Trabajo Práctico Final

COL

Elena Coronel, Diego Ramírez, Koichi Oguro, Ariel Méndez

Marzo - Abril 2013

Universidad Nacional de Asunción
Facultad Politécnica
Ingeniería Informática

Profesores
Ing. Jorge Villamayor
Ing. Klaus Pistilli

Índice general

Resumen

El sistema funciona de la siguiente manera:

El `main` contenido en `coldaemon.c` se ejecuta y realiza los pasos previos para la creación de un demonio, se encarga de llamar al módulo de configuración contenido en `config_module.c` el cuál a su vez utiliza un parser de `config_parser.c` para obtener los parámetros del sistema desde el fichero de configuración que se pasó como argumento al demonio al tiempo de invocarlo, estos parámetros determinan la ubicación del fichero de Control de Listas de Acceso, y el destino de los registros (logs), el tiempo máximo de espera en inactividad y el número de puerto donde escucha el servidor. Tras esto se entra al bucle principal del demonio que atiende por las señales, y lanza el hilo que escucha en el puerto definido.

Una vez lanzado el hilo que gestiona las conexiónes, el proceso que lanzó al hilo queda en espera de señales. El hilo lanzado, bloquea en escucha de una conexión nueva, valida si la conexión no excede el número de conexiones activas que tiene y lanza un nuevo hilo que atiende al cliente que sea conectado, este nuevo hilo gestiona las validaciones de los comandos introducidos por el usuario, autenticación y transacciones, y también gestiona el acceso a la base de datos, en todo momento el hilo registra en su bitácora las operaciones que se están realizando, y al finalizar indica al hilo principal que ha finalizado escribiendo en una estructura compartida.

1. Objetivos

1.1. Principales

1. Crear un demonio en el lenguaje C, que permita los cobros de los siguientes servicios, mediante sus parámetros de cobro.
2. Implementar en un solo script que implemente
    - Iniciar el Demonio
    - Apagar el Demonio
    - Reiniciar el Demonio
    - Recargar archivo de configuración del demonio
    - Estado actual del demonio

1.2. Secundarios

Utilización de:

- Hilos/Procesos
- IPC
- Demonios
- Sockets

2. Desarrollo

2.1. Implementación

2.1.1. coldaemon.c

`int main(int argc, char * argv[]);`

`argc` : número de argumentos en la invocación

`argv` : arreglo que contiene los argumentos

La función main tiene como función principal invocar al modulo de configuración del sistema y gestionar las señales que el sistema recibe, para ello hace uso de un modulo de configuración y sus rutinas de manejo de señales.

2.1.2. config_module.c

`char config_module(char * config_file, thread_arg * argumento);`

`config_file` : nombre completo del fichero de configuración

`argumento` : estructura que guarda los parámetros del sistema

La función `config_module` carga en argumento los parámetros que lee y parsea desde el fichero de configuración. retorna `0` en caso de éxito y cualquier otro valor en caso de error.

La función invoca al `cofnig_parser` para llenar los campos de la estructura argumento arma apropiadamente los paths de los ficheros `acl` y `log`, y verifica que el `acl` exista y pueda leerse para luego retornar.

`void dbg_print_thread_arg(thread_arg * argumento);`

`argumento` : estructura que guarda los parámentros del sistema

Esta función solo existe para propósitos de depuración, toma la estructura argumento e imprime sus valores en los registros del sistema (`/var/log/syslog` por defecto).

2.1.3. coredaemon.c

`void fin_hilo(thread_arg arg)`

`arg`: estructura que almacena los parámetros del sistema

Cierra adecuadamente la conexión de un hilo y elimina al hilo de la lista de hilos activos

`int recvtimeout(int socket, char *buffer, int len, int timeout)`

`socket` : descriptor del socket

`buffer` : espacio para almacenar la cadena leída

`len` : Longitud máxima de la cadena leible

`timeout` : Tiempo máximo de espera en inactividad

Este meodo implementa la función `recv` con el agregado de soportar desconexión por timeout.

Se hace uso de la función `select()` que toma el socket y verifica que haya paquetes en él con tiempo límite parametrizado, además está función hace uso de `limpiar_telnet()` que se encarga de sanear la cadena que llega de los clientes telnet.

`void * coredaemon(void * argumento)`

`argumento`: parámetros de configuración del sistema.

Esta función es el corazón lógico del demonio, es el hilo que realiza la interacción entre el cliente y el módulo de autenticación y la base de datos, formatea y muestra los mensajes al usuario y recibe e interpreta los mensajes enviados por el cliente.

`void limpiar_telnet(char * cadena);`

`cadena` : cadena a verificar si tiene dato satélite (añadido por el telnet)

Esta función busca el caracter `'\015'` dentro de la cadena y lo elimina de la misma, devolviendo una cadena limpia y libre de caracteres satélite.

2.1.4. thread_manager.c

`void * thread_manager(void * argumento);`

`argumento`: estructura que guarda los parámetros del sistema

Este es el administrador de hilos y gestor de las conexiones entrantes, escuha por conexiones nuevas y las asigna a un nuevo hilo si se pueden validar.

2.1.5. utils.c

`void writelog(int log_fd, const char * mensaje);`

`log_fd` : descriptor de fichero de logs

`mensaje` : mensaje a escribirse en el log

Esta sencilla función escribe en el log de manera bloqueante lo que cada hilo tenga para registrar.

`void thread_add(struct thread_list **lista, int index);`

`lista` : referencia a la lista hilos activos

`index` : número lógico de hilo por añadir

Este método opera sobre la estructura lista, la recorre y ubica un nuevo elemento en ella que corresponde a un hilo identificado por index.

`pthread_t * thread_get(struct thread_list *lista, int index);`

`lista` : referencia a la lista hilos activos

`index` : número lógico de hilo por añadir

Función que toma la lista de hilos y un índice, ubica el hilo referente a dicho índice y retorna un puntero a dicho hilo.

`void thread_del(struct thread_list **lista, int index);`

`lista` : referencia a la lista hilos activos

`index` : número lógico de hilo por añadir

Función que borra un hilo de la lista, referenciado por index.

`uint32_t hash( char * str)`

`str` : cadena de la cuál se obtendrá el hash

Esta función calcula el hash de una cadena de caractéres.

`char authentication (char * acl_file, char * user, uint32_t pass_buscado)`

`acl_file` : fichero de acl

`user` : nombre de usuario

`pass_buscado` : clave de usuario

Esta función busca en el fichero de ACL los pares user y pass_buscado (previo hash) para retornar 0 en caso de éxito o cualquier otro valor en caso de error.

2.1.6. Parser

2.1.6.1. config_parser

```
int config_parser(char * config_file, int * puerto, int * threads, int
*timeout, char ** logpath, char ** logfile, char ** aclpath, char ** aclfile)
```

`coldaemon.c` utiliza `config_parser` para obtener los datos que se encuentran parametrizados en el archivo de configuración `cold.properties`.

Mediante la utilización de token (`tokPtr`) se extraen los datos almacenados y se asignan a los parámetros:

- `puerto`: número de puerto.
- `threads`: cantidad de threads utilizados.
- `timeout`: tiempo de espera.
- `logpath`: ubicación del log.
- `logfile`: nombre del log.
- `aclopath`: ubicación del log de usuarios.
- `aclfile`: nombre del log de usuarios.

2.1.6.2. transa_parser

```
char col_parser (SERVICIO *servicio, char * patron,int log_fd)
```

`coredaemon` utiliza `col_parser` para obtener los datos correspondientes a los parámetros de servicios sobre los cuales se realizan las transacciones.

Es necesario parsear:

- `codser`: código de servicio.
- `numtran`: número de transacción.
- `fechahora`: fecha y hora de transacción.
- `tipofact`: tipo de factura.
- `comprobante`: número de comprobante.
- `monto`: monto de la factura.
- `vencimiento`: fecha de vencimiento.
- `verificador`: digito verificador.
- `prefijo`: prefijo.

- `número:` número telefónico.
- `nummed:` número de medidor.
- `abonado:` número de abonado.
- `mensaje:` Caracteres de mensaje.

Los campos son rellenados de acuerdo al Servicio a ser cobrados, aquellos que no son utilizados son inicializados.

```
char rev_parser(SERVICIO *servicio, char * patron)
```

`coredaemon` utiliza `rev_parser` de la cual obtiene los datos:

- `codser:` código de servicio.
- `numtran:` número de transacción.
- `fechahora:` fecha y hora de transacción.
- `mensaje:` Caracteres de mensaje.

los mismos son almacenados en una estructura `SERVICIO` para la posterior utilización por la Base de Datos.

2.1.7. Base de Datos

Base de datos utilizada: Postgresql

Estructura de la base de datos

- 3 tablas (pendientes, pagadas, transacciones)

1. Tabla pendientes:
   - `cod_serv tipo compr monto dig_verif prefijo numero medidor abonado id vencimiento`
2. Tabla pagadas:
   - `cod_serv tipo compr monto dig_verif prefijo numero medidor abonado transaccion fecha_hora usuario vencimiento`
3. Tabla transacciones:
   - `operacion cod_serv fecha_hora usuario mensaje tipo compr monto dig_verif prefijo numero medidor abonado transaccion`

Lógica de operación

1. Se recibe una operación junto con la estructura serv
2. Se determina el tipo de operación a realizar (col, rev, lastrx)
3. Si es col (cobro) se verifica que la factura a cobrar exista, no esté vencida, coincidan los montos y el codigo de transaccion sea el mismo, luego se agrega la factura a pagadas, se elimina de pendientes y se agrega la transacción a la tabla transacciones junto con el nombre del usuario que realizo el cobro.
4. Si es rev (reversa) se verifica que exista la transaccion en pagadas y se traslada de vuelta a la tabla pendientes, tambien se agrega una entrada a la tabla de transacciones.
5. Si es lastrx se consulta en la tabla de transacciones por las 3 ultimas transacciones del usuario actual y se concatenan en una cadena que luego es devuelta al cliente, la operacion lastrx es considerada como una transacción de consulta por lo que se agrega una entrada en la tabla de transacciones.

2.2. Operativa

2.2.1. coldaemon.c

Esta función empieza validando la cantidad de argumentos pasados al sistema que deben ser 2 y no más, el primer argumento es el nombre propiamente del demonio, por lo defecto cold y el segundo es el nombre completo del fichero de configuración que por defecto es /etc/cold.properties. Si esta validación se cumple el segundo argumento es pasado como parámetro a la función config_module, junto con una estructura que almacena los parámetros del sistema.

Por lo general hilo principal escribe sus logs sobre /var/log/syslog y solo registra estados del demonio y errores del mismo. Una vez que llega al bucle principal de trabajo. Establece las señales que serán atrapadas `TERM` y `HUP`, si es la primera vez en el buclea o se ha lanzado una señal HUP se crea un nuevo descriptor de socket y se lanza un hilo para manejar conexiones entrantes y lanzar nuevos hilos. En caso de no ser una señal de `HUP` o la primera entrada al bucle, el bucle solo verifica que no hayan llegado las señales `TERM` o `HUP`, de llegar `TERM` se cierran los logs y descriptores y se finaliza el demonio exitosamente. De llegar `HUP` se vuelve a llamar al módulo de configuración y se vuelve a crear el descriptor del socket para luego cancelar el hilo que gestione conexiones y volver a lanzar uno nuevo con los parámetros leídos por le modulo de configuración.

2.2.2. Coredaemon

`fin_hilo`: Este es un breve método que cierra la conexión que usa un hilo, bloquea exclusivamente la variable global ready que cuental los hilos activos para decrementarla, y elimina al hilo en cuestión de la lista de hilos activos.

`coredaemon`: La función empieza autenticando al cliente haciendo uso del módulo de autenticación, tras validar muestra los mensajes respectivos y llega al bucle principal donde lea los comandos del usuario los verifica y ejecuta las transacciones asociadas, en esta sección se hace uso frecuente del módulo de base de datos.

