspondiente a un metodo ACK
783 //en caso contrario se genera como la respuesta a un
784 //mensaje de confirmación o error luego de un INVITE
785 int gen_ack(int flag){
786 char* ptr;
787 ptr = sip_out_buffer;
788 gen_first_line(ptr, ACK);
789 strcat(ptr, "\n");
790
791 strcat(ptr, "call_id:");
792 copy_call_id(ptr);
793 strcat(ptr, "\n");
794
795 strcat(ptr,"via:");
796 generate_via(ptr);
797 strcat(ptr, "\n");
798
799 strcat(ptr,"max-forwards:");
800 generate_max_forwards(ptr);
801 strcat(ptr, "\n");
802
803 strcat(ptr,"from:");
804 copy_from(ptr);
```

```
805 strcat(ptr, "\n");
806
807 strcat(ptr,"to:");
808 copy_to(ptr);
809 strcat(ptr, "\n");
810
811 strcat(ptr,"cseq:");
812 if(flag == ACK){
813 copy_cseq_ack(ptr);
814 }
815 else{
816 copy_remote_cseq(ptr);
817 }
818 strcat(ptr, "\n");
819
820 return strlen(sip_out_buffer);
821 }
822 //FUNCION Gen
823 int gen_bye(void){
824 char* ptr;
825 ptr = sip_out_buffer;
826 gen_first_line(ptr, BYE);
827 strcat(ptr, "\n");
828
829 strcat(ptr, "call_id:");
830 copy_call_id(ptr);
831 strcat(ptr, "\n");
832
833 strcat(ptr,"via:");
834 generate_via(ptr);
835 strcat(ptr, "\n");
836
837 strcat(ptr,"max-forwards:");
838 generate_max_forwards(ptr);
839 strcat(ptr, "\n");
840
841 strcat(ptr,"from:");
842 copy_from(ptr);
843 strcat(ptr, "\n");
844
845 strcat(ptr,"to:");
846 copy_to(ptr);
847 strcat(ptr, "\n");
848
849 strcat(ptr,"cseq:");
850 generate_cseq(ptr, BYE);
851 strcat(ptr, "\n");
852
853 return strlen(sip_out_buffer);
854
855 }
856 //FUNCION Gen
857 int gen_response(int numero_respuesta){
858 char* ptr = sip_out_buffer;
859 if( numero_respuesta == 180 ){
860 strcpy(ptr, "SIP/2.0 180 Repicando");
861 strcat(ptr, "\n");
862
863 strcat(ptr, "call_id:");
```

```
864 copy_call_id(ptr);
865 strcat(ptr, "\n");
866
867 strcat(ptr, "via:");
868 copy_via_response(ptr);
869 strcat(ptr, "\n");
870
871 strcat(ptr,"max-forwards:");
872 generate_max_forwards(ptr);
873 strcat(ptr, "\n");
874
875 strcat(ptr,"contact:");
876 generate_contact_response(ptr);
877 strcat(ptr, "\n");
878
879 strcat(ptr,"from:");
880 copy_from_response(ptr);
881 strcat(ptr, "\n");
882
883 strcat(ptr,"to:");
884 generate_to_response(ptr);
885 strcat(ptr, "\n");
886
887 strcat(ptr,"cseq:");
888 copy_remote_cseq(ptr);
889 strcat(ptr, "\n");
890
891 strcat(ptr,"content-type:");
892 generate_content_type(ptr);
893 strcat(ptr, "\n");
894
895 strcat(ptr,"content-length:");
896 generate_content_length(ptr);
897 strcat(ptr, "\n");
898
899 //sdp
900
901 return strlen(sip_out_buffer);
902 }
903 else if( numero_respuesta == 200 ){
904 strcpy(ptr, "SIP/2.0 200 ok");
905 strcat(ptr, "\n");
906
907 strcat(ptr, "call_id:");
908 copy_call_id(ptr);
909 strcat(ptr, "\n");
910
911 copy_via_response(ptr);
912 strcat(ptr, "\n");
913
914 strcat(ptr,"max-forwards:");
915 generate_max_forwards(ptr);
916 strcat(ptr, "\n");
917
918 strcat(ptr,"contact:");
919 generate_contact_response(ptr);
920 strcat(ptr, "\n");
921
922 strcat(ptr,"from:");
923 copy_from_response(ptr);
```

