UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA PLATAFORMA BAJO TCP/IP PARA ESTABLECER COMUNICACIÓN TRANSACCIONAL Y POR LOTES, APLICADO A UN SISTEMA DE COMPENSACIÓN BANCARIA

TESIS

PARA OPTAR EL TITULO PROFESIONAL DE

INGENIERO ELECTRÓNICO

PRESENTADO POR

GERMÁN RAÚL HOYOS PARRAVICINO

PROMOCIÓN
1996 - II

LIMA - PERÚ
2004

“Dedico mi tesis a las más grandes personas en mi mundo, que siempre me han apoyado de manera incondicional a lograr mis objetivos y han sido un soporte en momentos de adversidades, me refiero a mi familia, mi adorable madre Rosa, mi muy estimado padre Germán y mi querida hermana Erika“

DESARROLLO DE UNA PLATAFORMA BAJO TCP/IP PARA ESTABLECER COMUNICACIÓN TRANSACCIONAL Y POR LOTES, APLICADO A UN SISTEMA DE COMPENSACIÓN BANCARIA AUTOMÁTICA

SUMARIO

La elaboración de la presente tesis consta del análisis y estudio de diferentes frentes con el fin de lograr la transferencia de información en forma rápida y segura en una red LAN / WAN, Algunos frentes que podemos mencionar: Estudio del funcionamiento de drivers de comunicación empleando primitivas de comunicación TCP/IP así como la elaboración de programas en C ANSI para el manejo de estas primitivas, tanto sobre plataforma LINUX como sobre plataforma WINDOWS 98/2000; el estudio de métodos de encriptación de información y mecanismos prácticos para el cifrado de la data en la transferencia de información a través de la red empleando para ello llaves dinámicas; también el manejo del concepto de comunicación Inter-proceso empleada en el Servidor (LINUX) que cumple diversas funciones, como validar a quien envía la información, llevar un control de los autorizados a transferir información, registro de fechas y horas de la información transferida y almacén de la misma información.

Adicional a los frentes citados anteriormente se ha tenido que emplear conocimientos de Base de Datos (MYSQL) en LINUX, así como la elaboración de programa en C++ sobre WINDOWS 98 / WINDOWS 2000 para generar DLL que contengan las funciones para la transferencia de

información (empleando las primitivas de TCP/IP) y que un módulo Visual Basic pueda mostrar al operador una pantalla amigable. Y por último llevar archivos de configuración los cuales constan de valor de chequeo para que en caso sean alterados pueda el Programa Cliente (Quien envía la data al *Servidor*) *bloquear la transferencia*.

INDICE

PROLOGO 1

SUMARIO 2

CAPÍTULO I 4

PLANTEAMIENTO DEL PROBLEMA 4

1. Descripción del Problema 4

2. Alternativa Planteada 5

3. Objetivos 7

4. Consideraciones 8

CAPÍTULO II 9

PROGRAMACIÓN EN REDES – PROTOCOLO DE COMUNICACIÓN TCP/IP 9

1. Conceptos de Programación en Redes 9

2. Modos de Servicio 9

3. Operaciones Síncronas y Asíncronas 10

4. Definición de cliente-servidor y Tipos de Servidores 11

5. Esquema genérico cliente-servidor 12

6. Definición de TCP/IP y sus Características 15

7. Definición de socket 18

8. Primitivas para crear sockets usando TCP/IP 18

9. Porque elegir TCP/IP 26

CAPÍTULO III 27

MECANISMOS DE COMUNICACIÓN INTERPROCESOS 27

1. Definición de Proceso 27

2. Creación de Procesos y sus Estados 30

3. Sistemas de tiempo compartido 36

4. Mecanismos de Comunicación Interproceso 36

CAPÍTULO IV 43

SEGURIDAD EN REDES LAN/WAN 43

1. Definición de seguridad en una red 43

2. Algoritmos Simétricos 43

3. Algoritmos de Llave Pública 45

4. Diferencias entre Algoritmos Simétricos y Algoritmos de Llave Pública 46

5. Elección de Algoritmos Simétricos y Algoritmos de Llave Pública 46

CAPÍTULO V 50

DESARROLLO DE UNA PLATAFORMA BAJO TCP/IP PARA ESTABLECER COMUNICACIÓN TRANSACCIONAL Y POR LOTES, APLICADO A UN SISTEMA DE COMPENSACIÓN BANCARIA AUTOMÁTICA 50

1. Modelo del Sistema – Esquema Compensación Bancario Planteado 50

2. Descripción del Sistema Servidor 52

3. Configuración del Sistema Servidor 52

4. Descripción de Procesos del Sistema Servidor 54

5. Sistema Cliente 81

6. Consideraciones para caídas de línea 86

7. Algoritmos usado para Autenticación y Cifrado de Información 86

CAPÍTULO VI 92

GENERALIZACION DEL SISTEMA – PLATAFORMA DE COMUNICACIÓN. APLICACIÓN COMERCIO ELECTRONICO 92

CAPÍTULO VII 98

COMPARACION CON OTRAS ALTERNATIVAS Y COSTO DEL PROYECTO 98

CONCLUSIONES Y RECOMENDACIONES 103

ANEXOS 106

ANEXO A 106

ANEXO B 107

ANEXO C 107

ANEXO D 108

ANEXO E 109

ANEXO F 111

ANEXO G 112

ANEXO H 113

ANEXO I 115

ANEXO J 116

ANEXO K 116

ANEXO L 116

ANEXO M 117

BIBLIOGRAFIA 118

PROLOGO

En estos días con el avance tecnológico incesante en los aspectos informáticos, así como en las comunicaciones, las empresas bancarias y otras relacionadas a operaciones de transacciones electrónicas, buscan una mayor rapidez, seguridad y eficiencia en sus operaciones, para lo cual constantemente buscan mejorías en su red de comunicaciones, ya sea en ampliar el ancho de banda del canal de transmisión, expandir la red a lugares más alejados y ampliar el mercado, pero sobretodo estas empresas buscan la mayor seguridad posible en las comunicaciones transaccionales.

La seguridad en redes actualmente es la base de cualquier empresa que pretenda prestar servicio de comercio electrónico a nivel nacional e internacional, ya sea para redes LAN como para redes WAN.

Es por todo lo anteriormente mencionado que he realizado esta tesis para dar una mejor alternativa a las actuales existentes como solución de bajo costo y alta fiabilidad en la transferencia de información.

CAPÍTULO I

PLANTEAMIENTO DEL PROBLEMA

1. Descripción del Problema

El explosivo crecimiento de redes de computadoras privadas y públicas ha dado como resultado un tremendo incremento en el volumen de data valiosa y sensitiva que es rutinariamente almacenada y transmitida digitalmente. Mensajes de computadoras viajando a través de redes con grandes sumas de dinero transferidas electrónicamente, el más grande desafío en este nuevo "mundo digital" es mantener esta información fuera de las manos de usuarios no autorizados quienes dañan sistemas de computadoras vulnerables. Como resultado, en estos días los principales objetivos que buscan las empresas cuyo rubro involucra la transferencia de información en forma electrónica, son la rapidez y la seguridad en trasmitirla, tanto para redes LAN como WAN. A continuación se dan algunos ejemplos bajo los cuales se realiza la transferencia de información en las redes LAN y WAN:

- Transferencia de transacciones electrónicas desde un adquirente (Cajeros, Kioskos, Páginas WEB, POS, etc.) a un servidor (Banco Emisor de Tarjetas, Institución Administradora de Procesos, etc.).
- Realización de consultas a una base de datos, en forma interactiva, como consultas de saldos desde páginas WEB.
- Transferencia de información para casos de contingencia (Información de respaldo), lo cual implica tener la información en dos equipos diferentes para de esta forma tener un backup por redundancia.
- Transferencia por lotes de información bancaria como envío de información de procesos de cierres, cargas de datos en forma masiva, etc.

2. Alternativa Planteada

Para cubrir con los requerimientos que resuelvan el problema planteado se necesitan drivers de comunicación que sean rápidos y eficientes, así como de elementos de seguridad que garanticen que la información sea transferida en forma segura, es decir, sin que pueda ser descifrada y/o manipulada en caso fuese capturada en el proceso de la transferencia electrónica. La figura 1.1 muestra como es la transmisión de información en una red.

Con lo anterior expuesto, se plantea desarrollar en base a estos requerimientos una aplicación orientada al comercio electrónico que se encargue de transferir información en la Red en forma rápida y segura, lo cual llevará a desarrollar los elementos necesarios para crear una medio

seguro donde se pueda transmitir información en forma rápida y segura, usando para ello API's de TCP/IP y algoritmos de cifrado en línea y como ejemplo se aplicará a transferencia electrónica de un sistema de compensación bancario automático.

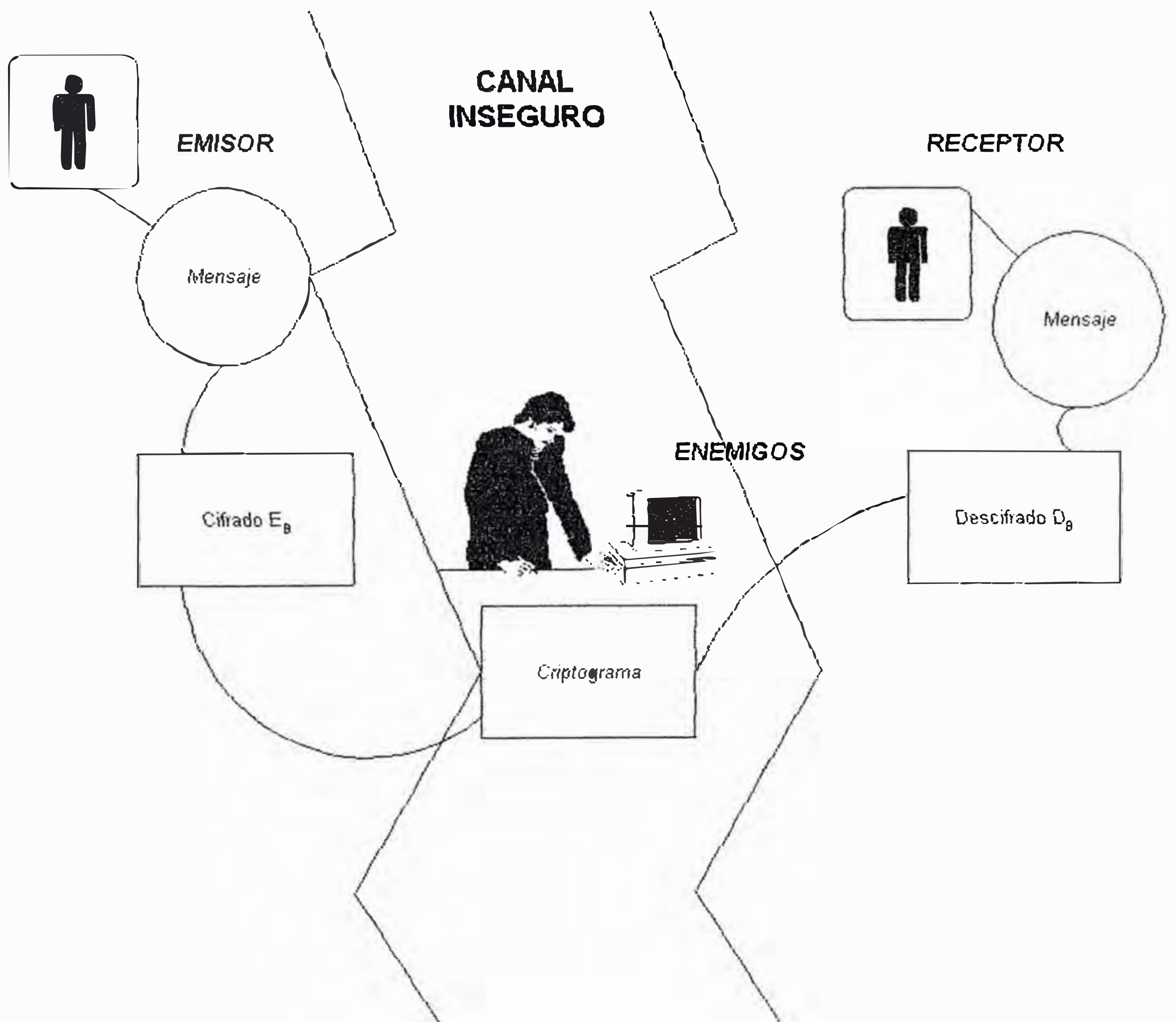


Figura 1.1: Esquema que representa un canal de transmisión público y lo peligroso que es transmitir a través de él sin cifrar la información.

3. Objetivos

Los objetivos podemos subdividirlos en dos partes, el objetivo general y los objetivos específicos. El objetivo general trata de explicar el porque la necesidad a solucionar el problema planteado en el ítem 1 de este capítulo, mientras que los objetivos específicos explican como es que en esta tesis se plantea resolver el problema.

El objetivo general es:

- Explicar la necesidad de implementar un esquema de seguridad en la transferencia de información en redes de comunicación, tanto para redes internas (LAN) como para redes externas (WAN).

Los objetivos específicos son:

- Explicar el propósito de los métodos de seguridad existentes actualmente en el medio y los métodos de seguridad más empleados.
- Que la solución planteada sea flexible para nuevos cambios y mejoras.
- Tener una solución que se acomode a los estándares empleados en nuestro medio, con énfasis al sector financiero.
- Resolver el problema planteado con un costo mínimo.

4. Consideraciones

Para el desarrollo de este proyecto estamos considerando lo siguiente:

- El proyecto ha sido desarrollado sobre LINUX Red HAT 8.0; por tanto una migración a cualquier otra plataforma UNIX requerirá de una recompilación y en algunos casos adecuación de programas para mantener las funcionalidades especificadas.
- El proyecto desarrollado es independiente del medio de *comunicación, por tanto, las características de los equipos de* comunicaciones existentes como medio de transporte de la información tienen que adecuarse a los requerimientos solicitados de tiempo de respuesta solicitados por el cliente.
- Si se emplean routers, firewalls, proxies, etc. estos deben ser configurados de tal modo que puedan permitir la comunicación a nivel aplicativo, habilitando para esto el paso a direcciones IP y puertos definidos para el transporte de información.
- Para la aplicación desarrollada en WINDOWS se requiere un mínimo de las siguientes características:

  1. 64 MB de RAM
  2. *Procesador Pentium III 1.3 Ghz*

CAPÍTULO II

PROGRAMACIÓN EN REDES – PROTOCOLO DE COMUNICACIÓN TCP/IP

1. Conceptos de Programación en Redes

La esencia de la programación en redes es la de manejar las comunicaciones entre procesos que se están ejecutando en diferentes computadoras. El modelo que sirve de base en la programación en redes es el denominado *"cliente-servidor"*. En este modelo un proceso en una *computadora (el servidor) espera por otro proceso en la misma u otra* computadora (cliente) para contactarlo.

2. Modos de Servicio

Por otro lado es importante definir los ***modos de servicio***, los cuales pueden dividirse principalmente en dos: *"orientado a conexión"* y *"no orientado a conexión"*.

Con *el servicio "orientado a conexión", un cliente y un servidor* establecen una ruta de comunicación sobre la cual ellos envían y reciben

datos. Esta clase de servicio es útil cuando un cliente y un servidor esperan tener un diálogo extenso. Una de las características del servicio orientado a conexión es que la comunicación es secuencial, confiable y libre de errores. Esto significa que toda la data enviada por un proceso es garantizada para ser recibida por el proceso destino, que la data llegue exactamente en el *orden en que fue enviada, y que la data llegue sin alteraciones o* corrupciones.

Por otro lado, con el servicio *"no orientado a conexión"*, el cliente y el servidor intercambian mensajes individuales. Esto es a veces apropiado cuando el servicio provisto por el servidor requiere muy poca interacción entre el servidor y el cliente. Cuando la "no conexión" es establecida, cada mensaje debe contener la dirección de la red destino, identificación de la computadora y la identificación del proceso local. Similarmente cada mensaje recibido debe contener la dirección de la red, identificación de la computadora y la identificación del proceso local del proceso que envió el mensaje.

3. Operaciones Síncronas y Asíncronas

Es importante también definir las operaciones síncronas y asíncronas. Por un lado el modo síncrono una función no retorna hasta que la operación se haya completado (o hasta que un error es detectado). Por ejemplo, si un proceso llama a una función de recibir datos en modo síncrono y no hay data disponible, la llamada se bloquea hasta que llegue data. Por otro lado, si el proceso llama a la función de recibir datos en modo asíncrono y no hay data

disponible, la llamada retorna inmediatamente con error (valor –1) y una variable global ***errno*** es actualizada a NODATA. Este concepto es importante porque lo usaremos más adelante para el caso de establecer comunicación entre procesos o para el intercambio de mensajes entre programas cuando veamos comunicación interprocesos.

4. Definición de cliente-servidor y Tipos de Servidores

El modelo *cliente-servidor* es el modelo estándar de ejecución de aplicaciones en una red. A continuación veremos un modelo general.

Un *servidor* es un proceso que se está ejecutando en un nodo de la red y que gestiona el acceso a un determinado recurso. Un *cliente* es un proceso que se ejecuta en el mismo o en diferente nodo y que realiza peticiones de servicio al servidor. Las peticiones están originadas por la necesidad de acceder al recurso que gestiona el servidor.

El servidor está continuamente esperando peticiones de servicio enviadas por el cliente. Cuando se produce una petición, el servidor despierta y atiende al cliente. Cuando el servicio concluye, el servidor vuelve al estado de espera.

De acuerdo con la forma de prestar servicio, podemos considerar dos tipos de servidores.

- *Servidores interactivos:* El servidor no solo recoge la petición de servicio, sino que el mismo se encarga de atenderla. Esta forma de trabajo presenta un inconveniente: si el servidor es lento en atender a los

clientes y hay una demanda de servicio muy elevada, se van a originar unos tiempos de espera muy grandes.

- *Servidores concurrentes:* El servidor recoge cada una de las peticiones de servicio y crea otros procesos para que se encarguen de atenderla. Este tipo de servidores sólo es aplicable en sistemas multiproceso, como es LINUX. La ventaja que tiene este tipo de servicio es que el servidor puede recoger peticiones a muy alta velocidad, porque está descargado de la tarea de atención al cliente.

5. Esquema genérico cliente-servidor

La forma genérica de los procesos servidores como la de los procesos clientes, es como se muestra en la figura 2.1.

Las acciones que deben llevar a cabo los programas servidores son las siguientes:

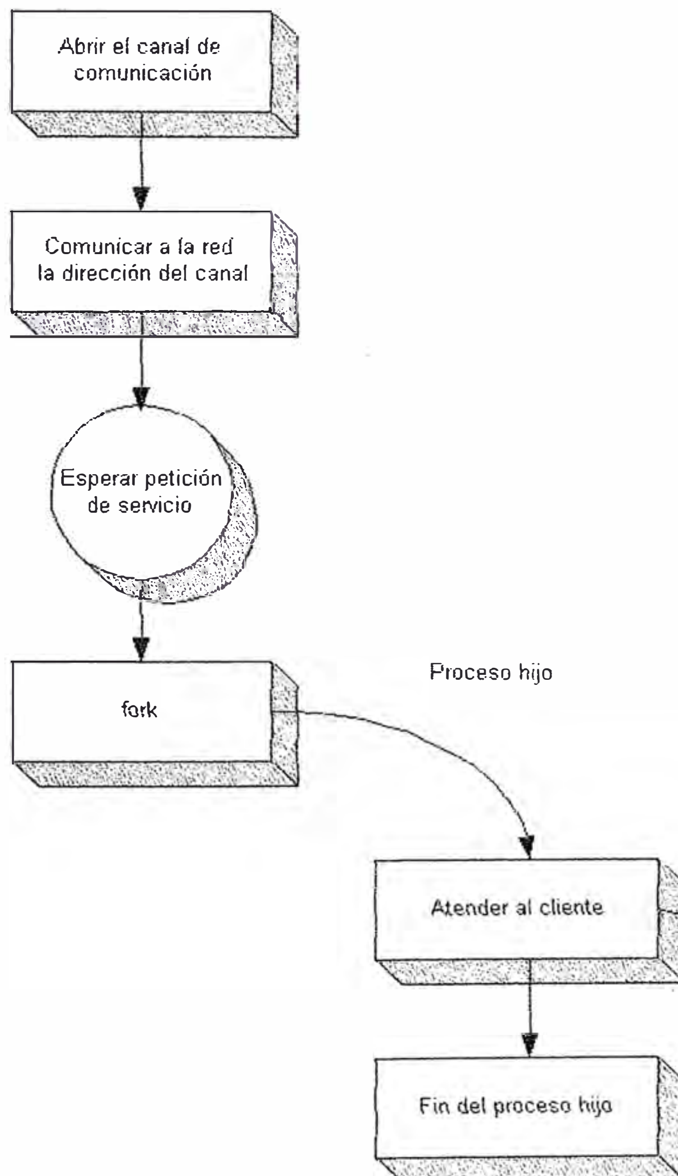
- Abrir el canal de comunicaciones e informar a la red tanto de la dirección a la que responderá como de su disposición para aceptar peticiones de servicio.
- Esperar a que un cliente solicite un servicio en la dirección que él tiene declarada.
- Cuando recibe una petición de servicio, si es un servidor interactivo, atenderá al cliente. Los servidores interactivos se suelen implementar cuando la respuesta que necesita el cliente es sencilla e implica poco tiempo de proceso.

