UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ciencias y Humanidades

Desarrollo de un manejador de colas de código libre compatible con MQSeries, como una opción gratuita para el manejo de transacciones en línea

Trabajo de investigación presentado para optar al grado de
Licenciado en Ciencias de la Computación
Por Carlos Alberto Molina Pérez

Guatemala
2003

Desarrollo de un manejador de colas de código libre compatible con MQSeries, como una opción gratuita para el manejo de transacciones en línea

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ciencias y Humanidades

Desarrollo de un manejador de colas de código libre compatible con MQSeries, como una opción gratuita para el manejo de transacciones en línea

Trabajo de investigación presentado para optar al grado de
Licenciado en Ciencias de la Computación
Por Carlos Alberto Molina Pérez

Guatemala
2003

PREFACIO

El presente trabajo de graduación pretende ayudar a las empresas guatemaltecas a interactuar con otras empresas por medio de transacciones en línea con un bajo costo, permitiendo acoplarse a los estándares del mercado.

El proyecto es compatible con la versión 6 de MQSeries, y no se garantiza su compatibilidad con futuras versiones de MQSeries.

Quiero agradecer a las siguientes personas:

- A mi asesor, Ing. Juan Antonio Cabrera Aguirre, por el tiempo dedicado en la asesoría del presente trabajo.
- Al Ing. Luis Masaya, Jefe del Departamento de Computación de la Universidad del Valle de Guatemala, por la aportación de ideas en la elaboración del proyecto.
- A mis maestros universitarios, por compartir sus conocimientos durante los cursos que me impartieron, y que hoy forman la base de los conocimientos utilizados para desarrollar este proyecto.
- Por último, pero no menos, a mis padres y hermanos, por el apoyo moral brindado durante mi formación universitaria.

CONTENIDO

Página

LISTA DE CUADROS

Cuadro Página

LISTA DE ILUSTRACIONES

Ilustración Página

RESUMEN

En Guatemala se ha tomado casi como un estándar el *software* de IBM, Websphere MQ (conocido también como MQSeries), como el medio de comunicación para la realización de transacciones en línea. El costo de adquirir esta solución de *software* puede ascender a miles de dólares estadounidenses.

En la actualidad existen manejadores de colas que son gratuitos, pero ninguno de ellos tiene la capacidad de comunicarse con MQSeries, por lo que apegarse a los estándares del mercado resulta una labor difícil.

El presente proyecto de tesis, presenta una opción gratuita a la implementación de transacciones en línea, por medio del desarrollo de un manejador de colas denominado MyOpenQ, el cual está desarrollado en el lenguaje de programación C y funciona sobre el sistema operativo Red Hat Linux. Como parte del proyecto se desarrolló el módulo MyOpenBridge, que funciona como un traductor de protocolos, logrando así la comunicación con MQSeries.

I. INTRODUCCIÓN

Con el surgimiento de la Internet se han derribado las barreras geográficas que existían para el comercio internacional. El mercado de hoy en día es un mercado global y las empresas que no lo miren como tal están perdiendo la oportunidad de expandirse. Las expectativas de los consumidores cada vez son más grandes, existe una verdadera guerra de proveedores por retener a sus clientes, los cuales ahora con el avance de la tecnología están a un solo clic de la competencia.

Todo esta tendencia de mercado ha llevado a las empresas ha formar alianzas estratégicas con otros proveedores, con el fin de brindarles a sus clientes más y mejores servicios.

Hoy en día, los clientes demandan que todas sus transacciones se realicen de forma inmediata, con lo cual las empresas se han visto obligadas a estar en línea. Esta modalidad genera la necesidad de una forma de comunicación segura, estable, rápida y confiable por medio de la cual los aliados estratégicos puedan intercambiar información.

Actualmente en Guatemala existe un gran número de empresas que ya prestan servicios en línea; en su mayoría estos consisten en la realización de pagos a través del sistema bancario, de servicios prestados por otras entidades como universidades, empresas de telefonía, empresas de servicio eléctrico, entre otras.

Para desarrollar estos servicios se ha tomado casi como un estándar el *software* de IBM MQSeries, que consiste en la interconexión de las dos partes por medio de colas de mensajería, con las cuales se puede intercambiar información de una manera segura y confiable.

Microsoft por su parte cuenta con el producto MSMQ, que permite realizar las mismas funcionalidades de MQSeries, pero que además puede utilizarse con el producto BizTalk Server, que permite manejar el envío de información incluyendo toda la lógica transaccional como parte del proceso de comunicación.

II. OBJETIVOS

El objetivo del proyecto es proporcionar a las empresas de una herramienta de *software* sin costo, que les ayude a realizar transacciones en línea de forma segura, y con la que se puedan acoplar a los estándares del mercado sin ningún problema.

Este objetivo se alcanzará por medio de MyOpenQ, para lo cual se requiere desarrollar:

- Un manejador de colas que maneje la lógica de comunicación entre dos puntos.
- Una serie de funciones que permitan realizar operaciones sobre colas, como escritura de mensajes y obtención de mensajes.
- Una herramienta administrativa que permitan la creación de las estructuras necesarias para la comunicación y almacenamiento de mensajes.
- Un traductor de protocolos, que permita establecer los parámetros de configuración para el mapeo de protocolos externos, con el protocolo de MyOpenQ.

Este proyecto pretende:

- Proveer a los desarrolladores de aplicaciones de un *software* intermediario orientado a mensajes (MOM. Message oriented middleware) que permita comunicar aplicaciones de una forma segura y sencilla.
- Fundar las bases, y publicar la primera versión de un proyecto de código libre, que pueda seguir creciendo con la contribución de personas alrededor del mundo.

III. MARCO TEÓRICO

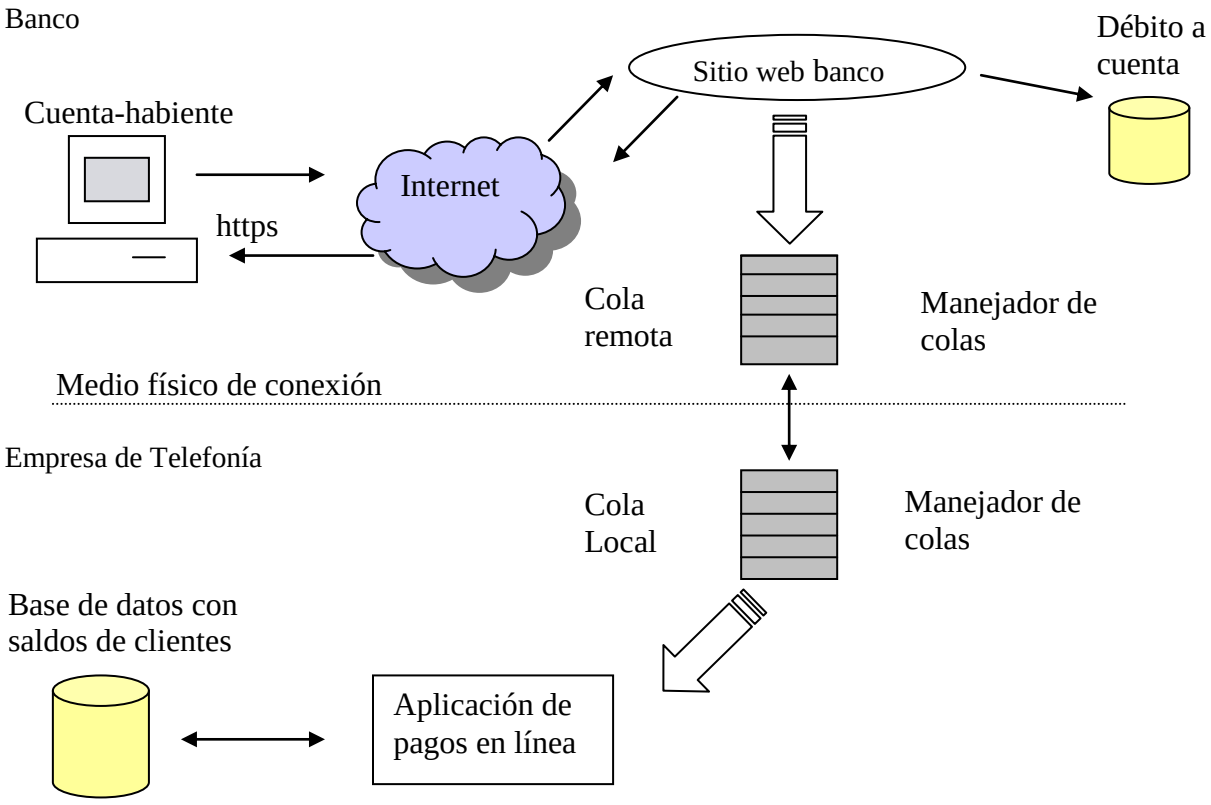
A. Esquema de comunicación

Para interconectar dos empresas es necesario contar con un medio físico que permita la comunicación. Para esto es necesaria la formación de una red, la cual deberá ser un medio confiable y seguro, que no sólo garantice la continuidad del servicio, sino la privacidad del mismo.

Para esto existen dos posibilidades, la formación de una red privada, que conecte a través de un cable físico a las dos instituciones en una conexión punto a punto, o entablar una red privada virtual (VPN. Virtual private network) que permita la comunicación segura entre las dos entidades utilizando como infraestructura la Internet.

Una vez se cuente con esta infraestructura, se deberá desarrollar un programa que permita la comunicación entre ambas; en la actualidad el medio más utilizado es el encolamiento. El encolamiento consiste en la creación de colas de mensajería, en las cuales se coloca la información que se desea enviar, de forma que la contraparte pueda atender ordenadamente las solicitudes.

Ilustración 1. Ejemplo de un pago telefónico en el sistema bancario, a través de Internet.