2.2.3. thread_manager.c

`thread_manager:` Esta función se encarga de abrir los logs, mostrar los primeros mensajes del sistema en su bitácora, enlaza el socket con su puerto al demonio y se pone en modo de escucha por conexiones entrantes. Al legar al bucle principal del programa se realiza llamada bloqueante accept() en espera de nuevas conexiones y se asignan las mismas a temp_sock_descriptor, verifica contra una variable compartida que todavía no se haya pasado el número máximo de hilos configurado, rechazando conexiones si es necesario, y cerrando el socket. Si todavía se permiten conexiones, se asignan adecuadamente las estructuras de parámetros del sistema y se cran los hilos.

2.2.4. Parser

2.2.4.1. config_parser

Para la configuración del demonio utilizamos `config_parser` el cual extrae los parámetros utilizados, los mismos se encuentran en el archivo de configuración utilizado por el mismo.

2.2.4.2. transa_parser

`col_parser:` realiza el parseo de la cadena de entrada, la cual contiene la transacción a ser realizada. En este caso se trata de cobro de servicios. El col_parser almacena los datos en la estructura a ser utilizada posteriormente por la Base de Datos, de manera a concretar la transacción.

`rev_parser:` realiza el parseo de la cadena obtenida, la misma contiene la transacción a ser revertida. Los datos son almacenados en la estructura SERVICIO para la utilización por la Base de Datos.

2.2.5. Base de Datos

Para realizar la conexión con la base de datos desde el demonio se utilizo la libreria `libpq-dev` que provee las siguientes funciones para la interacción con la base de datos:

- `PQconnectdb(conninfo)`

  Devuelve un objeto del tipo `PGconn` que es utilizado en las demas funciones de `libpq-dev` para realizar operaciones sobre la base de datos referenciada por el objeto `PGconn`.

- `PQexecParams`

  Envia un comando al servidor y aguarda por el resultado, con la habilidad de enviar parámetros en forma separada del texto del comando SQL.

  Este comando recibe un objeto del tipo `PGconn`, el comando SQL con los parámetros con la forma `$1, $2, $n`, la cantidad de parámetros y los valores de los parámetros en un vector, ademas recibe en casos especiales los tipos de los parametros enviados y el formato de resultado.

Los resultados de `PQexecParams` son verificados por las siguientes funciones:

- `PQresultStatus`

  Recibe un objeto tipo `PGresult` y devuelve su estado de ejecución, este resultado sera `PGRES_COMMAND_OK` si el comando fue ejecutado con exito.

- `PQgetvalue`

  Recibe el objeto `PGresult` y obtiene los valores referenciados por sus parametros, estos parametros apuntan a un numero de fila y columna especificos.

- `PQntuples`

  Devuelve la cantidad de filas retornada por el comando, recibe un objeto `PGresult`.

- `PQnfields`

  Devuelve la cantidad de columnas retornada por el comando, recibe un objeto `PGresult`.

Al finalizar una transaccion se limpia el objeto `PGresult` mediante la funcion `PQclear`

Para cerrar la conexión con la base de datos se utiliza `PQfinish` que recibe el objeto `PGconn`.

2.3. Section

Script de inicio del demonio

En la variable `$1` recibe el parametro que se le envia al script

- `start`
- `stop`
- `restart`
- `reload`
- `status`

Con `case` verificamos la orden recibida y se procede a la ejecucion del comando indicado

- `start`

  cold /etc/cold.properties

  Ejecuta el comando y lanza el demonio que se guardo en `/bin` y luego `cold.properties` se guardo en /etc/

- `stop`

  En la variable `pid` se guarda el `pid` del proceso llamado `cold`, `pidof` lanza el `pid` del proceso nombrado `cold`, si el proceso no esta corriendo devuelve vacio

  En el `if`, `-z` comprueba que lo que se encuentra en la variable `$pid` esta vacia

  - Si es verdadero, `pid="Vacio"` entonces no puede parar ya que no esta corriendo el demonio.
  - Si no da vacio detiene el servicio `kill -TERM` envia la señal de detenerse al demonio, mediante su `pid`.

- `restart`

  Se obtiene el pid del demonio.

  - Si es vacio no permite reiniciar.
  - Si no es vacio envia la señal `kill -TERM` al demonio para que se detenga, y luego lo lanza de nuevo.

- `reload`

  Se obtiene el pid del demonio.

  - Si es vacio no permite recargar.
  - Si no esta vacio envia la señal `kill -HUP` al demonio para que recargue el archivo de configuración.

- `status`

  Se obtiene el `pid` del demonio.

  - Si es vacio notifica que el demonio no esta activo.
  - Si no es vacio significa que el demonio esta corriendo y lo notifica con su numero de `pid`.
  - Si no recibio ninguno de estos parametros, notifica el modo en que se debe ser enviado.

Conclusiones

Durante la implementación del servicio se presentaros distintos problemas que fueron encarados con las herramientas de conocimiento.

Hicimos uso de sockets como esqueleto de la distribucion de las operaciones.

Utilizamos el parseo de configuraciones para obtener de manera individual las configuraciones del demonio tales como puerto, threads, timeout, entre otros. Ademas realizamos parseo de datos entrantes, para poder identificar cada uno de los parametros necesarios para la transacción.

Mediante hilos controlamos las peticiones de forma estricta (timeouts), capacidades de sobrecarga de peticiones y la bajada limpia del servidor sin provocar perdidas a los usuarios del mismo.

Realizamos uso de señales para hacer la modificacion de argumentos que interfieren en el uso del servicio.

La implementacion de semáforos se utilizo par bloquear la escritura en el archivo log.

Usamos scripts para el inicio del servicio al arrancar el sistema, asi como para dar reseteo, parada, recarga y otras propiedades como el estado del servicio de manera rápida.

Como base hacemos uso de la herramienta "demonio", para que el servicio se mantenga disponible en todo momento.

A. Anexos

A.1. Código

A.1.1. coldaemon (Script)

```
#!/bin/sh

case "$1" in
start)
echo "Iniciando servicio... "
# Aquí comando a ejecutar para arrancar el servicio
        cold /etc/cold.properties
;;
stop)
# Aquí comando a ejecutar para detener el servicio
        pid=`pidof cold`
        if [ -z $pid ]; then
                echo "No se puede detener el servicio, el demonio no esta corriendo
                    "
        else
                echo "Deteniendo servicio..."
                kill -TERM $pid
        fi
#TERM (15) senhal de detenerse

;;
restart)
# Aquí comando a ejecutar para reiniciar el servicio
        pid=`pidof cold`
        if [ -z $pid ]; then
                echo "No se puede reiniciar el servicio, el demonio no esta
                    corriendo"
        else
                echo "Reiniciando servicio..."
                kill -TERM $pid
                sleep 3
                cold /etc/cold.properties
        fi
#TERM (15)

;;
reload)
# Aquí comando a ejecutar para recargar el servicio
        pid=`pidof cold`
        if [ -z $pid ]; then
                echo "No se puede recargar.. el demonio no esta corriendo.."
        else
                echo "Recargando archivo de configuracion del servicio..."
                kill -HUP $pid
        fi
        #HUP (1)
```

```
;;
status)
        pid=‘pidof cold‘
        if [ -z $pid ]; then
                echo "El demonio no esta activo.."
        else
                echo "El demonio esta activo PID= $pid"

        fi

;;
*)
echo "Modo de empleo: sudo service coldaemon {start|stop|restart|reload|status}"
exit 1
;;
esac
exit 0
```

A.1.2. coldaemon.h

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <time.h>
#include <syslog.h>
#include <signal.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <postgresql/libpq-fe.h>
//#include <netbd.h>

/*

        Todas las variables en minúsculas, se usa guion bajo si es necesario
        ej; var, var_muy_larga
        Todos los define en mayúsculas
        ej: #define MACRO valor_macro
        todos los nombres de función en minúsculas

*/

// Definiciones
#define STR_LEN 512
#define PKG_LEN 16384

//Estructuras de datos
struct thread_list
{
        pthread_t hilo;
        int thread_index;
        struct thread_list * siguiente;
};

typedef struct
{
        int threads; // cantidad de hilos (thread manager)
        int puerto; // puerto de escucha
        char acl[STR_LEN]; // lista de control de acceso
        char log[STR_LEN]; // bitácora del demonio
        int timeout; // tiempo límite de espera
        struct thread_list * lista_hilo; // hilos en ejecución
        int thread_index; // identificador de hilo (thread worker)
        int socket_descriptor; // descriptor de socket (thread worker)
        struct sockaddr_in socket; // estructura socket (thread worker)
} thread_arg;

struct parameters
{
        thread_arg arg;
        struct parameters * siguiente;
};

typedef struct
{
        char *codser; //3 DIGITOS codigo de servicio
        int numtran; //6 DIGITOS numero de transaccion
        char *fechahora; //14 DIGITOS fecha y hora de transaccion
```

```
        char *tipofact; // 3 DIGITOS tipo de factura
        char *comprobante; //11 DIGITOS numero de comprobante
        long int monto; //12 DIGITOS monto de la factura
        char *vencimiento; //8 DIGITOS fecha de vencimiento
        int verificador; //1 DIGITOS digito verificador
        char *prefijo; //4 DIGITOS prefijo
        char *numero; //7 DIGITOS numero telefonico
        char *nummed; //15 DIGITOS numero de medidor
        char *abonado; //9 DIGITOS numero de abonado
        char *mensaje; //20 CARACTERES DE MENSAJE
}SERVICIO;

int ready;
pthread_mutex_t lock;

// Biblioteca de Funciones de:
// Hash, postgres, semaforos

#define OK 0
#define ARGUMENTOS_INVALIDOS 1
#define NO_CONFIG_FILE 2
#define INVALID_CONFIG_FILE 3
#define CANT_OPEN_ACL 13
#define LOG_ERROR 11
#define SOCK_DESCRIPTOR_ERROR 4
#define BINDING_ERROR 5
#define LISTENNING_ERROR 6
#define ACCEPT_CONNECTION_ERROR 7
#define CANT_FORK 8
#define SESSION_ERROR 9
#define CHDIR_ERROR 10
#define CONFIG_ERROR 12
#define NULL_THREAD 14
#define CANT_CLOSE_SOCKET 15
#define DB_EXIT_NICELY 20
#define CANT_READ_ACL 16
#define INVALID_USER 17

#define INVALID_COD_SERV 30
#define INVALID_YEAR 31
#define INVALID_DAY 32
#define INVALID_MONTH 33
#define INVALID_HOUR 34
#define INVALID_MIN 35
#define INVALID_SEC 36

/*
        acl_file es un puntero al nombre del fichero que tiene los datos de
            autenticación
        usuario es un puntero al nombre usuario
        clave es un puntero a la clave
        la función retorna 0 si la autenticación es exitosa y se debe definir
            códigos
        de error para cada caso de error
*/
//char authentication(char * acl_file, char * usuario, char * clave);


/*
        config_parser lee un archivo de configuración y establece los parámetros
            del demonio a partir del mismo
        retorna 0 si no hubo errores, establecer sus códigos de error en otro caso
        config_file, puntero al nombre del archivo de configuración
        puerto, threads, timeout, logpath, logfile, parámetros del demonio.
*/
int config_parser (char * config_file, int * puerto, int * threads, int * timeout,
```

```
    char ** logpath, char ** logfile, char ** aclpath, char ** aclfile);

/*
        EL CORE DAEMON
*/
void * coredaemon(void * argumento);
// Helper para limpiar el #015 que telnet envía como Retorno de Línea
void limpiar_telnet(char * cadena);

/*
        Parser de patrones de entrada
*/
char col_parser (SERVICIO *servicio, char * patron, int log_fd);
char rev_parser (SERVICIO *reversa, char * patron);

/*
        Módulo de Base de Datos
*/
int db_module(char * operacion, SERVICIO serv, char * usuario, int log_fd, char *
    resp);