Si el servidor es concurrente, creará un proceso mediante ***fork*** para que le dé servicio al cliente y pueda seguir esperando por otros servicios que le soliciten otros clientes.

- Volver al punto 2 para esperar nuevas peticiones de servicio.

El programa cliente, por su parte, llevará a cabo las siguientes acciones:

- Abre el canal de comunicaciones y se conecta a la dirección de red atendida por el servidor. Naturalmente, esta dirección debe ser conocida por el cliente y responderá al esquema de generación de direcciones de la familia de *sockets* que se está empleando.
- Enviar al servidor un mensaje de petición de servicio y esperar hasta recibir la respuesta. Su lógica debe considerar un tiempo máximo de espera para el caso de no recibir una respuesta por parte del servidor.
- Cierra el canal de comunicaciones y terminar la ejecución.



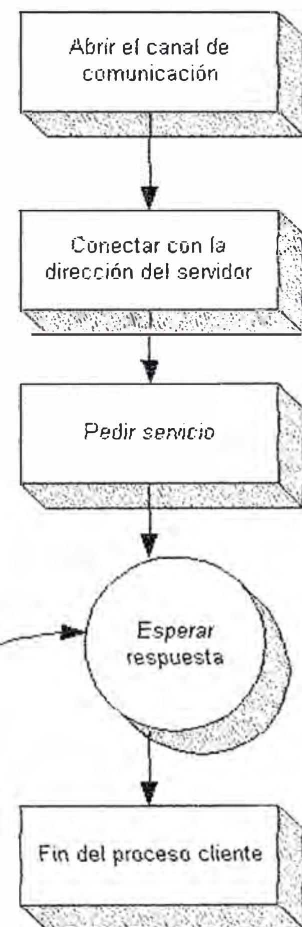


Figura 2.1: Este gráfico representa un esquema genérico de un cliente-servidor.

6. Definición de TCP/IP y sus Características

TCP/IP es un conjunto de protocolos de comunicación para comunicaciones **orientadas a conexión** que son independientes de la base de la tecnología de hardware de redes.

Entre las característica del protocolo TCP está que acepta mensajes de *longitud arbitrariamente grande procedentes de los procesos de usuario*, los separa en pedazos que no excedan de 64K octetos, y transmite cada pedazo como si fuera un datagrama separado. La capa de red no garantiza que los datagramas se entreguen apropiadamente, por lo que TCP deberá utilizar temporizadores y retransmitir los datagramas si fuera necesario. Los datagramas que consiguen llegar, pueden hacerlo en desorden; y dependerá de TCP el hecho de ensamblarlos en mensajes, con la secuencia correcta.

En la figura 2.2 se muestra la cabecera que se utiliza en TCP. La primera cosa que llama la atención es que la cabecera mínima de TCP sea de 20 octetos. Los campos Puerto fuente y Puerto destino identifican los puntos terminales de la conexión.

<table>
<tr><td colspan="8">Puerto Fuente</td><td>Puerto Destino</td></tr>
<tr><td colspan="9">Número de Secuencia</td></tr>
<tr><td colspan="9">Asentamiento en superposición</td></tr>
<tr><td>Longitud de la cabecera TCP</td><td></td><td>U
R
G</td><td>A
C
K</td><td>E
O
M</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td></td></tr>
<tr><td colspan="8">Código de Redundancia</td><td>Puntero Urgente</td></tr>
<tr><td colspan="9">Opciones (0 o mas palabras de 32 bits)</td></tr>
<tr><td colspan="9">Datos</td></tr>
</table>

Figura 2.2: Muestra la cabecera usada por el protocolo TCP

Los campos Número de secuencia y Asentamiento en superposición tienen una longitud de 32 bits, debido a que cada octeto de datos está numerado en TCP.

La longitud de la cabecera TCP indica el número de palabras de 32 bits que están contenidas en la cabecera de TCP. Esta información es necesaria porque el campo Opciones tiene una longitud variable, y por lo tanto la cabecera también. Después aparecen seis banderas de 1 bit. Si el puntero acelerado se está utilizando, entonces URG se coloca a 1. El puntero acelerado se emplea para indicar un desplazamiento en octetos a partir del número de secuencia actual en el que se encuentran datos acelerados. Esta facilidad se brinda en lugar de los mensajes de

interrupción. El bit SYN se utiliza para el establecimiento de conexiones. La solicitud de conexión tiene SYN = 1 y ACK = 0, para indicar que el campo de asentamiento en superposición no está utilizado. La respuesta a la solicitud de conexión sí lleva un asentamiento, por lo que SYN = 1 y ACK = 1. El bit FIN se utiliza para liberar la conexión; especifica que el emisor ya no tiene más datos. *Después de cerrar una conexión, un proceso puede seguir* recibiendo datos indefinidamente. El bit RST se utiliza para reiniciar una conexión que se ha vuelto confusa debido a SYN duplicados y retardados, o a caídas de los hosts. El bit EOM indica el fin de mensajes.

El control de flujos en TCP se trata mediante el uso de una ventana deslizante de tamaño variable. Es necesario tener un campo de 16 bits, porque la ventana indica el número de octetos que se pueden transmitir más allá del octeto asentido por el campo de ventana y no cuentas unidades de datos del protocolo de transporte.

Por otro lado, El código de redundancia también se brinda como un factor de seguridad extremo. El algoritmo del código de redundancia consiste en sumar simplemente todos los datos, considerados como palabras de 16 bits, y después tomar el complemento a 1 de la suma.

El campo Opciones se utiliza para diferentes cosas, por ejemplo para comunicar tamaños de tampones durante el procedimiento de establecimiento.

7. Definición de socket

Es la llamada para abrir un canal bidireccional de comunicaciones (apertura de un punto terminal en un canal), es decir, es un punto de comunicación por el cual un proceso puede emitir o recibir información. En el interior de un proceso un socket se identificará por un descriptor de la misma forma que identifican los archivos. Esta propiedad es esencial puesto que permite, por ejemplo, la redirección de los archivos de entrada/salida estándar a los sockets y por tanto, la utilización de aplicaciones estándar sobre la red. Esto significa también que todo proceso nuevo (creado por fork) hereda los descriptores de socket de su padre.

8. Primitivas para crear sockets usando TCP/IP

Las primitivas usadas para crear sockets se describen a continuación:

- *Socket*

La declaración de esta función es como sigue:

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (af, type, protocol)
int af, type, protocol;
```

El socket crea un punto terminal para conectarse a un canal y devuelve un descriptor. El descriptor de socket devuelto se usará en llamadas posteriores a funciones de la interfaz.

El argumento af (address family) especifica la familia de sockets o familia de direcciones que se desea emplear. Las distintas familias están definidas en el fichero de cabecera <sys/socket.h> y dependerá del

fabricante del sistema y de la configuración del hardware. Las siguientes familias suelen estar presentes en todos los sistemas:

AF_UNIX Protocolos internos UNIX. Es la familia de sockets empleada para comunicar procesos que se ejecutan en una misma máquina. Esta familia no requiere que esté presente el hardware especial de red, *puesto que en realidad no realiza accesos a ninguna red.*

AF_INET Protocolos Internet. Es la familia de sockets que se comunican mediante protocolos, tales como TCP o UDP.

El argumento type indica la semántica de la comunicación para el socket y puede tener los valores:

SOCK_STREAM Socket con un protocolo orientado a conexión.

SOCK_DGRAM Socket con un protocolo no orientado a conexión o datagrama.

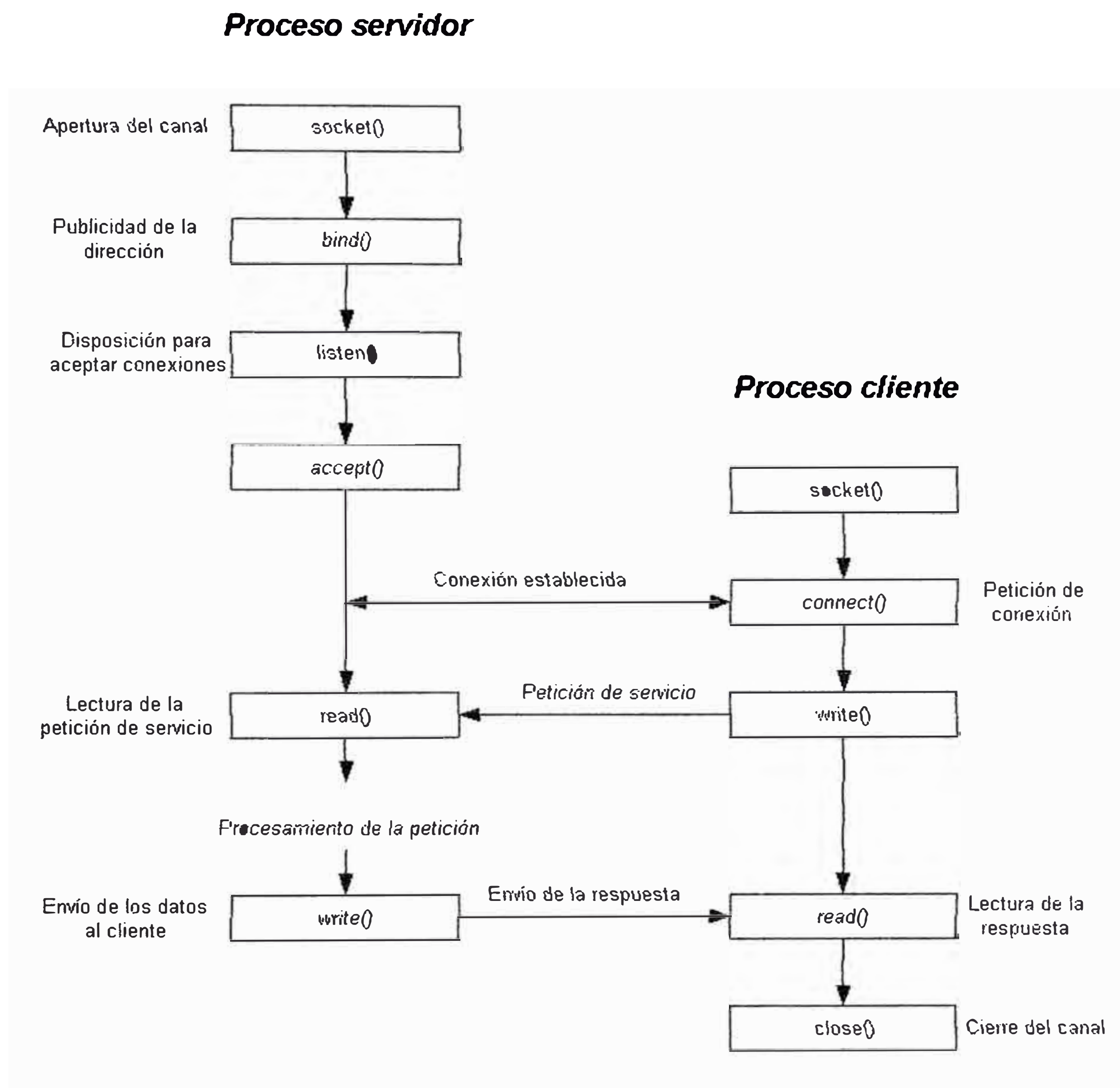


Figura 2.3: Secuencia de llamadas para crear el cliente y el servidor

El argumento protocol indica el protocolo particular que se va usar en el socket. Normalmente, cada tipo de socket tiene asociado solo un protocolo, pero si hubiera más de uno, se especificaría mediante este

argumento. Si este argumento es cero, la elección del protocolo la hará el sistema.

La figura 2.3 muestra una secuencia de las llamadas para realizar el servidor y el cliente cuando la conexión es virtual. En la figura vemos en los cuadros los nombres de las primitivas, las cuales se explicarán a continuación.

- *Bind*

Esta llamada se usa para unir un socket a una dirección de red determinada. Su declaración es la siguiente:

```
Int bind (sfd, addr, addrlen)
Int sfd;
Const void *addr;
Int addrlen;
```

Bind hace que el socket de descriptor sfd se una a la dirección de socket especificada en la estructura apuntada por addr, mientras addrlen especifica el tamaño de la dirección.

Es necesario saber que el formato de la dirección depende de la familia del socket, por lo que habrá que utilizar la estructura adecuada. Para la familia AF_INET la estructura es *struct sockaddr_in*. Y el valor del argumento addrlen se puede calcular con la función *sizeof*.

- *Listen*

Cuando se abre un socket orientado a conexión, el programa servidor indica que está disponible para recibir peticiones de conexión mediante la llamada *listen*, la cual se declara de la siguiente manera:

```
Int listen (sfd, backlog)
Int sfd, backlog;
```

La llamada a *listen* se suele ejecutar en el proceso servidor después de las llamadas a *socket* y *bind*. Listen habilita una cola asociada al socket descrito por sfd. Esta cola se va encargar de alojar las peticiones de conexión procedentes de los procesos clientes. La longitud de esta cola es la especificada por el argumento backlog. Para que la llamada a *listen* tenga sentido, el socket debe ser del tipo SOCK_STREAM (orientado a conexión).

La cola de conexiones es importante para los servidores del tipo interactivo porque mientras están procesando la petición de un cliente pueden llegar más peticiones de otros clientes.

- *Connect*

Para que un proceso cliente inicie una conexión con un servidor a través de un socket, es necesario que haga una llamada a connect. Esta función se declara de la siguiente forma:

Int connect (sfd, addr, addrlen)

El argumento sfd es el descriptor del socket que da acceso al canal y addr es un puntero a una estructura que contiene la dirección del socket remoto (servidor) al que queremos conectarnos y addrlen es el tamaño en bytes de la dirección de la dirección.

En el caso de socket del tipo SOCK_STREAM (orientado a conexión), *connect* intenta contactar con el ordenador remoto con objeto de realizar una conexión entre el socket remoto y el socket local. La llamada permanece bloqueada hasta que la conexión se completa, pero si el socket tiene activo el modo de acceso O_NDELAY (activado a través de una llamada a fcntl), la llamada a *connect* devuelve el control inmediatamente indicando que se ha producido un error de conexión.

- *Accept*

Los procesos servidores leen las peticiones de servicio de los clientes empleando la función accept, la cual se declara como sigue:

Int accept (sfd, addr, addrlen)

Esta llamada solo se usa con sockets orientados a conexión, como el tipo SOCK_STREAM. El argumento sfd es un descriptor de socket por una llamada previa a *socket* y unido a una dirección mediante *bind*. Accept extrae la primera petición de conexión que hay en la cola de peticiones creada con la función *listen*. Una vez extraída la petición de conexión, *accept crea un nuevo socket con las mismas propiedades que sfd y reserva un* nuevo descriptor de fichero (nsfd) para él.

Si no hay conexiones pendientes y el *socket* no tiene activo el modo de acceso no bloqueante (O_NDELAY), *accept* permanece bloqueado hasta que se reciba una nueva petición. Si el modo no bloqueante está activo, *accept* devolverá el control inmediatamente e indicará que se ha producido un error.

- *Recv*

Esta función es usada para recibir mensajes de un canal, y su declaración es como sigue:

recv(sdf, message, lenght, flag)

El parámetro sfd es el descriptor del socket, el parámetro message es un puntero char el cual recibirá la información, el parámetro lenght especifica la longitud máxima que se recibe y el parámetro flag da las características del modo de recepción, los cuales pueden ser 0 o una combinación de los bits siguientes:

MSG_PEEK: Cualquier dato leído del socket es tratado como si la lectura no se hubiera llevado a cabo, por lo que la siguiente operación de lectura va a leer los mismos datos.

MSG_OOB: Se utiliza para leer datos que van fuera de banda. Estos son datos a los que se les da un carácter de urgente, por lo que tienen preferencia en el envío y recepción.

En los sockets de la familia AF_UNIX no puede estar activo ninguno de los bits anteriores.

- *Send*

Esta función es usada para enviar mensajes a un canal, y su declaración es como sigue:

send (sdf, message, lenght, flag)

El parámetro sdf es el descriptor del socket, el parámetro message es un puntero char el cual tiene la información que se va enviar al canal, el parámetro legth especifica la longitud que se envía y el parámetro flag da las características del modo de envío. Las características del modo de envío *puede ser 0 o MSG_OOB, sin embargo, igual que el caso de "recv"* MSG_OOB no puede estar activo para sockets de la familia AF_UNIX.

- *Close*

  Esta función es usada para el cierre del canal.

9. Porque elegir TCP/IP

La tremenda aceptación y auge de TCP/IP en el ámbito comercial se debe a las siguientes razones:

- Libre comercialización y amplia disponibilidad.
- *Soporte para tecnologías diversas como Ethernet, Token Ring, X.25,* Frame Relay, ATM.
- *Se encuentra en computadoras de diferentes tamaños,* PCs, workstations, Minis, Mainframes.
- Soporte para tecnologías de enrutamiento dinámico (RIP,OSPF).

CAPÍTULO III

MECANISMOS DE COMUNICACIÓN INTERPROCESOS

1. Definición de Proceso

Proceso es una parte de un programa que ha sido cargado en memoria para su ejecución. En un proceso no sólo hay copia del programa, sino que además el kernel le añade información adicional para poder manejarlo.

Para tener un panorama más claro se repasará como se organizan los procesos en la memoria. En la mayoría de los casos simples de gestión de la memoria por el sistema es transparente, las peticiones de reserva dinámica (bien en la pila o en la zona de datos) son automáticas o se realizan mediante llamadas a la función de la biblioteca estándar *malloc*.

- *Espacio de direccionamiento de un proceso*

Con el fin de permitir la carga de programas en memoria de direcciones físicas cualesquiera, el código de los programas ejecutables contiene direcciones "virtuales". El encargado de traducir estas direcciones virtuales en direcciones físicas en el transcurso de la ejecución del programa es el módulo de gestión de memoria. Por tanto, un proceso dispone de un

espacio de direccionamiento virtual dividido tradicionalmente en tres regiones lógicas (texto que constituye el programa, datos y pila). La memoria física está dividida en páginas (unidad de memoria que se reserva) de tamaño variable según las máquinas, la asociación de una dirección física a una dirección virtual se divide en páginas lógicas y en un instante dado una tabla de páginas contiene las informaciones de las páginas lógicas (existencia o no de una página física, modificación o no de la página o edad de la página).

Además, cada una de las regiones dispone de su propia tabla de páginas. Otro punto interesante de este mecanismo es que, no es necesario haberlo cargado en su totalidad en memoria.

- *Organización General de la Memoria*

El espacio de direccionamiento virtual de un proceso puede esquematizarse como se muestra en la figura 3.1.

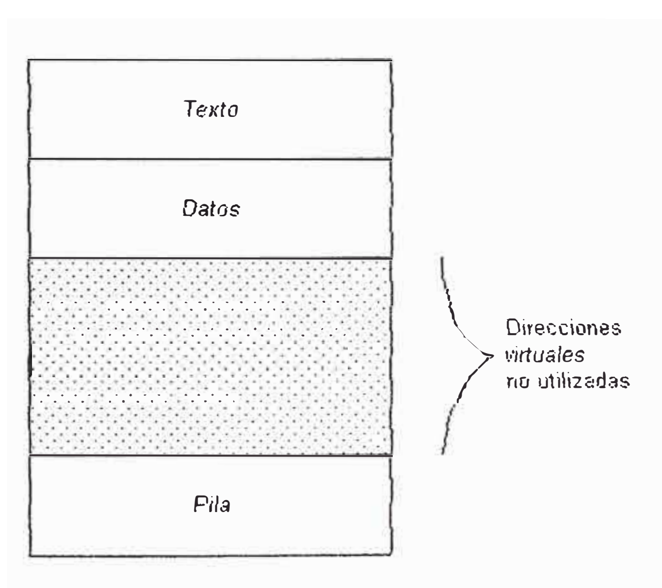


Figura 3.1: Organización general de la memoria

La zona de datos y la pila tienen la posibilidad de crecer. En efecto el tamaño de la pila cambia según las diferentes llamadas a las funciones (forman parte de esta región los parámetros y variables automáticas). En lo que concierne a la zona de datos puede dividirse en dos partes: una zona de tamaño fijo (correspondiente a variables estáticas y externas conocidas desde la compilación) y una zona dinámica cuyo tamaño puede variar en función de las peticiones de reserva de memoria por los procesos.

- *Punto de Ruptura*

La gestión de la pila está asegurada automáticamente por el sistema; por el contrario el usuario puede actuar sobre la zona de datos. En un instante dado, se llama *punto de ruptura* (breakpoint) a la dirección virtual más pequeña fuera de la región de datos. A priori, toda dirección de la región de datos superior a este valor es incorrecta. Sin embargo, es posible acceder a toda una página (deducida de la posición del punto de ruptura), con el mecanismo de reserva de memoria por páginas.

Por el contrario, toda tentativa de escritura más allá del límite de esta página provocará un error de memoria materializado por una señal específica SIGSEGV. Esta misma señal se envía al proceso cuando la pila alcanza el límite impuesto por el sistema.

El esquema de organización del espacio de direccionamiento de un proceso puede detallarse como se muestra en la figura 3.2.

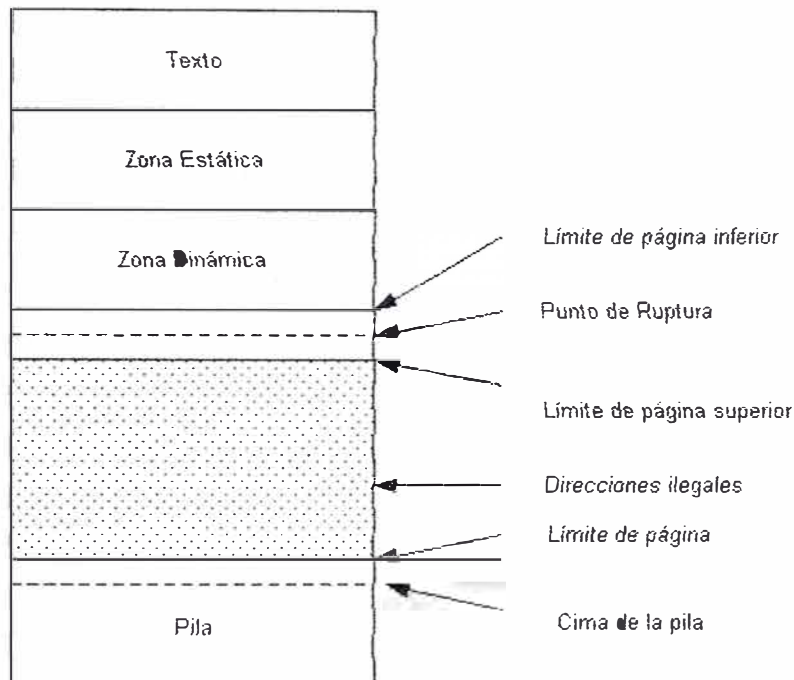


Figura 3.2: Esquema de organización del espacio de direccionamiento de un proceso

2. Creación de Procesos y sus Estados

La única forma de crear procesos en el sistema UNIX o LINUX es mediante la llamada ***fork***. El proceso que invoca a fork se llama proceso padre y el proceso creado es el proceso hijo. La declaración de fork es la siguiente:

```
#include <stdio.h>
pid_t fork();
```

La llamada a ***fork*** hace que el proceso actual se duplique. A la salida de fork, los dos procesos tienen una copia idéntica del contexto del nivel de usuario excepto el valor de ***pid***, que para el proceso padre toma el valor de PID del proceso hijo y para el proceso hijo toma el valor de 0. El proceso 0, creado por el *kernel* cuando arranca el sistema, es el único que no se crea *vía una llamada* ***fork***.

Si la llamada a fork falla devolverá un valor de –1 y en la variable ***errno*** el código de error producido. En la figura 3.3 se ilustra la creación del proceso.

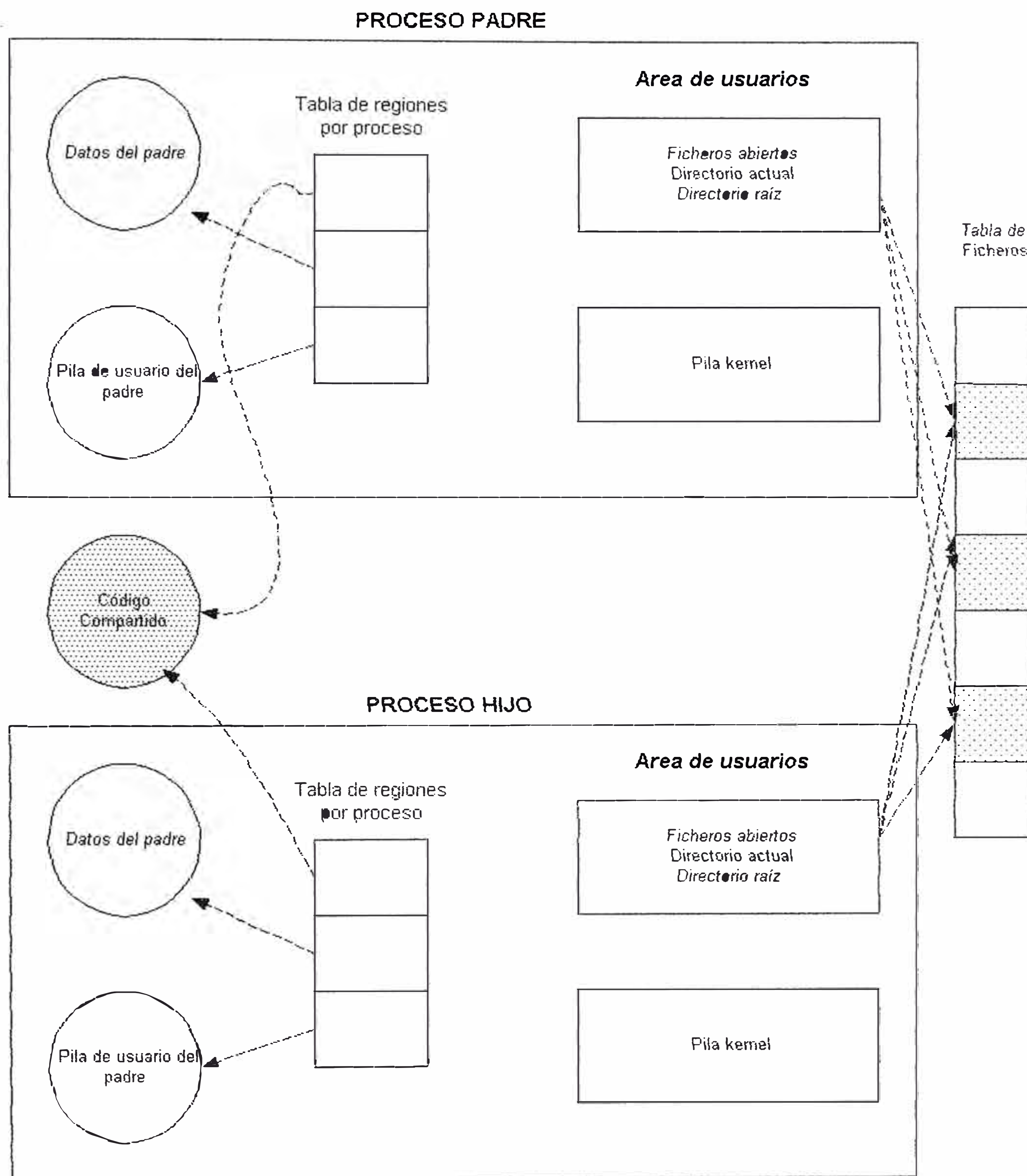


Figura 3.3: Creación de un proceso mediante fork

Los estados de un proceso están ligados al tiempo de vida de un proceso y pueden dividirse en un conjunto de estados, cada uno con unas características determinadas.

En un sistema monoprocesador no puede haber más de un proceso ejecutándose a la vez. Así, en los dos modos de ejecución (usuario y kernel) solo podrá haber un proceso en cada estado.

Un proceso no permanece siempre en un mismo estado, sino que está continuamente cambiando de acuerdo con unas reglas bien definidas. Estos cambios de estado vienen impuestos por la competencia que existe entre los procesos para compartir un recurso escaso como es la CPU. En la figura 3.4 se muestra un diagrama completo de los estados de un proceso, y los estados son:

- El proceso se está ejecutando en modo usuario.
- El proceso se está ejecutando en modo kernel.
- El proceso no se está ejecutando, pero está listo para ejecutarse tan pronto como el kernel lo ordene.
- El proceso está durmiendo cargando en memoria.
- El proceso está listo para ejecutarse, pero el swapper (proceso 0) debe cargar el proceso en memoria antes de que el kernel pueda ordenar que pase a ejecutarse.
- El proceso está durmiendo y el swapper ha descargado el proceso hacia una memoria secundaria (área de swap del disco) para crear espacio en la memoria principal donde cargar otros procesos.

- El proceso está volviendo del modo kernel al modo usuario, pero el kernel se apropia del proceso y hace un cambio de contexto, pasando otro proceso a ejecutarse en modo usuario.
- El proceso acaba de ser creado y está en un estado de transición, el proceso existe, pero ni está preparado para ejecutarse (estado 3), *ni durmiendo* (estado 4). Este estado es el inicial para todos los procesos, excepto el proceso 0.
- El proceso ejecuta la llamada *exit* y pasa al estado *zombi*. El proceso ya no existe, pero deja su proceso padre en un registro que contiene el código de salida y algunos datos estadísticos tales como los tiempos de ejecución. El estado de *zombi* es el estado final de un proceso.

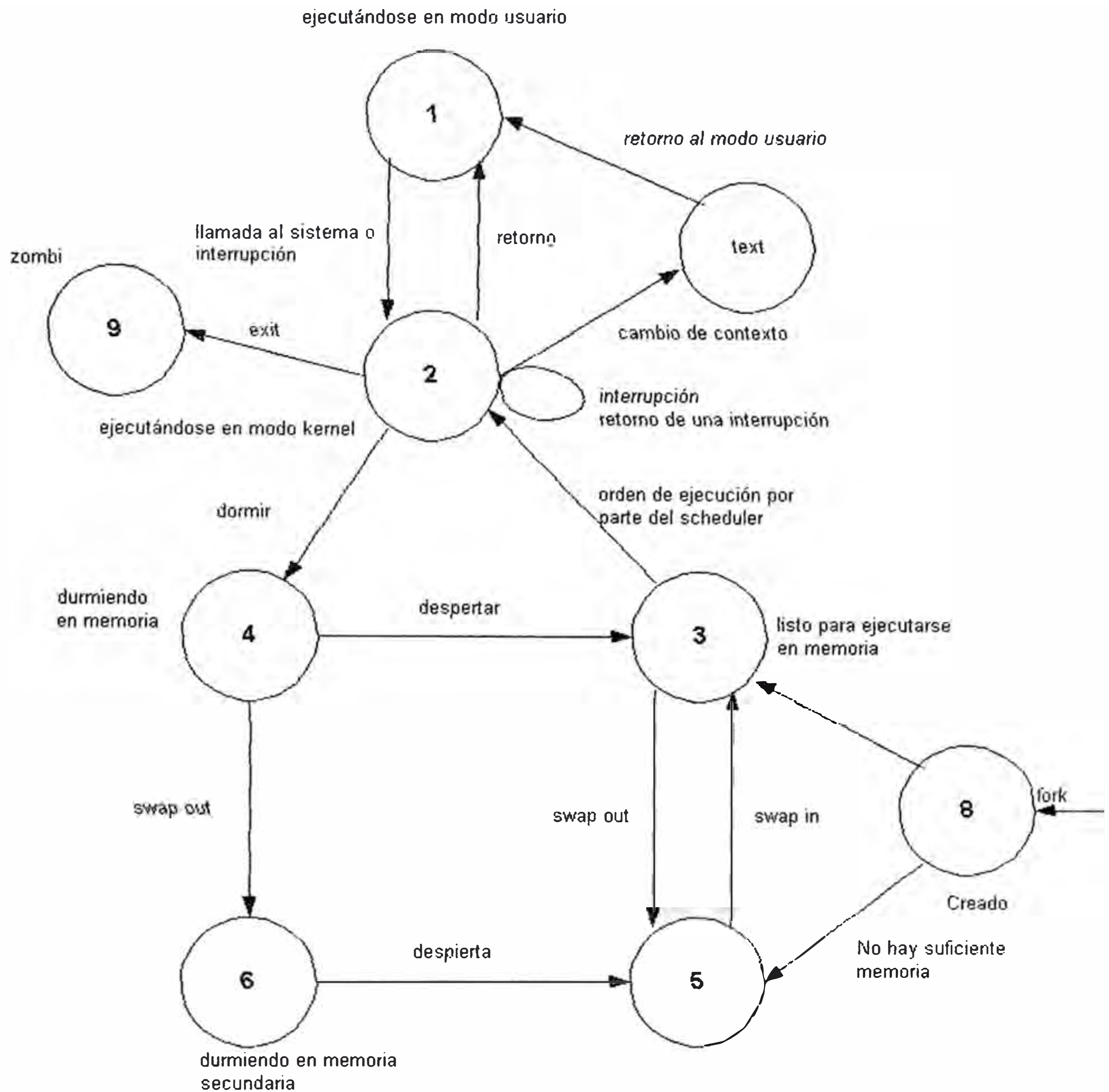


Figura 3.4: Diagrama completo de transición de estados de un proceso

3. Sistemas de tiempo compartido

Para el desarrollo del presente proyecto se ha tenido en cuenta el concepto de comunicación entre procesos con el objetivo de que el trabajo sobre la información se divida en más de un programa para tener mayor orden y a la vez control sobre cada rutina elaborada y sobre la actividad de los mismos procesos. Los conceptos que vamos a ver a continuación son los de colas y memorias compartidas.

4. Mecanismos de Comunicación Interproceso

Los mecanismos de comunicación interproceso básicamente están referidos a los conceptos de cola de mensajes y memorias compartidas, las cuales serán descritas a continuación.

Las colas de mensajes y sus primitivas son un mecanismo de comunicación que es implantado en UNIX bajo el concepto de buzón y permiten la comunicación indirecta entre procesos. Un proceso tiene la posibilidad de depositar mensajes o de extraerlos. Para facilitar la comunicación, cada mensaje está tipificado, la interpretación del tipo se deja a los propios procesos. La idea consiste, simplemente, en permitir que un proceso distinga entre los mensajes contenidos en una cola, los que quiere extraer.

Una propiedad esencial de un mensaje es que constituye un todo, es decir, que cuando un proceso lo extrae, lo hace en una sola operación. Así las colas de mensajes se gestionan según el modo FIFO.

El archivo que contiene las definiciones de las diferentes estructuras y constantes para la manipulación de las colas de mensajes se llama "/usr/include/sys/msg.h".

La estructura genérica de un mensaje predefinido en el archivo <sys/msg.h>, mencionado anteriormente, no puede usarse tal cual.

```
struct msgbuf{
        mtyp_t          mtype;
        char            mtext[MAXDATA];
}
```

Cuando se manipulan colas de mensajes, el usuario debe definir su propia estructura con nombre distinto de msgbuf (para evitar conflictos en la manipulación) y adaptada al uso que se vaya a hacer de las colas de mensajes.

Las *primitivas usadas para manipular las colas son las siguientes:*

- *msgget*

```
int msgget( clave, opcion)
        key_t clave;  //Clave de la Cola de mensajes
        int opcion;   //Opcion de creacion
```

Esta primitiva permite obtener la identificación de una cola de clave dado después de haberla creado.

- *msgsnd*

```
int msgsnd( msgid, p, len, opcion)
        int msgid;      //Identificación de la Cola
        struct msgbuf *p;//Puntero al msg que se va enviar
        int len;        //Longitud del mensaje
        int opcion;     //Opción de emisión
```

Una llamada así corresponde a un envío de un mensaje contenido en memoria en la cola identificada por msgid a la dirección p. Esta zona de memoria tiene, por un lado, el tipo de mensaje (estrictamente positivo) y, por otro, el propio mensaje. El parámetro len contiene la longitud del mensaje (es decir, sin tener en cuenta el espacio ocupado por el tipo). La primitiva devuelve el valor 0 si la emisión ha tenido éxito. El parámetro opcion permite especificar el comportamiento del proceso emisor en caso de que no pueda enviarse el mensaje por causa de la carga del mecanismo de colas de mensajes (demasiados mensajes o cola llena). Por defecto la primitiva msgsnd, en estas circunstancias, es bloqueante. Se despertará el proceso cuando sea posible la escritura. La utilización de la constante IPC_NOWAIT convierte la llamada bloqueante en no bloqueante y suministra un valor de retorno igual a –1, estando la variable errno posicionada con el valor ENOMSG.

Se puede interrumpir una llamada bloqueante a la primitiva: entonces, se suministra el valor de retorno –1 y errno = EINTR.

- *msgrcv*

```
int msgrcv( msgid, p, lenmax, tipo, opcion)
        int msgid;              // Identificación de la Cola
        struct msgbuf *p;//Puntero al msg que se va enviar
        int lenmax;// Long. máxima que se puede aceptar
        int tipo;//Tipo de msg que se espera recibir
        int opcion;             //Opción de emisión
```

Esta primitiva permite extraer de modo efectivo un mensaje de un tipo dado de una cola de mensajes y colocar el contenido en la dirección especificada. Una llamada así es, a priori, bloqueante: el proceso se bloquea hasta que llega un mensaje que satisface la petición, salvo que la opcion contenga el bit IPC_NOWAIT o que lo interrumpa una señal.

Además el parámetro lenmax da el tamaño máximo del mensaje que se puede extraer (que depende esencialmente del tamaño del espacio reservado para el campo mtext). La petición de extracción de un mensaje más largo supone un fracaso (errno=E2BIG), a menos que la opcion contenga el bit MSG_NOERROR. En este caso, se extrae el mensaje (en su totalidad) y se trunca al tamaño lenmax.

El parámetro tipo indica que mensaje se quiere extraer:

1. Tipo=0 se extrae el primero (cualquiera que sea su tipo);
2. Tipo>0 se extrae el primero del tipo dado;

3. Tipo<0 : se extrae el primer mensaje de tipo menor inferior o igual a |tipo|.

En los casos 1 y 2 el campo mtype del objeto p contiene, a su retorno, el tipo del mensaje recibido.

Por otro lado, el mecanismo de las memorias compartidas es el de compartir zonas físicas de datos, eventualmente estructuradas. Las zonas compartidas por diferentes procesos aparecen como recursos críticos y su utilización se hará a menudo conjuntamente con la de semáforos.

La *ventaja de este mecanismo frente, por ejemplo, a las colas de* mensajes es que no conlleva ninguna copia de los datos y en consecuencia se *muestra más eficiente*.

Las primitivas que se describen a continuación son para trabajar con memorias:

- *shmget*

```
int shmget (clave, tamaño, opcion)
        key_t  clave;           //Clave del segmento
        int tamaño;             //Tamaño del segmento
        int opcion;             //Opción de creación
```

Una llamada de este tipo permite obtener la identificación de un segmento de memoria después de haberlo creado. El tamaño pedido tiene que ser compatible con los límites impuestos por el sistema en caso de creación y con los del segmento si ya existe.

- *shmat*

```
char *shmat (shmid, adr, opcion)
        int shmid;          //Identificación del segmento
        char *adr;          //Dirección de enlace pedida
        int opcion;         //Opción de conexión
```

Una llamada ha esta primitiva realiza la conexión de un segmento de identificación shmid al espacio de direccionamiento del proceso. El valor devuelto por la primitiva es la dirección donde se ha realizado esta conexión de modo efectivo. Es decir, por la que el primer octeto será accesible. El problema *principal que se plantea es el de la elección de la dirección. Ni que* decir tiene que esta dirección no puede:

- *Entrar en conflicto con direcciones ya utilizadas*;
- Impedir el aumento del tamaño de la zona de datos y del de la pila;
- Violar *la forma de las direcciones deducida del tamaño* de las *páginas* (cualesquiera que sea su tamaño, un segmento estará formado de un número entero de páginas).

Por mediación de los parámetros adr y opcion, es posible realizar la elección de esta dirección de conexión con más o menos libertad según sus valores. La primitiva devuelve el valor –1 en caso de fallo.

- *shmdt*