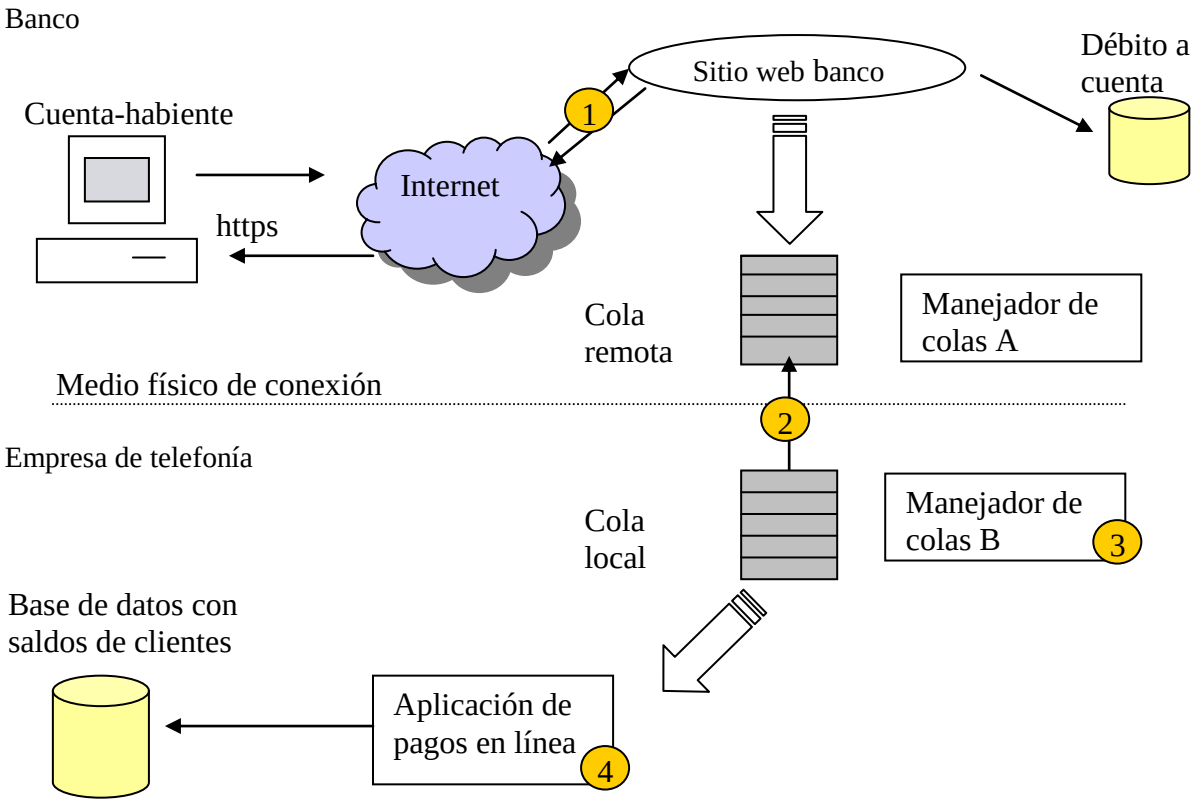
B. Ventajas de utilizar un manejador de colas

El esquema de transacciones en línea con un manejador de colas, garantiza un alto grado de disponibilidad. A diferencia de otros esquemas como *Web Services*, el manejador de colas entregará el mensaje aun cuando la contraparte no esté disponible temporalmente. Además presenta una forma fácil de integrar aplicaciones, reduciendo los tiempos de programación, ya que encapsula toda la lógica de comunicación.

La idea básica que se pretende con un manejador de colas, es que la aplicación que haga uso de él, se despreocupe de toda la lógica de comunicación, reintentos, manejo de errores, fallas en la conexión, etc.

La Ilustración 2 identifica los puntos de fallo que pueden existir en un esquema de transacciones en línea con un manejador de colas.

Ilustración 2. Puntos críticos en un esquema de transacciones en línea con un manejador de colas



Análisis sobre los puntos críticos de la Ilustración 2:

Punto 1: Efectivamente, si el sitio de Internet del banco no está disponible, no se pueden llevar a cabo transacciones, pero este ámbito está fuera del entorno del manejador de colas.
Punto 2: Si no existe una comunicación con la empresa de telefonía no importa, ya que el manejador de colas A recibe la petición, y la mantiene en la cola hasta que sea posible entregar el mensaje al manejador de colas B.
Punto 3: Si el manejador de colas B no está funcionando, el mensaje permanece en el manejador de colas A hasta que pueda ser entregado.
Punto 4: Si la aplicación que procesa los pagos en línea de la empresa telefónica no está funcionando, no importa, ya que el mensaje es entregado al manejador de colas B, y permanecerá en cola hasta que la aplicación de procesamiento de pagos funcione, y procese los mensajes que están en cola.

C. Teoría sobre los elementos utilizados

1. Protocolo de transferencia

El protocolo de transferencia de control (TCP. Transfer control protocol.) es el encargado del flujo de la transmisión y de asegurarse que los datos estén correctos. Mantiene un canal de comunicación abierto, en el cual aparenta que los datos que se transmiten desde el origen hacia su destino forman un circuito continuo. Tiene la capacidad de revisar errores, controlar el flujo y las interrupciones en la comunicación.

Cuando se envían datos utilizando este protocolo, TCP divide los datos en paquetes de 64 Kb máximo, los ordena por medio de una secuencia, añade información para el manejo de errores, y los transmite. El receptor recibe los paquetes de TCP, los revisa para validar su contenido, y los transforma nuevamente en los datos originales. Si existe algún error en un paquete, el TCP del receptor solicita que se envíen nuevamente los paquetes defectuosos.

El TCP es la entidad de transporte del protocolo de Internet (IP. Internet protocol), ya que IP no garantiza que los datagramas (unidades de información) se entreguen apropiadamente.

De esta forma, TCP provee una comunicación extremo a extremo de un programa que corre en una computadora, a otro que corre en otra computadora, las cuales están conectadas en red.

En la estructura de bloque de control de transmisión (TCB. Transmisión control block), se almacena información que sirve para que el TCP pueda asociar la conexión a un programa en particular. Para lograr esto, utiliza unos identificadores llamados *sockets*, los cuales están asociados a una dirección IP

y un número de puerto. Existen *sockets* locales (dirección IP y puerto de origen) y *sockets* remotos (dirección IP y puerto del destinatario).

Los puertos se utilizan para redireccionar la información enviada a las aplicaciones que esperan recibirla, de esta forma se puede diferenciar el flujo de información que se comparte entre dos computadoras con varias aplicaciones comunicándose entre sí.

Los comandos más habituales que acepta el TCP por parte de las aplicaciones son:

- Conectarse: Crea una conexión hacia otro *socket*.
- Recibir: Utilizado para leer información del *socket*, necesita estar en estado conectado.
- Enviar: Utilizado para enviar información, necesita estar en estado conectado.
- Cerrar: Cierra la conexión cuando la sesión ha finalizado.

Un paquete de TCP se divide en dos partes, encabezado y datos. Los campos que conforman el encabezado del paquete son:

- Número de puerto de origen (longitud de 16 *bits*).
- Número de puerto de destino (longitud de 16 *bits*).
- Números de secuencia: Posee dos números, el primero es el número de la última secuencia (SSN. Serial sequence number), de 32 *bits* de longitud. El segundo es el número de secuencia esperado de recepción (ACK. Acknowledge).
- Longitud del encabezado: Es el número de octetos que forma la cabecera del TCP dividido entre cuatro. Ocupa cuatro *bits* de longitud.
- Código de *bits*. Contiene contenidos específicos:
    - *Bit* URG: Marca el paquete como urgente.
    - *Bit* ACK: Indica si el campo de reconocimiento es válido o no.
    - *Bit* PSH: Por medio de este *bit*, el equipo remitente puede forzar el envío del segmento, esté o no lleno el buffer.
    - *Bit* RST: Utilizado para abortar la conexión, los buffer asociados se vacían.
    - *Bit* SYN. Utilizado para sincronizar los números de secuencia.
    - *Bit* FIN: Se activa cuando se está cerrando la conexión.
- Ventana: Indica el tamaño del búfer que tiene el emisor para la recepción de mensajes. Ocupa 32 *bits*.
- Opciones: Puede contener características que rigen las aplicaciones como puede ser el tamaño máximo del segmento TCP. Si el primer octeto tiene un valor de cero es que no hay opciones.
- Relleno: Campo de relleno que consiste de uno a tres *bytes* con valor cero, se utiliza para que la longitud de la cabecera sea divisible entre cuatro.
- Chequeo: Mecanismo utilizado para garantizar la integridad de los datos. Se realiza una suma de *bits* de todo el paquete TCP.

Cuadro 1. Estructura de un paquete TCP

<table>
<tr><th>División</th><th colspan="8">Campo</th><th>Longitud</th></tr>
<tr><td rowspan="10">Cabecera</td><td colspan="8">Número de puerto de origen</td><td>2</td></tr>
<tr><td colspan="8">Número de puerto de destino</td><td>2</td></tr>
<tr><td colspan="8">Número de Secuencia</td><td rowspan="2">4</td></tr>
<tr><td colspan="4">SSN</td><td colspan="4">Número de ACK</td></tr>
<tr><td colspan="4">Longitud de la cabecera</td><td colspan="4">uso futuro</td><td>1</td></tr>
<tr><td colspan="4" rowspan="2">uso futuro</td><td colspan="4">Código de bits</td><td rowspan="2">1</td></tr>
<tr><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td></tr>
<tr><td colspan="8">Ventana</td><td>4</td></tr>
<tr><td colspan="8">Opciones</td><td>Variable</td></tr>
<tr><td colspan="8">Relleno</td><td>1..3</td></tr>
<tr><td colspan="8">Chequeo</td><td>4</td></tr>
<tr><td colspan="9">Datos</td><td>Variable</td></tr>
</table>

2. MD5

El algoritmo MD5 es un algoritmo cuyo propósito es el de generar huellas de documentos; una cadena que identifica de forma única a un documento.

El sistema de encriptación de MD5 utiliza un algoritmo criptográfico que genera como salida 128 *bits*. Este algoritmo tiene como principal utilidad generar 128 *bits* como resumen de un conjunto de caracteres. El algoritmo es independiente del número de caracteres de entrada.

Es un algoritmo útil para validar información, ya que analizando el MD5 de un documento se puede saber si ha sufrido algún cambio desde que se selló originalmente. Se podría decir que es una especie de sello de seguridad, ya que si se toma el mensaje, se aplica el MD5 y el resultado es el mismo que la marca MD5, podemos estar seguros de que el mensaje no ha sufrido modificaciones por error en la transmisión.

3. Hilos de ejecución

Un hilo de ejecución es un flujo secuencial de control dentro de un programa. Un programa puede contener varios hilos de ejecución, logrando así la ejecución de tareas en paralelo[1].

En los sistemas operativos tradicionales, se asigna a cada proceso una dirección de memoria, y cada uno tiene un único hilo de control (contador de programa). Sin embargo, en ocasiones resulta

[1] Depende de la arquitectura de la computadora. Si es un multiprocesador se realiza paralelismo; si es uniprocesador se obtiene tiempo compartido.

conveniente que dos procesos trabajen conjuntamente, y de forma concurrente. Esto añade cierta complejidad, ya que los procesos son independientes y la comunicación entre ellos debe realizarse por algún mecanismo de comunicación entre procesos (IPC. Inter process communication).

Una solución razonable consistiría en que los procesos compartiesen el mismo segmento de memoria, de esta forma no sería necesaria su intercomunicación, ya que todos tendrían acceso a la misma zona de memoria.

Un proceso se compone de cinco partes fundamentales: Código, datos, pila, E/S a archivos, y tablas de señalización. Los procesos normales, o procesos de mucho peso (HWP. Heavy weight processes), toman un tiempo significante del procesador en realizar un cambio de contexto. El cambio de contexto se da cuando el proceso deja de hacer uso del procesador, y otro toma posesión de él. Para este cambio, es necesario que todas las tablas sean vaciadas y se llenen con los valores del nuevo proceso.