/*
        UTILERIA GENERAL
*/
void writelog(int log_fd, const char * mensaje);
void thread_add(struct thread_list **lista, int index);
pthread_t * thread_get(struct thread_list *lista, int index);
void thread_del(struct thread_list **lista, int index);
uint32_t hash( char * str);
char authentication (char * acl_file, char * user, uint32_t pass_buscado);

/*
        Administrador de Hilos
*/
void * thread_manager(void * argumento);

/*
        Módulo de Configuración
*/
char config_module(char * config_file, thread_arg * argumento);
void dbg_print_thread_arg(thread_arg * argumento);
```

A.1.3. coldaemon.c

```
#include "coldaemon.h"
#include <pthread.h>

int main(int argc, char * argv[])
{
        char * config_file;
        pid_t pid, sid;
        char * logpath;
        char * logfile;
        char * aclpath;
        char * aclfile;
        char buf[PKG_LEN];
        char printBuffer[STR_LEN];
        int i, len, log_fd, ret, j;
        int socket_descriptor;
        pthread_t manager;
        thread_arg  argumento;
        int create_thread_value;
        char flag_asignado_hilo;
        struct sockaddr_in sin;
        FILE * temp_file;
        ready = 0;

        // Validación de Argumentos, debe haber un argumento que especifique
        // el fichero de configuración del demonio
        if(argc != 2)
        {
                syslog(LOG_ERR,"Se necesita el fichero de configuración\n");
                return ARGUMENTOS_INVALIDOS;
        }

        // fichero de configuración de los parámetros del demonio
        config_file = argv[1];
        // MODULO DE CONFIGURACION
        if( (ret = config_module(config_file, &argumento)) != 0)
        {
                syslog(LOG_ERR,"Error con los parámetros de configuración
                    config_module\n",ret);
                return CONFIG_ERROR;
        }


        // Creando el proceso huérfano y terminando el proceso padre
        pid = fork();
        if(pid < 0)
        {
                syslog(LOG_ERR,"No se puede crear proceso hijo\n");
                return CANT_FORK;
        }
        if(pid > 0)
        {
                return OK;
        }

        // Convirtiendo al huérfano en lider de la sesión
        sid = setsid();
        if(sid < 0)
        {
                syslog(LOG_ERR,"No se puede crear la sesión\n");
                return SESSION_ERROR;
        }

        //Cambiando al directorio / como directorio de trabajo
        if( (chdir("/")) < 0)
```

```
{
        syslog(LOG_ERR,"No se puede cambiar de directorio\n");
        return CHDIR_ERROR;
}

// Cambiando Mascara
umask(0);
// Cerrando descriptores estándard
close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);

openlog("cold", LOG_PID, LOG_DAEMON);
syslog(LOG_INFO,"Cobros On Line Daemon\n");

argumento.lista_hilo = NULL;
i = 0;
while(1)
{
        // MANEJO DE SEÑALES
        sigset_t sigset;
        struct sigaction action;

        sigemptyset(&sigset); /* Inicializa el conjunto de señales */
        sigaddset(&sigset, SIGHUP); /* Reload Config file */
        sigaddset(&sigset, SIGTERM); /* Parar el demonio */
        sigprocmask(SIG_BLOCK, &sigset, NULL); /* Bloqueamos las señales */
        sigpending(&sigset); /* Comprueba las señales pendientes */

        if(!i)
        {
                // SOCKET PARA ESCUCHAR CONEXIONES
                socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
                if( socket_descriptor == -1)
                {
                        syslog(LOG_ERR,"No se puede obtener un descriptor
                            de socket\n");
                        exit(SOCK_DESCRIPTOR_ERROR);
                }

                argumento.socket_descriptor = socket_descriptor;
                pthread_create(&manager,NULL, thread_manager, (void *) &
                    argumento);
                i = 1;
        }

        if(sigismember(&sigset, SIGHUP)) {
                syslog(LOG_INFO, "SIGHUP\n");
                sigemptyset(&action.sa_mask);
                action.sa_handler = SIG_IGN;
                sigaction(SIGHUP, &action, NULL);
                /*
                        Reconfigurar el demonio
                */
                socket_descriptor = argumento.socket_descriptor;
                syslog(LOG_INFO,"Recargando fichero de configuración");
                if( (ret = config_module(config_file, &argumento)) != 0)
                {
                        syslog(LOG_ERR,"Error con los parámetros de
                            configuración config_module (%d)\n",ret);
                        return CONFIG_ERROR;
                }
                pthread_cancel(manager);
                //syslog(LOG_INFO,"Socket cerrado tcp(%d)\n",
                    socket_descriptor);
                /*
```

```
                        if( shutdown(socket_descriptor, 2) == -1)
                        {
                                syslog(LOG_ERR,"No se puede cerrar el socket\n");
                                return CANT_CLOSE_SOCKET;
                        }
                        */
                        close(socket_descriptor);
                        i= 0; //FLAG de administrador de hilos no corriendo
                        sigprocmask(SIG_UNBLOCK, &sigset, NULL);
                }
                if(sigismember(&sigset, SIGTERM)) {
                        syslog(LOG_INFO, "SIGTERM\n");
                        /* Ignora SIGTERM */
                        sigemptyset(&action.sa_mask);
                        action.sa_handler = SIG_IGN;
                        sigaction(SIGTERM, &action, NULL);
                        /*
                                Parar el demonio
                        */
                        closelog();
                        close(argumento.socket_descriptor);
                        /* Desbloquea SIGHUP */
                        sigprocmask(SIG_UNBLOCK, &sigset, NULL);
                        syslog(LOG_INFO,"Parando el demonio\n");
                        exit(OK);
                }
                sleep(1);
        }

        return OK;
}
```

A.1.4. config_module.c

```
#include "coldaemon.h"

char config_module(char * config_file, thread_arg * argumento)
{
        char * logpath;
        char * logfile;
        char * aclpath;
        char * aclfile;
        int ret;
        int temp_fd;

        // Invocación al config parser
        if( (ret = config_parser(config_file, &(argumento->puerto), &(argumento->
            threads), &(argumento->timeout), &logpath, &logfile, &aclpath, &aclfile
            )) != OK )
        {
                syslog(LOG_ERR,"Fichero de configuración inválido, config_parser(%d
                    )\n",ret );
                return ret;
        }

        // Estableciendo el nombre completo del fichero acl
        strcpy(argumento->acl,aclpath);
        strcat(argumento->acl,aclfile);

        // Estableciendo el nombre completo del fichero log
        strcpy(argumento->log,logpath);
        strcat(argumento->log,logfile);

        // Liberando la memoria reservada en config_parser
        free(aclpath);
        free(aclfile);
        free(logpath);
        free(logfile);

        // Hacer validaciones sobre los parámetros acl, y log
        if( ( temp_fd  = open(argumento->acl, O_RDONLY) ) < 0 )
        {
                syslog(LOG_ERR,"No existe el archivo %s o no se puede abrir\n",
                    argumento->acl);
                return CANT_OPEN_ACL;
        }
        close(temp_fd);

        // Just for debugging purposes
        //dbg_print_thread_arg(argumento);

        return OK;
}

void dbg_print_thread_arg(thread_arg * argumento)
{
        syslog(LOG_DEBUG, "Printing ARG");
        syslog(LOG_DEBUG, "threads = %d\n", argumento->threads);
        syslog(LOG_DEBUG, "puerto = %d\n", argumento->puerto);
        syslog(LOG_DEBUG, "timeout = %d\n", argumento->timeout);
        syslog(LOG_DEBUG, "acl = %s\n", argumento->acl);
        syslog(LOG_DEBUG, "log = %s\n", argumento->log);
        syslog(LOG_DEBUG, "thread_index = %d\n", argumento->thread_index);
        syslog(LOG_DEBUG, "socket_descriptor = %d\n", argumento->socket_descriptor)
            ;

        return;
}
```

A.1.5. config_parser.c

```
#include "coldaemon.h"

int config_parser(char * config_file, int * puerto, int * threads, int *timeout,
    char ** logpath, char ** logfile, char ** aclpath, char ** aclfile)
{
        char *tokenPtr; //puntero para los tokens
        char string[100]; //string donde se almacenan las lineas de configuracion
        int i; //variable para el for
        FILE *ficheroPtr;
        //syslog(LOG_ERR,"config_file = %s\n",config_file);
        if(( ficheroPtr = fopen( config_file, "r")) == NULL)
                return INVALID_CONFIG_FILE;
        /* Comparacion de string con cada argumento de configuracion */
        for(i=0; i<=6;i++)
        {
        /* Lectura de los argumentos de configuración */
                fscanf( ficheroPtr, "%s", string);

                tokenPtr= strtok( string, "="); //se extrae el argumento hasta =
                if(strstr(tokenPtr,"port")) //si es port
                {
                        tokenPtr = strtok(NULL," "); //se extrae la configuracion
                        sscanf(tokenPtr,"%d", puerto); //se almacena

//free(tokenPtr); //se libera la memoria
                }
                if(strstr(tokenPtr,"threads")) //si es threads
                {
                        tokenPtr = strtok(NULL," ");
                        sscanf(tokenPtr,"%d", threads); //se almacena
                }
                if(strstr(tokenPtr,"timeout"))//si es timeout
                {
                        tokenPtr = strtok(NULL," ");
                        sscanf(tokenPtr,"%d", timeout); //se almacena
                }
                if(strstr(tokenPtr,"logpath"))//si es logpath
                {
                        tokenPtr = strtok(NULL," ");
                        *logpath = (char *) malloc(sizeof(char)*strlen(tokenPtr));
                        strcpy(*logpath, tokenPtr);
                }
                if(strstr(tokenPtr,"logfile"))//si es logfile
                {
                        tokenPtr = strtok(NULL," ");
                        *logfile = (char *) malloc(sizeof(char)*strlen(tokenPtr));
                        strcpy(*logfile, tokenPtr);
                }
                if(strstr(tokenPtr,"aclpath"))//si es aclpath
                {
                        tokenPtr = strtok(NULL," ");
                        *aclpath = (char *) malloc(sizeof(char)*strlen(tokenPtr));
                        strcpy(*aclpath, tokenPtr);
                }
                if(strstr(tokenPtr,"aclfile"))//si es aclfile
                {
                        tokenPtr = strtok(NULL," ");
                        *aclfile = (char *) malloc(sizeof(char)*strlen(tokenPtr));
                        strcpy(*aclfile, tokenPtr);
                }
        }
        fclose(ficheroPtr);
        return OK;
}
```

A.1.6. coredaemon.c

```
#include "coldaemon.h"
#include <string.h>
#include <postgresql/libpq-fe.h>

void fin_hilo(thread_arg arg)
{
        close(arg.socket_descriptor);
        pthread_mutex_lock(&lock);
        --ready;
        thread_del(&(arg.lista_hilo), arg.thread_index);
        pthread_mutex_unlock(&lock);
        return;
}

int recvtimeout(int socket, char *buffer, int len, int timeout)
{
        fd_set fds;
        int n;
        struct timeval tv;
        FD_ZERO(&fds);
        FD_SET(socket, &fds);
        tv.tv_sec = timeout;
        tv.tv_usec = 0;
        n = select(socket+1, &fds, NULL, NULL, &tv);
        if (n == 0)
        {
                return -2; // timeout!
        }
        if (n == -1)
        {
                return -1; // error
        }

        n = recv(socket, buffer, len, 0);
        if( n > 0 )
        {
                buffer[n - 1] = '\0';
                limpiar_telnet(buffer);
                --n;
        }

        return n;
}


void * coredaemon(void * argumento)
{
        time_t current_time;
    char* c_time_string;

    /* Obtain current time as seconds elapsed since the Epoch. */
    current_time = time(NULL);

    /* Convert to local time format. */
    c_time_string = ctime(&current_time);


        thread_arg arg = *((thread_arg * ) argumento);
        char print_buffer[STR_LEN];
        char buffer[PKG_LEN];
        char resp[PKG_LEN];
        char temp[PKG_LEN];
        int len,error;
        char usuario[STR_LEN];
```