```
int *shmdt (adr)
    char adr;          //Dirección de enlace pedida
```

Efectúa el desenlace del segmento de memoria previamente enlazado (conectado) la dirección adr (mediante una llamada a la primitiva shmat). Por tanto, esta dirección se convierte en una dirección ilegal del proceso.

CAPÍTULO IV

SEGURIDAD EN REDES LAN/WAN

1. Definición de seguridad en una red

La seguridad en redes implica transferir la información a través de ellas en forma segura, y quien juega el rol más importante en estos casos es la criptografía. La criptografía provee seguridad, precisión y confiabilidad. Puede prevenir fraudes en comercios electrónicos y asegura la validez de las transacciones financieras. La criptografía acredita identidad o protege tu anonimato. Y en el futuro, a medida que el comercio y las comunicaciones continúan moviéndose hacia redes de computadoras, la criptografía llegará a ser cada vez más vital.

2. Algoritmos Simétricos

Básicamente un algoritmo criptográfico, también llamado un cifrador, es la función matemática usada para cifrar y descifrar data. Para cifrar un mensaje en texto plano, aplicamos un algoritmo de cifrado al texto plano.

Para descifrar el mensaje cifrado, aplicamos un algoritmo de descifrado al texto cifrado.

Para una real seguridad, todos los modernos algoritmos usan una llave, denotada por k. Esta llave puede tomar uno o varios valores (un número más grande es mejor).

El valor de la llave afecta a las funciones de cifrado y descifrado, así las funciones de cifrado y descifrado llegan a ser como sigue:

Ek(P)=C

Dk(C)=P

Y si la llave de cifrado y descifrado son las mismas entonces se cumple:

Dk(Ek(P))=P

Esto se muestra en la figura 4.1.

Figura 4.1: Cifrado y Descifrado con llaves (Key)

Hay dos formas generales de algoritmos de cifrado basados en llaves, simétrico y con llave pública.

Los algoritmos simétricos son algoritmos donde la llave de cifrado puede ser calculada de la llave de cifrado y viceversa. En muchos sistemas la llave de cifrado y descifrado son las mismas. Estos algoritmos llamados también *algoritmos de llave secreta*, *algoritmos de llave única*, o *algoritmos de una llave*, requieren al emisor y receptor agregar una llave antes de pasar los mensajes. *Esta llave debe de ser mantenida en secreto. La seguridad del* algoritmo simétrico reside en la llave; divulgar la llave significaría que nadie podría cifrar y descifrar mensajes en este sistema.

Por otro lado, los algoritmos simétricos pueden ser divididos en dos categorías. Algunas operan en el texto plano un solo bit al mismo tiempo; son llamados *algoritmos stream o cifradores stream.* Otros operan en el texto plano en grupos de bits. Los grupos de bits son llamados bloques, y los algoritmos son llamados *algoritmos de bloque o cifradores de bloque*.

Para algoritmos que son implementados en computadoras, un típico tamaño de bloque es 64bits. En ambos algoritmos, *stream y de bloque*, la misma llave es usada para cifrado y descifrado

3. Algoritmos de Llave Pública

Los algoritmos de llave pública son diseñados de modo que la llave usada para cifrado es diferente a la llave usada para descifrado. Por lo tanto, la llave de cifrado no puede ser calculada a partir de la llave de descifrado y viceversa. Estos sistemas son llamados de llave pública porque la llave de cifrado puede ser echa pública: un extraño puede usar la llave pública para

cifrar un mensaje, pero solamente con la correspondiente llave de descifrado se puede descifrar el mensaje. En estos sistemas, la llave de cifrado es a veces llamada llave pública y la llave de descifrado es a veces llamada llave privada.

4. Diferencias entre Algoritmos Simétricos y Algoritmos de Llave Pública

La seguridad de cualquier algoritmo es dependiente de la longitud de su llave (k). La mayoría de los algoritmos simétricos tienen una longitud de llave específica. Los algoritmos de llave Pública tienen longitudes de llaves variables. Por ejemplo, DES, con una llave de 56-bit es más seguro que un RSA con una llave de 40-bit, sin embargo, RSA con una llave de 1000-bit es más seguro que DES.

De otro lado, los algoritmos de Llave Pública pueden hacer cosas que los algoritmos simétricos nunca lo harían. Los algoritmos simétricos, sin embargo, es más rápido. Los algoritmos simétricos son ideales para cifrar archivos y canales de comunicación. Los algoritmos de Llave Pública son ideales para llaves cifradas y distribuidas, previendo de autenticación.

5. Elección de Algoritmos Simétricos y Algoritmos de Llave Pública

Como algoritmo simétrico he escogido el método DES (Data Encryption Standard), el cual es un cifrador de bloque, cifra data a bloques de 64-bit y

para ambos, cifrado y descifrado, usa el mismo algoritmo (con excepción de diferencias en el plan de las llaves).

La longitud de la llave es de 56 bits. (La llave es usualmente expresada como un número de 64-bit, pero cada ocho bit es usado para verificación de paridad y es ignorado). La llave puede ser cualquier número de 56 bit y puede ser cambiado en cualquier momento.

En este nivel más simple, el algoritmo es nada más que una combinación de dos técnicas básicas de cifrado: confusión y difusión. La construcción fundamental de bloques de DES es solo una combinación de estas técnicas (una sustitución seguida por una permutación) en el texto, basado en la llave. Esto es conocido como una *vuelta*. DES realiza 16 vueltas, aplica la misma combinación de técnicas en bloque del texto las 16 veces como se muestra en la figura 4.2.

Como algoritmo de llave Pública se ha reemplazado el método RSA por un método práctico que consiste en tener la llave pública en una Base de Datos (lado servidor) que identifica a cada cliente (quien envía la información), esta llave es solo para identificar al Cliente que a su vez su información de identificación está asociada a la data de cabecera la cual también es validada, así una vez identificado al Cliente se procede con el intercambio dinámico de llaves para el cifrado de la data.

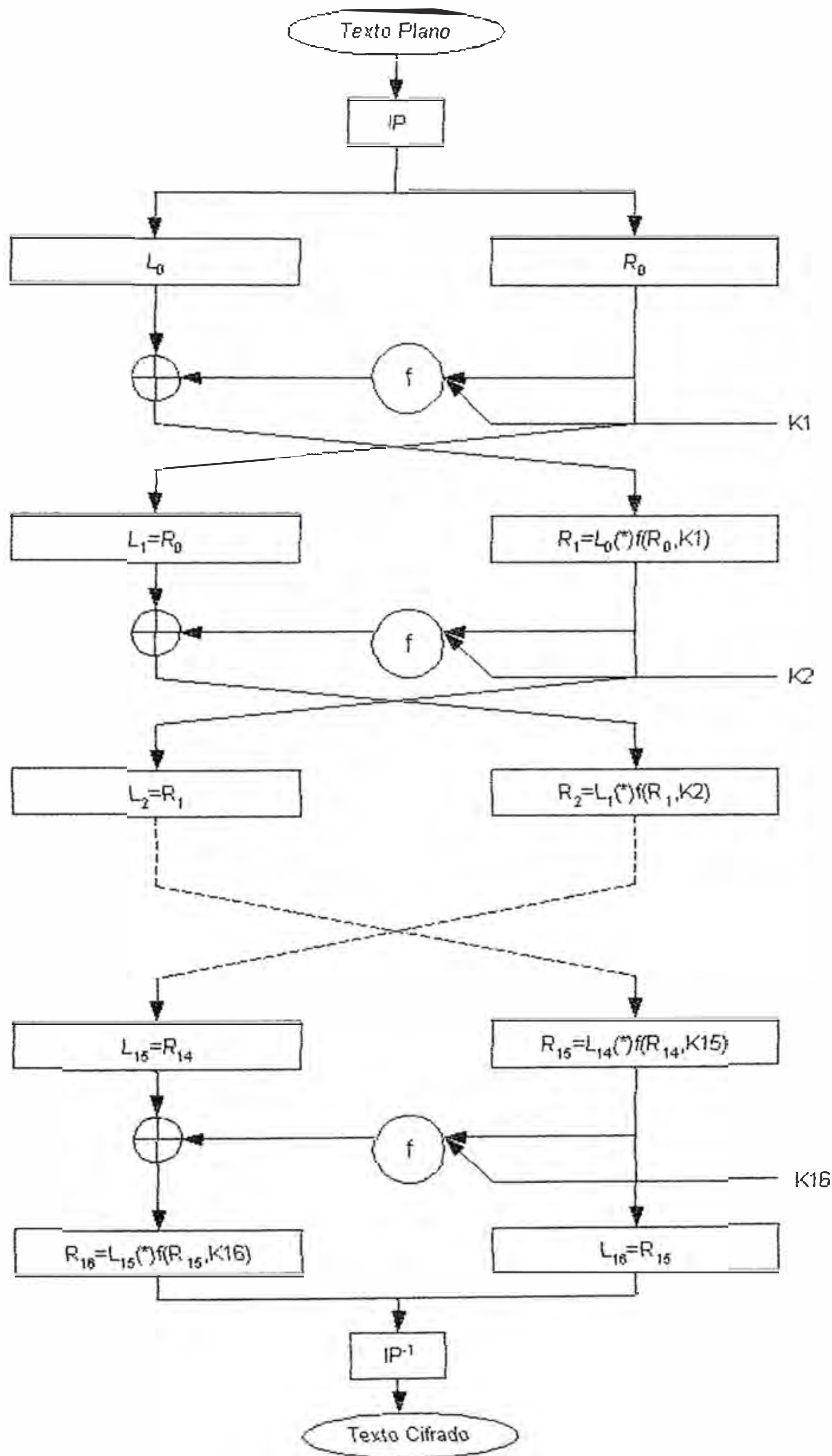


Figura 4.2: Diagrama de flujo del algoritmo DES

Para generar las dos llaves, elegir dos grandes números primos, p y q. Calcular el producto.

$$n = p \times q$$

Luego aleatoriamente elegir una llave de cifrado, e, tal que e y (p-1)x(q-1) *son primos relativamente. Finalmente, usar el algoritmo de Euclides* para calcular la llave de descrifrado, d, tal que:

$$e \times d = 1 \ (\text{mod } (p-1) \times (q-1))$$

en otras palabras,

$$d = e^{-1} \ (\text{mod}(p-1) \times (q-1))$$

Notar que d y n son primos relativamente. Los números e y n son las llaves públicas; el número d es la llave privada. Los dos primos, p y q, ya no son necesitados. Ellos deberían de ser descartados para nunca ser revelados.

CAPÍTULO V

DESARROLLO DE UNA PLATAFORMA BAJO TCP/IP PARA ESTABLECER COMUNICACIÓN TRANSACCIONAL Y POR LOTES, APLICADO A UN SISTEMA DE COMPENSACIÓN BANCARIA AUTOMÁTICA

1. Modelo del Sistema – Esquema Compensación Bancario Planteado

El esquema de compensación bancario surge bajo la necesidad de que actualmente todas las instituciones financieras en especial bancarias tiene la necesidad de realizar transferencias de información en una red LAN y/o WAN para lo cual requieren de una plataforma segura en donde se podrá realizar el intercambio de información con total seguridad y sobretodo confiabilidad en la información transferida. Es así como aplicamos este proyecto de tesis a un Sistema de Compensación Bancario, en el cual la institución bancaria realiza una transferencia de información a través de una red. Esta compensación bancaria se realiza transfiriéndose archivos de datos generados a partir de una extracción a una Base de Datos para poder ser procesados en un servidor principal. Como ejemplo podemos mencionar

la agencia A que la envía la información procesada durante todo el día al servidor bancario B. Para que pueda el servidor B procesar la información sin ningún problema primero debe cerciorarse de que la información recibida corresponde a una agencia autorizada y que además la información se encuentre completa y sobretodo que pueda viajar en forma segura (cifrada) para evitar manipulaciones en el camino. Todo esto nos lleva a enfocar el Sistema de Compensación Bancaria tal como se muestra en la Figura 5.1, en donde se observa clientes en un lado de la red y un servidor en el otro lado.

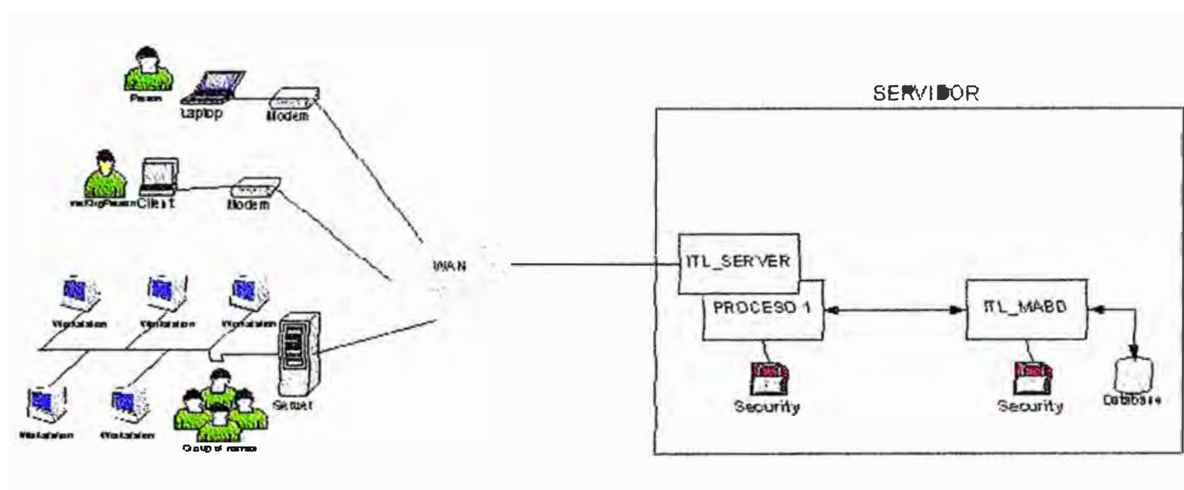


Figura 5.1: Esquema General Compensación Bancaria

A continuación entraremos en detalle técnico del modo de operación del Sistema Servidor y de cada uno de sus procesos requeridos para su funcionamiento como parte del Sistema de Compensación Bancaria y también del modo de operación del Sistema Cliente.

2. Descripción del Sistema Servidor

El Sistema servidor que se ha implementado en Linux consta de varios procesos que se han desarrollado empleando los conceptos de comunicación interprocesos, TCP/IP, Base de Datos y algoritmos de seguridad para hacer posible establecer una comunicación transaccional y por lotes entre HOST a través de una red LAN/WAN de tal modo que se transfiere información en forma segura y con un número de clientes determinado que pueden enviar información en cualquier momento durante las 24 horas del día. El Sistemas Servidor lo hemos divido en dos partes para su mejor explicación.

3. Configuración del Sistema Servidor

Para el desarrollo del presente proyecto hemos considerado tener variables de ambiente definidas, emplear archivos y tablas de configuración, todos con la finalidad de cumplir con las especificaciones requeridas, que se mencionan a continuación:

- Dar portabilidad al sistema.
- Dar una mayor facilidad al usuario al momento de inscribir nuevas instituciones que accederán al Sistema Servidor.
- Dar seguridad e independencia a cada institución inscrita en el Sistema.
- Poder llevar un control de los registros por cada institución conectada al sistema.
- Controlar los accesos al Sistema por institución.

Todos estos puntos mencionados anteriormente se cubren con las siguientes configuraciones:

- Configuración en Archivos

  Los archivos que son configurados en el Servidor son:

  1. bd_file.cfg

     Este archivo contiene los parámetros de conexión a la Base de Datos que estamos empleando, como son el nombre de la Base de Datos, el usuario y el password. El Anexo K muestra el formato de este archivo de configuración.

  2. config.cfg

     Este archivo contiene las llaves de las colas y shared memories que se crean al momento de levantar el sistema. Este punto es muy importante debido a que normalmente en cada equipo con UNIX ya se tienen creadas shared memories y colas para diversos usos y lo que se logra con este archivo es poder configurarlos de acuerdo a lo que se puede tener instalado previamente con el fin de que no hayan conflictos. En el Anexo L se muestra el formato del archivo.

  3. secur_file.cfg

     Este archivo contiene cifrada la llave maestra, la cual es utilizada para cifrar las llaves públicas antes de inscribirlas a la tabla ITLSECUR.

4. proc_tab

Este archivo es muy importante, puesto que aquí se encuentran todos los parámetros iniciales de los diversos programas que hacen posible la transferencia de información en línea. En el Anexo M se muestra como se configura este archivo.

- Configuración en Tablas

Las Tablas configurables son:

1. *ITLSECUR*

Esta tabla contiene los valores que identifican al Cliente y son *usados para su validación al momento de conectarse. En el Anexo* B se muestra el formato de esta tabla.

2. ITLTRFMSG

Esta tabla contiene los campos necesarios para poder llevar una administración de los lotes enviados por los diferentes Clientes y a su vez poder llevar un control al momento de la transferencia. En el *Anexo G se muestra su estructura.*

4. Descripción de Procesos del Sistema Servidor

Son básicamente 4 procesos que se han definido para la transferencia de información en línea, además de programas para cifrado de llaves, inscripción de clientes y monitoreo de transferencia de información. A

continuación se detallan los procesos para la transferencia de información en línea:

ITL_UP : Proceso que tiene como función de levantar los programas que se encargan de la transferencia de información en línea, que básicamente son: el driver TCP/IP (ITL_SERVER), encargado de la comunicación con los clientes y del proceso aplicativo servidor (ITL_MABD) encargado de procesar la información enviada por los clientes. El diagrama de flujo de este proceso está indicado en el Anexo A y consta de las siguientes principales funciones:

- Conexión a Base de Datos

El proceso ITL_UP debe conectarse a la Base de Datos en primera instancia con el objetivo de poder extraer datos del cada cliente configurado para poder ser cargado a memoria. La función para la conexión a la Base de Datos viene dada por:

```
connect_bd(MYSQL *idsql)
{
	char use_bd[30];
	char user[10];
	char pass[10];
	char name_bd[20];

	ENTER(connect_bd);

	memset(user,0,sizeof(user));
	memset(pass,0,sizeof(pass));
	memset(name_bd,0,sizeof(name_bd));
	if(cargar_parametros_bd(user,pass,name_bd)==BAD)
	{
		LOG(DEB,"Error cargando parametros BD");
		return BAD;
	}

	mysql_init(idsql);
	mysql_real_connect(idsql,NULL,user,pass,name_bd,0,NULL,0);
	if(mysql_errno(idsql))
	{
		LOG(DEB,"Error conectandome a la Base de Datos [%s] user [%s]
",mysql_error(idsql),user);
		return BAD;
	}


	LOG(DEB,"Conectado a BD [%s] user [%s] ",name_bd,user);
	return GOOD;
}
```

De aquí podemos apreciar que existe la función **cargar_parametros_bd**, la cual obtiene los valores para conexión a la Base de Datos a través del archivo “bd_file.cfg“.

- Obtener Datos de Archivos de Configuración.

Esta función es muy importante y consiste en obtener los datos de programas que se van a cargar a memoria así como de las direcciones de las colas y shared memories que se van a crear. A continuación se muestra la función para obtener los programas que se van a cargar a memoria, así como la función para obtener las direcciones de las colas y shared memories a crear.

Obtener los programas a Cargar a Memoria

```
     if ( (fd_proc = fopen( proc_table, "r") ) == NULL ){
             LOG(DEB,"Error abriendo %s",proc_table);
             exit(1);
      }
     proc[0] = (struct _proc*)malloc( sizeof(struct _proc ) *
MAX_NUM_PROC );
     for ( i=1 ; i<MAX_NUM_PROC ; i++)     proc[i] = proc[i-1] + 1;
     GetProcTable();

GetProcTable()
{
    int   i, j;
    unsigned char  buff[120], tmp[30];

    ENTER(GetProcTable);

    LOG(DEB,"Levantando Proc_Table a memoria...");
    num_proc = 0;
    while(fgets((char *)buff, 120,fd_proc) != NULL && num_proc <
MAX_NUM_PROC) {
          if ( buff[0] != '#'  &&  buff[0] != ' ' ) {
                proc[num_proc]->proctype = buff[0] ;
                proc[num_proc]->intext   = buff[2] ;
                j= 0; i= 6;
                while ( buff[i] != ';' ) {
                    proc[num_proc]->prog[j++] = (buff[i] == ' ' ?  '\
0':buff[i] );
                    i++ ;
                }
                proc[num_proc]->prog[17] = buff[4]  ;
                j = 0; i++;
                while ( buff[i] != ';' ) {
                    proc[num_proc]->ruta[j++] = (buff[i] == ' ' ?  '\
0':buff[i] );
                    i++ ;
                }
                j = 0; i++;
                while ( (tmp[j++] = buff[i++] ) != ';' ) ;
                tmp[j] = '\0';
                sscanf ( (char *) &tmp[0], "%d", &proc[num_proc]->redid );
                i++;
                while ( buff[++i] != ';' ) ;
                j = 0; i++;
                while ( (proc[num_proc]->param[j] = buff[i++] ) != ';' )
j++;
                proc[num_proc]->param[j] = '\0';
                num_proc++;
          }
    }
    fclose ( fd_proc ) ;
    LOG(DEB,"numero de procesos inscritos : %d", num_proc );
}
```

Obtener direcciones de Colas y Shared Memories