Un hilo de ejecución es similar a un proceso real, sin embargo, un hilo se considera un proceso de poco peso (LWP. Light weight process), porque se ejecuta dentro del contexto de un programa completo y se aprovecha de los recursos asignados al programa y del entorno de éste.

De esta forma, todos los hilos de un programa utilizan la misma memoria, facilitando la comunicación entre ellos, ya que pueden visualizar las mismas variables.

Beneficios de la programación multi-hilo

- La creación de un nuevo hilo es mucho más rápida que la creación de un nuevo proceso.
- Al terminar su ejecución, un hilo sólo libera el espacio asignado, siendo este procedimiento mucho más rápido que la terminación de un proceso.
- Se facilita la comunicación entre los hilos, ya que comparten la memoria.

Existen dos tipos de hilos:

- Hilos de usuario (ULT. User level threads)
- Hilos de sistema operativo (KLT. Kernel level threads).

En los ULT, la administración de los hilos se realiza a nivel de la aplicación, sin la intervención del sistema operativo. El sistema operativo no está consciente de la existencia de hilos, él únicamente ve un proceso realizando un requerimiento.

Ventajas de los ULT:

- Son portables a cualquier sistema operativo, ya que no dependen de él.

- El cambio de contexto es más rápido.
- La planificación de ejecución de los hilos se puede programar a conveniencia.

Desventajas de los ULT:

- El sistema operativo lo ve como un único proceso, por lo que una llamada a sistema bloqueará el proceso completo, aun cuando los otros hilos del proceso pudieran aprovechar el fragmento de tiempo que les asigno el procesador.
- En un ambiente multiprocesador no se aprovecha el paralelismo, ya que el proceso se ejecuta únicamente en un procesador.

En los KLT el sistema operativo sí está consciente de los diferentes hilos de un proceso. El sistema operativo mantiene información del proceso en general y de cada hilo del proceso en particular, y es el encargado de realizar la calendarización de la ejecución de los diferentes hilos.

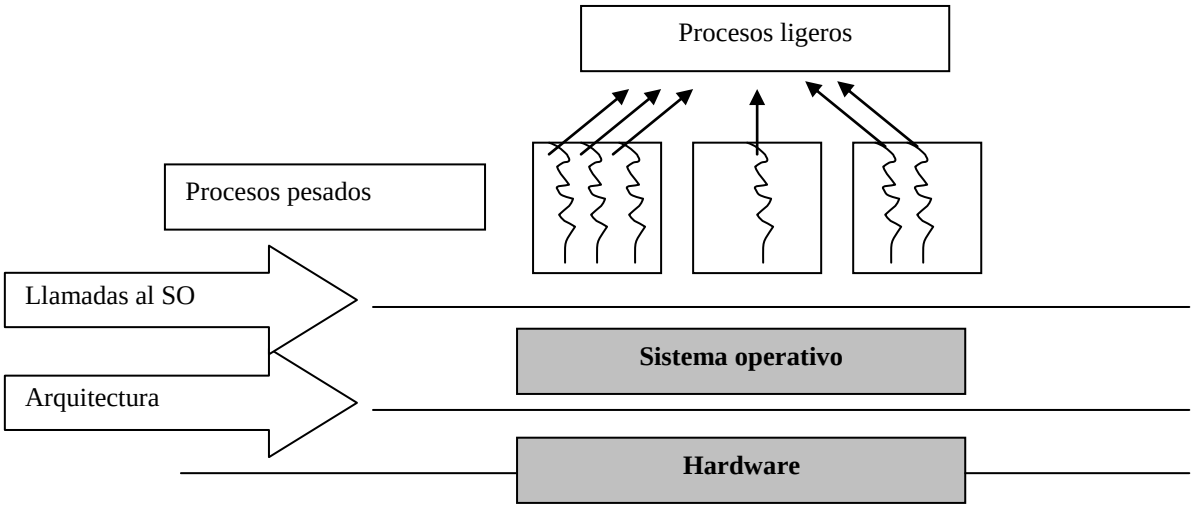
Ventajas de los KLT:

- En un ambiente multiprocesador, los diferentes hilos de un mismo proceso, pueden estarse ejecutando en diferentes procesadores al mismo tiempo.
- No existen problemas de bloqueo por llamadas al sistema.

Desventajas de los KLT:

- Existe un mayor tiempo de procesador en realizar un cambio de contexto entre hilos.
- Dependen del soporte del sistema operativo.

Ilustración 3. Esquema de procesos con hilos



4. Semáforos de exclusión mutua (mutex)

Los semáforos sirven para controlar condiciones entre los hilos de un proceso. Los semáforos se pueden utilizar para solucionar el acceso a las regiones críticas del programa. Estas regiones son fragmentos de código en que se debe tener un acceso único a un recurso en particular, es decir, solo un hilo de ejecución puede tener acceso a la vez.

El mutex es un dispositivo de exclusión mutua, utilizado para proteger estructuras de datos compartidas de modificaciones concurrentes, y llevar un buen manejo de las regiones críticas.

El mutex tiene dos posibles estados: abierto, cuando no está tomado por ningún hilo, y cerrado, cuando está tomado por algún hilo. Un mutex no puede estar tomado por dos hilos simultáneamente. Cuando se intenta cerrar un mutex que esta tomado por otro hilo, el hilo queda suspendido hasta que el hilo dueño ejecuta un abrir, liberando el mutex.

VI. DESARROLLO DE LA PROPUESTA

El proyecto consiste en desarrollar un MOM, que se encargue de manejar toda la lógica de envío de mensajes de forma asíncrona, y que garantice la entrega de mensajes. Para lograr esto se desarrolló un manejador de colas, que permite la definición de colas locales y remotas.

A. Selección de elementos

La primera tarea a realizar fue la de seleccionar un sistema operativo sobre el cual se ejecutaría el manejador de colas. Se tomaron en consideración los siguientes elementos:

- Estable, y con renombre en el ámbito mundial, de forma que inspire confianza al usuario.
- Un sistema operativo de bajo costo, para motivar el uso de la aplicación.
- Que tenga un manejo eficiente de hilos.

Considerando esto, se decidió que la mejor opción era utilizar el sistema operativo *Red Hat Linux*, que también surgió como un proyecto de código abierto.

La segunda tarea a realizar fue la de seleccionar el lenguaje de programación en el cual se desarrollaría el manejador de colas. Las características que se buscaban eran:

- Que fuera un lenguaje conocido a nivel mundial, ya que por el carácter código abierto del proyecto, esto era un requisito indispensable.
- Un lenguaje que permitiera el manejo de hilos, *sockets*, de una manera eficiente.

Considerando esto se presentan dos opciones claras, Java y C. Se decidió seleccionar el lenguaje C, ya que al utilizarse con la librería portable de sistema operativo (POSIX. Portable Operating System Interface) de Linux se puede tener hilos de ejecución de sistema operativo, los cuales permiten aprovechar arquitecturas con varios procesadores. Además de que provee un mejor rendimiento y es más veloz, al no ejecutarse sobre una máquina virtual.

Según lo que se explicó en el marco teórico del presente trabajo existen dos posibles manejos de hilos, de usuario y de sistema operativo.

La librería POSIX en Linux permite tomar todas las ventajas de hilos a nivel de sistema operativo, sin sufrir las desventajas presentadas.

El tiempo de procesador requerido para efectuar un cambio de contexto en Linux, ha sido minimizado por medio de la ejecución del comando clonar, el cual permite definir específicamente que segmentos se comparten por medio de las banderas clone_vm, clone_fs, clone_files, clone_sighand.

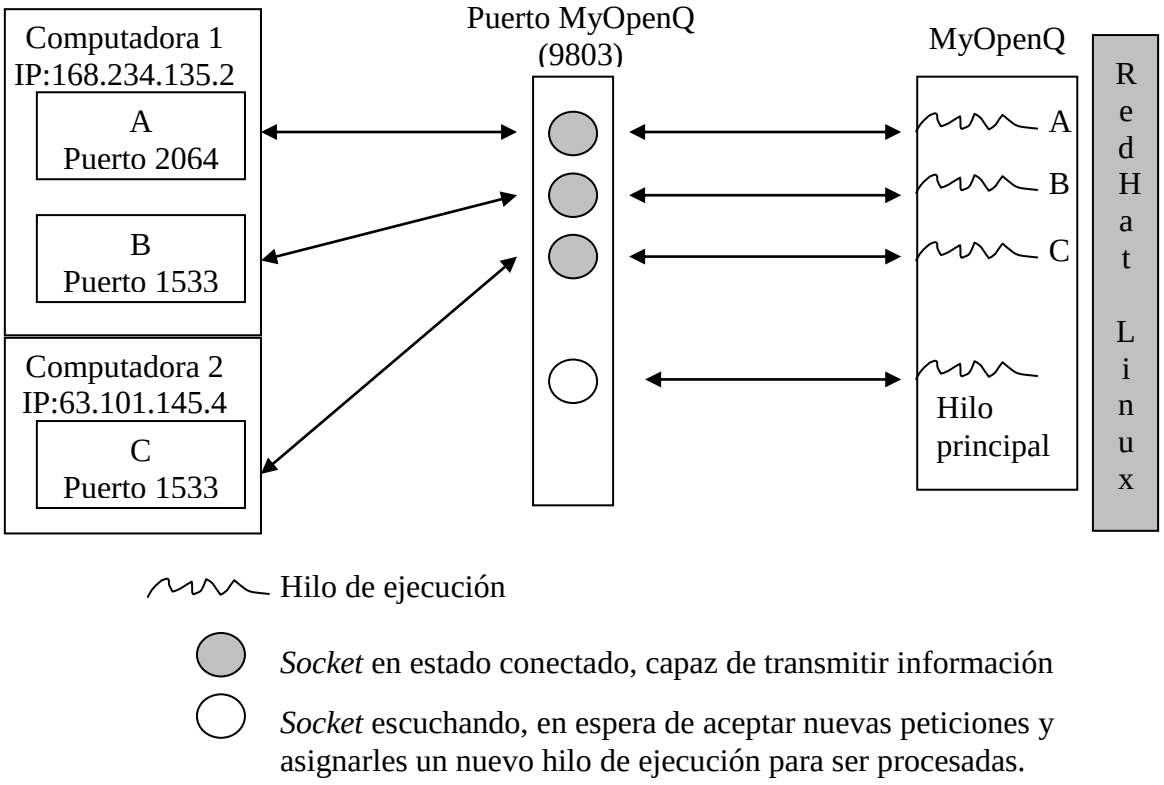
Una vez seleccionado el sistema operativo y el lenguaje programación se dio inicio al desarrollo del proyecto.

Se decidió que toda la comunicación hacia el servidor se realizaría por la misma vía, es decir, tanto la escritura de mensajes, como la administración del servidor, ya sea de forma local o remota, se realizaría por un único punto de acceso.