```
char clave[STR_LEN];
int log_fd;
SERVICIO serv;

free(argumento);

if( (log_fd = open(arg.log, O_CREAT | O_WRONLY | O_APPEND, 0666)) < 0 )
{
        syslog(LOG_ERR,"No se puede abrir el fichero %s (%d)\n",arg.log, log_fd);
        exit(LOG_ERROR);
}

writelog(log_fd,"Iniciando Autenticación\n");
sprintf(resp,"Usuario: ");

if(send(arg.socket_descriptor, resp, strlen(resp),0) == -1)
{
        writelog(log_fd,"No se puede enviar\n");
        fin_hilo(arg);
        return;
}
if( ( len = recvtimeout(arg.socket_descriptor, buffer, 16384,arg.timeout) ) < 0 )
{
        writelog(log_fd, "No se puede recibir: ");
        if(len == -2)
        {
                writelog(log_fd, "Timeout\n");
        }else{
                writelog(log_fd, "Error de I/O\n");
        }
        fin_hilo(arg);
        return;
}
if(len == 0)
{
        writelog(log_fd, "Conexión abortada\n");
        fin_hilo(arg);
        return;
}
strcpy(usuario,buffer);

sprintf(resp,"Clave: ");
if(send(arg.socket_descriptor, resp, strlen(resp),0) == -1)
{
        writelog(log_fd,"No se puede enviar\n");
        fin_hilo(arg);
        return;
}
if( ( len = recvtimeout(arg.socket_descriptor, buffer, 16384,arg.timeout) ) < 0 )
{
        writelog(log_fd, "No se puede recibir\n");
        if(len == -2)
        {
                writelog(log_fd, "Timeout\n");
        }else{
                writelog(log_fd, "Error de I/O\n");
        }
        fin_hilo(arg);
        return;
}
if(len == 0)
{
        writelog(log_fd, "Conexión abortada\n");
        fin_hilo(arg);
        return;
}
```

```
strcpy(clave,buffer);

// AUTENTICACIÓN
if(authentication (arg.acl, usuario, hash(clave)) != 0)
{
        writelog(log_fd,"Fallo de autenticación\n");
        sprintf(resp,"Credenciales inválidas\n");
        if(send(arg.socket_descriptor, resp, strlen(resp),0) == -1)
        {
                writelog(log_fd,"No se puede enviar\n");
        }
        sprintf(temp,"[Hilo %d] Cerrando la conexión\n",arg.thread_index);
        writelog(log_fd,temp);
        fin_hilo(arg);
        return;
}
sprintf(temp,"Se autenticó exitosamente al usuario %s\n",usuario);
writelog(log_fd, temp);

sprintf(resp,"Bienvenido al sistema de Cobros en Linea\nEscriba help para ayuda\n$ ");
if(send(arg.socket_descriptor, resp, strlen(resp),0) == -1)
{
        writelog(log_fd,"No se puede enviar\n");
        fin_hilo(arg);
        return;
}
if( ( len = recvtimeout(arg.socket_descriptor, buffer, PKG_LEN,arg.timeout) ) < 0 )
{
        writelog(log_fd, "No se puede recibir\n");
        if(len == -2)
        {
                writelog(log_fd, "Timeout\n");
        }else{
                writelog(log_fd, "Error de I/O\n");
        }
        fin_hilo(arg);
        return;
}

while( len > 0)
{
        // Motor de Inferencia
        if(strcmp(buffer,"close") == 0)
        {
                sprintf(temp,"[%s::%s::close::Conexion terminada]\n",c_time_string,usuario);
                writelog(log_fd, temp);
                fin_hilo(arg);
                return;
        }
        if(strcmp(buffer,"help") == 0)
        {
                //sprintf(temp,"IMPRIMIR LA AYUDA EN PANTALLA\n");
                sprintf(temp,"[%s::%s::help::Ayuda]\n",c_time_string,usuario);
                writelog(log_fd, temp);
                if( db_module("help", serv,usuario,log_fd,resp) != 0)
                {
                        writelog(log_fd,"Intentando mostrar la ayuda\n");
                }else{
                        if(send(arg.socket_descriptor, resp,strlen(resp),0) == -1)
                        {
                                writelog(log_fd,"No se puede enviar\n");
                                fin_hilo(arg);
                                return;
                        }
                }
        }
```

```
if(strcmp(buffer,"lastrx") == 0)
{
        sprintf(temp,"Imprimir las 3 últimas transacciones\n");
        writelog(log_fd, temp);
        if( db_module("lastrx", serv,usuario,log_fd,resp) != 0)
        {
                writelog(log_fd,"EXPLOTO BD Intentando mostrar las útlimas 3 transa
        }else{
                if(send(arg.socket_descriptor, resp,strlen(resp),0) == -1)
                {
                        writelog(log_fd,"No se puede enviar\n");
                        fin_hilo(arg);
                        return;
                }
        }
}
if(strncmp(buffer,"col ",4) == 0)
{
        buffer[len-1] = '\n';
        if((error = col_parser(&serv,buffer+4,log_fd)) != 0)
        {

                sprintf(temp,"ERROR: ");
                writelog(log_fd, temp);
                switch (error)
                {
                case 30:
                        sprintf(temp,"INVALID_COD_SERV\n");
                        writelog(log_fd, temp);
                        sprintf(resp,"ERROR: Codigo de servicio invalido. Verifique
                        break;
                case 31:
                        sprintf(temp,"INVALID_YEAR\n");
                        writelog(log_fd, temp);
                        sprintf(resp,"ERROR: Año invalido. Verifique y reintente \n$
                        break;
                case 32:
                        sprintf(temp,"INVALID_DAY\n");
                        writelog(log_fd, temp);
                        sprintf(resp,"ERROR: Dia invalido. Verifique y reintente \n$
                        break;
                case 33:
                        sprintf(temp,"INVALID_MONTH\n");
                        writelog(log_fd, temp);
                        sprintf(resp,"ERROR: Mes invalido. Verifique y reintente \n$
                        break;
                case 34:
                        sprintf(temp,"INVALID_HOUR\n");
                        writelog(log_fd, temp);
                        sprintf(resp,"ERROR: Hora invalida. Verifique y reintente \n
                        break;
                case 35:
                        sprintf(temp,"INVALID_MIN\n");
                        writelog(log_fd, temp);
                        sprintf(resp,"ERROR: Minutos invalidos. Verifique y reinten
                        break;
                case 36:
                        sprintf(temp,"INVALID_SEC\n");
                        writelog(log_fd, temp);
                        sprintf(resp,"ERROR: Segundos invalidos. Verifique y reinte
                        break;
                }

                if(send(arg.socket_descriptor, resp,strlen(resp),0) == -1)
                {
                        writelog(log_fd,"No se puede enviar\n");
```

```
                                fin_hilo(arg);
                                return;
                        }
                }else{
                        //IMPRIMIR LOS RESULTADOS BLAH BLHA BLHA Y LOGGEAR
                        if(db_module("col",serv,usuario,log_fd,resp) != 0)
                                writelog(log_fd,"Database Error\n");
                        if(send(arg.socket_descriptor, resp,strlen(resp),0) == -1)
                        {
                                writelog(log_fd,"No se puede enviar\n");
                                fin_hilo(arg);
                                return;
                        }
                }
        }
        if(strncmp(buffer,"rev ",4) == 0)
        {
                buffer[len-1] = '\n';
                if( (error = rev_parser(&serv,buffer+4)) != 0)
                {
                        sprintf(temp,"ERROR: ");
                        writelog(log_fd, temp);
                        switch (error)
                        {
                        case 30:
                                sprintf(temp,"INVALID_COD_SERV\n");
                                writelog(log_fd, temp);
                                sprintf(resp,"ERROR: Codigo de servicio invalido. Verifique y
                                break;
                        case 31:
                                sprintf(temp,"INVALID_YEAR\n");
                                writelog(log_fd, temp);
                                sprintf(resp,"ERROR: Año invalido. Verifique y reintente\n$ ")
                                break;
                        case 32:
                                sprintf(temp,"INVALID_DAY\n");
                                writelog(log_fd, temp);
                                sprintf(resp,"ERROR: Dia invalido. Verifique y reintente\n$ ")
                                break;
                        case 33:
                                sprintf(temp,"INVALID_MONTH\n");
                                writelog(log_fd, temp);
                                sprintf(resp,"ERROR: Mes invalido. Verifique y reintente\n$ ")
                                break;
                        case 34:
                                sprintf(temp,"INVALID_HOUR\n");
                                writelog(log_fd, temp);
                                sprintf(resp,"ERROR: Hora invalida. Verifique y reintente\n$ "
                                break;
                        case 35:
                                sprintf(temp,"INVALID_MIN\n");
                                writelog(log_fd, temp);
                                sprintf(resp,"ERROR: Minutos invalidos. Verifique y reintente\
                                break;
                        case 36:
                                sprintf(temp,"INVALID_SEC\n");
                                writelog(log_fd, temp);
                                sprintf(resp,"ERROR: Segundos invalidos. Verifique y reintente
                                break;
                        }

                        if(send(arg.socket_descriptor, resp,strlen(resp),0) == -1)
                        {
                                writelog(log_fd,"No se puede enviar\n");
                                fin_hilo(arg);
                                return;
```

```
                                }
                        }else{
                                //IMPRIMIR LOS RESULTADOS BLAH BLHA BLHA Y LOGGEAR
                                if(db_module("rev",serv,usuario,log_fd,resp) != 0)
                                        writelog(log_fd,"EXPLOTO BD\n");
                                if(send(arg.socket_descriptor, resp,strlen(resp),0) == -1)
                                {
                                        writelog(log_fd,"No se puede enviar\n");
                                        fin_hilo(arg);
                                        return;
                                }
                        }
                }

                sprintf(resp,"$ ");
                if(send(arg.socket_descriptor, resp,strlen(resp),0) == -1)
                {
                        writelog(log_fd,"No se puede enviar\n");
                        fin_hilo(arg);
                        return;
                }

                if( ( len = recvtimeout(arg.socket_descriptor, buffer, PKG_LEN,arg.timeout) ) < 0 )
                {
                        writelog(log_fd, "no se puede recibir\n");
                        if(len == -2)
                        {
                                writelog(log_fd, "Timeout\n");
                        }else{
                                writelog(log_fd, "Error de I/O\n");
                        }
                        fin_hilo(arg);
                        return;
                }

        }

        sprintf(print_buffer,"Hilo %d cerrando la conexión\n", arg.thread_index);
        writelog(log_fd,print_buffer);
        close(log_fd);

        fin_hilo(arg);
        return;
}

void limpiar_telnet(char * cadena)
{
        int i = 0;

        while(cadena[i] != '\0')
        {
                if(cadena[i] == '\015')
                {
                        cadena[i] = '\0';
                        break;
                }
                i++;
        }

        return;
}
```

A.1.7. db_coldaemon.sql

```
DROP TABLE IF EXISTS pagadas, pendientes, transacciones;

CREATE TABLE pagadas(cod_serv INTEGER, tipo INTEGER, compr BIGINT, monto BIGINT,
    dig_verif INTEGER, prefijo INTEGER, numero INTEGER, medidor BIGINT, abonado
    BIGINT, transaccion INTEGER, fecha_hora BIGINT,  usuario TEXT, vencimiento
    BIGINT);

CREATE TABLE pendientes(cod_serv INTEGER,tipo INTEGER,compr BIGINT,monto BIGINT,
    dig_verif INTEGER,prefijo INTEGER,numero INTEGER ,medidor BIGINT, abonado
    BIGINT, id INTEGER,vencimiento BIGINT);

CREATE TABLE transacciones(operacion TEXT, cod_serv INTEGER, fecha_hora TEXT,
    usuario TEXT, mensaje TEXT,tipo INTEGER, compr BIGINT, monto BIGINT, dig_verif
    INTEGER, prefijo INTEGER, numero INTEGER, medidor BIGINT, abonado BIGINT,
    transaccion INTEGER);