```
924 strcat(ptr, "\n");
925
926 strcat(ptr,"to:");
927 copy_to_response(ptr);
928 strcat(ptr, "\n");
929
930 strcat(ptr,"cseq:");
931 copy_remote_cseq(ptr);
932 strcat(ptr, "\n");
933
934 }
935 else if( numero_respuesta == 486 ){
936 strcpy(ptr, "SIP/2.0 486 Ocupado");
937 strcat(ptr, "\n");
938
939 strcat(ptr, "call_id:");
940 copy_call_id(ptr);
941 strcat(ptr, "\n");
942
943 strcat(ptr, "via:");
944 copy_via_response(ptr);
945 strcat(ptr, "\n");
946
947 strcat(ptr,"max-forwards:");
948 generate_max_forwards(ptr);
949 strcat(ptr, "\n");
950
951 strcat(ptr,"contact:");
952 generate_contact_response(ptr);
953 strcat(ptr, "\n");
954
955 strcat(ptr,"from:");
956 copy_from_response(ptr);
957 strcat(ptr, "\n");
958
959 strcat(ptr,"to:");
960 generate_to_response(ptr);
961 strcat(ptr, "\n");
962
963 strcat(ptr,"cseq:");
964 copy_remote_cseq(ptr);
965 strcat(ptr, "\n");
966 }
967 else if( numero_respuesta == 487 ){
968 strcpy(sip_out_buffer, "SIP/2.0 487 Error");
969 strcat(ptr, "\n");
970
971 strcat(ptr, "call_id:");
972 copy_call_id(ptr);
973 strcat(ptr, "\n");
974
975 strcat(ptr, "via:");
976 copy_via_response(ptr);
977 strcat(ptr, "\n");
978
979 strcat(ptr,"max-forwards:");
980 generate_max_forwards(ptr);
981 strcat(ptr, "\n");
982
983 strcat(ptr,"contact:");
```

```
984 generate_contact_response(ptr);
985 strcat(ptr, "\n");
986
987 strcat(ptr,"from:");
988 copy_from_response(ptr);
989 strcat(ptr, "\n");
990
991 strcat(ptr,"to:");
992 generate_to_response(ptr);
993 strcat(ptr, "\n");
994
995 strcat(ptr,"cseq:");
996 copy_remote_cseq(ptr);
997 strcat(ptr, "\n");
998 }
999 return (strlen(sip_out_buffer));
1000 }
1001
1002 /*void respuesta(int mensaje_primera_linea)
1003 La funcion coloca una respuesta provicional
1004 en la variable bandera_respuesta,
1005 basada en el estado del SIP y de la transaccion
1006 Esta resputesta provisional, puede ser luego modificada
1007 por la revision de cada cabecera del mesnaje SIP
1008 */
1009 void respuesta(int mensaje_primera_linea){
1010
1011 switch(mensaje_primera_linea){
1012 case (INVITE):
1013 switch(dialog.estado){
1014 case (ESPERA):
1015 switch(dialog.estado_tran){
1016 case (ESPERA):
1017 bandera_respuesta = R_REPICANDO;
1018 dialog.estado_tran = PROCESANDO;
1019 break;
1020 default:
1021 bandera_respuesta = R_OCUPADO;
1022 break;
1023 }
1024 break;
1025 default:
1026 bandera_respuesta = R_OCUPADO;
1027 break;
1028 }
1029 break; // case INVITE
1030 case (ACK):
1031 switch(dialog.estado){
1032 case(EN_LLAMADA):
1033 dialog.estado_tran = ESPERA;
1034 break;
1035 default:
1036 bandera_respuesta = R_ERROR;
1037 }
1038 break; //case ACK
1039 case (BYE):
1040 switch(dialog.estado){
1041 case(EN_LLAMADA):
1042 bandera_respuesta = 200;
```