Este punto de acceso es un *socket* de TCP que escucha peticiones en un puerto específico, al cual todas las aplicaciones que necesiten efectuar operaciones se deberán de conectar.

Considerando que se tendrán varias aplicaciones realizando operaciones sobre el manejador de colas, se tomó la decisión de que cada petición será procesada por un hilo distinto creado bajo demanda, dedicado a atender únicamente peticiones que provengan por el mismo canal de comunicación que le fue asignado, hasta que se finalice la sesión.

Ilustración 4. Procesamiento de peticiones



El hilo principal del programa posee un *socket* que se encuentra escuchando, y cualquier petición de crear un nuevo canal de comunicación es atendida y asignada a un nuevo hilo luego de ejecutar el comando aceptar, por medio del cual se crea un nuevo *socket* que es capaz de recibir y transmitir información.

En el lenguaje C se puede seleccionar entra diferentes tipos de *socket*:

Socket de flujo**:** Orientado a conexión, se utilizan para comunicarse utilizando el protocolo TCP, y para esto hay que en primer lugar establecer una conexión entre un par de *sockets*. Mientras uno de los *sockets* atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos *sockets* estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

Socket de datagramas**:** Utiliza el protocolo de Datagramas de Usuario (UDP. User Datagram Protocol) Es un servicio de transporte sin conexión. Son más eficientes que TCP, pero en su utilización no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

Socket sencillo**:** Dan acceso directo a la capa de *software* de red subyacente o a protocolos de más bajo nivel.

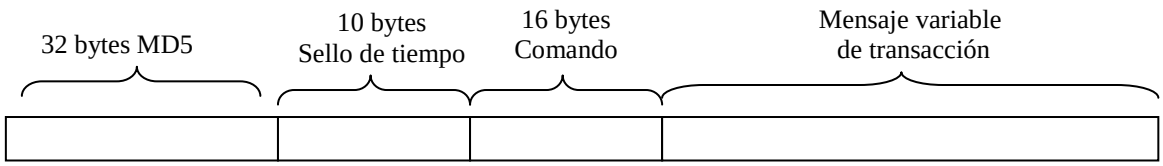
Es necesario que el manejador de colas garantice la entrega de los mensajes, es por esto que se seleccionó la opción de *socket* de flujo, que es una implementación del protocolo TCP, que garantiza la entrega de paquetes en orden, evita duplicidad y maneja el reenvío en caso de error.

El TCP entabla la conexión por medio de una secuencia de tres pasos, en la que se envía de primero un paquete sincronización (SYN. Por su abreviatura en inglés) para sincronizar las secuencias, la contraparte responde con un ACK, y luego el que inició la comunicación devuelve otro ACK. Esto hace que el TCP sea un poco ineficiente en abrir y cerrar conexiones, por esta razón se decidió que las conexiones entre dos manejadores de colas permanecerán abiertas, y se cerrarán cuando finalice el funcionamiento de uno de los dos manejadores de colas, o cuando por fallos en la red se deba reestablecer la conexión.

Una vez tomada esta decisión se analizaron las diferentes operaciones que el manejador de cola debería realizar, para poder así determinar la estructura del mensaje que se enviará al *socket*.

B. Estructura del paquete de MyOpenQ

La estructura básica del paquete es la siguiente:



El valor del MD5 nos proporciona un método seguro para colocarle un sello único a cada mensaje. De esta forma se puede evitar que se reciba dos veces el mismo mensaje; además permite verificar la integridad del mismo.

Se podría pensar que el TCP maneja la validación de duplicados y estructura. Pero la verificación de TCP no posee un algoritmo fuerte. Por esto se consideró que valdría la pena agregar este campo para validar efectivamente el mensaje.

Además hay que considerar que sucedería en el caso de que el manejador de colas se detenga de una manera precipitada. Por ejemplo: Supongamos que se tiene una definición de cola remota Qa en el servidor A, que refleja la imagen de una cola local Qb definida en el servidor B. Si se coloca un mensaje M en la cola Qa, el manejador de colas A tomará el mensaje y lo enviará a B. Si luego de que el mensaje es recibido por B, el servidor de colas A tiene un problema con el fluido eléctrico, el mensaje M nunca fue eliminado de la cola Qa, y al reiniciarse el servicio, enviará nuevamente el mensaje, causando duplicidad en los mensajes.

De situaciones parecidas a la anterior, se puede deducir la necesidad de la validación del MD5, para evitar duplicidad.

El sello de tiempo ayuda al algoritmo MD5 a generar distintos identificadores a pesar de transmitir varias veces el mismo mensaje. El sello de tiempo permitirá monitorear los mensajes en la cola, con el fin de eliminar los mensajes cuyo tiempo de expiración se haya cumplido.

Cada transacción tiene valores que la identifican como tal, y posee una estructura de mensaje en particular. Funcionalmente podemos dividir las transacciones en dos tipos: administrativas y de usuario. Para tener acceso a estas operaciones, se desarrollaron una serie de programas que permiten realizar las diferentes transacciones.

C. Comandos de MyOpenQ

Al ejecutarse un comando se abre un *socket* hacia el servidor de colas especificado, y se envía el paquete con la estructura correspondiente.

MyOpenQ crea un nuevo hilo que es el encargado de atender la petición. La primera tarea que realiza es obtener la IP del cliente que envía la transacción; esto lo realiza por medio del comando getpeername(). Se verifica que la IP se encuentre en el listado de permisos. Si el cliente no posee permisos, se devuelve el código de error MOQ – 4001, y no se efectúa nada más.

Una vez se hayan validado los permisos, se procede a validar el mensaje por medio del campo MD5. Este campo es opcional y se valida únicamente si tiene valor. Esta decisión mejora el rendimiento del manejador de colas, y se basó en que los comandos que se ejecutan de forma local no viajan a través de la red, y por lo tanto no es posible una transmisión errónea del paquete.

Una vez aprobada la validación del mensaje, se verifica que el mensaje no sea un mensaje duplicado. Esto se realiza por medio de la verificación de un listado en el que se almacenan los últimos valores de MD5 recibidos. El tamaño de esta lista se configura por medio del parámetro MD5List del archivo de configuración. Esta lista se mantiene en memoria, y se actualiza en disco por medio de un hilo de control.

Todas las transacciones que realizan operaciones sobre una cola, efectúan el acceso por medio del nombre de la cola; esto implica que se tiene que realizar una búsqueda basada en el nombre, de la estructura de la cola en memoria para poder realizar las operaciones. Las colas se encuentran organizadas en un arreglo de punteros a estructuras, y la búsqueda se efectúa por medio de un recorrido lineal del arreglo en busca de la estructura que posea el nombre especificado. Si durante esta búsqueda secuencial se altera el contenido del arreglo, es posible que la información quede inconsistente. Para controlar esto se utiliza un mutex que regula el acceso a las estructuras. Se realizó una función de búsqueda, la cual permite centralizar el acceso al mutex de estructuras en un sólo punto del código del programa (siempre que sea por motivos de búsqueda). Cuando se encuentra la estructura, la función libera el mutex y retorna el puntero a la estructura encontrada.

1. Comandos administrativos

a. Crear

El comando crear se utiliza para la creación de colas tanto locales como remotas.

Estructura del Paquete

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'CREATE'
Variable	Nombre	32 *bytes*	Nombre de la cola
	Tamaño	5 *bytes*	Número máximo de mensajes
	Longitud	5 *bytes*	Longitud máxima de mensajes
	Tipo	1 *byte*	'L' Local 'R' Remota
	En caso el tipo sea remota		
	IP	15 *bytes*	IP del manejador de colas
	Puerto	5 *bytes*	Puerto del manejador de colas

En este caso en particular, la función de búsqueda es utilizada para verificar que no exista otra cola con el mismo nombre.

El servidor aloca en memoria un espacio para almacenar los valores de la estructura, ejecuta un cierre sobre el mutex de estructuras, aumenta el contador de colas, y agrega un puntero en el arreglo de colas con la dirección al espacio de memoria alocado para la estructura. Por último libera el mutex de estructuras y envía la respuesta de éxito.

b. Borrar

El comando borrar se utiliza para la eliminación de colas locales o remotas.

Estructura del Paquete:

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'DELETE'
Variable	Nombre	32 *bytes*	Nombre de la cola

El servidor de colas utiliza la función de búsqueda para localizar la cola en memoria. Se ejecuta un cierre sobre el mutex de estructuras y el mutex específico de la cola. Se procede a liberar la memoria utilizada por todos los mensajes, y se libera la memoria utilizada por la estructura. Se reorganiza el listado de punteros a colas y se libera el *mutex de estructuras*.

Las colas remotas tienen un hilo que se encarga de enviar los mensajes; este hilo se crea al iniciar el manejador de colas, y es necesario detenerlo con anterioridad, pues si la estructura se elimina antes y el hilo realiza algún acceso a la cola, generará error. Esto se realiza por medio del comando pthread_Kill().

c. Apagar

El comando apagar se utiliza para detener el servicio de colas.

Estructura del Paquete :

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'SHUTDOWN'

El servidor recibe la petición de detener el servicio de colas. Se cambia el valor de la bandera de apagado, la cual toma el valor de uno y se envía la respuesta de éxito.

Todos los hilos del manejador revisan esta bandera dentro de los ciclos de ejecución cuando se encuentran en una región no crítica. Si el valor de la bandera esta en uno, ejecutan un pthread_exit(), lo cual permite que el servidor de colas se detenga de una forma estable. El hilo principal espera la ejecución de todos los hilos por medio de un pthread_join(), y procede a realizar una última actualización de las estructuras a disco.

2. Comandos de usuario

Además de las herramientas administrativas se desarrolló una serie de comandos que permiten hacer uso de las colas.

a. Enviar

El comando enviar se utiliza para escribir un mensaje a la cola.

Estructura del paquete:

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'PUT'
Variable	Nombre	32 *bytes*	Nombre de la cola
	Longitud	5 *bytes*	Longitud del mensajes
	Mensaje	n *bytes*	Texto

El servidor de colas utiliza la función de búsqueda para localizar la estructura de la cola. Se crea una nueva estructura de mensaje en memoria con los valores especificados. Ejecuta un cierre sobre el mutex de la cola, almacena el puntero a la nueva estructura en el arreglo de mensajes y libera el mutex de la cola. Envía la respuesta de éxito.

b. Obtener

El comando obtener se utiliza para leer un mensaje de la cola.

Estructura del paquete:

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'GET'
Variable	Nombre	32 *bytes*	Nombre de la cola

El servidor de colas utiliza la función de búsqueda para localizar la estructura de la cola. Se ejecuta un cierre sobre el mutex de la cola, se extrae el mensaje, y se reorganiza el arreglo de mensajes. Por último se libera el espacio de memoria alocado, libera el mutex y envía la respuesta.

c. Visualizar

El comando visualizar se utiliza para leer el mensaje en determinada posición de la cola sin necesidad de retirarlo de la misma.

