Biblioteca C-open-smpp-3.4

Este trabajo está dirigido a programadores C con conocimientos básicos del protocolo SMPP, al menos en la especificación y manejo de sesiones. Extendiendo un poco el desarrollo de la biblioteca, el alcance de la misma podría involucrar un metodo de desarrollo de protocolos sobre TCP/IP.

Historial de revisiones

Revisión 1.2 2012-01-02 Revisado por: RT
Version 1.10 de la librería, se agrega funcion de unpack2, para resolver la identificacion del CMD_ID desde el buffer.
Revisión 1.1 2006-10-28 Revisado por: RT
Version 1.8 de la librería, resolución de pendientes.
Revisión 1.0 2006-06-03 Revisado por: RT
Versión inicial

Tabla de contenidos

1. Introducción

El principal enfoque de la biblioteca, es trabajar en el empaquetado y desempaquetado de estructuras de dato. Así independientemente de que esta implementación sea del protocolo SMPP-3.4, el potencial de la misma es generar de manera simple cualquier protocolo propietario sobre TCP.

1.1. El objetivo planteado

Todo desarrollador que intente entrar en el mundo SMPP, invariablemente pasa por el kannel (http://www.kannel.org/). Sin embargo, el proyecto kannel es lo suficientemente grande como para desanimar a quien quiere implementar alguna solución que involucre solamente el SMPP.

La actual biblioteca, se basa en algunos trucos del código fuente del kannel, generando una forma para manipular estructuras de datos en base al precompilador de C. El resultado es una serie de funciones que empacan y desempacan estructuras de datos hacia o desde un buffer.

La intención de la misma, es proveer de una implementación de protocolo SMPP-3.4 solo en la parte de manejo de PDUs. Independizando de la misma el manejo de la conexión TCP y de la sesión SMPP. Como se vé, la biblioteca apunta a solucionar solamente el tema del protocolo, dejando a criterio del desarrollador el manejo de los otros niveles en la comunicación.

La actual versión de librería 1.8, incorpora los dos PDUs pendientes en la versión 1.7 completando la totalidad de los PDUs del SMPP-3.4. Los nuevos PDUs son SUBMIT_MULTI y SUBMIT_MULTI_RESP. Dichos PDUs son especiales ya que incorporan listas dinámicas de destinos. La implementación de las mismas fueron hechas siguiendo las primitivas de los parámetros opcionales.

1.2. Definición de la API

La API está definida en 4 funciones:

- **Función de empaquetado:** En esta función, el primer y último parámetro son representativos de la estructura de datos que se quiere empacar, el primero indentifica dicha estructura y el último es un puntero al objeto de dato.

```
int smpp34_pack( uint32_t type,    /* in  */
                 uint8_t *ptrBuf,  /* out */
                 int      ptrSize, /* out */
                 int     *ptrLen,  /* out */
                 void    *tt       /* in  */  )
```

- **Función de desempacado:** En esta función, se le pasa un puntero al buffer y la longitud del mismo, y la función devuelve un puntero a una estructura definida por el campo type.

```
int smpp34_unpack( uint32_t type,   /* in  */
                   void    *tt,     /* out */
                   uint8_t *ptrBuf, /* in  */
                   int     ptrLen,  /* in  */ )
```

- **Función de volcado de estructura (DUMP):** En esta función, los parámetros de entrada corresponden a la dirección de un objeto de dato, y la función parsea los valores de dicha estructura, dejando dicho texto en otro puntero de salida.

```
int smpp34_dumpPdu( uint32_t type,      /* in  */
                    uint8_t *dest,      /* out */
                    int      size_dest, /* in  */
                    void    *tt         /* in  */ )
```

- **Función de volcado de buffer (DUMP):** En esta función, los parámetros de entrada corresponden a la dirección de un buffer, la función imprime en un buffer de salida en formato imprimible el contenido del buffer.

```
int smpp34_dumpBuf( uint8_t *dest,  /* out */
                    int      destL, /* in  */
                    uint8_t *src,   /* in  */
                    int      srcL   /* in  */ )
```

Además de las funciones, se definen las variables globales que completan la API de desarrollo.

```
int smpp34_errno;
char smpp34_strerror[2048];
```