```
/* Obtener Key memories and queues */
memset(config_file,0,sizeof(config_file));
sprintf(config_file,"%s/config.cfg",dirtab);
if ( (fd_keys = fopen( config_file, "r") ) == NULL ){
        LOG(DEB,"Error abriendo %s",config_file);
        exit(1);
}
key_main=get_key_main();
key_001=get_key("KEY_001");
GetQKeys();

int get_key_main()
{
    FILE     *fp;
    unsigned char  buff[120];
    char     *p;
    char     temp[20];
    int            key_main=-1;
    int            j;
    char     *dirtab;
    char     config_file[100];

    ENTER(get_key_main);

    dirtab=getenv("TAB");
    if(dirtab==NULL)
    {
          LOG(DEB,"No puedo obtener variable ambiente TAB");
          return BAD;
    }
    memset(config_file,0,sizeof(config_file));
    sprintf(config_file,"%s/config.cfg",dirtab);

    if((fp=fopen(config_file,"r"))==NULL)
    {
          LOG(DEB,"No pudo abrir archivo %s",config_file);
          return BAD;
    }
    while(fgets((char *)buff, 120,fp) != NULL) {
         if ( strncmp(buff,"KEY_MAIN",8)==0 ) {
                  p=strstr(buff,"0x");
                  if(p!=NULL)
                  {
                        memset(temp,0,sizeof(temp));
                        strcpy(temp,strstr(buff,"0x")+2);
                        for(j=0;j<strlen(temp);j++)
                              if(temp[j]==0x0a) temp[j]=0;
                        key_main=atoi(temp);
                  }
                  else
                        return -1;
         }
    }
    return key_main;
}
```

- Crear Colas y Shared Memories

Una vez obtenido los valores de programas a cargar a memoria, así como de las direcciones de las colas y shared memories, se debe crear las colas y shared memories, empleando para ello las API de comunuicación interproceso, tal como se muestra a continuación:

Crear Colas

```
crea_colas()
{
int qid;
int i;

        ENTER(crea_colas);

        for(i=0;i<numq;i++)
        {
                if((qid=msgget(queue[i],0666|IPC_CREAT))<0)
                {
                        LOG(DEB,"Error %d al obtener COLA %x\
n",errno,queue[i]);
                        exit(-1);
                }
        }
}
```

Crear Shared Memories

```
    /*-----------------------
    * Crear Shared Memory's
    */
     LOG(DEB,"Revisando existencia de SHR_MEM_001......   ");
     if((regshm1=(ST001*)get_shm(key_001, 1))==(ST001*) -1)
             LOG(DEB,"Error Creando Shared Memory 001"),exit(1);

     LOG(DEB,"Revisando existencia de SHR_MEM_MAIN......   ");
     if((regshm_main=(STMAIN*)GetShmid(key_main, 1, &IdShm))==(STMAIN *)
-1)
             LOG(DEB,"Error Creando Shared Memory Main"),exit(1);
```

- Cargar Datos a Memoria

Después de crear las shared memories estamos en la capacidad de cargar los datos más importantes y a la vez más empleados a memoria con el fin de disminuir los tiempos de procesamiento por cada transferencia de información. El procedimiento de carga de datos a memoria viene dado por:

Cargar datos a memoria

```
memset(regshm1,0,sizeof(ST001)*TOT_REG_001);
if(cargar_ids(&idsql,regshm1)==BAD)
{
	LOG(DEB,"Error cargando llaves ");
	shmdt(	(char *)regshm1);
	exit(1);
}

int cargar_ids(MYSQL *idsql, ST001 *regshm1)
{
	MYSQL_RES *result;
	MYSQL_ROW row;
	unsigned int num_fields;
	unsigned long *lengths;
	int i,j;
	char masterkey[17];
	char keyid[9];
	char masterkey_bin[9];
	char row_bin[9];
	char aux[50];
	int len;

	ENTER(carga_ids);

	memset(masterkey,0,sizeof(masterkey));

	mysql_query(idsql,"select id,keyid from itlsecur");
	result = mysql_use_result(idsql);
	num_fields=mysql_num_fields(result);
	j=0;
```

....continua

```
....continuación

        while((row = mysql_fetch_row(result)))
        {
                lengths=mysql_fetch_lengths(result);
                for(i=0;i<num_fields;i++)
                {
                        switch(i)
                        {
                                case 0:
                                        strncpy((regshm1+j)->id,row[i],lengths[i]);
                                        break;
                                case 1:
                                        if(get_masterkey(masterkey)==BAD)
                                        {
                                                LOG(DEB,"Error grave get_materkey");
                                                return BAD;
                                        }
                                        memset(masterkey_bin,0,sizeof(masterkey_bin)
);

                                        memset(row_bin,0,sizeof(row_bin));
                                        hex_to_bin(masterkey_bin,masterkey,16);
                                        hex_to_bin(row_bin,row[i],16);
                                        memset(keyid,0,sizeof(keyid));
                                        decrypdatakey(masterkey_bin,row_bin,keyid,8,
8);

                                        memset(aux,0,sizeof(aux));
                                        bin_to_hexchar(aux,keyid,16);
                                        strncpy((regshm1+j)->keyid,keyid,8);
                                        break;
                        }
                }
                j++;
        }

        if(!mysql_eof(result))
        {
                LOG(DEB,"Error en select  %s",mysql_error(idsql));
                return BAD;
        }
        LOG(DEB,"Fin de Carga Ids");
        return GOOD;
}
```

En esta carga de datos a memoria, estamos almacenando en la memoria principal los ID (identificadores de Clientes) con sus respectivas llaves públicas de cifrado de información. Hay que tener en cuenta que la llave pública de cifrado del cliente que estamos cargando a memoria es la llave del Cliente que teníamos en la Tabla ITLSECUR descifrado bajo la llave principal que esta almacenada en el archivo cifrado secur_file.cfg.

- Levantar los procesos

Para levantar los procesos se tiene que cargar los programas a la memoria, esto se realiza ejecutando los programas tal como se muestra a continuación:

Levantar Procesos

```
hijos()
{
     char nombre[30];
     int  num;

     ENTER(hijos);

     for ( num=0 ; num<num_proc ; num++ )   {
         nombre[0] = '\0';
         if (  proc[num]->proctype == 'a' ) {
                  if ( fork() == 0 )   {
                  LOG(DEB,"levantando proceso [%s]\n", proc[num]-
>prog);
                  fprintf(stdout,"\nlevantando proceso [%s]\n",
proc[num]->prog);
                                  fflush(stdout);
                        exe_process( regshm_main, num, (char*)0 );
                   LOG(DEB,"UP, FRACASO LLAMADA A EXE_PROCESS\n");
                        exit( 0);
                  }
         }
     }

}
```

```
exe_process(ptr, slot, cad )
    STMAIN *ptr;
    int     slot;
    char    cad[];
{
    char nombre[80], prm[80], ruta[80], *param[8];
    int  i, l, num, posp;

    if ( cad!=NULL && (posp=prmonline(cad)) != -1 )
        strcpy ( prm, &cad[posp+1] );
    else
        strcpy ( prm, (ptr+slot)->param);

    strcpy ( nombre, (ptr+slot)->prog );

    i = num = 0;
    l = strlen ( prm );
    param[num++] = nombre;
    while ( i < l ) {
          while ( prm[i] == ' ' ) i++;
          param[num++] = &prm[i];
          while ( prm[i] != ' '&& prm[i] != '\0' ) i++;
          prm[i++] = 0;
    }
    param[num] = 0;
    sprintf( ruta, "%s/%s", (ptr+slot)->ruta, nombre );
    return ( execv ( ruta, param ) );
}
```

ITL_DOWN: Proceso que tiene como función bajar los procesos ITL_SERVER e ITL_MABD. El diagrama de flujo de este proceso está indicado en el Anexo C y las principales funciones se describen a continuación.

- Obtener las direcciones de Colas y Shared Memories

Este paso consiste en obtener las direcciones de las colas y shared memories con las cuales han sido creadas con el fin de poder destruirlas en los siguientes pasos. Las funciones empleadas para este caso son:

Funciones para Obtener las Shared Memories

```
key_main=get_key_main();
key_001=get_key("KEY_001");

if ( ( shmidM = shmget(key_main, 1, 0666) ) == -1)
       printf("\nError tomando la memoria principal"),exit(1);

if ( ( shmid1 = shmget(key_001, 1, 0666) ) == -1)
       printf("\nError tomando la memoria 001"),exit(1);
```

Funciones para Obtener las Colas

```
numq=0;
GetQKeys();

for(i=0;i<numq;i++)
{
       if((qid=msgget(queue[i],0666))<0)
       {
              LOG(DEB,"Error %d al obtener COLA %x\
n",errno,queue[i]);
              exit(-1);
       }
}
```

- Eliminar los procesos activos

Con esto tenemos que desactivar los procesos, es decir, parar la ejecución de los programas en memoria. Esto lo logramos enviando señales a cada proceso y el método se muestra a continuación:

Eliminar los procesos activos

```
      for ( i=MAX_NUM_PROC-1 ; i > -1 ; i-- ) {
                 printf("\ndown programa %s \t
Estatus:%c",(reg_shm_main+i)->prog,(reg_shm_main + i)->status) ;
             if ( (reg_shm_main + i)->status == 'a' )   {
                  pid = ( reg_shm_main + i )->pid ;
             if(strcmp((reg_shm_main+i)->prog,"itl_server")==0)
                      signo=SIGKILL ;
             else if(strcmp((reg_shm_main+i)->prog,"itl_mabd")==0)
                      signo=SIGKILL;
             else  signo=SIGTERM;
             printf("\nprograma:%s\tpid:%d",(reg_shm_main+i)-
>prog,pid);
                  if ( kill ( pid, signo ) == -1 )   {
                      if ( errno == 3 )
                          printf("\nproceso: %s. No Owner ni
SuperUser",
                                 ( reg_shm_main +i)-> prog  );
                      else
                          printf("\nproceso: %s. No pude matarlo",
                                 ( reg_shm_main +i)-> prog  );
                      killok = 0;
                  }
                  else
                      printf("\nproceso: %s. muerto", ( reg_shm_main
+i)-> prog  );
             }
      }
```

- Eliminar las colas y shared memories

Por último eliminamos las colas y shared memories que fueron creadas para el proceso de transferencia de información. Las funciones empleadas para destruir las colas y shared memories son:

Destruir las shared memories

`if ( shmctl ( shmidM, IPC_RMID, shmstat ) == -1 )`
`      LOG(DEB,"Error destruyendo la memoria principal"),exit(1);`
`if( shmctl( shmid1, IPC_RMID,shmstat) == -1)`
`      LOG(DEB,"Error al destruir shm_001");`

Destruir las colas

`for(i=0;i<numq;i++)`
`{`
`      if(msgctl(qid[i],IPC_RMID,0)<0)`
`      {`
`            LOG(DEB,"Error %d al eliminar COLA %d\`
`n",qid[i]);`
`            exit(-1);`
`      }`
`}`

ITL_SERVER*:* Es un driver desarrollado en lenguaje C ANSI y que emplea las primitivas del protocolo TCP/IP y tiene como función atender las llamadas realizadas por los clientes y enviarlas al proceso aplicativo servidor ITL_MABD. Este driver trabaja indefinidamente atendiendo las llamadas de los clientes. Para este análisis partimos que el Servidor tiene registrado solo 10 instituciones como máximo que pueden ser clientes y por tanto estar conectados al Servidor al mismo tiempo y enviarle información en cualquier momento y en forma concurrente si fuera el caso, sin presentar problema alguno al driver TCP. El diagrama de flujo de este proceso está indicado en el Anexo D y a continuación se detallan las principales funciones realizadas por el proceso ITL_SERVER.

- Obtener Cola y Tipo de Mensaje para escribir en la Cola

El proceso ITL_SERVER así como driver necesita enviar la información recibida a otro proceso para que la valide para lo cual emplea cola de intercambio de mensajes. En esta cola se deben escribir los mensajes y con un identificador para que el proceso que lea la cola sepa de quien viene el mensaje y cual es el propósito del mismo. La manera como se obtiene la cola y como se define el tipo de mensaje que se va escribir en la cola viene dado por las funciones primitivas de comunicación interproceso y han sido empleadas tal como se muestra:

Obtener Cola y Tipo de Mensaje

```
queue=GetQueue("QUEUE_SERVER");

if((qid=msgget(queue,0666|IPC_CREAT))<0)
{
        fprintf(stdout,"\nError al obtener cola %x\n",queue);
        exit(-1);
}


if((tipo=get_q_mtype("itl_server"))<0)
        printf("\nitl_server:Error en obtener Tipo de Mensaje a
Enviar a itl_mabd");
```

Se ve claramente que se emplea la primitiva de IPC (Comunicación InterProceso) para pegarse a la cola creada por el proceso ITL_UP y usarla para intercambio de información con el proceso ITL_MABD. También vemos que definimos un tipo de datos para escribir en la cola, para que de esta forma en ampliaciones futuros tengamos ordenados los tipos de mensajes en la cola de acuerdo al origen del mensaje.

- Inscribir proceso en la shared memory principal

Al inscribir el proceso en la shared memory principal podemos llevar un control sobre el proceso activo, es decir, que el proceso puede ser monitoreado en su ejecución y podrá ser notificada su inactividad a través de la shared memory. En la presente tesis nuestro principal objetivo es transferir la información a través de la red en forma segura y llevar un control de la transferencia, sin embargo en relación al punto de monitoreo de procesos no nos hemos evocado tanto, pero de igual modo este tema puede ser abordado en la mejoría para el control de procesos activos y dar alarmas en caso de inactividad de procesos.

- Crear el socket para escuchar llamadas de Clientes

Una vez inscrito el proceso en la memoria e identificada las colas de trabajo para el intercambio de mensajes con el proceso ITL_MABD, debemos crear el socket para escuchar las llamadas de los clientes. Este socket que crearemos atenderá como máximo a 10 clientes, que a su vez es el máximo numero permitido por la función listen para escuchar de un puerto. Entonces planteamos que si se quiere atender más clientes (más de 10) debemos multiplicar nuestro proceso ITL_SERVER apuntando además a otros números de puerto para de esta forma atender cuantos clientes se desee configurar. Hay que tener en cuenta que este servidor está orientado a atender clientes que a su vez son instituciones que tiene una red y que cada una de ellas puede transferir información de cuantas estaciones estén

colgadas a su red, es como se muestra en la Figura 5.1. A continuación se muestra como crear un socket empleando las primitivas de IPC del LINUX.

Crear socket para atender llamadas de Clientes

```
if((s=socket(AF_INET,SOCK_STREAM,0))==-1)
 {
     r=close(s);
     LOG(DEB,"error creando socket..");
   LOG(DEB,"cierro close(s):%d-errno:%d",r,errno);
}

memset((char *)&sin,0,sizeof(sin));
sin.sin_family=AF_INET;
sin.sin_port=htons(port);
sin.sin_addr.s_addr=htonl(INADDR_ANY);

if(setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) != 0) {
       LOG(DEB,"setsockopt address error errno(%d) [%s]",
               errno, strerror(errno));
       return(-4);
}

if(bind(s,(struct sockaddr *)&sin,sizeof(sin))==-1){
       r=close(s);
       LOG(DEB,"Error en bind..%d",errno);
       LOG(DEB,"cierro close(s):%d-errno:%d",r,errno);
}

if(listen(s,10)==-1){
       r=close(s);
       LOG(DEB,"Error listen");
       LOG(DEB,"cierro close(s):%d-errno:%d",r,errno);
}
```

- Esperar llamada del cliente

Una vez creado el socket para escuchar las llamadas de los clientes, se debe esperar por la llamada y por cada llamada entrante generar un proceso que se dedique a atender la llamada y de esta forma evitar una congestión en el puerto que atiende a clientes. Para poder crear procesos que se dediquen exclusivamente a atender las llamadas entrantes hemos necesitado usar la función ***fork*** del sistema operativo y el modo como lo hemos empleado está resumido como se muestra:

Esperar llamada del Cliente

```
	if((child_pid = fork()) < 0)
		LOG(DEB,"can't fork a new children process errno(%d)",
errno);
	else if(child_pid == 0) {
		LOG(DEB,"llamando a tcp_child [%d] new_sock[%d] pid_dad [%d]
pid_child [%d]",sock,newsock,pid_dad,child_pid);
		close(sock);
		tcp_child(newsock,ipstr,progid_h,serv,binacq);
		exit(0);
	}
	close(newsock);
```

Aquí podemos apreciar que realizamos un fork ante una llamada al puerto y aquí llamamos a la función tcp_child la cual es la encargada de generar dos procesos para atender a la llamada entrante y así liberar al proceso principal ITL_SERVER para que continúe atendiendo más llamadas y no se quede colgado todo el tiempo que dure esta.

Cabe mencionar que el proceso ITL_SERVER también consta de una lógica para validar las direcciones IP de los clientes y de esta forma rechazar todas las llamadas de clientes no registrados en la Base de Datos ITLSECUR.

ITL_MABD: Proceso encargado de atender los mensajes enviados por los clientes. Este proceso recibe la data de los clientes en forma cifrada y debe validar sus datos, registrarlos y responder al cliente como notificación de una buena recepción. Para lograr esto es necesario que este proceso emplee funciones de seguridad para cifrado y descifrado. Como planteamos en un

inicio estamos aplicando el algoritmo síncrono DES el cual tiene sus valores de las tablas seleccionados y mostrados en el Anexo E.

Como acabamos de mencionar este proceso debe emplear funciones de seguridad para descifrado y cifrado de información pero además debe acceder a la base de datos, registrar la información enviada y llevar un control de la transferencia de información. Todas estas actividades se muestran definidas en el diagrama de Flujo del proceso en el Anexo F y a continuación se describe cada función del proceso.

- Conexión a la Base de Datos

  Como el proceso ITL_MABD debe llevar un control de la transferencia de información, decidimos para el desarrollo del proceso realizarlo a través de la tabla ITLTRFMSG, cuya estructura está definida en el Anexo G y que nos sirve de control para cada registro enviado por los clientes y de esta forma ayudar al sincronismo en casos de caídas de línea. En esta tabla iremos grabando cada registro enviado, el ID del Cliente, el nombre del archivo a transferir así como el tamaño del mismo. Es por este motivo que requerimos que el proceso ITL_MABD tenga conexión a la Base de Datos.

- Obtener Cola y Tipos de Mensaje de Cola

  Para intercambiar los mensajes con el proceso ITL_SERVER necesitamos que el proceso ITL_MABD interactúe con él a través de las colas para comunicación interprocesos. Debido a esto obtenemos los ID

de las colas y los tipos de mensajes de colas para poder realizar el intercambio de información. A continuación se muestran las funciones empleadas para satisfacer estos requerimientos.

Obtener Colas y Tipos de Mensajes de Cola

```
        queue=GetQueue("QUEUE_MABD");

        if((qid=msgget(queue,0666))<0) {
                LOG(DEB,"Error obteniendo COLA :%x
errno:%d",queue,errno);
                exit(1);
        }
        if((tipo=get_q_mtype("itl_mabd"))<0){
                LOG(DEB,"Error en obtener tipo de mensaje");
                el_fin();
        }
        queuersp=GetQueue("QUEUE_SERVER");

        if((qidrsp=msgget(queuersp,0666))<0) {
                LOG(DEB,"Error obteniendo COLA :%x
errno:%d",queuersp,errno);
                exit(1);
        }
        if((tiporsp=get_q_mtype("itl_mabd"))<0){
                LOG(DEB,"Error en obtener tipo de mensaje");
                el_fin();
        }
```

Notar que estamos usando dos colas para el intercambio de mensajes con el proceso ITL_SERVER. Una cola la estamos empleando para las transacciones enviadas por los clientes, mientras que la otra la empleamos para las respuestas enviadas a los clientes. El motivo de emplear dos colas es debido a que estimamos el tiempo de máximo trafico en el Servidor en donde todos los clientes envíen información en forma simultánea entonces requerimos una liberación en las colas puyes estas al llenar su tope máximo comenzarán a perderse los mensajes por más control que le podamos poner

a los procesos y deberá requerirse una sincronización por parte del cliente más a menudo.

- Pegarse a la shared memory

El proceso ITL_SERVER necesita pegarse a la shared memory donde se encuentran los datos del Cliente, especialmente para obtener la llave de cifrado de intercambio de información.

- Esperar mensajes del Cliente

El proceso ITL_MABD debe esperar los mensajes del cliente y esto se realiza leyendo de la cola de intercambio de mensajes provenientes del proceso ITL_SERVER, y la manera como definimos esta espera de mensaje viene dada por la función msgrcv definida en los capítulos previos.

- Validar información

Una vez que un mensaje es leído, tenemos que validar la información proveniente del Cliente para lo cual primero necesitamos descifrar el mensaje y validar su contenido de acuerdo al cliente que la envía. Para realizar este proceso tenemos dos noveles bien marcados:

1. Validar la información en sí, es decir, usando los algoritmos de cifrado de llave simétrica y llave pública se debe verificar que la información enviada por cada cliente es válida. Para esta operación existe un intercambio de

llaves entre el Cliente y el Servidor, lo cual se explicará en detalle en el ítem 7 de este capítulo.

2. Se lleva un control de la secuencia de la información enviada por el Cliente, así como del tamaño total de la información enviada, para lo cual se cuenta con la tabla ITLTRFMSG. En el Anexo H se muestran las 3 funciones principales que se han creado para acceder a la tabla ITLTRFMSG y llevar el control de la transferencia de información.

Cabe mencionar que como se ha establecido un pseudo protocolo para el intercambio de mensajes con los Clientes, entonces la información proveniente del Cliente tiene un HEADER con un formato definido tanto para el inicio de la transferencia establecida por el Cliente como para el resto de información enviada. En el Anexo I se muestra los formatos de mensajes definidos para el intercambio de información así como los códigos de respuesta que se han definido para llevar un control de la situación en que se encuentra cada proceso (lado Cliente y lado Servidor).

A continuación se explica en detalle la validación de la información enviada por el Cliente teniendo en cuenta las definiciones establecidas en los Anexos H e I respectivamente.

Primero, cuando el cliente quiere enviar una información primero envía una data de requerimiento de envío de información, en donde a la vez envía la llave privada con la cual enviará la información, pero cifrada bajo la llave pública del Servidor. El algoritmo se muestra a continuación:

```
if((coderr=read_header(&idsql,mensaje,bin,key,&lendata,filename,
&filelen,&nrototreg,&nroreg,&status,hdr_clear))!=GOOD)
{
	get_values_rsp(hdr_clear,fileid,&nroregtmp,fecha_hora);
	LOG(DEB,"Error BIN [%s] No registrado
coderr[%d]",bin,coderr);
}
else
{
	get_values_rsp(hdr_clear,fileid,&nroreg,fecha_hora);
	if(status==ST_ENVDAT && lendata<=0)
	{
		LOG(DEB,"Error lendata [%d]",lendata);
		coderr=ST_ERRRCP;
	}
	else
	switch(status)
	{
		case 1:/*Inicio Envio*/
		/*verificar si ya existe*/
		if((coderr=find_dataheader(idsql,bin,filename,fileid,
fecha_hora,&nroreg_bd,&status_bd,&nrototreg))!=GOOD)
		{
			LOG(DEB,"Envio no Registrado");
			if((coderr=put_dataheader(&idsql,bin,status,file
name,filelen,nrototreg,nroreg,fileid,fecha_hora))!=GOOD)
			{
				LOG(DEB,"Error leyendo datos adicionales
del Header");
			}
		}
		else
		{
			/*reenvio */
			LOG(DEB,"Reenvio de Archivo
nroreg[%d]",nroreg_bd);
			nroreg=nroreg_bd;
			coderr = ST_ERRDUP;
		}
				break;
...
...
...
```

Según como se muestra la función read_header, es la función que se encarga de leer el header completo del mensaje recibido por el cliente, validar su ID e identificar la llave privada del Cliente, una vez que pase esta validación se procede a extraer los datos restantes del header como son el *nombre del archivo a transferir, su longitud y el número de archivos a ser*

enviados para esta transferencia. Luego la función find_dataheader descrita anteriormente tiene como función detectar si la transferencia se realiza por primera vez o es un reenvío de información. Para los casos de reenvío de información podemos asumir que ha existido una caída de línea y el cliente quiere enviar la información que le quedó pendiente y por este motivo los *reenvíos son rechazados con el número de registro enviado por el cliente y* registrado en la tabla de control ITLTRFMSG. Para los casos de ser un envío de información por primera vez registramos los datos completos de la información que se recibirá del cliente a través de la función put_dataheader, tal como se muestra en el cuadro anterior.

Segundo, una vez recibida la primera trama del cliente, vamos a tener que seguir llevando un control exhaustivo de cada registro recibido. Esto lo logramos empleando las funciones de acceso a Base de Datos descritas anteriormente. El algoritmo se muestra a continuación:

```
case 2:/*Envio secuencial */
case 3:/*Fin de envio */
      memset(filename,0,sizeof(filename));
      if((coderr=find_dataheader(idsql,bin,filename,fileid,fecha_
hora,&nroreg_bd,&status_bd,&nrototreg))!=GOOD)
      {
            LOG(DEB,"Envio no Registrado");
      }
      else
      {
            if(nroreg == nroreg_bd + 1)
            {
                  if((coderr=update_dataheader(&idsql,bin,status,n
roreg,fileid,fecha_hora))!=GOOD)
                  {
                        LOG(DEB,"Error actualizando datos
adicionales del Header");
                        coderr=ST_ERRUPD;
                  }
                  else
                        coderr=write_data(key,mensaje,filename,len
,lendata,fecha_hora);
                        if(coderr == ST_OK)
                        {
                              if(nroreg == nrototreg && status==3)
                              {
                                    LOG(DEB,"Termino transferencia
de Archivo %s SATISFACTORIAMENTE !!! ",filename);
                              }
                        }
            }
            else
            {
                  LOG(DEB,"Nro de registro recibido invalido [%d]
",nroreg);
                  coderr=ST_ERRNREG;
            }
      }
break;
default:
      LOG(DEB,"Error Status NO VALIDO [%d]",status);
      coderr=ST_ERRSTS;
      break;
}

}
```

Se observa que la función find_dataheader, la cual es empleada para verificar si la información enviada ya ha sido registrada, en caso pase esta

verificación continúa la verificación del número de registro enviado. Con esto llevamos un control para la sincronización en casos de caídas de línea.

- Grabar la información

En el cuadro anterior también vemos la función write_data, que es la que se encarga de grabar la información enviada por el cliente en el directorio $HOME/data y además de grabar esta información sin el cifrado de la transferencia, es decir, en clear text, tal como el archivo está en el cliente. El detalle de esta función se muestra en el siguiente cuadro.

```
int write_data(char *key,char *mensaje,char *filename,int len,
int lendata,char *fecha_hora)
{
	int	dout;
	int	lentot;
	int	coderr;
	char	*dirdata;
	char	fileout[100];
	char	data[MAXDATA+1];
	char	data_clear[MAXDATA+1];

	ENTER(write_data);

	lentot = len - LENHDRSEQ - LENBIN;
	memset(data,0,sizeof(data));
	memcpy(data,mensaje+LENHDRSEQ,lentot);
	memset(data_clear,0,sizeof(data_clear));
	decrypdatakey(key,data,data_clear,lendata,lentot);
	dirdata=getenv("DATA");
	if(dirdata==NULL)
	{
		LOG(DEB,"Error obteniendo variable ambiente");
		coderr=ST_ERRGEN;
	}
	memset(fileout,0,sizeof(fileout));
	sprintf(fileout,"%s/%s_%s",dirdata,filename,fecha_hora);
	if((dout=open(fileout,O_WRONLY|O_CREAT|
O_APPEND,0660))<0)
	{
		LOG(DEB,"No puedo generar archivo errno
[%d]",errno);
		coderr=ST_OPENFILE;
	}
	if(write(dout,data_clear,lendata)!=lendata)
	{
		LOG(DEB,"Error al grabar en archivo [%s] errno
[%d]",fileout,errno);
		coderr=ST_WRFILE;
	}
	else
	{
		coderr=ST_OK;
	}
	close(dout);

	return coderr;

}
```

- Responder

Una vez la información haya sido recibida por el cliente siempre debe haber una respuesta, ya sea por notificar un error o por notificar que la data enviada ha sido enviada satisfactoriamente, todo esto cumpliendo con los códigos de respuesta establecidos para esta transferencia de información. Cabe mencionar que esta respuesta tiene el mismo formato que el HEADER de información enviada por el cliente y es solo el HEADER el que se responde, no hay data adicional, pues en el análisis no se ha visto necesario emplearla.

5. Sistema Cliente

El Sistema cliente se ha desarrollado para trabajar sobre plataformas Windows 95, Windows 98, Windows 2000 e incluso sobre Windows NT y es quien realiza el requerimiento al Servidor para enviarle información, por lo tanto, es el cliente quien inicia la comunicación y es el quien tiene la información bancaria que se desea transferir al Servidor en forma rápida y segura. Esta información bancaria a transferir del Cliente al Servidor debe ser un archivo el cual fue generado de la extracción de datos a una Base de Datos que se encuentre en cualquier plataforma y haya sido enviada a la PC-Cliente por cualquier medio de transporte de información (disquetes, CD'd, FTP, etc.). Para cumplir con los requisitos en el sistema cliente se ha desarrollado un programa Visual (Visual Basic con funciones realizadas en Visual C) que cumple con las siguientes tareas:

- Cargar una configuración específica por Cliente.
- Tener la facilidad de escoger el archivo a transferir.
- Poder iniciar la comunicación con el Servidor.
- Cifrar la información del archivo a transferir y enviarla al Servidor.
- Recibir la Respuesta del Servidor e interpretarla.
- Guardar evidencia de la transferencia de información.
- Tener la Capacidad de recuperación frente a condiciones de excepción, tal como caídas de línea.

A continuación se describirá como se realiza cada tarea:

- Cargar una configuración específica por Cliente.

Se han definido dos archivos de seguridad en el Cliente para su configuración, la cual debe ser única por cliente.

Primero, el archivo de configuración "itlconfig.ini" lleva los valores asociados al Cliente, tal como su ID de identificación, su descripción y la llave que se ha designado para la transferencia. Al final del archivo se tiene un valor de seguridad del archivo frente a modificaciones que se realicen en forma manual, es decir, cualquier modificación sobre el archivo será detectada por el aplicativo y no habría transferencia de información. En el Anexo J se muestra la estructura del archivo con un ejemplo.

Segundo, el archivo "secur_file.cfg" contiene la masterkey bajo la cual fue cifrada la llave que se encuentra en el archivo "itlconfig.ini". Este archivo "secur_file.cfg" se encuentra cifrado.

- Tener la facilidad de escoger el archivo a transferir.

Para este punto se ha dispuesto de facilidades que otorga el Visual Basic con su entorno Visual, motivo por el cual fue escogido para desarrollar el programa que será visto por el Cliente – Operador.

- Poder iniciar la comunicación con el Servidor.

Para iniciar la comunicación con el Servidor se ha desarrollado funciones con empleando las API's TCP/IP en Visual C y la DLL generada está siendo llamada desde el programa del Operador (Visual Basic). La función para la comunicación con el Servidor realizada en Visual C se detalla en el próximo cuadro, en donde se puede apreciar que para establecer la comunicación con el Servidor en una primera etapa solo se necesita del IP del Servidor y del Número de Puerto de escucha del Servidor los cuales están representados en la función como REMOTEHOST y REMOTEPORT respectivamente.

Cabe mencionar que el Servidor a este nivel valida el IP del Cliente, motivo por el cual si el Cliente no está inscrito en el Servidor toda conexión de este será rechazada.

```
init_client(int REMOTEPORT,char REMOTEHOST[10]){
int veces=0;

inicia_again:;

       if( (sc=socket(AF_INET,SOCK_STREAM,0)) < 0)
                  {printf("\nError en obtener socket..");return -
1;}

       /* INICIALIZAR ESTRUCTURA */
       memset((char *)&sin,0,sizeof(sin));
       sin.sin_family=AF_INET;
       sin.sin_port=htons(REMOTEPORT);
       if((hp=gethostbyname(REMOTEHOST))==NULL)
         {
              printf("server:cierro close(sc):%d-errno:%d\
n",close(sc),errno);
            printf("server:Error gethostbyname\n");
            return -1;
         }

       memcpy((char *)&sin.sin_addr,(char *)hp->h_addr,hp-
>h_length);

       if(connect(sc,(struct sockaddr *)&sin,sizeof(sin))<0)
         {
            if(errno==146 && veces<=10){
                  printf("cliente:Inicio nuevamente\n");
                  printf("cliente:cierro close(sc):%d-errno:%d\
n",close(sc),errno);
                  tiempo(0.7);
                  veces++;
                  goto inicia_again;
            }

            printf("cliente:Error en Connect ERRNO:%d\n",errno);
            return -1;
         }

return 0;