Estructura del paquete:

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'BROWSE'
Variable	Nombre	32 *bytes*	Nombre de la cola
	Número	5 *bytes*	Número de mensaje

El servidor de colas utiliza la función de búsqueda para localizar la estructura de la cola. Se ejecuta un cierre sobre el mutex de la cola, se extrae el mensaje en la posición especificada, se libera el mutex y envía la respuesta.

d. Desplegar

El comando desplegar se utiliza para visualizar la estructura de una cola.

Estructura del paquete

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'DISPLAY'
Variable	Nombre	32 *bytes*	Nombre de la cola

Se utiliza la función de búsqueda para localizar la estructura de la cola. Se ejecuta un cierre sobre el mutex de estructuras, se arma una cadena que describa la estructura, y se libera el mutex. Luego se envía la cadena de caracteres armada como respuesta.

e. Contar

El comando contar se utiliza para realizar un conteo de la cantidad de mensajes que hay en una cola.

Estructura del paquete

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'COUNT'
Variable	Nombre	32 *bytes*	Nombre de la cola

El servidor retorna el valor del contador de mensajes de la cola.

f. Purgar

El comando purgar se utiliza para eliminar todos los mensajes de una cola.

Estructura del paquete

Sección	Campo	Longitud	Valor
Fija	MD5	32 *bytes*	MD5 del mensaje
	Sello de tiempo	10 *bytes*	Numérico
	Comando	10 *bytes*	'PURGE'
Variable	Nombre	32 *bytes*	Nombre de la cola

El manejador de colas utiliza la función de búsqueda para localizar la estructura de la cola. Se ejecuta un cierre sobre el mutex de la cola, se libera el espacio de los mensajes, y se actualiza el valor del contador de mensajes a cero. Por último, se libera el uso del mutex y se envía la respuesta de éxito.

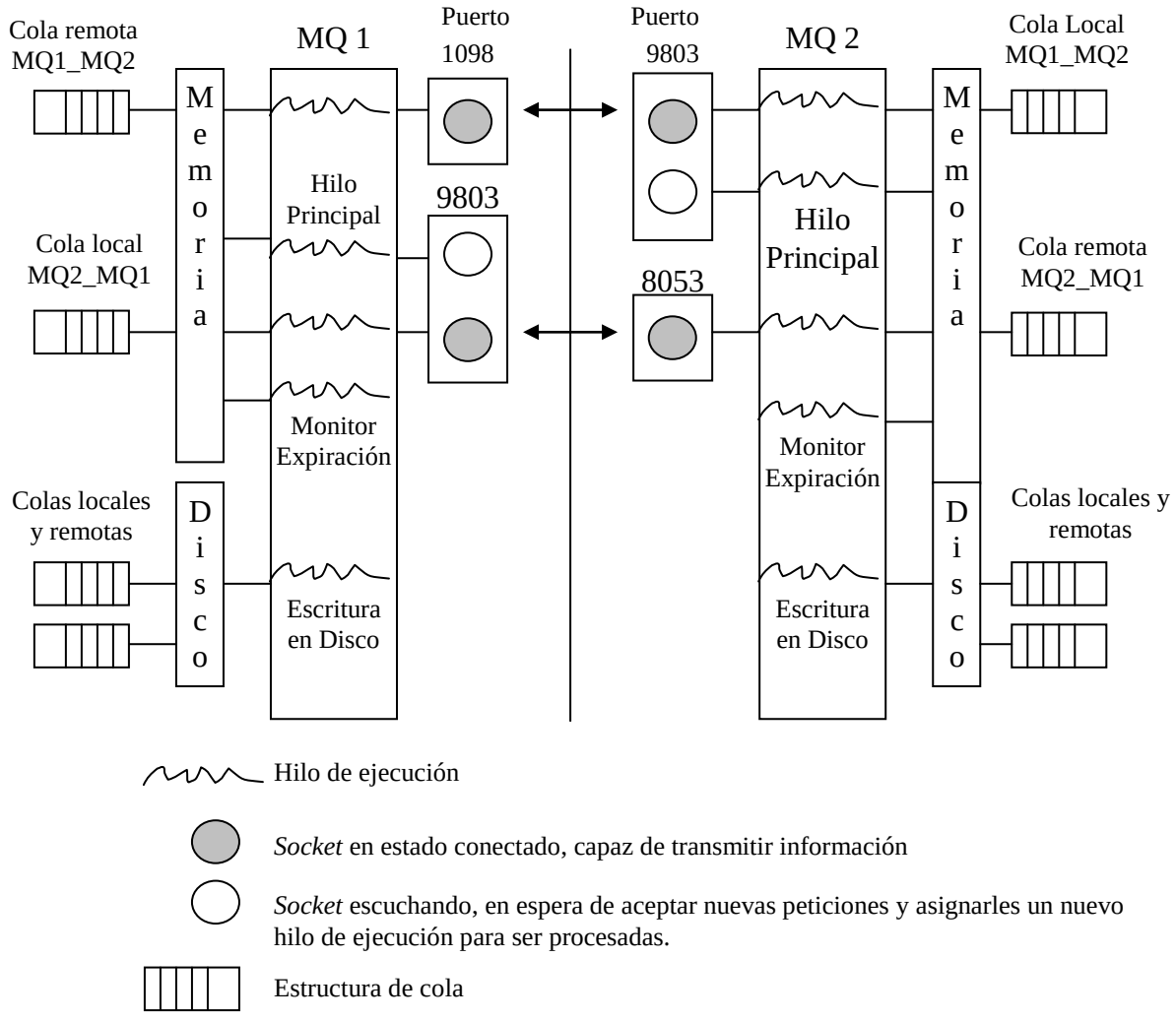
D. Hilos de control de MyOpenQ

Para la realización de diversas tareas internas de funcionamiento de MyOpenQ, se utilizan hilos de ejecución, a los cuales se les ha asignado una tarea específica. Por ejemplo:

- Hilo encargado de monitorear mensajes expirados
- Hilo encargado de almacenar los cambios realizados en memoria a disco

Cada definición de cola remota genera al momento de iniciar el servidor, un hilo de ejecución dedicado, el cual se encarga de levantar un canal de comunicación al manejador de colas especificado en la estructura de la cola. Este hilo revisa constantemente la cola por mensajes pendientes de ser enviados, y efectúa un sched_yield() si no hay mensajes pendientes; de esta forma puede ceder voluntariamente el procesador a otro hilo para que sea aprovechado, mejorando el desempeño de la aplicación.

Ilustración 5. Hilos de control



E. Seguridad de MyOpenQ

Se manejan dos esquemas de seguridad. Permisos de usuario y de administración.

Los permisos de usuario permiten efectuar los comandos de usuario (enviar, obtener, visualizar, desplegar, contar, purgar), y se definen en el archivo user.allow. Para dar permisos a una nueva computadora sobre una cola, se necesita agregar una línea con el formato:

<IP>, <Nombre de la cola>

Los permisos de administración permiten efectuar comandos de administración (crear, borrar, apagar), además de dar acceso de usuario a todas las estructuras definidas en el servidor. Se definen en el archivo admin.allow. Para dar permisos administrativos a una nueva computadora, es necesario agregar una línea con el formato

<IP>

F. Estructuras de información

El manejo de las estructuras y datos se lleva en memoria. Para evitar el acceso simultáneo a las estructuras, se utilizaron semáforos de exclusión mutua que regulan el acceso tanto a las estructuras como a los datos.

Existe un mutex para modificaciones a estructuras, las cuales se realizan por medio de los comandos administrativos.

Para la modificación de información, se utiliza un mutex por cola. Con esto se evita que dos hilos puedan realizar comandos de escritura y lectura a una cola simultáneamente, causando inconsistencia en los datos. Los mutex entre dos colas distintas no intervienen uno con el otro.

La información de las colas se almacena en disco duro por un hilo de control, que se ejecuta periódicamente según el parámetro de escritura del archivo de configuración. La estructura se almacena en el archivo queues.data en formato de texto y contiene los siguientes campos:

Campo	Tamaño
Nombre	32 *bytes*
Tamaño	5 *bytes*
Longitud	5 *bytes*
Tipo	1 *byte*

Campo	Tamaño
IP	15 *bytes*
Puerto	5 *bytes*

Los mensajes que contiene cada cola, son almacenados en un archivo aparte. Dentro del directorio de datos, se genera un archivo plano con el nombre de la cola. En él se almacena la estructura de cada mensaje en formato de texto.

G. Conexión a MQSeries

Para conectar MyOpenQ a MQSeries, se desarrolló un módulo aparte que se llamó MyOpenBridge. Este módulo se encarga de traducir protocolos de otros manejadores de colas, al protocolo que maneja MyOpenQ.

MyOpenBridge es capaz de aprender nuevos protocolos, de forma que casi cualquier manejador de colas puede ser conectado con MyOpenQ.

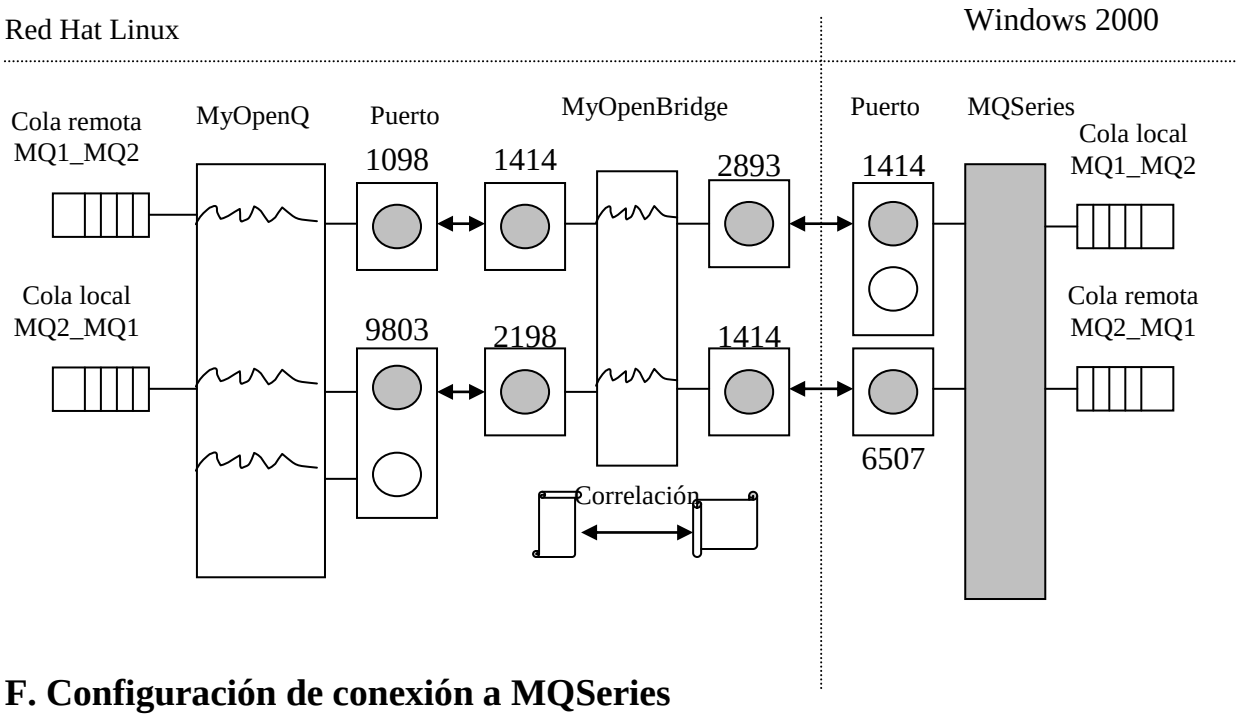
MyOpenBridge realiza una correlación entre la cadena de *bytes* de un mensaje de un protocolo y la estructura de transacciones de MyOpenQ. Esto se logra por medio de un mapa que se configura en el archivo myopenb.conf.