Se definen además todas las estructuras de datos propias del protocolo que se quiere implementar. En este caso hay una estructura de dato por cada PDU del SMPP-3.4.

```
typedef struct tlv_t tlv_t;
typedef struct dad_t dad_t;              /* used in SUBMIT_MULTI PDU      */
typedef struct udad_t udad_t;            /* used in SUBMIT_MULTI_RESP PDU */
typedef struct bind_transmitter_t bind_transmitter_t;
typedef struct bind_transmitter_resp_t bind_transmitter_resp_t;
typedef struct bind_receiver_t bind_receiver_t;
typedef struct bind_receiver_resp_t bind_receiver_resp_t;
typedef struct bind_transceiver_t bind_transceiver_t;
typedef struct bind_transceiver_resp_t bind_transceiver_resp_t;
typedef struct outbind_t outbind_t;
typedef struct unbind_t unbind_t;
typedef struct unbind_resp_t unbind_resp_t;
typedef struct generic_nack_t generic_nack_t;
typedef struct submit_sm_t submit_sm_t;
typedef struct submit_sm_resp_t submit_sm_resp_t;
typedef struct submit_multi_t submit_multi_t;
typedef struct submit_multi_resp_t submit_multi_resp_t;
typedef struct deliver_sm_t deliver_sm_t;
typedef struct deliver_sm_resp_t deliver_sm_resp_t;
typedef struct data_sm_t data_sm_t;
typedef struct data_sm_resp_t data_sm_resp_t;
typedef struct query_sm_t query_sm_t;
typedef struct query_sm_resp_t query_sm_resp_t;
typedef struct cancel_sm_t cancel_sm_t;
```

```
typedef struct cancel_sm_resp_t cancel_sm_resp_t;
typedef struct replace_sm_t replace_sm_t;
typedef struct replace_sm_resp_t replace_sm_resp_t;
typedef struct enquire_link_t enquire_link_t;
typedef struct alert_notification_t alert_notification_t;
```

La descripción de cada una de estas estructura esta detallada en la especificación misma del protocolo. Como otro de los objetivos planteados, la implementación de cada estructura no difiere en nada de la especificación del protolo (salvo en el tema de parámetros opcionales, donde se usa una lista de datos dinámicos).

Para manejar parámetros opcionales dentro del protocolo SMPP-3.4 se han agregado 2 funciones que manejan listas dinamicas de TLVs.

```
int build_tlv( tlv_t **dest, tlv_t *source );
int destroy_tlv( tlv_t *sourceList );
```

Para manejar las listas dinámicas de los destinos en los PDUs de SUBMIT_MULTI y SUBMIT_MULTI_RESP se agregan las funciones.

```
int build_dad( dad_t **dest, dad_t *source );
int destroy_dad( dad_t *sourceList );
int build_udad( udad_t **dest, udad_t *source );
int destroy_udad( udad_t *sourceList );
```

Consulte la especificación del protocolo SMPP-3.4 así como los ejemplos provistos, para manipular dichos PDUs.

A continuación, se verá en más detalle cada una de estas funciones. La biblioteca cuenta con ejemplos para todos los PDUs del protocolo SMPP-3.4.

2. Función de empaquetamiento de estructuras

Como vemos la función `smpp34_pack( ... )` toma cinco parámetros y retorna un valor entero que describe el resultado de la operación. Un valor distinto de 0 en el retorno, indica que hubo un error en el intento de empaquetado, luego hay una descripción en modo texto en la variable global `smpp34_strerror`.

```
extern int smpp34_errno;
extern char smpp34_strerror[2048];

int smpp34_pack( uint32_t type,    /* in  */
                 uint8_t *ptrBuf,  /* out */
                 int      ptrSize, /* out */
                 int     *ptrLen,  /* out */
                 void    *tt       /* in  */  )
```

Donde:

type: es el command_id del PDU que se quiere empacar, el valor de este parámetro está directamente relacionado a una estructura de datos especifica.

ptrBuf: es un puntero a un buffer, donde va a almacenarse el PDU empacado. La memoria debe ser reservada de manera externa, ya sea dinámica o estática.

ptrSize: es un entero que describe el largo del buffer destino (el parámetro anterior).

ptrLen: en caso de éxito en la llamada, en esta variable queda el largo de la data dentro del buffer. Obviamente, siempre **ptrLen < ptrSize**.

tt: Es un puntero a una de las estructuras de datos listadas en la introducción y que se corresponde con el valor del primer parámetro.

2.1. Ejemplo de uso

Se detalla a continuación un pequeño ejemplo de uso, en el mismo, se detalla la creación del objeto de dato, la carga de información en la estructura, y el empaquetamiento de la misma.