INSERT INTO pendientes VALUES
    (001,001,12345678912,000000100000,7,0,0,0,0,1,20131212235959);

INSERT INTO pendientes VALUES
    (002,001,0,000000100000,0,0644,9876543,0,0,2,20131212235959);

INSERT INTO pendientes VALUES
    (003,0,0,000000100000,0,0,0,987654321012345,0,3,20131212235959);

INSERT INTO pendientes VALUES
    (004,0,0,000000100000,0,0644,987654,0,0,4,20131212235959);

INSERT INTO pendientes VALUES
    (005,0,0,000000100000,0,0,0,0,987654321,5,20131212235959);
```

A.1.8. db_module.c

```
#include "coldaemon.h"

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    return;
}

int existe_factura(const char * cod_serv, const char * compr,const char * medidor,
    const char * prefijo, const char * numero, const char * abonado,const char *
    monto, const char * vencimiento, PGconn * conn, PGresult * res,int log_fd,char
    * resp){

        const char *parametros[8];
        parametros[0] = compr;
        parametros[1] = medidor;
        parametros[2] = abonado;
        parametros[3] = prefijo;
        parametros[4] = numero;
        parametros[5] = monto;
        parametros[6] = vencimiento;
        parametros[7] = cod_serv;
        int entero = 0;
        char temp[512];
        res = PQexecParams(conn,"SELECT * FROM pendientes WHERE (compr=$1 OR
            medidor=$2 OR abonado=$3 OR (numero=$4 AND prefijo=$5)) AND (monto=$6
            AND vencimiento>=$7 AND cod_serv=$8);",
                                            8,
                                            NULL,
                                            parametros,
                                            NULL,
                                            NULL,
                                            0);

        if (PQntuples(res) == 0)
        {
                sprintf(resp,"Factura invalida\n");
                sprintf(temp,"Factura inválida\n");
                writelog(log_fd,temp);
                entero = 1;
        }

        PQclear(res);
        return entero;
}

int existe_trx(const char * transaccion, PGconn * conn, PGresult * res, int log_fd,
     char * resp){
        const char *parametros[1];
        parametros[0] = transaccion;
        int entero = 0;
        char temp[512];
        res = PQexecParams(conn, "SELECT * FROM pagadas WHERE transaccion=$1",
                                            1,
                                            NULL,
                                            parametros,
                                            NULL,
                                            NULL,
                                            0);
        if (PQntuples(res) == 0)
        {
                sprintf(resp,"No existe transaccion a reversar\n");
                sprintf(temp,"No existe transaccion a reversar\n");
```

```
                writelog(log_fd,temp);
                entero = 1;
        }
        PQclear(res);
        return entero;
}

int db_module(char * operacion, SERVICIO serv, char * usuario, int log_fd, char *
    resp)
{
        time_t current_time;
        char* c_time_string;

        //Se obtiene el tiempo en segundos
        current_time = time(NULL);

        //Se convierte al formato de hora local
        c_time_string = ctime(&current_time);

        char temp[512] = {0};
        const char      *conninfo;
        PGconn          *conn;
        PGresult        *res;

        const char *paramValues[15];
        const char *paramValues2[14]; //utilizado para comandos de eliminacion
        const char *paramRev[14];
        int         paramLengths[14];
        int         paramFormats[14];
        int             t,f,tuples;
        char aux_monto[512];
        char aux_nummed[512];
        char aux_numtran[512];
        char aux_venc[512];
        char aux_verificador[2];

        conninfo = "dbname = coldaemon";

        paramValues[0] = serv.codser;
        paramValues[1] = serv.tipofact;
        paramValues[2] = serv.comprobante;
        sprintf(aux_monto,"%lu",serv.monto);
        paramValues[3] = aux_monto;
        sprintf(temp,"Monto = %s\n",paramValues[3]);
        sprintf(aux_verificador,"%d",serv.verificador);
        paramValues[4] = aux_verificador;
        paramValues[5] = serv.prefijo;
        paramValues[6] = serv.numero;
        paramValues[7] = serv.nummed;
        paramValues[8] = serv.abonado;
        sprintf(aux_numtran,"%d",serv.numtran);
        paramValues[9] = aux_numtran;
        paramValues[10] = serv.fechahora;
        paramValues[11] = usuario;
        paramValues[12] = "20131212235959";
        paramValues[13] = operacion;
        paramValues[14] = serv.mensaje;
        /*strcpy(aux_venc,serv.vencimiento);
        strcat(aux_venc,"235959");
        paramValues[12] = aux_venc;*/
        paramRev[0]=operacion;
        paramRev[1]=serv.codser;
        paramRev[2]=serv.fechahora;
        paramRev[3]=usuario;
        paramRev[4]=serv.mensaje;
        paramRev[5]=serv.tipofact;
```

```
    paramRev[6]=serv.comprobante;
    paramRev[7]=aux_monto;
    paramRev[8]=aux_verificador;
    paramRev[9]=serv.prefijo;
    paramRev[10]=serv.numero;
    paramRev[11]=serv.nummed;
    paramRev[12]=serv.abonado;
    paramRev[13]=aux_numtran;

    // Se realiza la conexión a la base de datos
conn = PQconnectdb(conninfo);

    // Se chequea si la conexión backend ha sido establecida
if (PQstatus(conn) != CONNECTION_OK)
{
    sprintf(temp, "Connection to database failed: %s",
    PQerrorMessage(conn));
            writelog(log_fd,temp);
    exit_nicely(conn);
}
    //INICIO DE OPERACIONES EN LA BASE DE DATOS
    if(strcmp(operacion, "col") == 0){
            sprintf(temp,"[%s::%s::%s::Peticion de cobro]\n",c_time_string,
                usuario,operacion);
            writelog(log_fd,temp);
            /*
            *       se envia serv.fecha hora a existe_factura y no serv.
                vencimiento
            *       se cambia paramValues[10] por paramValues[12]
            */
            if(existe_factura(paramValues[0],paramValues[2],paramValues[7],
                paramValues[5],paramValues[6],paramValues[8],paramValues[3],
                paramValues[10],conn,res,log_fd,resp) == 0){
                    //insertar registro en la tabla de pagadas
                    res = PQexecParams(conn,
                   "INSERT INTO pagadas VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,
                       $11,$12,$13);",
                   13,       // 13 parametros
                   NULL,    // let the backend deduce param type
                   paramValues,
                   NULL,    // don't need param lengths since text
                   NULL,    // default to all text params
                   0);      // ask for non binary results
        if (PQresultStatus(res) != PGRES_COMMAND_OK)
            {
                    //SE GENERA EL MENSAJE DE RETORNO
                    sprintf(resp,"%s%s%s001Fallo el cobro\n",
                                    serv.codser,aux_numtran,serv.fechahora)
                        ;
                    sprintf(temp,"insert command failed: %s", PQerrorMessage(
                        conn));
                    writelog(log_fd,temp);
                    PQclear(res);
                    exit_nicely(conn);
            }

                    PQclear(res);
                    //eliminar registro de la tabla pendientes
                    //asignar valores a paramValues (si! hace falta)
                    if (strcmp(serv.codser,"001") == 0)
                    {
                            //eliminar aguas
                            paramValues2[0] = serv.comprobante;
                            res = PQexecParams(conn,
                                            "DELETE FROM pendientes WHERE compr
                                                =$1;",
```

```
                                1,
                                NULL,
                                paramValues2,
                                NULL,
                                NULL,
                                0);
        }else if(strcmp(serv.codser,"002") == 0){
                //eliminar fijo
                paramValues2[0] = serv.prefijo;
                paramValues2[1] = serv.numero;

                res = PQexecParams(conn,
                                "DELETE FROM pendientes WHERE
                                    numero=$2 AND prefijo=$1;",
                                2,
                                NULL,
                                paramValues2,
                                NULL,
                                NULL,
                                0);
        }else if(strcmp(serv.codser,"003") == 0){
                //eliminar electricidad
                paramValues2[0] = serv.nummed;

                res = PQexecParams(conn,
                                "DELETE FROM pendientes WHERE
                                    medidor=$1;",
                                1,
                                NULL,
                                paramValues2,
                                NULL,
                                NULL,
                                0);

        }else if(strcmp(serv.codser,"004") == 0){
                //eliminar movil
                paramValues2[0] = serv.prefijo;
                paramValues2[1] = serv.numero;
                res = PQexecParams(conn,
                                "DELETE FROM pendientes WHERE (
                                    prefijo=$1 AND numero=$2);",
                                2,
                                NULL,
                                paramValues2,
                                NULL,
                                NULL,
                                0);
        }else if(strcmp(serv.codser,"005") == 0){
                //eliminar cable
                paramValues2[0] = serv.abonado;
                res = PQexecParams(conn,
                                "DELETE FROM pendientes WHERE
                                    abonado=$1;",
                                1,
                                NULL,
                                paramValues2,
                                NULL,
                                NULL,
                                0);
        }
        if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
        //SE GENERA EL MENSAJE DE RETORNO
        sprintf(resp,"%s%s%s001Fallo el cobro\n",serv.codser,
            aux_numtran,serv.fechahora);
        sprintf(temp,"delete command failed: %s", PQerrorMessage(
```

```
                    conn));
                        writelog(log_fd,temp);
                PQclear(res);
                exit_nicely(conn);
        }

                PQclear(res);
                //agregar a transacciones
        res = PQexecParams(conn,
        "INSERT INTO transacciones VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,
            $11,$12,$13,$14);",
        14,       // 13 parametros
    NULL,    // let the backend deduce param type
    paramRev,
    NULL,    // don't need param lengths since text
    NULL,    // default to all text params
    0);      // ask for non binary results

                if (PQresultStatus(res) != PGRES_COMMAND_OK)
        {
                //SE GENERA EL MENSAJE DE RETORNO
                sprintf(resp,"%s%s%s001Fallo el cobro\n",serv.codser,
                    aux_numtran,serv.fechahora);
                sprintf(temp,"fallo de envio a transaccion: %s",
                    PQerrorMessage(conn));
                        writelog(log_fd,temp);
                PQclear(res);
                exit_nicely(conn);
        }
        sprintf(resp,"%s%s%s000Cobro Exitoso\n",serv.codser,aux_numtran,
            serv.fechahora);
        writelog(log_fd,resp);
        PQclear(res);
        }
}else if(strcmp(operacion, "rev") == 0){
        sprintf(temp,"[%s::%s::%s::Peticion de reversa]\n",
                                c_time_string,usuario,operacion);
        writelog(log_fd,temp);
        if(existe_trx(paramValues[9],conn,res,log_fd,resp) == 0){
                sprintf(temp,"Existe la transaccion\nMoviendo a pendientes\
                    n");
                writelog(log_fd,temp);

                //mover a pendientes
                res = PQexec(conn,"INSERT INTO pendientes               (
                    cod_serv,tipo,compr,monto,vencimiento,dig_verif,prefijo
                    ,numero,medidor,abonado) SELECT cod_serv,tipo,compr,
                    monto,vencimiento,dig_verif,prefijo,numero,medidor,
                    abonado  FROM pagadas;");

                if (PQresultStatus(res) != PGRES_COMMAND_OK)
                {
                sprintf(temp,"moving command failed: %s", PQerrorMessage(
                    conn));
                writelog(log_fd,temp);
                PQclear(res);
                exit_nicely(conn);
                }

                //borrar de pagadas
                sprintf(temp,"borrando de pagadas\n");
                writelog(log_fd,temp);
                paramValues2[0] = aux_numtran;
                res = PQexecParams(conn,
               "DELETE FROM pagadas WHERE transaccion=$1;",
               1,       // 1 parametro
```

```
                NULL,    // let the backend deduce param type
                paramValues2,
                NULL,    // don't need param lengths since text
                NULL,    // default to all text params
                0);      // ask for non binary results

                 if (PQresultStatus(res) != PGRES_COMMAND_OK)
                 {
                         sprintf(temp,"delete rev command failed: %s",
                             PQerrorMessage(conn));
                         writelog(log_fd,temp);
                         PQclear(res);
                         exit_nicely(conn);
                 }