En este archivo se establece una plantilla que permite definir valores estáticos que no cambian entre transacciones del mismo tipo. Además se configuran las posiciones y tamaños en que vienen definidos los valores dinámicos, y con que campos se mapean en la transacción de MyOpenQ.

Cuando se envía un mensaje de MyOpenQ hacia otro manejador de colas, MyOpenBridge toma la plantilla definida para la transacción y arma un mensaje dinámicamente, colocando el valor de los campos variables en las posiciones especificadas. De esta forma, traduce la transacción a una estructura legible por el otro manejador de colas.

Cuando se recibe una transacción hacia MyOpenQ, se realiza el proceso inverso. Para identificar la transacción se utilizan los campos TransactionIndex y TransactionLength, que definen la posición en la cual se puede encontrar el patrón de reconocimiento que se le asocia a la transacción.

Ilustración 6. Esquema de conexión a MQSeries



F. Configuración de conexión a MQSeries

Para efectuar la configuración del traductor hacia MQSeries, fue necesario utilizar un programa espía de paquetes, que permitiera analizar los paquetes TCP que utilizaban dos MQSeries para conectarse entre sí.

Por medio de ingeniería reversa, en base a observación y análisis de las diferencias entre varios paquetes, se dedujo la estructura de cada uno de los paquetes utilizados para posteriormente definir la plantilla de correlación.

1. Paquetes de inicio de canal

Se efectuó la captura de los paquetes de inicio de canal, y se determinó que los paquetes que envía son:

- Una conexión de 3 pasos para entablar una conexión TCP.
- Paquete específico de inicio de canal
- La contraparte envía una respuesta de inicio de canal, con los datos locales.

El paquete de inicio de canal tiene el siguiente formato:

```
0000:  00 50 BA 85 D4 50 00 E0 4C 7A 21 7C 08 00 45 00 .P...P..Lz!|..E.
0010:  00 AC 0A 01 40 00 80 06 6E 45 C0 A8 00 5A C0 A8 ....@...nE...Z..
0020:  00 5B 04 25 05 86 91 4C F5 07 BD 7C 69 D6 50 18 .[.%...L...|i.P.
0030:  44 70 13 A7 00 00 54 53 48 20 00 00 00 84 02 01 Dp....TSH ......
0040:  01 00 00 00 00 00 00 00 00 00 22 02 00 00 B5 01 ..........".....
0050:  00 00 49 44 20 20 07 07 00 00 00 00 32 00 FE 7F ..ID  ......2...
```

```
0060:  00 00 00 00 40 00 FF C9 9A 3B 63 68 2E 63 6C 6F ....@....;ch.clo
0070:  6E 2E 68 70 20 20 20 20 20 20 20 20 20 20 07 00 n.hp          ..
0080:  B5 01 51 4D 5F 63 6D 6F 6C 69 6E 61 20 20 20 20 ..QM_cmolina
0090:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00A0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00B0:  20 20 2C 01 00 00 00 00 73 20                  ,.....s
```

Los primeros 54 *bytes* son el encabezado del protocolo TCP, por lo tanto no se toman en cuenta en el conteo de *bytes*.

Se identificaron los siguientes campos

Campo	Inicio	Longitud	Valor
Nombre del manejador de colas	76	48	QM_cmolina
Nombre del canal	52	20	ch.clon.hp
Tipo de transacción	8	2	0x0201
Bit de dirección	10	1	0x01

El paquete de respuesta al Inicio de canal, tiene el siguiente formato:

```
0000:  00 E0 4C 7A 21 7C 00 50 BA 85 D4 50 08 00 45 00 ..Lz!|.P...P..E.
0010:  00 AC 3D 4F 40 00 80 06 3A F7 C0 A8 00 5B C0 A8 ..=O@...:....[..
0020:  00 5A 05 86 04 25 BD 7C 69 D6 91 4C F5 8B 50 18 .Z...%.|i..L..P.
0030:  43 EC 85 E5 00 00 54 53 48 20 00 00 00 84 02 01 C.....TSH ......
0040:  00 00 00 00 00 00 00 00 00 00 22 02 00 00 B5 01 ..........".....
0050:  00 00 49 44 20 20 07 07 00 00 00 00 32 00 FE 7F ..ID  ......2...
0060:  00 00 00 00 40 00 FF C9 9A 3B 63 68 2E 63 6C 6F ....@....;ch.clo
0070:  6E 2E 68 70 20 20 20 20 20 20 20 20 20 20 07 00 n.hp          ..
0080:  B5 01 51 4D 5F 63 61 72 6C 69 74 6F 73 77 61 79 ..QM_carlitosway
0090:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00A0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00B0:  20 20 2C 01 00 00 00 00 73 20                  ,.....s
```

Los primeros 54 *bytes* son el encabezado del protocolo TCP, por lo tanto no se toman en cuenta en el conteo de *bytes*.

Se identificaron los siguientes campos

Campo	Inicio	Longitud	Valor
Nombre del manejador de colas	76	48	QM_carlitosway
Nombre del canal	52	20	ch.clon.hp
Tipo de transacción	8	2	0x0201
Bit de dirección	10	1	0x00

El *bit* de dirección en este caso viene con valor de 0, para indicar que es una respuesta al inicio de canal.

2. Paquetes de envío de mensajes

Se efectuó la captura de los paquetes de inicio de canal, y se determinó que tienen la siguiente estructura:

```
0000:  00 50 BA 85 D4 50 00 E0 4C 7A 21 7C 08 00 45 00 .P...P..Lz!|..E.
0010:  02 08 A6 E1 40 00 80 06 D0 08 C0 A8 00 5A C0 A8 ....@........Z..
0020:  00 5B 04 93 05 86 82 55 49 58 EF 93 04 E1 50 18 .[.....UIX....P.
0030:  43 EC 76 DA 00 00 54 53 48 20 00 00 01 E0 02 04 C.v...TSH ......
0040:  30 00 5A BA 75 3F 10 00 09 01 22 02 00 00 B5 01 0.Z.u?....".....
0050:  00 00 4D 53 48 20 01 00 00 00 04 00 00 00 00 00 ..MSH ..........
0060:  00 00 B0 01 00 00 58 51 48 20 01 00 00 00 4D 79 ......XQH ....My
0070:  48 50 51 20 20 20 20 20 20 20 20 20 20 20 20 20 HPQ
0080:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0090:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 51 4D                QM
00A0:  5F 63 61 72 6C 69 74 6F 73 77 61 79 20 20 20 20 _carlitosway
00B0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00C0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 4D 44                MD
00D0:  20 20 01 00 00 00 00 00 00 00 08 00 00 00 FF FF   ..............
00E0:  FF FF 00 00 00 00 22 02 00 00 B5 01 00 00 4D 51 ......".......MQ
00F0:  53 54 52 20 20 20 00 00 00 00 00 00 00 00 41 4D STR   ........AM
0100:  51 20 51 4D 5F 63 6D 6F 6C 69 6E 61 20 20 AD B4 Q QM_cmolina  ..
0110:  75 3F 20 00 2B 01 00 00 00 00 00 00 00 00 00 00 u? .+..........
0120:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0130:  00 00 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ..
0140:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0150:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0160:  20 20 51 4D 5F 63 6D 6F 6C 69 6E 61 20 20 20 20   QM_cmolina
0170:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0180:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0190:  20 20 41 20 20 20 20 20 20 20 20 20 20 20 16 01   A           ..
01A0:  05 15 00 00 00 FA 4F 0C 2F 23 F3 F6 63 16 C0 EA ......O./#..c...
01B0:  32 E9 03 00 00 00 00 00 00 00 00 00 00 0B 20 20 2.............
01C0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
01D0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 0B 00                ..
01E0:  00 00 57 65 62 53 70 68 65 72 65 20 4D 51 5C 62 ..WebSphere MQ\b
01F0:  69 6E 5C 61 6D 71 73 70 75 74 2E 65 78 65 32 30 in\amqsput.exe20
0200:  30 33 30 39 32 38 30 31 33 30 34 37 35 39 20 20 03092801304759
0210:  20 20 68 6F 6C 61                                  hola
```

Los primeros 54 *bytes* son el encabezado del protocolo TCP, por lo tanto no se toman en cuenta en el conteo de *bytes*.

Se identificaron los siguientes campos

Campo	Inicio	Longitud	Valor
Tamaño total del mensaje	6	2	X01E0
Tipo de transacción	8	2	0x0204
Longitud del mensaje	36	1	0x04
Nombre de la cola	56	48	MyHPQ
Manejador destino	104	48	QM_cariltosway
Manejador origen	300	48	QM_cmolina

Campo	Inicio	Longitud	Valor
Sello de tiempo	456	16	2003092801304759
Mensaje	476		Hola

3. Correlación de campos

En base al análisis de identificación de campos de las transacciones analizadas, se configuró el mapa de conexión a MQSeries.