Ejemplo 1. Ejemplo de pack y dumpBuff.

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <netinet/in.h>
#include "smpp34.h"
#include "smpp34_structs.h"
#include "smpp34_params.h"

char bufPDU[2048];
int  bufPDULen = 0;
char bPrint[2048];

    int
main( int argc, char *argv[] )
{

    int  ret = 0;
    enquire_link_t pdu;

    /* Init PDU ***************************************************/
    memset(&pdu, 0, sizeof(enquire_link_t));
    pdu.command_length   = 0;
    pdu.command_id       = ENQUIRE_LINK;           /* defined in smpp34.h */
    pdu.command_status   = ESME_ROK;               /* defined in smpp34.h */
    pdu.sequence_number  = 1;

    /* Linealize PDU to buffer ************************************/
    memset(&bufPDU, 0, sizeof(bufPDU));
    ret = smpp34_pack( pdu.command_id,
            bufPDU, sizeof(bufPDU), &bufPDULen, (void*)&pdu);
```

```
    if( ret != 0 ){
        printf("Error in smpp34_pack():%d:\n%s\n",
                smpp34_errno, smpp34_strerror);
        return( -1 );
    };

    /* Print Buffer ******************************************************/
    memset(bPrint, 0, sizeof(bPrint));
    ret = smpp34_dumpBuf(bPrint, sizeof(bPrint), bufPDU, bufPDULen);
    if( ret != 0 ){
        printf("Error in smpp34_dumpBuf():%d:\n%s\n",
                smpp34_errno, smpp34_strerror );
        return( -1 );
    };

    printf("The PDU enquire_link is packet in\n%s", bPrint);
    return( 0 );
};
```

Por último, este mismo ejemplo nos va a servir en la descripción de la función `smpp34_dumpBuf( ... )` . El ejemplo se compila y ejecuta:

```
[rtremsal@localhost dist]$ gcc -o lo lo.c -I./include -static -L./lib -lsmpp34
[rtremsal@localhost dist]$ ./lo
The PDU enquire_link is packet in
   00 00 00 10 00 00 00 15   00 00 00 00 00 00 00 01   ........ ........
```

3. Función de desempaquetamiento de estructuras

Como vemos la función `smpp34_unpack( ... )` toma cuatro parámetros y retorna un valor entero que describe el resultado de la operación. Un valor distinto de 0 en el retorno, indica que hubo un error en el intento de desempaquetado, luego hay una descripción en modo texto en la variable global `smpp34_strerror`.

```
extern int smpp34_errno;
extern char smpp34_strerror[2048];

int smpp34_unpack( uint32_t type,   /* in  */
                   void    *tt,     /* out */
                   uint8_t *ptrBuf, /* in  */
                   int     ptrLen,  /* in  */ )
```

Donde:

type: es el command_id del PDU que se quiere desempacar, el valor de este parámetro esta directamente relacionado a una estructura de datos específica.

tt: Es un puntero a una de las estructuras de datos listadas en la introducción y que se corresponde con el valor del primer parámetro. El puntero a dicha esctructura describe donde va a volcare el contenido del buffer.

ptrBuf: es un puntero a un buffer, donde va a almacenado el PDU empacado.

ptrLen: en esta variable va el largo del buffer anteriormente descripto.

3.1. Ejemplo de uso

Se detalla a continuación un pequeño ejemplo de uso, en el mismo, se detalla la creación de un buffer con data binaria. Luego se aplica la función de desempacado y se carga una estructura de datos con la data.

Ejemplo 2. Ejemplo de unpack y dumpPdu.

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <netinet/in.h>
#include "smpp34.h"
#include "smpp34_structs.h"
#include "smpp34_params.h"

char bufPDU[] = { 0x00, 0x00, 0x00, 0x36, 0x00, 0x00, 0x00, 0x01,
                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
                  0x73, 0x79, 0x73, 0x74, 0x65, 0x6D, 0x5F, 0x69,
                  0x64, 0x00, 0x70, 0x61, 0x73, 0x73, 0x00, 0x73,
                  0x79, 0x73, 0x74, 0x65, 0x00, 0x00, 0x02, 0x01,
                  0x61, 0x64, 0x64, 0x72, 0x65, 0x73, 0x73, 0x5F,
                  0x72, 0x61, 0x6E, 0x67, 0x65, 0x00 };

int  bufPDULen = 0;
char bPrint[2048];

    int
main( int argc, char *argv[] )
{

    int  ret = 0;
    bind_receiver_t pdu;
    uint32_t tempo;
    uint32_t cmd_id;

    /* Init PDU ***************************************************/
    memset(&pdu, 0, sizeof(bind_receiver_t));
    memset(&bPrint, 0, sizeof(bPrint));
    memcpy(&tempo, bufPDU+4, sizeof(uint32_t));
    cmd_id = ntohl( tempo );
```

```
    /* unpack PDU ***********************************************************/
    ret = smpp34_unpack(cmd_id, (void*)&pdu, bufPDU, sizeof(bufPDU));
    if( ret != 0){
        printf( "Error in smpp34_unpack():%d:%s\n",
                smpp34_errno, smpp34_strerror);
        return( -1 );
    };

    /* Print PDU ************************************************************/
    memset(bPrint, 0, sizeof(bPrint));
    ret = smpp34_dumpPdu(cmd_id, bPrint, sizeof(bPrint), (void*)&pdu );
    if( ret != 0){
        printf( "Error in smpp34_dumpPdu():%d:%s\n",
                smpp34_errno, smpp34_strerror );
        return( -1 );
    };
    printf("El PDU recibido es: \n%s\n", bPrint);

    return( 0 );
};
```