```
1043 dialog.estado = ESPERA;
1044 dialog.estado_tran = ESPERA;
1045 break;
1046 default:
1047 bandera_respuesta = R_ERROR;
1048 }
1049 break; //case BYE
1050 case (R_REPICANDO):
1051 switch(dialog.estado){
1052 case(INICIO_LLAMADA):
1053 dialog.estado_tran = PROCESANDO;
1054 bandera_respuesta = 0;
1055 break;
1056 default:
1057 break;
1058 }
1059 break;
1060 case (R_OK):
1061 switch(dialog.estado){
1062 case(INICIO_LLAMADA):
1063 dialog.estado_tran = ESPERA;
1064 dialog.estado = EN_LLAMADA;
1065 bandera_respuesta = ACK;
1066 break;
1067 case(EN_LLAMADA):
1068 case(ESPERA):
1069 dialog.estado_tran = ESPERA;
1070 bandera_respuesta = 0;
1071 default:
1072 break;
1073 }
1074 break;
1075 case (R_OCUPADO):
1076 switch(dialog.estado){
1077 case(INICIO_LLAMADA):
1078 dialog.estado_tran = ESPERA;
1079 dialog.estado = ESPERA;
1080 bandera_respuesta = ACK;
1081 break;
1082 default:
1083 break;
1084 }
1085 break;
1086 case (R_ERROR):
1087 switch(dialog.estado){
1088 case(INICIO_LLAMADA):
1089 dialog.estado_tran = ESPERA;
1090 dialog.estado = ESPERA;
1091 bandera_respuesta = ACK;
1092 break;
1093 default:
1094 break;
1095 }
1096 break;
1097 }
1098 }
1099
1100 /*int decode_first_line( char* line)
1101 la funcion se encarga de revisar la primera linea de
```

```
1102 un mensaje SIP, o linea de inicio.
1103 y llamar ala funcion 'respuesta()' para que coloque una
1104 respuesta provisioinal en la variable bandera_respuesta
1105 */
1106 int decode_first_line( char* line){
1107 if( strncmp(line, "INVITE", 6) == 0 ){
1108 respuesta(INVITE);
1109 }
1110 else if( strncmp(line, "ACK", 3) == 0 ){
1111 respuesta(ACK);
1112 }
1113 else if( strncmp(line, "BYE", 3) == 0 ){
1114 respuesta(BYE);
1115 }
1116 else if( strncmp( line + 8 , "180", 3) == 0){
1117 respuesta(R_REPICANDO);
1118 }
1119 else if( strncmp( line + 8 , "200", 3) == 0){
1120 respuesta(R_OK);
1121 }
1122 else if( line[8] == '3' ){
1123 respuesta(R_ERROR);
1124 }
1125 else if( strncmp( line + 8 , "486", 3) == 0 ){
1126 respuesta(R_ERROR);
1127 }
1128 else if( strncmp( line + 8 , "487", 3) == 0 ){
1129 respuesta(R_ERROR); }
1130 else if( line[8] == '4' ){
1131 respuesta(R_ERROR);
1132 }
1133 else if( line[8] == '5' ){
1134 respuesta(R_ERROR);
1135 }
1136 else if( line[8] == '6' ){
1137 respuesta(R_ERROR);
1138 }
1139 return EXIT_SUCCESS;
1140 }
1141
1142 /*void sip_reinit(void)
1143 Reinicia los estados de SIP destruye los valores
1144 en call_id, via, y from, from_tag
1145 resetea el temporizador timerA
1146 */
1147 void sip_reinit(void){
1148 dialog.estado = ESPERA;
1149 dialog.estado_tran = ESPERA;
1150 T1a = 800;
1151 wait_timer(&timerA, 2 );
1152 destroy_call_id();
1153 destroy_via();
1154 destroy_from();
1155 }
1156
1157 //void send( int tamanyo )
1158 //envia el paquete en sip_buffer_out por UDP
1159 void send( int tamanyo ){
1160 sock_fastwrite( &dialog.sip_socket, sip_out_buffer, tamanyo
```