        //agregar a transacciones
        res = PQexecParams(conn,
        "INSERT INTO transacciones VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,
            $11,$12,$13,$14);",
        14,       // 13 parametros
    NULL,    // let the backend deduce param type
    paramRev,
    NULL,    // don't need param lengths since text
    NULL,    // default to all text params
    0);      // ask for non binary results

        if (PQresultStatus(res) != PGRES_COMMAND_OK)
        {
                sprintf(temp,"fallo de envio a transaccion: %s",
                    PQerrorMessage(conn));
                        writelog(log_fd,temp);
                PQclear(res);
                exit_nicely(conn);
        }

        PQclear(res);
        }
}else if(strcmp(operacion, "lastrx") == 0){
        sprintf(temp,"[%s::%s::%s::Peticion de listado]\n",c_time_string,
            usuario,operacion);
        writelog(log_fd,temp);
        paramValues2[0] = usuario;
        int retorno = 0;
        res = PQexecParams(conn, "SELECT * FROM transacciones WHERE usuario
            =$1;",
                                        1,
                                        NULL,
                                        paramValues2,
                                        NULL,
                                        NULL,
                                        0);
        if (PQntuples(res) == 0)
        {
                sprintf(temp,"No hay transacciones recientes\n");
                writelog(log_fd,temp);
                sprintf(resp,"No hay transacciones recientes\n");
                retorno = 1;
        }else{
                resp[0] = '\0';
                tuples = PQntuples(res) - 3;
                if (PQntuples(res) < 3)
                        tuples = 0;
                for(t = tuples; t < PQntuples(res); t++)
                {
                        for (f = 0; f < PQnfields(res); f++)
                        {
```

```
                                    //concatenar resultados de PQgetvalue
                                    sprintf(temp,"%s ", PQgetvalue(res, t, f));
                                        strcat(resp,temp);
                                        writelog(log_fd,temp);
                                }
                                sprintf(temp,"\n");
                                strcat(resp,temp);
                                writelog(log_fd,temp);
                        }
                }
                PQclear(res);
                //agregar a transacciones
                /*
                *       Se podria guardar la fecha/hora del sistema
                */
                paramRev[2] = c_time_string;
                res = PQexecParams(conn,"INSERT INTO transacciones VALUES ($1,$2,$3
                    ,$4,'0',0,0,0,0,0,0,0,0,0);",
                4,       // 14 parametros
            NULL,    // let the backend deduce param type
            paramRev,
            NULL,    // don't need param lengths since text
            NULL,    // default to all text params
            0);      // ask for non binary results

                        if (PQresultStatus(res) != PGRES_COMMAND_OK)
                {

                        sprintf(temp,"fallo de envio a transaccion: %s",
                            PQerrorMessage(conn));
                                writelog(log_fd,temp);
                        PQclear(res);
                        exit_nicely(conn);
                }
                PQclear(res);

        }else if(strcmp(operacion, "help") == 0){
                sprintf(resp,"Comandos:\n- col <parametros> Realiza un cobro con la
                     transaccion indicada por parametros.\n- rev <parametros>
                    Realiza una reversa de la transaccion indicada por paramtros.\n
                    - lastrx Consulta las ultimas transacciones hechas por el
                    usuario.\n- close Cierra la conexion con el servidor\n");

        }
        //Se cierra la conexión a la base de datos
    PQfinish(conn);
        return OK;
}
```

A.1.9. thread_manager.c

```
#include "coldaemon.h"

void * thread_manager(void * argumento)
{
        thread_arg arg = *((thread_arg *) argumento);

        char * log = arg.log;
        int puerto = arg.puerto;
        int threads = arg.threads;
        int log_fd;
        char printBuffer[STR_LEN];
        //SOCKET
        struct sockaddr_in pin;
        struct sockaddr_in sin;
        int temp_sock_descriptor;
        int address_size;
        char buf[PKG_LEN];
        int create_thread_value;
        int ret;
        thread_arg * arg_for_thread = NULL;

        if( (log_fd = open(log, O_CREAT | O_WRONLY | O_APPEND, 0666)) < 0 )
        {
                syslog(LOG_ERR,"No se puede abrir el fichero %s (%d)\n",log, log_fd
                    );
                exit(LOG_ERROR);
        }
        strcpy(printBuffer,"[cold] Cobros On-Line Daemon\n");
        writelog(log_fd,printBuffer);

        bzero(&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(puerto);

        if( (ret = bind(arg.socket_descriptor, (struct sockaddr *)&(sin), sizeof(
            sin))) == -1 )
        {
                syslog(LOG_ERR,"No se puede usar el puerto %d\n",puerto);
                exit(BINDING_ERROR);
        }

        if( listen(arg.socket_descriptor, threads) == -1 )
        {
                syslog(LOG_ERR,"No se puede escuchar en el puerto %d\n",puerto);
                exit(LISTENNING_ERROR);
        }

        sprintf(printBuffer,"Esperando conexiones en el puerto %d\n",puerto);
        writelog(log_fd,printBuffer);

        while(1)
        {
                temp_sock_descriptor = accept(arg.socket_descriptor, (struct
                    sockaddr *)& pin, &address_size);
                if( temp_sock_descriptor == -1 )
                {
                        syslog(LOG_ERR,"No se puede aceptar la conexión\n");
                        exit(ACCEPT_CONNECTION_ERROR);
                }
                //strcpy(printBuffer,"Conexión Entrante\n");
                //writelog(log_fd, printBuffer);

                // Asignar argumentos y ejecutar hilos
```

```
                if( ready <  arg.threads )
                {
                        ready++;
                        //syslog(LOG_DEBUG,"Ejecutando el hilo %d\n",ready);
                        arg_for_thread = (thread_arg *)malloc(sizeof(thread_arg));

                        *arg_for_thread = arg;

                        arg_for_thread->thread_index = ready;
                        arg_for_thread->socket_descriptor = temp_sock_descriptor;
                        arg_for_thread->socket = pin;

                        thread_add(&(arg_for_thread->lista_hilo), ready);
                        create_thread_value = pthread_create(thread_get(
                            arg_for_thread->lista_hilo,ready),NULL, coredaemon, (
                            void *) arg_for_thread);

                        //syslog(LOG_DEBUG,"create_thread_value = %d\n",
                            create_thread_value);
                        //syslog(LOG_DEBUG,"Usando Hilo %d\n",ready);
                }else{
                        // Rechazar la conexión
                        sprintf(printBuffer,"El servidor no acepta más conexiones
                            en este momento\nPor favor aguarde un momento y
                            reintente conectarse nuevamente\n");
                        if(send(temp_sock_descriptor, printBuffer, strlen(
                            printBuffer), 0) == -1)
                        {
                                syslog(LOG_ERR,"Error al informar que ya no se
                                    aceptan más conexiónes\n");
                        }
                        strcpy(printBuffer, "No hay hilos disponibles\n");
                        writelog(log_fd, printBuffer);
                        close(temp_sock_descriptor);
                }
        }

        return;
}
```

A.1.10. transa_parser.c

```
#include "coldaemon.h"

char col_parser (SERVICIO *servicio, char * patron,int log_fd)
{
        int contcar = 0; //contador de caracteres
        char string[58] = {0}; //rubro
        int tipo = 0;
        char *auxiliar; //string auxiliar
        int entero; //entero auxiliar
        int biciesto = 0; //booleano de año biciesto
        //Asignar el tamaño de codigo de servicio
        auxiliar = (char *) calloc(sizeof(char),3);
        //char patron[] =
            "00112345620131212163215001123456789120000001000002013121 27\n";
        int index = 0;
        char tmp[512];

        //establecer el rubro de transaccion
        for(contcar=0 ; contcar <=2; contcar++)
        {
                string[contcar] = patron[index++]; //lectura del caracter
                strcat(auxiliar, string+contcar); //concatenacion en el auxiliar
                if((string[contcar] != '0') && contcar <= 1)
                        return INVALID_COD_SERV;//CODIFICAR ERROR Y SALIR DE LA
                            FUNCION. ERROR EN LOS DOS PRIMEROS DIGITOS DEL SERVICIO
                            .
        }

        //****CONTROLAR QUE TIPO SEA UN NUMERO VALIDO DE SERVICIO***** SI NO ES
            VALIDO SALIR!!

        tipo = atoi(string+2);  //definir el tipo de servicio
        //printf("Tipo:%d\n",tipo);

        servicio->codser = (char *)calloc(sizeof(char),3); //Asignar tamaño al
            codigo de servicio
        strcpy(servicio->codser, auxiliar); // Asignar el valor del auxiliar al
            codigo de servicio
        //printf("Auxiliar: %s\n", auxiliar);
        //printf("codser: %s\n", servicio->codser);
        free(auxiliar); //liberar el auxiliar

        //establecer el tamaño del auxiliar al de transaccion
        auxiliar = (char *)calloc(sizeof(char),6);

        //establecer el numero de transaccion
        for(contcar=0 ; contcar <=5; contcar++)
        {
                string[3+contcar] = patron[index++]; //lectura del caracter
                strcat(auxiliar,string+3+contcar); //concatenación en el auxiliar
        }

        servicio->numtran = atoi(auxiliar); //se convierte el string de transacción
             a int y se asigna al numero de transaccion del servicio
        //printf("Auxiliar: %s\n", auxiliar);
        free(auxiliar); //se libera el auxiliar
        //printf("numtran: %d\n", servicio->numtran);

        //Asignar el tamaño del auxiliar a fechahora
        auxiliar = (char *)calloc(sizeof(char),14);

        //Asignar el tamano del fecharhora del Servicio
        servicio->fechahora = (char *)calloc(sizeof(char),14);
```

```
//establecer la fecha de la transaccion
        for(contcar=0 ; contcar<=13; contcar++)
        {
                string[9+contcar] = patron[index++];//lectura del caracter
                strcat(auxiliar, string+9+contcar);//concatenacion en el
                    auxiliar
                if(contcar == 3) //ya se leyo el año completo
                {
                        entero = atoi(auxiliar); //convertir el char año a
                            entero
                        //printf("%d\n", entero);
                        if(entero < 2013)
                                return INVALID_YEAR;
                        if((entero%4 == 0) && (entero%100 != 0) || (entero
                            %400 == 0)) //verificacion de año biciesto
                                biciesto = 1; //año biciesto
                }
                if(contcar == 7) //ya se leyo el mes y el dia completo
                {
                        entero = (string[13]-'0')*10 + (string[14]-'0'); //
                            asignar el mes a entero
                        if(entero > 12 || entero < 1)
                        return INVALID_MONTH;

                if((entero == 1) || (entero == 3) || (entero == 5) || (
                    entero == 7) || (entero == 8) || (entero == 10) || (
                    entero ==12)) // si el mes tiene 31 dias
                {
                        entero = (string[15]-'0')*10 + (string[16]-'0');
                        if((entero < 1) || (entero > 31))
                                return INVALID_DAY;
                }
                else if((entero == 4) || (entero == 6) || (entero == 9) ||
                    (entero == 11)) //si el mes tiene 30 dias
                        {
                                entero = (string[15]-'0')*10 + (string[16]-
                                    '0');
                                if((entero < 1) || (entero > 30))
                                        return INVALID_DAY;
                        }
                else if(biciesto == 0) //si es febrero y el año NO es
                    biciesto
                        {
                                entero = (string[15]-'0')*10 + (string[16]-
                                    '0');
                                if((entero < 1) || (entero > 28))
                                        return INVALID_DAY;
                        }
                else //si el año es biciesto y es febrero
                {
                        entero = (string[15]-'0')*10 + (string[16]-'0');
                        if((entero < 1) || (entero > 29))
                                return INVALID_DAY;
                }
        }
}

entero = (string[17]-'0')*10 + (string[18]-'0');
if((entero > 24) || (entero < 1)) // verificar si la hora es valida
        return INVALID_HOUR;
entero = (string[19]-'0')*10 + (string[20]-'0');
if((entero > 59) || (entero < 0)) //verificar si los minutos son validos
        return INVALID_MIN;
entero = (string[21]-'0')*10 + (string[22]-'0');
if((entero > 59) || (entero < 0)) //verificar si los segundos son validos
        return INVALID_SEC;
```