Por último, este mismo ejemplo nos va a servir en la descripción de la función `smpp34_dumpPdu( ... )` .

El ejemplo se compila y ejecuta:

```
[rtremsal@localhost dist]$ gcc -o ll ll.c -I ./include -static -L ./lib -lsmpp34
[rtremsal@localhost dist]$ ./ll
El PDU recibido es:
command_length                [00000036] - [54]
command_id                    [00000001] - [BIND_RECEIVER]
command_status                [00000000] - [ESME_ROK]
sequence_number               [00000001] - [1]
system_id                     [system_id]
password                      [pass]
system_type                   [syste]
interface_version             [00]       - [0]
addr_ton                      [02]       - [TON_National]
addr_npi                      [01]       - [NPI_ISDN_E163_E164]
address_range                 [address_range]
```

4. Otra función de desempaquetamiento de estructuras

La función `smpp34_unpack2( ... )` tiene el mismo efecto que `smpp34_unpack( ... )`, pero no necesita el parámetro de command_id. En la librería todas las funciones de pack/unpack necesitan el command_id ya que el parsing del buffer depende de este parámetro. La función toma tres parámetros y retorna un valor entero que describe el resultado de la operación. Un valor distinto de 0 en el retorno, indica que hubo un error en el intento de desempaquetado, luego hay una descripción en modo texto en la variable global `smpp34_strerror`. El detalle del código de la función se muestra abajo.

```
extern int smpp34_errno;
extern char smpp34_strerror[2048];

int smpp34_unpack2( void    *tt,     /* out */
                    uint8_t *ptrBuf, /* in  */
                    int     ptrLen,  /* in  */ )
{
    uint32_t cmdid;
    uint32_t tempo;
    memcpy(&tempo, ptrBuf + 4, sizeof(uint32_t)); /* get command_id PDU */
    cmdid = ntohl( tempo );
    return( smpp34_unpack(cmdid, tt, ptrBuf, ptrLen) );
};
```

Donde:

tt: Es un puntero a una de las estructuras de datos listadas en la introducción y que se corresponde con el valor del primer parámetro. El puntero a dicha esctructura describe donde va a volcare el contenido del buffer.

ptrBuf: es un puntero a un buffer, donde va a almacenado el PDU empacado.

ptrLen: en esta variable va el largo del buffer anteriormente descripto.

5. Función de volcado de estructuras (DUMP)

Como vemos la función `smpp34_dumpPdu( ... )` toma cuatro parámetros y retorna un valor entero que describe el resultado de la operación. Un valor distinto de 0 en el retorno, indica que hubo un error en el intento de volcado, luego hay una descripción del error en modo texto en la variable global `smpp34_strerror`.

```
extern int smpp34_errno;
extern char smpp34_strerror[2048];

int smpp34_dumpPdu( uint32_t type,      /* in  */
                    uint8_t *dest,      /* out */
                    int      size_dest, /* in  */
                    void    *tt         /* in  */ )
```

Donde:

type: es el command_id del PDU que se quiere volcar, el valor de este parámetro está directamente relacionado a una estructura de datos específica.

dest: es un puntero a un buffer, donde va a almacenarse el PDU volcado. La memoria debe ser reservada de manera externa, ya sea dinámica o estática.

size_dest: es un entero que describe el largo del buffer destino (el parámetro anterior).

tt: Es un puntero a una de las estructuras de datos listadas en la introducción y que se corresponde con el valor del primer parámetro.