La plantilla definida en el archivo myopenb.conf quedo así:

Para la identificación del tipo de transacción:

```
#Template For Incoming Transactions
#Character at which transaction could be identified
TransactionIndex=8

#Length in bytes for matching transaction type
TransactionLength=2

#Patterns for different transactions, hex values
START_CHANNEL=x0201
PUT=x0204
ACK=x0205
```

Para inicio de canal:

```
#Channel Parameters
CH_MANAGER_NAME_POS=76
CH_MANAGER_NAME_LENGTH=48
CH_CHANNEL_NAME_POS=52
CH_CHANNEL_NAME_LENGTH=20
#Template for channel Starting
CH_TEMPLATE=x5453482000000084020101000000000000000000022020000B501000049442020070700000000003
200FE7F000000004000FFC99A3B202020202020202020202020202020202020202020200700B501202020202020202
020202020202020202020202020202020202020202020202020202020202020202020202020202020202020202C0100000
0007320
CH_MESSAGE_LENGTH=132
```

Para envío de mensajes:

```
#Put Parameters
PUT_MESSAGE_SIZE_POS=6
PUT_SRC_MANAGER_NAME_LENGTH=48
PUT_DEST_MANAGER_NAME_POS=104
PUT_QUEUE_NAME_POS=56
PUT_QUEUE_NAME_LENGTH=48
PUT_SRC_MANAGER_NAME_POS=300
PUT_DEST_MANAGER_NAME_LENGTH=48
PUT_MESSAGE_POS=476
PUT_MESSAGE_LENGTH_POS=36
PUT_SELLO DE TIEMPO_POS=456

#Template for Put Message
PUT_TEMPLATE=x54534820000001E0020430005ABA753F1000060722020000B50100004D534820070000000A00
000000000000B60100005851482001000000202020202020202020202020202020202020202020202020202020
202020202020202020202020202020202020202020202020202020202020202020202020202020202020202020
202020202020202020202020202020202020202020202020204D442020010000000000000008000000FFFFFFFF00
```

```
00000022020000B50100004D515354522020200000000000000000414D5120514D5F636D6F6C696E612020ADB4
753F2000150D0000000000000000000000000000000000000000000000000000000020202020202020202020
2020202020202020202020202020202020202020202020202020202020202020202020202020202020202020
2020202020202020202020202020202020202020202020202020202020202020202020202020202020204120202020
2020202020202016010515000000FA4F0C2F23F3F66316C0EA32E9030000000000000000000000B202020202020
2020202020202020202020202020202020202020202020202020200B00000057656253706865726 5204D515C6269
6E5C616D71737075742E6578650000000000000000000000000000303720202020
```

Con esta configuración, se logró conectar MyOpenQ con MQSeries. Para desarrollar plantillas de conexión a otros manejadores de colas, el proceso sería el mismo: capturar paquetes, deducir estructuras, y configurar la plantilla de correlación.

V. RESULTADOS Y DISCUSIÓN

A. Con respecto a los objetivos

Para verificar los resultados de la herramienta, se simuló un escenario similar al de la Ilustración 2. Se desarrolló una pagina en PHP ejecutándose sobre un apache web server; esta página utiliza a MyOpenQ como medio de comunicación para realizar un pago de teléfono en línea. La empresa de telefonía posee un MQSeries.

Estos son los tiempos del desarrollo:

- La configuración de las colas y permisos en el puente tomó 20 minutos.
- El desarrollo de la página en PHP tomó cerca de 10 minutos.
- Pruebas 15 minutos

Total de tiempo empleado en la solución, entre 30 – 60 minutos.

Desarrollar esta solución sin el uso del manejador de colas, y garantizar la entrega del mensaje, aun cuando el servicio de la empresa telefónica no se encuentre disponible, sería una labor que tomaría días.

Otra posible opción sería adquirir un MQSeries, pero entonces los costos del proyecto ascenderían grandemente por la adquisición de este *software*.

B. Resultados de MyOpenBridge

Al enviar el mensaje de conexión, MyOpenBridge generó el paquete de apertura de canal, haciendo que MQSeries levantara el canal de escucha con éxito.

Al realizar el comando enviar a una cola remota de MyOpenQ, se realizó la petición al servidor de MyOpenBridge, el cual generó el paquete de escritura hacia la cola de MQSeries. Inicialmente, solo se lograba que el MQSeries almacenara el primer mensaje enviado a la cola. Se pensó que esto se debía a los seis *bits* de relleno que tenía el mensaje con valores de 0x20 al final del paquete de red, cuando la comunicación era entre dos MQSeries, y que ésta era una especie de seguridad que bloqueaba la recepción de más paquetes.

Por tanto se tomó un rumbo equivocado en el proyecto, y se comenzó a analizar la manera de generar estos seis *bits* de relleno con valor de 0x20, en vez de 0x00, que era como los generaba el *socket* de transmisión.

Se intentó efectuar la comunicación por medio de un *socket* simple, para poder enviar los *bits* deseados, pero esto implicaba una complejidad mayor, ya que se tendrían que realizar todos los ACK y retransmisiones y chequeos de error que ya efectúa el TCP.

Finalmente se descubrió que la causa del error se debía al paquete de aceptación del mensaje de MQSeries, que no estaba siendo enviando. Al generar este paquete, se logró colocar los paquetes siguientes también en la cola de MQSeries, sin necesidad de reabrir el canal.

VI. CONCLUSIONES

- MyOpenQ funciona como un MOM, que permite unir aplicaciones y realizar transacciones en línea de una forma rápida, reduciendo los tiempos de programación.
- MyOpenQ es un proyecto de código abierto y uso libre, por lo que generar transacciones en línea por medio de colas no tendrá costo alguno en lo que respecta a la adquisición de *software* de comunicación.
- MyOpenBridge hace compatible a MyOpenQ con MQSeries, por lo que ahora es posible acoplarse a los estándares del mercado.
- MyOpenQ utiliza hilos de sistema operativo, logrando así un mejor desempeño en arquitecturas multiprocesador; los hilos se pueden ejecutar paralelamente en procesadores distintos.
- MyOpenQ garantiza la portabilidad hacia otros sistemas de la familia Linux y Unix por medio de la utilización de librerías estandarizadas.

VII. RECOMENDACIONES

Se recomienda la utilización de MyOpenQ y MyOpenBridge sobre el sistema operativo Linux. La creación de hilos de sistema operativo y el cambio de contexto en Linux es más eficiente que en otros sistemas operativos. Se decidió utilizar la librería POSIX en esta plataforma, ya que implementa hilos de sistema operativo, pero es posible que en otros sistemas operativos la librería no se comporte de igual forma, y maneje los hilos como si fueran de usuario.

VIII. REFERENCIAS BIBLIOGRAFICAS

- Walton, Sean. *The Linux Documentation Project*. [en línea] Estados Unidos. 21 enero 1997. <http://www.tldp.org/FAQ/Threads-FAQ/> [consulta: 01 junio 2003]

- Universidad Nacional de la Plata. *Sistemas Operativos Threads.* [en línea] Argentina. Mayo 2002. <http://extension.info.unlp.edu.ar/so/files/Threads.doc> [consulta: 10 julio 2003]

- Universidad Politécnica de Cataluña. Programación utilizando C threads *Introducción al concepto de Hilos.* [en línea] España. <http://dsl.upc.es/netscout/doc/tutorial-threads/3_1_introduccion.htm> [consulta 15 junio 2003]

- Garrido Gonzáles, Luis. *Protocolos de Red Protocolo TCP*. [en línea] España. Abril2003. <http://www.garriwp.com/protocolos_red/transporte/tcpip/tcp/> [consulta 23 junio 2003]

- Red Hat Linux. Linux Programmer's Manual *sockets, tcpip, pthread, clone.* Versión 7.0 Manual del Sistema Operativo

- Microsoft Message Queuing (MSMQ) Center. *MSMQ Overview.* [en línea]. Estados Unidos. 28 Febrero 2000. <http://www.microsoft.com/msmq/> [consulta: 28 junio 2003]

- Boston Systems Group. *Open Source Message Queue (OSMQ).* [en línea]. Estados Unidos. 2003. <http://www.osmq.org/> [consulta: 19 junio 2003]

- IBM. *Websphere MQ Classes for Java Messaging System (JMS) [en línea]. Estados Unidos. 2003.* <http://www.ibm.com/software/integration/mqfamily/api/mqjava.html> [consulta: 20 junio 2003]

IX. APÉNDICE

A. Glosario

ACK: Acrónimo utilizado para denominar a los mensajes de reconocimiento.

Cola: Estructura que permite organizar eventos, de forma que el primero que ingresa es el primero que sale de la estructura de almacenamiento.

Datagrama: Unidad de información.

Hilo de ejecución: Es un flujo secuencial de control dentro de un programa. También se denomina proceso ligero.

IP: Protocolo de Internet. Utilizado también para describir la dirección de una máquina en dicho protocolo.

MOM: Clasificación de *software* en la que se agrupan las aplicaciones intermedias utilizadas como medio de comunicación entre dos aplicaciones o más, en la que la mensajería es la base de la comunicación.

POSIX: Estándar que asegura la portabilidad hacia otros sistemas operativos. En el contexto del trabajo se utiliza para la librería POSIX de hilos.

Socket: Identificador que asocia una dirección IP y un número de puerto de comunicación.

TCP: Protocolo de transferencia de control. Encargado del flujo de la transmisión y de asegurarse que los datos estén correctos

VPN: Red privada virtual. Es una red simulada sobre Internet que se establece de una forma segura.

B. Paquetes de comunicación

1. Paquete de inicio de canal.

Generado por MyOpenBridge abriendo una conexión hacia MQSeries.

```
0000:  00 E0 4C 7A 21 7C 00 08 A1 2B F7 0C 08 00 45 00 ..Lz!|...+....E.
0010:  00 B8 05 6F 40 00 40 06 B2 8B C0 A8 00 9B C0 A8 ...o@.@.........
0020:  00 5A 04 02 05 86 E3 C2 D5 2E 5C 27 A7 AC 80 18 .Z........\'....
0030:  7D 78 1C DB 00 00 01 01 08 0A 00 23 3C CC 00 00 }x.........#<...
0040:  00 00 54 53 48 20 00 00 00 84 02 01 01 00 00 00 ..TSH ..........
0050:  00 00 00 00 00 00 22 02 00 00 B5 01 00 00 49 44 ......".......ID
0060:  20 20 07 07 00 00 00 00 32 00 FE 7F 00 00 00 00   ......2.......
0070:  40 00 FF C9 9A 3B 63 68 2E 6D 79 6F 70 65 6E 71 @....;ch.myopenq
0080:  2E 63 6C 6F 6E 20 20 20 20 20 07 00 B5 01 4D 79 .clon     ....My
0090:  4F 70 65 6E 51 20 20 20 20 20 20 20 20 20 20 20 OpenQ
00A0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00B0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 2C 01              ,.
00C0:  00 00 00 00 73 20                              ....s
```