```
//asignar la fechahora del servicio
strcpy(servicio->fechahora, auxiliar);
//liberar auxiliar
//printf("Auxiliar: %s\n", auxiliar);
free(auxiliar);
//printf("fechahora: %s\n", servicio->fechahora);

switch (tipo)
{

                case 1:
                        //Asignar recursos al tipo de factura
                        servicio->tipofact = (char *)calloc(sizeof(char),3)
                            ;

                        //Asignar recursos al auxiliar
                        auxiliar = (char *)calloc(sizeof(char),3);

                        //printf("Agua\n");
                        for(contcar = 0; contcar <= 2; contcar++)
                        {
                                string[23+contcar] = patron[index++];//
                                    lectura del caracter
                                strcat(auxiliar, string+23+contcar);//
                                    concatenacion en el auxiliar
                        }

                        //asignar el tipo de factura
                        strcpy(servicio->tipofact, auxiliar);
                        //liberar el auxiliar
                        //printf("Auxiliar: %s\n", auxiliar);
                        free(auxiliar);
                        //printf("tipofact: %s\n", servicio->tipofact);


                        //Asignar recursos al comprobante
                        servicio->comprobante = (char *)calloc(sizeof(char)
                            ,11);
                        //Asignar recursos al auxiliar
                        auxiliar = (char *)calloc(sizeof(char),11);

                        for(contcar = 0; contcar <= 10; contcar++)
                        {
                                string[26+contcar] = patron[index++]; //
                                    lectura del caracter
                                strcat(auxiliar, string+26+contcar); //
                                    concatenacion en el auxiliar
                        }

                        //asignar el comprobante
                        strcpy(servicio->comprobante, auxiliar);
                        //liberar auxiliar
                        //printf("Auxiliar: %s\n", auxiliar);
                        free(auxiliar);
                        //printf("Comprobante: %s\n", servicio->comprobante
                            );


                        //Asignar recursos al auxiliar
                        auxiliar = (char *)calloc(sizeof(char),12);

                        for(contcar = 0; contcar <= 11; contcar++)
                        {
                                string[37+contcar] = patron[index++];
                                strcat(auxiliar, string+37+contcar);
```

```
}

//Asignar el monto
servicio->monto = atoi(auxiliar);
//Liberar el auxiliar
//printf("Auxiliar: %s\n", auxiliar);
free(auxiliar);
//printf("Monto: %d\n", servicio->monto);

//Asignar recursos a Vencimiento
servicio->vencimiento = (char *)calloc(sizeof(char)
    ,8);
//Asignar recursos a Auxiliar
auxiliar = (char *)calloc(sizeof(char),8);
//Establecer que NO es año biciesto

for(contcar = 0; contcar <= 7; contcar++)
{
        string[49+contcar] = patron[index++];
        strcat(auxiliar, string+49+contcar);
        //printf("contcar=%d\naux=%s\n", contcar,
            auxiliar);
        if(contcar == 3) //ya se leyo el año
            completo
        {
                entero = atoi(auxiliar); //
                    convertir el char año a entero
                if(entero < 2013)
                        return INVALID_YEAR;
                if((entero%4 == 0) && (entero%100
                    != 0) || (entero%400 == 0)) //
                    verificacion de año biciesto
                        biciesto = 1; //año
                            biciesto
        }
        if(contcar == 7) //ya se leyo el mes y el
            dia completo
        {
                entero = (string[53]-'0')*10 + (
                    string[54]-'0'); //asignar el
                    mes a entero

                if(entero > 12 || entero < 1)
                        return INVALID_MONTH;

                if((entero == 1) || (entero == 3)
                    || (entero == 5) || (entero ==
                    7) || (entero == 8) || (entero
                    == 10) || (entero ==12)) // si
                    el mes tiene 31 dias
                {
                        entero = (string[55]-'0')
                            *10 + (string[56]-'0');
                             //asignar el mes a
                            entero

                        if((entero < 1) || (entero
                            > 31))
                                return INVALID_DAY;
                }
                else if((entero == 4) || (entero ==
                     6) || (entero == 9) || (entero
                     == 11)) //si el mes tiene 30
                    dias
                {
                        entero = (string[55]-'0')
```

```
                                        *10 + (string[56]-'0');
                                         //asignar el mes a
                                        entero

                                        if((entero < 1) || (entero
                                        > 30))
                                            return INVALID_DAY;
                                }
                                else if(biciesto == 0) //si es
                                    febrero y el año NO es biciesto
                                {
                                        entero = (string[55]-'0')
                                        *10 + (string[56]-'0');
                                         //asignar el mes a
                                        entero

                                        if((entero < 1) || (entero
                                        > 28))
                                            return -15;
                                }
                                else //si el año es biciesto y es
                                    febrero
                                {
                                        entero = (string[55]-'0')
                                        *10 + (string[56]-'0');
                                         //asignar el mes a
                                        entero
                                        if((entero < 1) || (entero
                                        > 29))
                                            return INVALID_DAY;
                                }
                        }
                }
                //Asignar el Vencimiento
                strcpy(servicio->vencimiento, auxiliar);
                //Liberar el auxiliar
                //printf("Auxiliar: %s\n", auxiliar);
                free(auxiliar);
                //printf("Vencimiento: %s\n", servicio->vencimiento
                    );

                //Obtener el digito verificador
                string[57] = patron[index++];
                servicio->verificador = atoi(string+57);
                //printf("Verificador: %d\n\n", servicio->
                    verificador);

                servicio->prefijo = "0";
                servicio->numero = "0";
                servicio->nummed = 0;
                servicio->abonado = "0";
                servicio->mensaje = "0";
                break;
        case 2:
                //printf("Telefono Fijo\n");

                //Asignar recursos al prefijo
                servicio->prefijo = (char *)calloc(sizeof(char),4);

                //Asignar recursos al auxiliar
                auxiliar = (char *)calloc(sizeof(char),4);
                string[23] = patron[index++];
                strcpy(auxiliar,string+23);
                for(contcar = 1; contcar <= 3; contcar++)
                {
                        string[23+contcar] = patron[index++];
```

```
                        strcat(auxiliar, string+23+contcar);
                }

                //Asignar el prefijo
                strcpy(servicio->prefijo, auxiliar);

                //Liberar el auxiliar
                free(auxiliar);
                //printf("Prefijo: %s\n", servicio->prefijo);
                //Asignar recursos al numero
                servicio->numero = (char *)calloc(sizeof(char),7);
                //Asignar recursos al auxiliar
                auxiliar = (char *)calloc(sizeof(char),7);

                for(contcar = 0; contcar <= 6; contcar++)
                {
                        string[27+contcar] = patron[index++];
                        strcat(auxiliar, string+27+contcar);
                }

                //Asignar el numero
                strcpy(servicio->numero, auxiliar);
                //Liberar el auxiliar

                free(auxiliar);
                //printf("Numero: %s\n", servicio->numero);

                //Asignar recursos al auxiliar
                auxiliar = (char *)calloc(sizeof(char),12);

                for(contcar = 0; contcar <= 11; contcar++)
                {
                        string[34+contcar] = patron[index++];
                        strcat(auxiliar, string+34+contcar);
                }

                //Asignar el monto
                servicio->monto = atoi(auxiliar);
                //Liberar el auxiliar
                //printf("Auxiliar: %s\n", auxiliar);
                free(auxiliar);
                //printf("Monto: %d\n", servicio->monto);

                servicio->comprobante = "0";
                servicio->abonado = "0";
                servicio->nummed = "0";
                servicio->mensaje = "0";
                servicio->tipofact = "0";
                servicio->vencimiento = "0";
                servicio->verificador = 0;
                break;
        case 3:
                //printf("Suministro Electrico\n");
                //Asignar recursos al auxiliar
                servicio->nummed = (char *)calloc(sizeof(char),12);
                auxiliar = (char *)calloc(sizeof(char),12);

                for(contcar = 0; contcar <= 14; contcar++)
                {
                        string[23+contcar] = patron[index++];
                        strcat(auxiliar, string+23+contcar);
                }

                //Asignar el numero de medidor
                strcpy (servicio->nummed, auxiliar);
                //Liberar el auxiliar
```

```
                //printf("Auxiliar: %s\n", auxiliar);
                free(auxiliar);
                //printf("Numero de Medidor: %d\n", servicio->
                    nummed);

                //Asignar recursos al auxiliar
                auxiliar = (char *)calloc(sizeof(char),12);

                for(contcar = 0; contcar <= 11; contcar++)
                {
                        string[38+contcar] = patron[index++];
                        strcat(auxiliar, string+38+contcar);
                }

                //Asignar el monto
                servicio->monto = atoi(auxiliar);
                //Liberar el auxiliar
                //printf("Auxiliar: %s\n", auxiliar);
                free(auxiliar);
                //printf("Monto: %d\n", servicio->monto);
                servicio->tipofact = "0";
                servicio->comprobante = "0";
                servicio->vencimiento = "0";
                servicio->verificador = 0;
                servicio->prefijo = "0";
                servicio->numero = "0";
                servicio->abonado = "0";
                servicio->mensaje = "0";
                break;
        case 4:
                //printf("Telefono Movil\n");
                //Asignar recursos al prefijo
                servicio->prefijo = (char *)calloc(sizeof(char),4);
                //Asignar recursos al auxiliar
                auxiliar = (char *)calloc(sizeof(char),4);

                for(contcar = 0; contcar <= 3; contcar++)
                {
                        string[23+contcar] = patron[index++];
                        strcat(auxiliar, string+23+contcar);
                }

                //Asignar el prefijo
                strcpy(servicio->prefijo, auxiliar);
                //Liberar el auxiliar

                free(auxiliar);
                //printf("Prefijo: %s\n", servicio->prefijo);

                //Asignar recursos al numero
                servicio->numero= (char *)calloc(sizeof(char),6);
                //Asignar recursos al auxiliar
                auxiliar = (char *)calloc(sizeof(char),6);

                for(contcar = 0; contcar <= 5; contcar++)
                {
                        string[27+contcar] = patron[index++];
                        strcat(auxiliar, string+27+contcar);
                }

                //Asignar el numero
                strcpy(servicio->numero, auxiliar);
                //Liberar el auxiliar

                free(auxiliar);
                //printf("Numero: %s\n", servicio->numero);
```

```
                        //Asignar recursos al auxiliar
                        auxiliar = (char *)calloc(sizeof(char),12);

                        for(contcar = 0; contcar <= 11; contcar++)
                        {
                                string[33+contcar] = patron[index++];
                                strcat(auxiliar, string+33+contcar);
                        }

                        //Asignar el monto
                        servicio->monto = atoi(auxiliar);
                        //Liberar el auxiliar
                        //printf("Auxiliar: %s\n", auxiliar);
                        free(auxiliar);
                        //printf("Monto: %d\n", servicio->monto);
                        servicio->tipofact = "0";
                        servicio->comprobante = "0";
                        servicio->vencimiento = "0";
                        servicio->verificador = 0;
                        servicio->nummed = 0;
                        servicio->abonado = "0";
                        servicio->mensaje = "0";
                        break;
                case 5:
                        //printf("Cable TV\n");

                        //Asignar recursos al comprobante
                        servicio->abonado = (char *)calloc(sizeof(char),9);
                        //Asignar recursos al auxiliar
                        auxiliar = (char *)calloc(sizeof(char),9);

                        for(contcar = 0; contcar <= 8; contcar++)
                        {
                                string[23+contcar] = patron[index++]; //
                                    lectura del caracter
                                strcat(auxiliar, string+23+contcar); //
                                    concatenacion en el auxiliar
                        }