}
```

- Cifrar la información del archivo a transferir y enviarla al Servidor.

Una vez obtenido el archivo a transferir desde la selección realizada por el Operador y establecida la conexión al *Servidor*, se procede a cifrar la información del archivo empleando las mismas funciones para cifrado y *descifrado desarrolladas en el Servidor para el proceso ITL_MABD y que*

fueron descritas anteriormente. Estas funciones las hemos reflejado también en Visual C y la DLL está siendo empleada por el programa en Visual Basic (Aplicativo del Cliente-Operador).

- Recibir la Respuesta del Servidor e interpretarla.

La información recibida del Servidor consta de un HEADER donde informa sobre la situación actual del archivo transferido, en la cual describe si fue la información enviada fue registrada satisfactoriamente por el Servidor o si fue rechazada y para este último caso da el detalle del rechazo. También para descifrar la información recibida al igual como para enviarla estamos empleando las funciones desarrolladas en el Servidor para el proceso ITL_MABD migradas ahora al Visual C.

- Guardar evidencia de la transferencia de información.

Toda información que es enviada, así como la información recibida la estamos registrando en un archivo de control con el motivo de poder ser empleado para el caso de caídas de línea y también para control de la transferencia de información.

- Tener la Capacidad de recuperación frente a condiciones de excepción, tal como caídas de línea.

La recuperación frente a caídas de línea está relacionada al punto anterior, pues al contar con un archivo de control estamos en la capacidad de saber en que número de registro enviado se cayó la línea y

por ende a partir de este número debemos reiniciar la transferencia y evitar de esta forma transferir nuevamente todo el archivo.

6. Consideraciones para caídas de línea

En toda transferencia de información a través de medios electrónicos existe la posibilidad de que se pierda la comunicación en cualquier momento, por uno u otro motivo, a esto le llamamos caídas de línea. Estas caídas de línea vienen a ser un factor muy importante y el cual debe ser considerado cuando se realizan los drivers de comunicación.

Para nuestro caso, estamos considerando que ante una caída de línea, primero debe haber una conversación entre el driver cliente y el servidor a penas se reestablezca la comunicación y decidan a partir de que punto se debe continuar con la transmisión de la información. Para hacer posible esto, se manejan tramas comparables cada cierto período definido, de esta forma el cliente va guardar siempre la última trama comparable enviada al servidor antes de una caída de línea, entonces cuando se reestablezca la comunicación la enviará al servidor, en caso el servidor no la encuentre, deberá reenviarla; y en el caso que si la encuentre deberá enviar la información a partir de donde se quedo.

7. Algoritmos usado para Autenticación y Cifrado de Información

Los algoritmos de autenticación del cliente así como del cifrado de información son muy importantes para realizar una transferencia en forma segura través de una red.

7.1 Autenticación

Con la autenticación verificamos que el Cliente es permitido para enviar información al Servidor. Para esto estamos empleando:

1. El IP del Cliente que es registrado en la Base de Datos del Servidor en la tabla ITLSECUR y es verificado por cada requerimiento de conexión al Servidor.
2. Una llave pública y única por cliente que es registrada y administrada por el Servidor en la tabla ITLSECUR.
3. Una llave pública del Servidor que se encuentra registrada en el Cliente en el archivo itlsecur.cfg.

Este esquema de funcionamiento es valido para autenticar tanto en redes locales como en redes públicas tal como se muestra en la Figura 5.2

7.2 Cifrado de Información

La información que se transfiere entre el Cliente y el Servidor siempre va cifrada empleando el algoritmo simétrico DES y empleando la lógica de Llave Pública. Esto es:

Primero, el Cliente genera en forma aleatoria su llave privada y la cifra bajo la llave pública del Servidor y le envía al Servidor esta información.

Segundo, si el Servidor acepta su llave privada, entonces le realiza un requerimiento para que el cliente envíe la información, para lo cual emplea la llave pública del Cliente.

Tercero, el Cliente envía la información que requiere cifrada bajo su llave privada y este paso se repite esta terminar de enviar toda su información. Este procedimiento se ilustra en la Figura 5.3

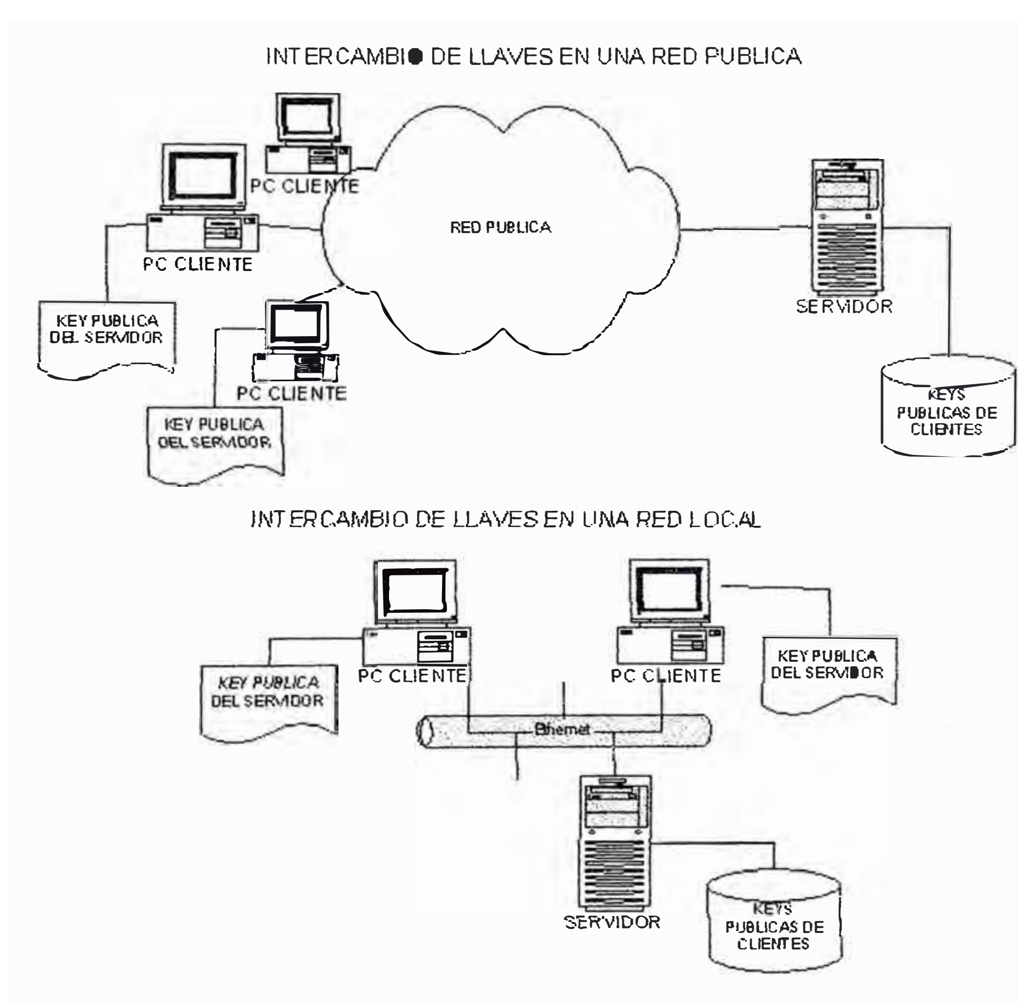


Figura 5.2

Como se muestra en la Figura 5.2 el intercambio de llaves basados en algoritmos simétricos y asimétricos funciona de igual modo en redes públicas (WAN) como en redes locales.

PROCEDIMIENTO DE CIFRADO - LADO CLIENTE

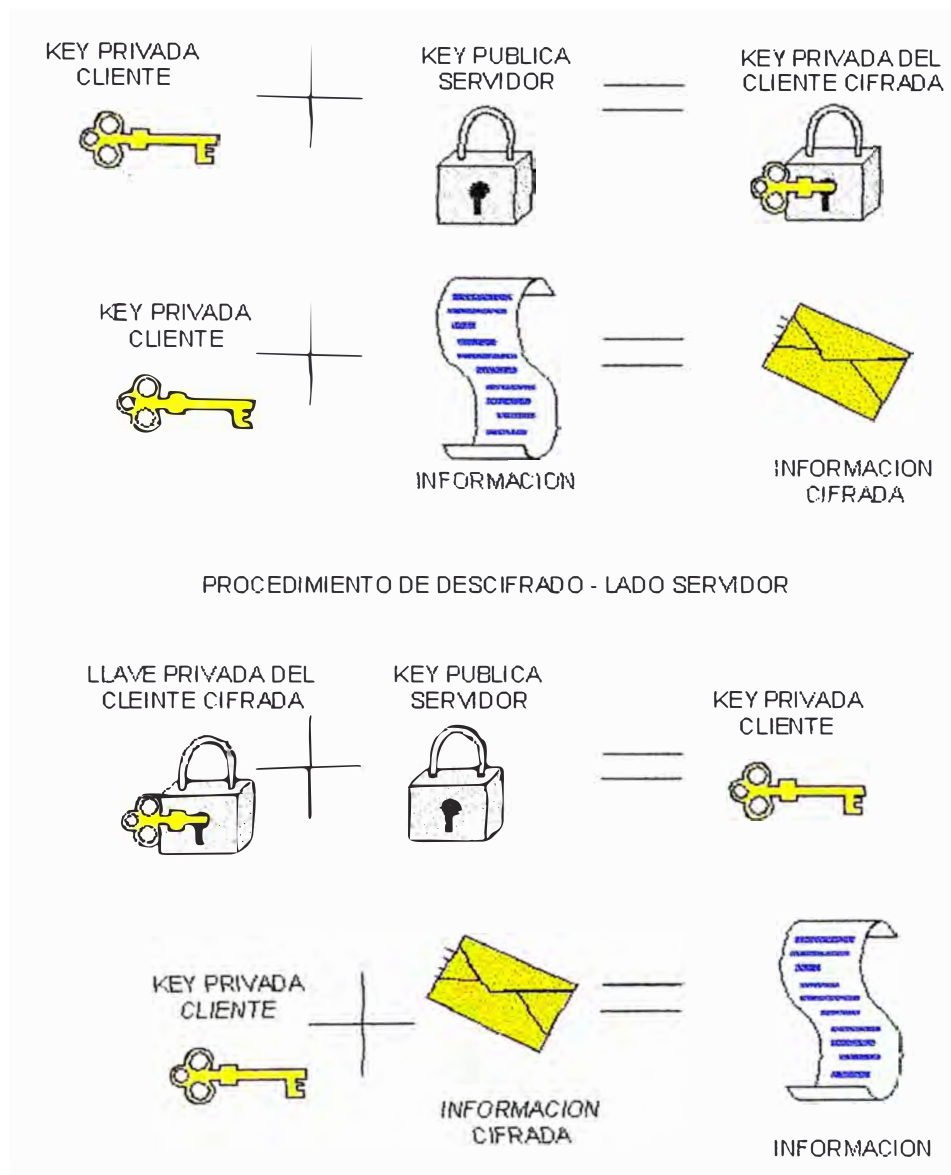


Figura 5.3

•

Como se puede apreciar en la Figura 5.3 hay dos etapas del cifrado en la transmisión de información del Cliente al Servidor y esta etapa se repite para cuando el Servidor le envía una respuesta al Cliente pero en sentido inverso, tal como se muestra en la Figura 5.4.

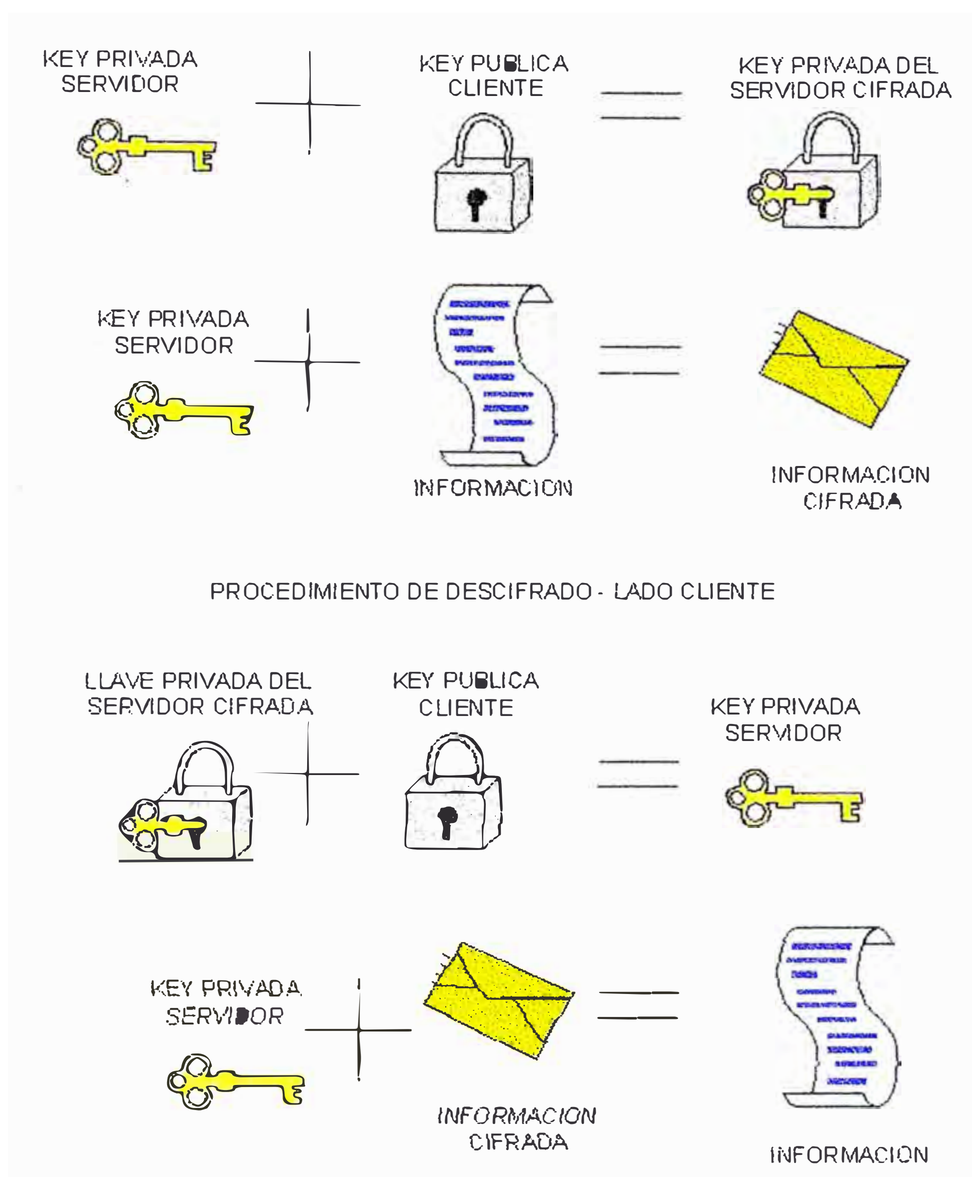


Figura 5.4

Cabe mencionar que las llaves públicas se almacenan tanto en el Cliente como en el Servidor, sin embargo las llaves privadas con generadas en forma aleatoria por transferencia de información completa (envío completo del archivo) y son almacenadas en memoria, mientras dura la transferencia de la información.

7.3 Cifrado a Nivel Administración de Archivos-Cliente

Debido a que en el Cliente hay un riesgo de de las llaves públicas puedan ser alteradas por lo usuarios se ha convenido desarrollar un algoritmo de cifrado del archivo de configuración, el cual se genera en la última línea del archivo de configuración y es validado por el aplicativo TRFLOTES al momento de cargarse a memoria (ejecutarlo).

El valor de seguridad que se genera en el archivo de configuración itl_secur.cfg del Cliente es generado por un programa independiente a la transferencia de información al momento de tener todos los parámetros establecidos, tal como:

- El ID del Cliente.
- El IP de la PC-Cliente.
- Llave Pública del Cliente.
- Llave Pública del Servidor.

Con todo lo mencionado estamos logrando que logramos que ninguna PC-Cliente pueda tener acceso a la llave real de intercambio de información, sino a una llave cifrada la cual a su vez tampoco podrá ser modificada del archivo de configuración (ver anexo J) debido a que el aplicativo de transferencia de información lo detectará al validar la línea de seguridad.

CAPÍTULO VI

GENERALIZACION DEL SISTEMA – PLATAFORMA DE COMUNICACIÓN. APLICACIÓN COMERCIO ELECTRONICO

El sistema que se ha desarrollado, "Sistema de Compensación Bancaria", es una base para múltiples desarrollos de transferencia de información en forma segura en redes LAN y/o WAN. Su generalización puede ser aplicada a la transferencia de información electrónica en diversos campos, tanto así como transferencias electrónicas en línea como para transferencias electrónicas por lotes. Esta generalización es posible debido a que en el presente proyecto se han desarrollado drivers de comunicación TCP/IP con el propósito de conectar redes LAN y/o WAN, además se cuenta con procesos encargados de cifrar la información y del sistema servidor base para ser empleado en comercio electrónico con algunas variaciones como por ejemplo en el formato de mensaje a transferir o el modo de operación del Servidor y/o del Cliente. Como ejemplos de aplicación de comercio electrónico posibles en donde se podría aplicar el sistema tenemos:

1. Transacciones electrónicas en línea, así como transacciones electrónicas efectuadas desde una página WEB o de una ventanilla y que debe ser autorizada por un servidor. Cabe mencionar que este esquema es muy común actualmente entre las instituciones financieras a nivel mundial y como ejemplos de este modo de operación tenemos:
    - Banco de Crédito, operaciones de Cajeros y Ventanillas.
    - TIM – Prepago Virtual y OLC, para cargar tarjetas TIM.
    - CMR – Saga Falabella, para compras en línea.
    - Banco Exterior de Venezuela, para procesar transacciones de VISA y MasterCard.
    - Etc.
2. Transacciones por Lotes realizadas para compensar las operaciones efectuadas en un sistema, cuando este no está provisto de un servidor y la compensación debe ser efectuada por un servidor principal, como ejemplos tenemos:

    - CMR- Saga Falabella Compensación de Compras efectuadas en POS.
    - Sistema de Compensación Financiera en empresas que manejan un solo servidor y tienen múltiples agencias, como Hiraoka, Metro, etc.

Todas estas aplicaciones posibles se podrán realizar en base a este sistema de compensación bancaria realizado, ya que contempla la parte de comunicaciones así como la parte de seguridad.

A continuación desarrollamos un ejemplo de un sistema transaccional en línea para ver el empleo que le damos a los procesos desarrollados.

En la Figura 6.1 se muestra un esquema general de un sistema transaccional en línea en donde la transacción electrónica que se realiza tiene como propósito principal debitar de la tarjeta de ahorros (puede ser tarjeta de crédito o propietaria) a través de una PC que tiene conectada una lectora de tarjetas.

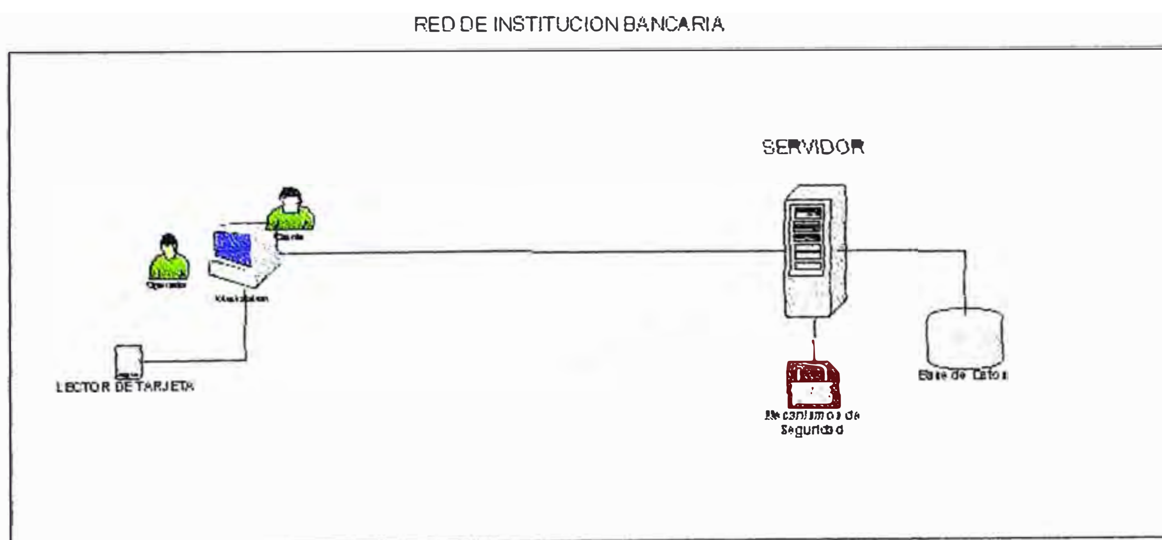


Figura 6.1: Esquema General de un Sistema Transaccional en línea.

Así como está planteado en el esquema de la Figura 6.1, es muy similar a como el Banco del Trabajo está desarrollando un esquema de inscripción de

tarjetas y consulta sobre estados de tarjetas, sin embargo, el sistema desarrollado en este proyecto va más allá puesto que maneja los dispositivos de seguridad, por tanto se puede realizar las operaciones normales de pagos de servicios con tarjetas de débito, crédito ó propietarias.

Ahora si dividimos en módulos el sistema de transacciones electrónicas en línea, *veremos que por el lado Cliente tenemos:*

1. Interfase al Operador que consta con un modulo de acceso Visual para un mejor desempeño, el cual es desarrollado siempre de acuerdo a los requerimientos de la empresa.
2. El módulo de acceso a la lectora de tarjetas, el cual es una DLL que tiene un conjunto de funciones para acceder a la lectora y son proporcionadas por el proveedor de lectoras de tarjetas, tal como HYPER en el Perú.
3. El módulo de formateo de la data siempre es ajustado de acuerdo al formato de mensajes del servidor.
4. El módulo para la conexión al servidor y envío de las transacciones, el cual ya ha sido desarrollado en el presente proyecto.

Y *por el lado servidor tenemos:*

1. La interfase para la conexión al cliente, la cual ha sido desarrollada en este proyecto.
2. El formateo de la información, la cual siempre se ajusta al formato de mensajes empleada por el Servidor.
3. La parte de seguridad, la cual ha sido desarrollada en el presente proyecto.

4. Almacenar la información en la Base de Datos, la cual también ha sido desarrollada en el presente proyecto con MySQL.

Cabe mencionar que en todo sistema los formatos de mensajes son variables en todos los sistemas, así como las funciones de acceso a la Base de Datos, esto debido a que se tiene muchas opciones posibles al respecto. Así tenemos por ejemplo algunos formatos de mensajes:

- Formato ISO8583 de VISA
- Formato ISO8583 de MasterCard Crédito
- Formato VISAII
- Formato
- Formato PRICE
- Formatos propietarios

Y por el lado de Base de Datos tenemos por ejemplo:

- Oracle
- Sybase
- DB2
- Ctree
- Informix

Luego concluimos con este ejemplo, que el presente proyecto tiene todo lo necesario para ser incorporado en un sistema de transacciones electrónicas en línea o incluso de crear un sistema nuevo puesto que los módulos

principales han sido desarrollados y los que faltan de formatos y accesos a Base de Datos se ajustan en relación a cada empresa y sus interfases.

CAPÍTULO VII

COMPARACION CON OTRAS ALTERNATIVAS Y COSTO DEL PROYECTO

Para realizar las comparaciones alguna otra aplicación similar en el mercado tendremos que dividir los métodos aplicados en el Sistema de Compensación Bancario en dos partes, primero la transferencia de información en redes Lan y/o WAN y segundo la seguridad en la transferencia de información.

En relación a la transferencia de información en redes podríamos realizar una comparación con los actuales métodos aplicados en el mercado nacional como son los provistos por las principales empresas en el medio como son Novatronic y SLM quienes actualmente trabajan solo con plataformas UNIX AIX, Sun Solaris, Tru64 y SCO, cuyos costos resultan ser altos para los clientes a comparación de Linux que es de costo inferior a los $5. Los costos de licencias de AIX, Sun Solaris, Tru64 o SCO bordean los $5000, mientras que el Linux es libre y tiene las mismas funcionalidades que las otras plataformas, y adicional a este costo de licencia habría que agregarle el costo del producto de encriptación de datos desarrollados por las empresas arriba mencionadas.

Un cuadro comparativo de precios de las plataformas se muestra en el siguiente cuadro:

Plataforma	AIX (4.3.3)	Tru64	Sun Solaris	Linux *SUSE*	Linux RedHat
Costo ($)	19000	20000	20000	**100**	**80**

Por otro lado, en relación a la seguridad de la transferencia de información, podríamos mencionar alguna de las actuales herramientas de seguridad más empleadas en el medio, como son las provistas por empresas proveedoras de equipos ROUTER y de empresas proveedoras de líneas dedicadas. Empresas que proveen equipos tenemos GMD y Telefónica Data.

Para el presente estudio de costos se ha escogido a Telefónica Data como referencia, ya que ellos también proveen más líneas dedicadas en el medio y el costo por paquete completo de equipo más línea disminuye en comparación a las demás empresas.

En tal sentido hemos realizado un análisis considerando dos opciones:

1. Servidio Digired (Enlace Dedicado)

   Este punto consiste en solicitar una línea dedicada entre dos puntos de tal forma que hasta cierto punto la información viaja segura, sin embargo, este método de solo enviar la información a través de un canal dedicado no es muy seguro, además sus costos son relativamente altos y acompañados a un programa de cifrado provisto por cualquier empresa el costo se incrementa en una suma muy

elevada. En la Figura 7.1 se muestra el costo de una línea dedicada de acuerdo a los precios de telefónica a la fecha Julio 20003.

Costos de Línea Dedicada 128Kbps DIGIRED

Inversion Inicial	
Conexión a la Red:	1180
Instalación por puerta	236
	1416
Pago Mensual	
Acceso a la Res hasta 128Kbps	414.18
02 Modems:	141.6
	555.78

Año	Mes	Pago Línea dedicada 128Kbps DIGIRED	
		Inversion Inicial	Pago Mensual
0	0	1416	
1	1		555.78
1	2		555.78
1	3		555.78
1	4		555.78
1	5		555.78
1	6		555.78
1	7		555.78
1	8		555.78
1	9		555.78
1	10		555.78
1	11		555.78
1	12		**555.78**
INVERSION ANUAL			**8085.36**

Figura 7.1

2. Red IP-VPN

Consiste en emplear equipos router que tengan la opción de cifrado de información, es decir, que estos equipos tengan la opción VPN habilitada y que por tanto no habría necesidad de comprar la línea

dedicada ni tampoco habría que emplear encriptación, solo sería necesaria la aplicación de transferencia de información bancaria por lotes. En la Figura 7.2 se muestran los costos por esta implementación.

Costos Empleando RED IP-VPN

Inversion Inicial

Conexión a la Red:	500
Instalación por puerta	100

Router

Pago Unico	110
Pago Mensual	68
Gestion de Router	20

Año	Mes	Pago Línea dedicada 128Kbps DIGIRED	
		Inversion Inicial	Pago Mensual
0	0	710	
1	1		88
1	2		88
1	3		88
1	4		88
1	5		88
1	6		88
1	7		88
1	8		88
1	9		88
1	10		88
1	11		88
1	12		**88**
INVERSION ANUAL			1766

Figura 7.2

Mientras que empleando la aplicación propuesta en esta tesis, solo bastaría con una conexión a INTERNET, puesto que el cifrado ya lo tenemos en el software. Una vez realizada estas comparaciones podemos concluir que por costo y funcionalidad, este proyecto resulta ser el adecuado en el mercado de comercio electrónico en estos días.

CONCLUSIONES Y RECOMENDACIONES

De todo lo expuesto anteriormente podemos sacar las siguientes conclusiones:

- Hemos notado que hay muchas empresas grandes en el medio que están usando como plataforma *UNIX* tales como *SCO*, *Tru64* y *AIX* las cuales requieren un mantenimiento de uso caro, sin embargo, *podrían emplear como plataforma base para sus servidores* Linux cuyo costo es mucho menor y sobretodo es de igual confiabilidad que los otros sistemas.
- La seguridad debe ser implementada en todo sistema de comercio electrónico si es que se quiere ofrecer confiabilidad a la misma transferencia de información electrónica.
- Es posible crear una red de comunicación segura empleando Software como una alternativa confiable y que además se puede ajustar a la medida de los requerimientos. Este punto es muy importante ya que los métodos de seguridad empleando hardware resultan muy caros.

- La alternativa de seguridad empleando Software es muchas veces más flexible como para adecuarse a cambios rápidos en los mismos algoritmos de seguridad.
- Hemos podido implementar un Sistema de Compensación Bancario en una red de comunicaciones enviando data en forma segura y con mínimo costo invertido. Esto se debe a que para el desarrollo del Sistema hemos empleado como plataforma base Linux como Servidor, cuyo costo es ínfimo y hemos usado PC con Windows NT, Windows 95 , 98 y/o Windows 2000 como plataforma base para los clientes, cuyo costo también es absorbido por la institución bancaria para el trabajo de sus operadores.
- En este Sistema hemos podido emplear el método de seguridad Simétrico DES para la seguridad de la transferencia de información electrónica y hemos cubierto el empleo de las Llaves Públicas con los métodos de identificación de la PC-Cliente con su ID y el IP, obteniéndose una seguridad completa para el sistema en su conjunto.
- El Sistema de Compensación Bancario desarrollado tiene una fácil administración cuando se trata de incrementar el número de clientes, pues se cuenta con archivos de configuración y tablas de control las cuales pueden registrar nuevos Clientes sin necesidad de bajar el Sistema, es decir, contamos con un Sistema que no requiere bajarse en ningún momento.

- El Sistema de Compensación Bancario desarrollado en el presente proyecto nos sirve como base para poder transferir todo tipo de información electrónica, ya sea transferir transacciones electrónicas en *línea* o transferir transacciones por lotes, ambas empleadas actualmente en el medio de comercio electrónico a nivel mundial.
- El Sistema de Compensación Bancario desarrollado es escalable, ya que podemos emplear sus funciones primitivas para la transferencia de información electrónica para intercambiar información en cualquier formato de mensajes. Es así como se puede establecer comunicación con un VAP de VISA o un MIP de MasterCard e intercambiar información en formato ISO8583, o también se podría establecer comunicación con instituciones privadas e intercambiar mensajes en formatos propietarios, los cuales se van adecuando en las rutinas del sistema.
- Los sistemas de transferencia de información electrónica a nivel mundial están en aumento y por tanto la competencia es mayor, sin embargo, lo que los clientes buscan es minimizar costos pero a su vez tener la garantía de que la información que se transfiere en sus redes viaja en forma segura, como así lo plantea este Sistema de Compensación Bancaria.

ANEXOS

ANEXO A

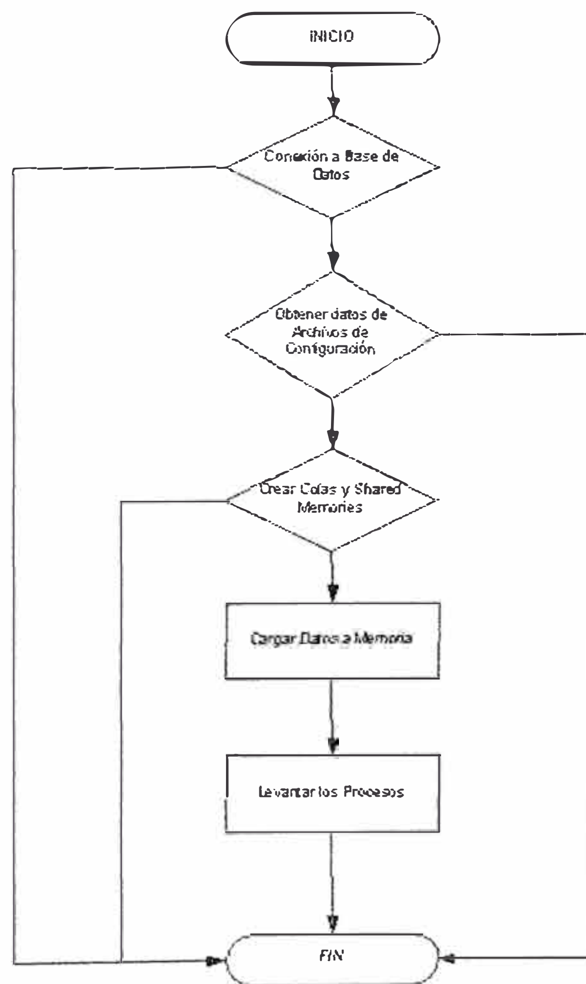


Diagrama de Flujo ITL_UP

ANEXO B

Campo	Tipo	Descripción
Id	varchar(12)	Identificador del Cliente
Ip	varchar(15)	IP del Cliente
Keyid	varchar(17)	Llave de Intercambio del Cliente cifrado bajo la MASTERKEY
Timeout	int(6)	TimeOut del Cliente
nrotimeout	int(3)	Númeo de TimeOuts del Cliente

ANEXO C

Diagrama de Flujo ITL_DOWN

ANEXO D

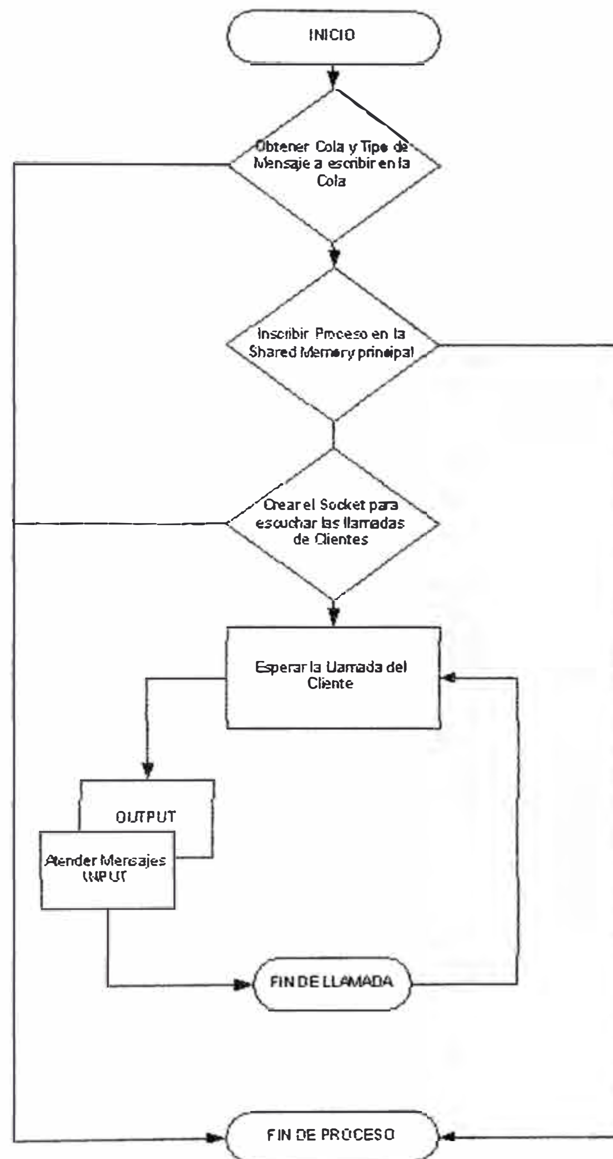


Diagrama de Flujo ITL_SERVER

ANEXO E