5.1. Ejemplo de uso

El Ejemplo 2 descripto en la sección anterior, describe el uso de esta función.

6. Función de volcado de buffer (DUMP)

Como vemos la función `smpp34_dumpBuff( ... )` toma cuatro parámetros y retorna un valor entero que describe el resultado de la operación. Un valor distinto de 0 en el retorno, indica que hubo un error en el intento de volcado.

```
extern int smpp34_errno;
extern char smpp34_strerror[2048];

int smpp34_dumpBuf( uint8_t *dest,  /* out */
                    int      destL, /* in  */
                    uint8_t *src,   /* in  */
                    int      srcL   /* in  */ )
```

Donde:

dest: es un puntero a un buffer, donde va a almacenarse el buffer volcado. La memoria debe ser reservada de manera externa, ya sea dinámica o estática.

destL: es un entero que describe el largo del buffer destino (el parámetro anterior).

src: es un puntero al buffer binario de origen.

srcL: es un entero que describe el largo del buffer origen (el parámetro anterior).

6.1. Ejemplo de uso

El Ejemplo 1 descripto anteriormente, describe el uso de esta función.

7. Manejo de parámetros opcionales

Como vemos las funciones `build_tlv( ... )` y `destroy_tlv( ... )` permiten crear y destruir una lista dinámica de parámetros, que es parte de algunos PDUs del protocolo y que se denominan parámetros opcionales.

```
int build_tlv( tlv_t **dest,
               tlv_t  *source );

int destroy_tlv( tlv_t *sourceList );
```

Donde:

dest: es un puntero a una referencia a una estructura de tipo tlv_t. En el ejemplo que vemos más adelante se muestra el porque de esta doble referencia.

source: es un puntero a una estructura tlv_t.

sourceList: Este puntero es una referencia a tlv_t. En la funcion `destroy_tlv( ... )` el parámetro es una referencia a una lista linkeada.

7.1. Ejemplo de uso

El uso de parámetros opcionales esta limitado a un campo de tipo tlv_t que no existe en todas las estructuras de datos del PDU. Se muestra a continuación la forma en que debe manejarse estas estructuras.

```
#define TEXTO "mensaje de texto numero 01"
:
{
    submit_sm_t pdu;
    tlv_t tlv;

    memset(&tlv, 0, sizeof(tlv_t));
    tlv.tag = TLVID_user_message_reference;   /* tag present in submit_sm */
    tlv.length = sizeof(uint16_t);
    tlv.value.val16 = 0x0024;                                  /* valor */
    build_tlv( &(pdu.tlv), &tlv );    /* value attached to main structure */

    memset(&tlv, 0, sizeof(tlv_t));
    tlv.tag = TLVID_more_messages_to_send;    /* tag present in submit_sm */
    tlv.length = sizeof(uint8_t);
    tlv.value.val8 = 0x24;                                     /* valor */
    build_tlv( &(pdu.tlv), &tlv );    /* value attached to main structure */

    memset(&tlv, 0, sizeof(tlv_t));
    tlv.tag = TLVID_message_payload;          /* tag present in submit_sm */
    tlv.length = strlen(TEXTO);
    memcpy(tlv.value.octet, TEXTO, tlv.length);                /* valor */
    build_tlv( &(pdu.tlv), &tlv );    /* value attached to main structure */

    /* Pack and send data in pdu */
```

```
    destroy_tlv( pdu.tlv ); /* Free pdu list */
}
```

8. Manejo de listas de destinos en los PDUs SUBMIT_MULTI y SUBMIT_MULTI_RESP

Como en el caso anterior, ante la necesidad de manipular listas dinámicas de objetos de datos, esta vez en los PDUs SUBMIT_MULTI y SUBMIT_MULTI_RESP, se generan las funciones: `build_dad( ... )`, `build_udad( ... )`, `destroy_dad( ... )` y `destroy_udad( ... )`, que permiten crear y destruir listas dinámica de parámetros, en este caso de direcciones destinos.

```
int build_dad( dad_t **dest,
               dad_t  *source );
int destroy_dad( dad_t *sourceList );
int build_udad( udad_t **dest,
               udad_t  *source );
int destroy_udad( udad_t *sourceList );
```

Refierase a los ejemplos respectivos de `submit_multi_test` y `submit_multi_resp_test` para ver como manipular las listas. Si usted ya maneja parámetros opcionales no debería tener problemas para entender este funcionamiento.