```
);
1161 }
1162
1163
1164 void sip_send(void){
1165 if( bandera_respuesta == R_REPICANDO ){
1166 send( gen_response(R_REPICANDO) );
1167 bandera_respuesta = 0;
1168 dialog.estado = REPICANDO;
1169 (*ringing)();
1170 }
1171 else if( bandera_respuesta == R_OK ){
1172 send( gen_response(R_OK) );
1173 bandera_respuesta = 0;
1174 }
1175 else if( bandera_respuesta == R_OCUPADO ){
1176 send( gen_response(R_OCUPADO) );
1177 bandera_respuesta = 0;
1178 }
1179 else if( bandera_respuesta == R_ERROR ){
1180 send( gen_response(R_ERROR) );
1181 bandera_respuesta = 0;
1182 }
1183 else if( bandera_respuesta == INVITE ){
1184 send( gen_invite() );
1185 bandera_respuesta = 0;
1186 }
1187 else if( bandera_respuesta == BYE ){
1188 send( gen_bye() );
1189 bandera_respuesta = 0;
1190 }
1191 else if( bandera_respuesta == ACK ){
1192 if( dialog.estado == EN_LLAMADA )
1193 send( gen_ack(ACK) );
1194 else
1195 send( gen_ack(0) );
1196
1197 bandera_respuesta = 0;
1198 }
1199 else if( ((*init_call)()) && dialog.estado == ESPERA ){
1200 (*get_uri)();
1201 send( gen_invite() );
1202 set_timer(&timerA,T1);
1203 set_timer(&timerB, 32*T1 );
1204 dialog.estado_tran = LLAMANDO;
1205 dialog.estado = LLAMANDO;
1206 }
1207 else if( ((*pick_up)()) && dialog.estado == REPICANDO ){
1208 send( gen_response(200) );
1209 ( *end_ringing )();
1210 }
1211 else if( ((*put_down)()) && dialog.estado == EN_LLAMADA ){
1212 send( gen_bye() );
1213 sip_reinit();
1214 }
1215
1216 }
1217
1218 /*int sip_init(void)
```

```
1219 Inicia el socket UDP utilizado por SIP
1220 revisa por la existencia de un archivo de configuracion
1221 para SIP, si este existe obtiene los datos de proxy
1222 y uri local sino se asignan dos valores por defecto
1223 inicia los valores de estado de SIP
1224 Coloca el valor de T1 para inicio de los temporizadores
1225 A y B en un valor de 800 milisegundos
1226 Obtiene la direccion IP local
1227 genera los valores cseq y to tag aleatoriamente
1228 */
1229 int sip_init(void){
1230 char temp[250], *ptr, *ptr2;
1231 int c;
1232 FILE* fptr;
1233 if ( !udp_open(&(dialog.sip_socket), \
1234 5060, \
1235 -1, \
1236 5060, \
1237 NULL) ) {
1238 return EXIT_FAILURE;
1239 }
1240 if( sock_recv_init(&(dialog.sip_socket), \
1241 sip_in_buffer, \
1242 SIP_BUFFER_SIZE) ){
1243 return EXIT_FAILURE;
1244 }
1245
1246 fptr = fopen("sip.cfg","r");
1247 if( fptr != NULL ){
1248 ptr = temp;
1249 while( ((*ptr) = fgetc(fptr)) != EOF ){
1250
1251 ptr++;
1252 }
1253 *ptr = '\0';
1254 ptr = temp;
1255 ptr2 = temp;
1256 while( ((*ptr) != '\n') && ((*ptr) != EOF)){
1257 ptr++;
1258 }
1259 *ptr = '\0';
1260 ptr++;
1261 strcpy( dialog.proxy_name, ptr2);
1262
1263 ptr2 = ptr;
1264 while( (*ptr) != '\n' && (*ptr) != EOF){
1265 ptr++;
1266 }
1267 *ptr = '\0';
1268 strcpy( dialog.uri.local, ptr2);
1269 }
1270 else{
1271 strcpy( dialog.proxy_name, "server@ucab.edu.ve");
1272 strcpy( dialog.uri.local, "sip:local@ucab.edu.ve");
1273 }
1274 dialog.estado = ESPERA;
1275 dialog.estado_tran = ESPERA;
1276 bandera_respuesta = 0;
1277 T1 = 800;
```