                        //asignar el comprobante
                        strcpy(servicio->abonado, auxiliar);
                        //liberar auxiliar
                        //printf("Auxiliar: %s\n", auxiliar);
                        free(auxiliar);
                        //printf("Abonado: %s\n", servicio->abonado);

                        //Asignar recursos al auxiliar
                        auxiliar = (char *)calloc(sizeof(char),12);

                        for(contcar = 0; contcar <= 11; contcar++)
                        {
                                string[32+contcar] = patron[index++];
                                strcat(auxiliar, string+32+contcar);
                        }

                        //Asignar el monto
                        servicio->monto = atoi(auxiliar);
                        //Liberar el auxiliar
                        //printf("Auxiliar: %s\n", auxiliar);
                        free(auxiliar);
                        //printf("Monto: %d\n", servicio->monto);
                        servicio->tipofact = "0";
                        servicio->comprobante = "0";
                        servicio->vencimiento = "0";
                        servicio->verificador = 0;
```

```
                                servicio->prefijo = "0";
                                servicio->numero = "0";
                                servicio->nummed = 0;
                                servicio->mensaje = "0";
                                break;
                        default:
                                //printf("Servicio Inexistente\n"); // CODIFICAR
                                    ERROR Y SALIR DE LA FUNCION. ERROR EN EL DIGITO
                                     FINAL DEL CODIGO DE TIPO
                                break;
        }
        return OK;
}

char rev_parser(SERVICIO *servicio, char * patron)
{
        int contcar = 0; //contador de caracteres
        char string[23] = {0}; //rubro
        char *auxiliar; //string auxiliar
        int entero; //entero auxiliar
        int biciesto = 0; //booleano de año biciesto
        char *tokenPtr; //puntero para los tokens
        char caracter[2]={'\0'};
        //char patron[] = "00512345620131212163215Factura incorrecta.\n";
        int index = 0;
        //Asignar el tamaño de codigo de servicio
        auxiliar = (char *) calloc(sizeof(char),3);

        //establecer el rubro de transaccion
        for(contcar=0 ; contcar <=2; contcar++)
        {
                string[contcar] = patron[index++]; //lectura del caracter
                strcat(auxiliar, string+contcar); //concatenacion en el auxiliar
                if((string[contcar] != '0') && contcar <= 1)
                        return INVALID_COD_SERV; //CODIFICAR ERROR Y SALIR DE LA
                            FUNCION. ERROR EN LOS DOS PRIMEROS DIGITOS DEL SERVICIO
                            .
        }
        //****CONTROLAR QUE TIPO SEA UN NUMERO VALIDO DE SERVICIO***** SI NO ES
            VALIDO SALIR!!

        servicio->codser = (char *)calloc(sizeof(char),3); //Asignar tamaño al
            codigo de servicio
        strcpy(servicio->codser, auxiliar); // Asignar el valor del auxiliar al
            codigo de servicio
        //printf("Auxiliar: %s\n", auxiliar);
        //printf("codser: %s\n", servicio->codser);
        free(auxiliar); //liberar el auxiliar

        //establecer el tamaño del auxiliar al de transaccion
        auxiliar = (char *)calloc(sizeof(char),6);

        //establecer el numero de transaccion
        for(contcar=0 ; contcar <=5; contcar++)
        {
                string[3+contcar] = patron[index++]; //lectura del caracter
                strcat(auxiliar,string+3+contcar); //concatenación en el auxiliar
        }

        servicio->numtran = atoi(auxiliar); //se convierte el string de transacción
             a int y se asigna al numero de transaccion del servicio
        //printf("Auxiliar: %s\n", auxiliar);
        free(auxiliar); //se libera el auxiliar
        //printf("numtran: %d\n", servicio->numtran);

        //Asignar el tamaño del auxiliar a fechahora
```

```
auxiliar = (char *)calloc(sizeof(char),14);

//Asignar el tamano del fecharhora del Servicio
servicio->fechahora = (char *)calloc(sizeof(char),14);

//establecer la fecha de la transaccion
for(contcar=0 ; contcar<=13; contcar++)
{
        string[9+contcar] = patron[index++];//lectura del caracter
        strcat(auxiliar, string+9+contcar);//concatenacion en el auxiliar
        if(contcar == 3) //ya se leyo el año completo
        {
                entero = atoi(auxiliar); //convertir el char año a entero
                if(entero < 2013)
                        return INVALID_YEAR;
                if((entero%4 == 0) && (entero%100 != 0) || (entero%400 ==
                    0)) //verificacion de año biciesto
                        biciesto = 1; //año biciesto
        }
        if(contcar == 7) //ya se leyo el mes y el dia completo
        {
                entero = (string[13]-'0')*10 + (string[14]-'0'); //asignar
                    el mes a entero

                if(entero > 12 || entero < 1)
                        return INVALID_MONTH;
                if((entero == 1) || (entero == 3) || (entero == 5) || (
                    entero == 7) || (entero == 8) || (entero == 10) || (
                    entero ==12)) // si el mes tiene 31 dias
                {
                        entero = (string[15]-'0')*10 + (string[16]-'0'); //
                            asignar el mes a entero
                        if((entero < 1) || (entero > 31))
                                return INVALID_DAY;
                }
                else if((entero == 4) || (entero == 6) || (entero == 9) ||
                    (entero == 11)) //si el mes tiene 30 dias
                        {
                                entero = (string[15]-'0')*10 + (string[16]-
                                    '0'); //asignar el mes a entero
                                if((entero < 1) || (entero > 30))
                                        return INVALID_DAY;
                        }
                else if(biciesto == 0) //si es febrero y el año NO es
                    biciesto
                        {
                                entero = (string[15]-'0')*10 + (string[16]-
                                    '0'); //asignar el mes a entero
                                if((entero < 1) || (entero > 28))
                                        return INVALID_DAY;
                        }
                else //si el año es biciesto y es febrero
                {
                        entero = (string[15]-'0')*10 + (string[16]-'0'); //
                            asignar el mes a entero
                        if((entero < 1) || (entero > 29))
                                return INVALID_DAY;
                }
        }
}

entero = (string[17]-'0')*10 + (string[18]-'0'); //asignar el mes a entero
if((entero > 24) || (entero < 1)) // verificar si la hora es valida
        return INVALID_HOUR;
entero = (string[19]-'0')*10 + (string[20]-'0'); //asignar el mes a entero
if((entero > 59) || (entero < 0)) //verificar si los minutos son validos
```

```
                return INVALID_MIN;
        entero = (string[21]-'0')*10 + (string[22]-'0'); //asignar el mes a entero
        if((entero > 59) || (entero < 0)) //verificar si los segundos son validos
                return INVALID_SEC;

        //asignar la fechahora del servicio
        strcpy(servicio->fechahora, auxiliar);
        //liberar auxiliar
        //printf("Auxiliar: %s\n", auxiliar);
        free(auxiliar);
        //printf("fechahora: %s\n", servicio->fechahora);

        auxiliar = (char *)calloc(sizeof(char),20);
        auxiliar[0] = '\0';
        while((caracter[0] = patron[index++]) != '\n')
        {
                strcat(auxiliar, caracter);
        }
        caracter[0] = '\0';
        strcat(auxiliar, caracter);
        servicio->mensaje = (char *)calloc(sizeof(char),20);
        strcpy(servicio->mensaje, auxiliar);
        //printf("Auxiliar:%s\n", auxiliar);
        free(auxiliar);
        //printf("Mensaje:%s\n", servicio->mensaje);
        return OK;
}
```

A.1.11. utils.c

```
#include "coldaemon.h"

void writelog(int log_fd, const char * mensaje)
{
        pthread_mutex_lock(&lock);
        write(log_fd, mensaje, strlen(mensaje));
        pthread_mutex_unlock(&lock);
        return;
}

void thread_add(struct thread_list **lista, int index)
{
        if(*lista == NULL)
        {

                *lista = (struct thread_list *)malloc(sizeof(struct thread_list));
                (*lista)->thread_index = index;
                (*lista)->siguiente = NULL;
        }else{
                struct thread_list * temp  = *lista;
                while(temp->siguiente != NULL)
                {
                        temp = temp->siguiente;
                }
                temp->siguiente = (struct thread_list *)malloc(sizeof(struct
                    thread_list));
                temp = temp->siguiente;
                temp->thread_index = index;
                temp->siguiente = NULL;
        }
        return;
}

pthread_t * thread_get(struct thread_list *lista, int index)
{
        if(lista == NULL)
        {
                //syslog(LOG_ERR,"[CRITICAL ERROR]thread_list llega nulo!\n");
                exit(NULL_THREAD);
        }
        do
        {
                if(index == lista->thread_index)
                {
                        return &(lista->hilo);
                }
        }while(lista->siguiente != NULL && (lista = lista->siguiente) );

        //syslog(LOG_ERR,"No existe el hilo buscado");
        return NULL;
}

void thread_del(struct thread_list **lista, int index)
{
        struct thread_list * temp = *lista;
        struct thread_list * anterior = NULL;
        if(*lista == NULL)
        {
                //syslog(LOG_ERR,"No se pueden borrar los hilos");
                exit(NULL_THREAD);
        }
        do{
                if(index == temp->thread_index)
                {
```

```
                        if(anterior != NULL)
                        {
                                anterior->siguiente = temp->siguiente;

                        }else
                        {
                                *lista = temp->siguiente;
                        }
                        free(temp);
                        return;
                }
                anterior = temp;
        }while(temp->siguiente != NULL && (temp = temp->siguiente));

        //syslog(LOG_ERR,"No hay hilos ejecutandose\n");
        return;
}

/*generador de hash de contrasenha recibe el char ingresado por teclado de la
    contrasenha y lo hashea retornando el long hash del pass*/
uint32_t hash( char * str)
{
    //printf ("lk%s", &str);
    uint32_t hash = 5381;
    int c;

     while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    // printf ("asdn%s", hash);
     return hash;
}

/*recibe el puntero al archivo, el puntero al usuario y contrasenha a verificar*/
char authentication (char * acl_file, char * user, uint32_t  pass_buscado)
{
       char user1[50] = {'\0'};
        char *tokenPtr;
       char *pass_file;
        int x = 0;
       uint32_t pass1=0;
                FILE * acl;

                if( ( acl = fopen(acl_file,"r") ) == NULL)
                {
                        syslog(LOG_ERR,"No se pudo abrir %s\n",acl_file);
                        return CANT_READ_ACL; //Código de error para "No se puede
                            leer ACL"
                }

               strcpy(user1, "");//vacia la variable user1
               fscanf(acl, "%s", user1);//lee una linea del archivo
               pass_file=(strpbrk(user1, ":")+2);//extrae lo que encuentra depues
                   de '::' pass
               pass1 = atoi(pass_file);//convierte el pass string leido de archivo
                   a long

               while (memcmp(user1,user,x)!=0 && pass1!=pass_buscado && !feof(acl))
               {/*si el usuario y la contrasenha no son iguales, y no es fin de
                   archivo leer la siguiente linea del archivo acl*/
                     fscanf(acl, "%s", user1);
                     pass_file=(strpbrk(user1, ":")+2);//extrae de la linea user::
                         pass lo que esta despues del :: "pass"
                     pass1 = atoi(pass_file);//convierte la cadena leida a long
                         para luego comparar
               }
```

```
                /*int memcmp(const void *s1, const void *s2, size_t n);
Compara los primeros n caracteres del objeto apuntado por s1 (interpretado como
unsigned char) con los primeros n caracteres del objeto apuntado por s2 (
    interpretado
como unsigned char).Devuelve 0 en caso que sean iguales*/
                 tokenPtr= strtok(user1,"::");
                if((pass_buscado==pass1 && strcmp(tokenPtr,user)) == 0)
                {/*verifica si la variable registrada el final de recorrer la lista
                    de archivos es igual al usuario y contrasenha buscados para la
                    autenticacion*/
                        return OK;
                }
                else//si no es por que recorrio todo el archivo y no encontro
                    coincidencias de user y pass
                {
                        return INVALID_USER;
                }

}//fin autenticacion
```

Bibliografía