9. Ejemplos incluidos en la librería.

9.1. Ejemplos para cada PDU.

La biblioteca tiene una aplicación de ejemplo por cada PDU del protocolo SMPP. Las aplicaciones se ejecutan sin parámetros y la idea es probar las funciones de la API a través de una serie de pasos:

1. Declarar 2 variables del mismo tipo, y cargar los campos que la componen en una de ellas.
2. Hacer un dump del PDU para ver la totalidad de los campos de la estructura.
3. Empacar la estructura en un buffer y hacer un dump del mismo.
4. Desempacar el buffer sobre la segunda variable definida en el punto 1.
5. Hacer un dump de la segunda estructura.

9.2. Un ESME de ejemplo.

Se incorpora a todos los ejemplos un pequeño ESME que hace el submit de un mensaje y es posible configurarlo según la siguiente sintaxis:

```
[rtremsal@localhost bin]$ ./esme
Error in parameters
usage: ./esme  -c file.xml [-h]
    -c /path/to/file.xml: config file path.
    -h : Help, show this message.

[rtremsal@localhost bin]$ more esme.xml
<?xml version="1.0"?>
<config>
    <conn_tcp host="127.0.0.1" port="9090"/>
    <conn_smpp system_id="sytem" password="asdfg34" system_type="type01"/>
    <smpp_msg src="5565" dst="0911110000" msg="Este es un ejemplo 01"/>
</config>
```

El archivo de configuración tiene parámetros para hacer la conexión tcp, la conexión smpp y por último los parámetros necesarios para enviar un mensaje. La aplicación se ejecuta como:

```
[rtremsal@localhost bin]$ ./esme -c esme.xml
Error in connect(127.0.0.1:9090)
Error in tcp connect.

[rtremsal@localhost bin]$ ./esme -c esme.xml
------------------------------------------------------------
SENDING PDU
command_length                [00000029] - [41]
command_id                    [00000002] - [BIND_TRANSMITTER]
command_status                [00000000] - [ESME_ROK]
:
:
:
------------------------------------------------------------
RECEIVE BUFFER
   00 00 00 10 80 00 00 06   00 00 00 00 00 00 00 03   ........ ........

RECEIVE PDU
command_length                [00000010] - [16]
command_id                    [80000006] - [UNBIND_RESP]
command_status                [00000000] - [ESME_ROK]
sequence_number               [00000003] - [3]
```

En el primer caso, no habia un server smpp activo, por lo cual el error se da a nivel de tcp. En el segundo caso, y ayudado por un smpp server de pruebas `sctt` - (SMPP Client Test Tool) disponible en SMS Forum (http://www.smsforum.com).

En el segundo caso, se establece el envío del mensaje mediante los siguientes pasos.

- Realizar la conexión a nivel TCP.
- Realizar la conexión a nivel SMPP. Uso del PDU BIND con los datos de validación (hacemos la conexión en modo TRANSMITER donde solo podemos enviar mensajes). Esperamos una confirmación de conexion exitosa en el PDU BIND_TRANSMITER_RESP.
- Enviamos el mensaje (PDU de SUBMIT_SM). Recibimos la confirmación de envio en el PDU SUBMIT_SM_RESP. A partir de aquí, estamos seguros que el mensaje se entrego al otro peer.
- Realizamos la desconexión a nivel SMPP. Envio de PDU de UNBIND. Esperamos confirmación a través de UNBIND_RESP.
- Realizamos la desconexión a nivel de TCP.

Si bien este ejemplo, es muy básico, permite ver más en detalle el uso que un ESME le puede dar a la biblioteca. Por otro lado, se recomienda estudiar y modificar el ejemplo para completar el entendimiento.

10. Conclusiones y futuras actualizaciones.

El principal enfoque de la biblioteca, se basa en el manejo de estructuras de datos, si bien el protocolo implementado es el SMPP-3.4 la idea es expandir la biblioteca a un esquema de protocolos parametrizables.

El siguiente objetivo es la implementación de la biblioteca SMPP-5.0, así como la posibilidad de crear protocolos propietarios y eficientes.