```
/* Tables defined in the Data Encryption Standard documents */

/* initial permutation IP */
static char ip[] = {
        58, 50, 42, 34, 26, 18, 10,  2,
        60, 52, 44, 36, 28, 20, 12,  4,
        62, 54, 46, 38, 30, 22, 14,  6,
        64, 56, 48, 40, 32, 24, 16,  8,
        57, 49, 41, 33, 25, 17,  9,  1,
        59, 51, 43, 35, 27, 19, 11,  3,
        61, 53, 45, 37, 29, 21, 13,  5,
        63, 55, 47, 39, 31, 23, 15,  7
};

/* final permutation IP^-1 */
static char fp[] = {
        40,  8, 48, 16, 56, 24, 64, 32,
        39,  7, 47, 15, 55, 23, 63, 31,
        38,  6, 46, 14, 54, 22, 62, 30,
        37,  5, 45, 13, 53, 21, 61, 29,
        36,  4, 44, 12, 52, 20, 60, 28,
        35,  3, 43, 11, 51, 19, 59, 27,
        34,  2, 42, 10, 50, 18, 58, 26,
        33,  1, 41,  9, 49, 17, 57, 25
};

/* expansion operation matrix
 * This is for reference only; it is unused in the code
 * as the f() function performs it implicitly for speed
 */
#ifdef notdef
static char ei[] = {
        32,  1,  2,  3,  4,  5,
         4,  5,  6,  7,  8,  9,
         8,  9, 10, 11, 12, 13,
        12, 13, 14, 15, 16, 17,
        16, 17, 18, 19, 20, 21,
        20, 21, 22, 23, 24, 25,
        24, 25, 26, 27, 28, 29,
        28, 29, 30, 31, 32,  1
};
#endif

/* permuted choice table (key) */
static char pc1[] = {
        57, 49, 41, 33, 25, 17,  9,
         1, 58, 50, 42, 34, 26, 18,
        10,  2, 59, 51, 43, 35, 27,
        19, 11,  3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
         7, 62, 54, 46, 38, 30, 22,
        14,  6, 61, 53, 45, 37, 29,
        21, 13,  5, 28, 20, 12,  4
};

/* number left rotations of pc1 */
static char totrot[] = {
        1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
};

/* permuted choice key (table) */
static char pc2[] = {
        14, 17, 11, 24,  1,  5,
         3, 28, 15,  6, 21, 10,
        23, 19, 12,  4, 26,  8,
        16,  7, 27, 20, 13,  2,
        41, 52, 31, 37, 47, 55,
        30, 40, 51, 45, 33, 48,
        44, 49, 39, 56, 34, 53,
        46, 42, 50, 36, 29, 32
};
```

Tablas DES

```
/* The (in)famous S-boxes */
static char si[8][64] = {
        14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7,
         0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
         4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
        15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13,

        15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10,
         3, 13,  4,  7, 15,  2,  8, 14, 12,  0, 1,  10,  6,  9,  11, 5,
         0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
        13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9,

        10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
        13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
        13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
         1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12,

         7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15,
        13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
        10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
         3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14,

         2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9,
        14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
         4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
        11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3,

        12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
        10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
         9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
         4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13,

         4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
        13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
         1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
         6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12,

        13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
         1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
         7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
         2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11
};

/* 32-bit permutation function P used on the output of the S-boxes */
static char p32i[] = {
        16,  7, 20, 21,
        29, 12, 28, 17,
         1, 15, 23, 26,
         5, 18, 31, 10,
         2,  8, 24, 14,
        32, 27,  3,  9,
        19, 13, 30,  6,
        22, 11,  4, 25
};
/* End of DES-defined tables */
```

Tablas DES

ANEXO F

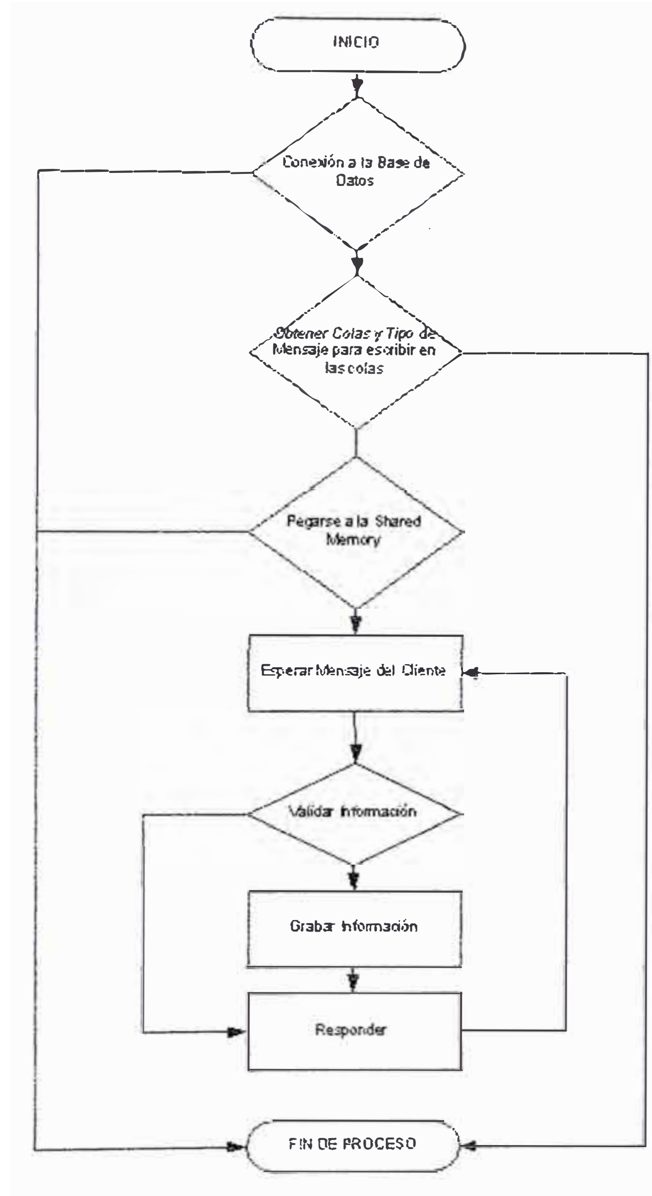


Diagrama de Flujo ITL_MABD

ANEXO G

Campo	Tipo	Descripción
Id	varchar(9)	Identificador del Cliente
Fileid	varchar(7)	Identificador del Archivo a transferir
Filename	varchar(25)	Nombre del Archivo a transferir
filelen	int(10)	Longitud del archivo a transferir
fechahora	varchar(13)	Fecha y hora del inicio de la transferencia del archivo
nrototreg	int(8)	Número total de registro a ser transferidos
nroseqreg	int(3)	Número de registro enviado
Status	int(3)	Estado de la transferencia. Los detalles de códigos de respuesta se indican en el Anexo I.

ANEXO H

Funciones para Acceso a la Base de Datos (Tabla ITLTRFMSG)

```
/*-----------------------------------------------*/
/* put_dataheader: Poner Datos en la Base de Datos */
/*-----------------------------------------------*/
int put_dataheader(MYSQL *idsql, char *bin,int status,char
*filename,int filelen,int nrototreg, int nroreg, char
*fileid,char *fecha_hora)
{
	char  query_insert[1000];

	ENTER(put_dataheader);

	memset(query_insert,0,sizeof(query_insert));
	sprintf(query_insert,"insert into itltrfmsg
values('%s','%s','%s',%d,'%s', %d, %d, %d)",bin, fileid,
filename, filelen,fecha_hora,nrototreg,nroreg,status);
	mysql_query(idsql,query_insert);
	if(mysql_errno(idsql))
	{
		LOG(DEB,"Error al insertar registro [%d] Descripcion
[%s]",mysql_errno(idsql),mysql_error(idsql));
		return ST_ERRINS;
	}
	LOG(DEB,"REGISTRO INSERTADO EN itltrfmsg");

	return GOOD;

}

/*-----------------------------------------------*/
/* update_dataheader: Poner Datos en la Base de Datos */
/*-----------------------------------------------*/
int update_dataheader(MYSQL *idsql,char *bin,int status, int
nroseqreg,char *fileid, char *fecha_hora)
{
	char  query_update[1000];

	ENTER(update_dataheader);

	memset(query_update,0,sizeof(query_update));
	sprintf(query_update,"update itltrfmsg set nroseqreg= %d,
status= %d  WHERE bin='%s' and fileid='%s' and
fechahora='%s'",nroseqreg,status,bin, fileid, fecha_hora);
	mysql_query(idsql,query_update);
	if(mysql_errno(idsql))
	{
		LOG(DEB,"Error al actualizar registro [%d]
Descripcion [%s]",mysql_errno(idsql),mysql_error(idsql));
		return ST_ERRUPD;
	}
	LOG(DEB,"REGISTRO ACTUALIZADO EN itltrfmsg");

	return GOOD;

}
```

```
int find_dataheader(MYSQL idsql,char *bin, char *filename, char
*fileid, char *fecha_hora,int *nroreg, int *status, int
*nrototreg)
{
	MYSQL_RES *result;
	MYSQL_ROW row;
	unsigned int num_fields=0;
	unsigned int num_rows;
	char  query_find[1000];
	int   nroseqreg;
	int   nstatus;
	int   numtot;
	int   i,r,find_reg=0;

	ENTER(find_dataheader);

	memset(query_find,0,sizeof(query_find));
	sprintf(query_find,"select
filename,nroseqreg,status,nrototreg from itltrfmsg  WHERE
bin='%s' and fileid='%s' and fechahora='%s'",bin, fileid,
fecha_hora);
	r=mysql_query(&idsql,query_find);
	r=mysql_field_count(&idsql);
	if(mysql_errno(&idsql))
	{
		LOG(DEB,"Error al buscar registro [%d] Descripcion
[%s]",mysql_errno(&idsql),mysql_error(&idsql));
		return ST_REGNOTFOUND;
	}
	result = mysql_use_result(&idsql);
	num_fields=mysql_num_fields(result);
	find_reg=0;
	while((row = mysql_fetch_row(result))){
		find_reg=1;
		for(i=0;i<num_fields;i++)
		{
			switch(i)
			{
				case 0:
					strcpy(filename,row[i]);
					break;
				case 1:
					nroseqreg=atoi(row[i]);
					*nroreg = nroseqreg;
					break;
				case 2:
					nstatus=atoi(row[i]);
					*status = nstatus;
					break;
				case 3:
					numtot=atoi(row[i]);
					*nrototreg = numtot;
					break;
			}
		}

	}
	if(find_reg)
		return GOOD;
	else
		return BAD;
}
```

ANEXO I

Header de Inicio de Transferencia

```
struct header_ini
{
        char   status[2];
        char   fileid[6];
        char   nroseqreg[7];
        char   lendata[5];
        char   bin[8];
        char   fecha_hora[12];
        char   filename[24];
        char   filelen[9];
        char   nrototreg[7];
        char   xchgkey[16];
};
```

Header por registro enviado en la Transferencia

```
struct header_seq
{
        char   status[2];
        char   fileid[6];
        char   nroseqreg[7];
        char   lendata[5];
        char   bin[8];
        char   fecha_hora[12];
};
```

Código de Respuesta	Descripción
00	Recepción satisfactoria
01	Inicio de envío
02	Envío de Data
03	Fin de Envío de Data
10	Error en recepción de envío de data
11	Reenvío de data
12	Error de formato
13	Número de registro incorrecto
14	ID no registrado
15	Status enviado no válido
16	Error en insertar registro en Tabla de Control
17	Error en actualizar registro en Tabla de Control
18	Archivo enviado no encontrado en Tabla de Control

ANEXO J

```
## Archivo de Configuracion ##
HostServer   = 128.2.10.110
PortServer   = 2020

Institucion  = 500000
KeyCliente   = 0123456789ABCDEF
KeyServidor  = FEDCBA9876543210

Descripcion  = Banco ACME Envio de Lotes
Key          = 9714566DFDC2B848
Dirlog       = E:\Documentos\Raul\Log


IDFILE = 8765414319876555FC9876618765414319876555FC987661
FC9876618765414319876555FC987661
```

ANEXO K

```
#Archivo de Parámetros de conexión a la Base de Datos

name_bd=BD_BANK01
user=user01
pass=pass01
```

ANEXO L

```
[MAIN]
KEY_MAIN = 0x00000064
KEY_001 = 0x00000074

[QUEUES]
QUEUE_SERVER = 0x0000010
QUEUE_MABD = 0x0000020
```

ANEXO M

```
# PROC_TAB. procesos del ITL.
#
# notas:
#       - no deben haber espacios despues del separador de campo ';'
#                   - EL MAXIMO NUMERO DE PROCESOS A INSCRIBIR ES 30.
#
#PROCTYPE;INTEXT;OBLIGATORIO;NOMBRE;RUTA;REDID;SYSNAME;PARAMETROS;COMENTARIOS;
#longitudes maximas:
#  (1)     (1)    (1)      (15)  (30)              (25)
#
a;i;s;itl_server       ;/home/raul/bin ;1 ;INTERFAZ_ITL ;-P7105;
a;i;s;itl_mabd     ;/home/raul/bin ;1 ;INTERFAZ_ITL ;;
```

BIBLIOGRAFIA

[1] Academia de Networking de Cisco Systems: Guía del Primer año

Autor: Vito Amato - Redactor

[2] Applied Cryptography

Autor: Bruce Schneier

[3] Análisis de seguridad de la familia de protocolos TCP/IP y sus *servicios asociados*

Autor: Raúl Siles Peláez

[4] Artículo: Data and Computer Security

Autor: Dr. Michael J. Ganley

[5] Artículo: General Cryptographic Knowledge

Autor: Pierre-Antoine Benatar

[6] Artículo: Security in Banking

Autor: Brian Pugh

[7] Artículo: Why Cryptography is Harder than It looks

Autor: Bruce Schneier

[8] Comunicaciones en UNIX

Autor: Jean – Marie Rifflet

[9] Desarrollo de Aplicaciones Distribuidas con Visual Basic 6.0 y COM+

Autor: Telematic

[10] High Speed Implementation of DES

Autor: S.K. Banerjee

[11] Ingeniería del Software – Un enfoque práctico

Autor: Roger S. Pressman

[12] Linux – Manual de Referencia

Autor: Richard Petersen

[13] Securing your WEB site For Business – A step by step guide for secure on-line commerce

Autor: Verisign

[14] SCO OpenServer Development System– **Network *Programmer's Guide and Reference***

[15] UNIX – Programación Avanzada

Autor: Fco. Manuel Márquez

[16] Visual C-Microsoft

Autor: Fco. Javier Ceballos