2. Paquete de envío

Generado por MyOpenBridge escribiendo un mensaje hacia una cola de MQSeries.

```
0000:  00 E0 4C 7A 21 7C 00 08 A1 2B F7 0C 08 00 45 00 ..Lz!|...+....E.
0010:  02 19 05 73 40 00 40 06 B1 26 C0 A8 00 9B C0 A8 ...s@.@..&......
0020:  00 5A 04 02 05 86 E3 C2 D5 B2 5C 27 A8 30 80 18 .Z........\'.0..
0030:  7D 78 32 EE 00 00 01 01 08 0A 00 23 3D 9A 00 04 }x2........#=...
0040:  7A FE 54 53 48 20 00 00 01 E5 02 04 30 00 5A BA z.TSH ......0.Z.
```

```
0050:  75 3F 10 00 06 07 22 02 00 00 B5 01 00 00 4D 53 u?....".......MS
0060:  48 20 07 00 00 00 09 00 00 00 00 00 00 00 B6 01 H ..............
0070:  00 00 58 51 48 20 01 00 00 00 4D 79 4C 6F 63 61 ..XQH ....MyLoca
0080:  6C 51 20 20 20 20 20 20 20 20 20 20 20 20 20 20 lQ
0090:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00A0:  20 20 20 20 20 20 20 20 20 20 51 4D 5F 63 6D 6F           QM_cmo
00B0:  6C 69 6E 61 20 20 20 20 20 20 20 20 20 20 20 20 lina
00C0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00D0:  20 20 20 20 20 20 20 20 20 20 4D 44 20 20 01 00           MD  ..
00E0:  00 00 00 00 00 00 08 00 00 00 FF FF FF FF 00 00 ................
00F0:  00 00 22 02 00 00 B5 01 00 00 4D 51 53 54 52 20 ..".......MQSTR
0100:  20 20 00 00 00 00 00 00 00 00 41 4D 51 20 51 4D   ........AMQ QM
0110:  5F 63 6D 6F 6C 69 6E 61 20 20 AD B4 75 3F 20 00 _cmolina  ..u? .
0120:  15 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0130:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 20 ..............
0140:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0150:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0160:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 4D 79               My
0170:  4F 70 65 6E 51 20 20 20 20 20 20 20 20 20 20 20 OpenQ
0180:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0190:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 41 20               A
01A0:  20 20 20 20 20 20 20 20 20 20 16 01 05 15 00 00           ......
01B0:  00 FA 4F 0C 2F 23 F3 F6 63 16 C0 EA 32 E9 03 00 ..O./#..c...2...
01C0:  00 00 00 00 00 00 00 00 00 0B 20 20 20 20 20 20 ..........
01D0:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
01E0:  20 20 20 20 20 20 20 20 20 20 0B 00 00 00 57 65           ....We
01F0:  62 53 70 68 65 72 65 20 4D 51 5C 62 69 6E 5C 61 bSphere MQ\bin\a
0200:  6D 71 73 70 75 74 2E 65 78 65 32 30 30 33 30 39 mqsput.exe200309
0210:  32 37 31 38 32 35 30 31 30 37 20 20 20 20 4D 65 2718250107    Me
0220:  6E 73 61 6A 65 20 30                            nsaje 0
```

3. Paquete de reconocimiento

Este paquete lo genera MQSeries cuando recibe un mensaje, se debe contestar con otro paquete similar (diferenciado únicamente por el ***bit* de dirección**) para poder continuar con la escritura del siguiente mensaje a la cola.

```
0000:  00 08 A1 2B F7 0C 00 E0 4C 7A 21 7C 08 00 45 00 ...+....Lz!|..E.
0010:  00 44 CD B2 40 00 80 06 AA BB C0 A8 00 5A C0 A8 .D..@........Z..
0020:  00 9B 05 14 05 86 F6 E2 A6 19 E9 B0 5A 07 50 18 ............Z.P.
0030:  43 EC 6E 88 00 00 54 53 48 20 00 00 00 1C 02 05 C.n...TSH ......
0040:  01 00 84 CE 6D 3F 10 00 17 01 22 02 00 00 B5 01 ....m?...."..... 
0050:  00 00                                           ..
```

C. Manual de Comandos

1. Crear

El comando crear se utiliza para la creación de colas tanto locales como remotas.

La sintaxis del comando es:

create -s [server_ip] -p [server_port] -size [size] -timeout [timeout] -ml [length] -t [type] -rs [remote_server_ip] -rp [remote_port] -q queue_name

Los parámetros entre corchetes no son requeridos.

-s server_ip: Especifica la dirección IP del servidor en el que se desea crear la cola.

Valor predeterminado: 127.0.0.1

-p server_port: Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones.
Valor predeterminado: 9803

-size size: Un número entero, que define el número máximo de mensajes que la cola puede retener. Cuando la cantidad de mensajes pendientes en la cola es igual a este número, los mensajes siguientes serán rechazados, hasta que se disponga de espacio al retirar un mensaje de la cola.
Valor predeterminado: 1024.

-t timeout: Un número entero, que define el tiempo en segundos que el mensaje permanecerá en la cola antes de ser eliminado. Para que el mensaje permanezca en la cola indefinidamente, el timeout deberá tener el valor 0.
Valor predeterminado: 0.

-ml length: Un número entero, menor a 1024 (o el valor con que se haya compilado en la variable _MESSAGE_MAX_LENGTH), que define la longitud máxima que puede tener un mensaje.
Valor predeterminado: 1024.

-t type: "local" o "remote". Especifica el tipo de cola a crear. Una cola local es en la que se escribe mensajes y éstos se quedan en el servidor de colas local. En una cola remota, los mensajes escritos viajan a una cola local que esta definida en otro manejador de colas.

En caso el tipo de la cola sea remoto, se deberán de definir los siguientes parámetros adicionales:

-rs remote_ip: Dirección IP del manejador de colas al cual viajarán los mensajes escritos en esta cola.

-rp remote_port: Puerto en que el manejador de colas remoto recibe peticiones.

-q queue_name: Nombre con el cual se identificará la cola. Cadena menor a 64 *bytes*.

En caso de éxito devuelve:

SUCCESS

Códigos de error:

MOQ – 1001 El nombre de la cola es inválido.

MOQ – 1002 El número máximo de mensajes especificado no es válido.

MOQ – 1004 Ya existe una definición de cola con ese nombre.

MOQ – 3001 El número máximo de colas se ha alcanzado. Por lo que no se puede crear otra cola.

2. Borrar

El comando borrar se utiliza para la eliminación de la definición de colas locales o remotas.

La sintáxis del comando es:

delete -s [server_ip] -p [server_port] -q queue_name

Los parámetros entre corchetes no son requeridos.

-s server_ip: Especifica la dirección IP del servidor de colas.
Valor predeterminado: 127.0.0.1

-p server_port: Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones.
Valor predeterminado: 9803

-q queue_name: Nombre de la definición de cola que se desea crear.

En caso de éxito devuelve:

SUCCESS

Códigos de error:

MOQ – 1003 La cola especificada no existe.

MOQ – 4001 No posee permisos para efectuar el comando.

3. Apagar

El comando apagar se utiliza para detener el servicio de colas.

La sintáxis del comando es:

shutdown -s [server_ip] -p [server_port]

Los parámetros entre corchetes no son requeridos.

-s server_ip: Especifica la dirección IP del servidor en el que se desea detener. Valor predeterminado: 127.0.0.1

-p server_port: Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones.
Valor predeterminado: 9803

En caso de éxito devuelve:

SUCCESS

Códigos de error:

MOQ – 4001 No posee permisos para efectuar el comando.

4. Enviar

El comando enviar se utiliza para escribir un mensaje a la cola.

Estructura del paquete

La sintáxis del comando es:

put -s [server_ip] -p [server_port] -q queue_name -m message

Los parámetros entre corchetes no son requeridos.

-s server_ip: Especifica la dirección IP del servidor al que se desea enviar el mensaje. Valor predeterminado: 127.0.0.1

-p server_port: Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones.
Valor predeterminado: 9803

-q queue_name: Nombre de la cola en la que se colocará el mensaje.

En caso de éxito devuelve:

SUCCESS

Códigos de error:

MOQ - 2001 El tamaño del mensaje excede la longitud máxima establecida por la cola.

MOQ – 4001 No posee permisos para efectuar el comando.

5. Obtener

El comando obtenet se utiliza para leer un mensaje de la cola.

La sintáxis del comando es:

get -s [server_ip] -p [server_port] -q queue_name -m message

Los parámetros entre corchetes no son requeridos.

-s server_ip: Especifica la dirección IP del servidor del que se desea obtener el mensaje.
Valor predeterminado: 127.0.0.1

-p server_port: Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones.
Valor predeterminado: 9803

-q queue_name: Nombre de la cola de la cual se leerá el mensaje.

En caso de éxito devuelve:

El texto del mensaje que retiro de la cola.

Códigos de error:

MOQ - 1003 La cola especificada no se encontró.

MOQ – 1007 Mensaje no encontrado

MOQ – 4001 No posee permisos para efectuar el comando.

6. Visualizar

El comando visualizar se utiliza para leer el mensaje en determinada posición de la cola sin necesidad de retirarlo de la misma.

La sintáxis del comando es:

browse: -s [server_ip] -p [server_port] -n [number] -q queue_name

Los parámetros entre corchetes no son requeridos.

-s server_ip: Especifica la dirección IP del servidor del que se desea leer el mensaje.
Valor predeterminado: 127.0.0.1

-p server_port: Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones.
Valor predeterminado: 9803

-q queue_name: Nombre de la cola a consultar.

-n number: Especifica el número de mensaje a leer de la cola.
Valor predeterminado: 1.

En caso de éxito devuelve:

El texto del mensaje en esa posición.

Códigos de error:

MOQ - 1003 La cola especificada no se encontró.

MOQ – 1007 Mensaje no encontrado

MOQ – 4001 No posee permisos para efectuar el comando.

7. Desplegar

El comando desplegar se utiliza para visualizar la estructura de una cola.

Estructura del paquete

La sintáxis del comando es:

display: -s [server_ip] -p [server_port] -q queue_name

Los parámetros entre corchetes no son requeridos.

-s server_ip: Especifica la dirección IP del servidor en el que esta definida la cola.
Valor predeterminado: 127.0.0.1

-p server_port: Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones.
Valor predeterminado: 9803

-q queue_name: Nombre de la cola a consultar.

En caso de éxito devuelve:

Texto describiendo la estructura de la cola.

Códigos de error:

MOQ - 1003 La cola especificada no se encontró.

MOQ – 4001 No posee permisos para efectuar el comando.

8. Contar

El comando contar se utiliza para realizar un conteo de la cantidad de mensajes que hay en una cola.

La sintáxis del comando es:

count: -s [server_ip] -p [server_port] -q queue_name

Los parámetros entre corchetes no son requeridos.

-s server_ip:	Especifica la dirección IP del servidor en el que esta definida la cola. Valor predeterminado: 127.0.0.1
-p server_port:	Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones. Valor predeterminado: 9803
-q queue_name:	Nombre de la cola a consultar.

En caso de éxito devuelve:

Texto describiendo la estructura de la cola.

Códigos de error:

MOQ - 1003 La cola especificada no se encontró.

MOQ – 4001 No posee permisos para efectuar el comando.

9. Purgar

El comando purgar se utiliza para eliminar todos los mensajes de una cola.

La sintáxis del comando es:

purge -s [server_ip] -p [server_port] -q queue_name

Los parámetros entre corchetes no son requeridos.

-s server_ip:	Especifica la dirección IP del servidor en el que esta definida la cola a borrar. Valor predeterminado: 127.0.0.1
-p server_port:	Un número entero, que especifica el puerto en el que el manejador de colas recibe las peticiones. Valor predeterminado: 9803
-q queue_name:	Nombre de la cola.

En caso de éxito devuelve:

SUCCESS

Códigos de error:

MOQ - 1003 La cola especificada no se encontró.

MOQ – 4001 No posee permisos para efectuar el comando.