```
1278 T1a = T1;
1279 dialog.address.local = gethostid();
1280 dialog.cseq.local = rand();
1281 strcpy( dialog.dialog_id.local_tag, itoa(rand(), temp,10) );
1282 dialog.sdp_content = FALSE;
1283 return EXIT_SUCCESS;
1284 }
1285
1286 //void sip_close(void)
1287 //cierra el socket utilizado para SIP
1288 //terminando la comunicacion
1289 void sip_close(void){
1290 sock_close(&dialog.sip_socket);
1291 }
1292
1293 //int sip_estado(host_remoto_struct* host)
1294 //retrorna el estado de SIP y en una estructura
1295 //la direccion IP y el puerto remoto utilizado
1296 //para la comunicacion estos son valores utiles
1297 //para iniciar la comunicacion RTP
1298 int sip_estado(host_remoto_struct* host){
1299 (*host).IP = dialog.address.remote;
1300 (*host).puerto = 5004;
1301 return dialog.estado;
1302 }
1303
1304 void sip_recv(void){
1305 #define LINE_SIZE 1500
1306 int len, seen_nonws, i, temp_length;
1307 int bandera_primera_linea=TRUE;
1308 u_int8_t *line = sip_recv_buffer;
1309 u_int8_t * temp_ptr;
1310 packet_t pkt;
1311
1312
1313 //se lee el socket para revisar si hay un nuevo paquete SIP
1314 if( temp_length = sock_recv( &dialog.sip_socket, \
1315 sip_recv_buffer, \
1316 SIP_RECV_BUFFER_SIZE, \
1317 0 )) {
1318 pkt.contents = sip_recv_buffer;
1319 pkt.current = sip_recv_buffer;
1320 pkt.apparent_end = pkt.end = (u_int8_t*)(sip_recv_buffer
+ temp_length);
1321 while(1){
1322 len = get_packet_line(line, LINE_SIZE, &pkt);
1323 if (len <= 0)
1324 break;
1325
1326 //si se tiene una linea de solo espacioes,
1327 //es el fin del mensaje SIP
1328 seen_nonws = 0;
1329 for (i=0; i<strlen(line); i++){
1330 if (!isspace(line[i])){
1331 seen_nonws = 1;
1332 break;
1333 }
1334 }
1335 if ( seen_nonws == 0 )
```

```
1336 break;
1337
1338 //convierte cada linea en minusculas para simplificar
1339 //las comparaciones
1340 if( !bandera_primera_linea )
1341 for (i=0; line[i] == ':'; i++)
1342 line[i] = tolower(line[i]);
1343
1344 //Inicio de comparaciones
1345 if( !bandera_primera_linea ){
1346 temp_ptr = line;
1347 while( *temp_ptr != ':' )
1348 temp_ptr++;
1349 }
1350
1351 i=0;
1352 if( bandera_primera_linea ){
1353 decode_first_line( line );
1354 bandera_primera_linea = FALSE;
1355 }
1356 else {
1357 do{
1358 //se compara cada cadena de caracteres
1359 //en smatriz contra la linea actual
1360 //si hay una coincidencia en la comparacion
1361 //se llama a la funcion correspondiente
1362 //de tipo *cabecera*_Func
1363 //si no hay coincidencia se ignora
1364 //la cabecera
1365 if( strncmp(smatriz[i], \
1366 line, \
1367 max__(strlen(smatriz[i]), \
1368 (temp_ptr-line)) ) == 0 ) {
1369 (*smatrizFunc[i]) ( line );
1370 i = 0;
1371 break;
1372 }
1373 i++;
1374 if (i >= MAXCOMP)
1375 break;
1376 }//end do
1377 while (1);
1378 }//end else
1379 }//end while
1380
1381 #if 0
1382 //para futra implementacion SDP
1383 if( dialog.sdp_content > 0 ){
1384 dump_sdp(pkt, dialog.content_length );
1385 dialog.sdp_content = 0;
1386 }
1387 #endif
1388
1389 }//end if temp_lenght = sock_recv
1390 }//end funcion
1391
1392 /*void sip_check_timer(void)
1393 revisa los temporizadores TimerA y TimerB
1394 en caso de qe el TimerA haya exprado se reretransmite un
```

```
1395 mensaje INVITE
1396 si el timerB expira se reinia SIP, puesto no se ha obtenido
1397 respuesta al mensaje INVITE
1398 */
1399 void sip_check_timer(void){
1400 if( dialog.estado_tran == LLAMANDO && check_timer(&timerA)
){
1401 send( gen_invite() );
1402 T1a *= 2;
1403 set_timer(&timerA, T1a );
1404 }
1405 if( dialog.estado_tran == LLAMANDO && check_timer(&timerB)
){
1406 sip_reinit();
1407 }
1408 }
1409
1410 /*void sip_tick(void)
1411 funcion que es llamada de manera externa por el telefono
1412 para manejar los paquetes SIP y las respuestas
1413
1414 La funcion revisa por paquetes recibidos
1415 Envia respuestas o mensajes iniciales en caso de
1416 ser necesario
1417 y revisa los temporizadores para respuestas automaticas
1418 */
1419 void sip_tick(void){
1420 sip_recv();
1421 sip_send();
1422 sip_check_timer();
1423 }
1424
```