Obligatorio 2

Introducción a las Redes de Computadoras

Grupo 26

Sebastian Caggiano – 4.737.478

Fabian Rodriguez – 4.123.587

Sebastian Suttner – 4.216.541

24/04/2011

Contenido

1. Descripción del documento

Se presenta con este documento las características más relevantes de nuestro sistema implementado, cumpliendo con lo pedido en el obligatorio. A continuación se presenta una breve descripción de las secciones redactadas.

Implementación

Estructuras definidas: Se describen las variables y tipos de datos definidos con el fin de lograr el almacenamiento de datos configurados a través de la administración del servidor proxy.

Concurrencia: Se describe de qué manera se diseñó la realización de la mutua-exclusión en el sistema.

Librerías: Se detallan las librerías que se utilizó en la implementación, y las operaciones primitivas de las mismas.

Diseño

Diseño funcional: Se describe el funcionamiento general del servidor frente a las peticiones que se le pidió que implemente.

Diagrama de estados: Se muestra un diagrama de los estados y transiciones frente a estímulos externos por los que se convierten los distintos componentes del sistema frente.

Pseudocódigo: Se realiza el pseudocódigo diseñado en el componente central del sistema, el cuál es el encargado de realizar la acción y respuesta en cuanto a las peticiones realizadas por los usuarios o administradores.

Extras

Se describe características secundarias del sistema. Es decir, aquellas en las que no determinan el funcionamiento del sistema, pero sí establece una buena organización y correcta implementación del sistema.

2. Implementación

2.1 Estructura definida

A continuación se detallan las estructuras en la cuál se almacena las configuraciones de la administración del servidor proxy.

```
bool booldenyPOST;
bool booldenyGET;
list<string>  listaDW;
list<string>  listaDUW;
```

Descripción:

boolDenyPOST: Variable a nivel global (única en el sistema, donde todo hilo de atención del servidor accederá a la misma). En ella se almacenará el valor correspondiente a verdadero, si actualmente se permite el pedido http de tipo *POST*, falso en caso contrario.

boolDenyGET: Misma descripción que la variable *boolDenyPOST,* con diferencia que corresponde al pedido http de tipo *GET*.

listaDW: Variable a nivel global de tipo lista (única en el sistema, donde todo hilo de atención del servidor accederá a la misma). En ella se almacena aquellas palabras en la cual se deberá filtrar en el contenido de las páginas web requeridas por el explorador web.

listaDUW: Variable a nivel global de tipo lista (única en el sistema, donde todo hilo de atención del servidor accederá a la misma). En ella se almacena aquellas palabras en la cual se deberá bloquear la respuesta del pedido http recibido, si la url del mismo, contiene alguna de las dichas palabras.

2.2 Concurrencia

Luego de analizar las configuraciones que se pueden administrar en el servidor, y debido a que el mismo deberá soportar la atención a N peticiones concurrentemente (sean peticiones administrativas o web). Se decidió crear 4 semáforos binarios, con el fin de que el sistema cuente con una consistencia en los valores de los datos de la configuración administrativa. Se nombran a continuación:

```
pthread_mutex_t mutexListDW   = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexListDUW  = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexdenyPOST = PTHREAD_MUTEX_INITIALIZER;
```

```
pthread_mutex_t mutexdenyGET  = PTHREAD_MUTEX_INITIALIZER;
```

Descripción

mutexListDW: Se utiliza para realizar una mutua-exclusión en el acceso de la lista *listaDW*. Ya que para los distintos hilos de administrador, se debe permitir manipular la lista únicamente de a 1 hilo a la vez, y para los hilos de petición web, no se debe estar modificando la lista mientras se lista sus valores. Lo cuál quiere decir que éste semáforo, formará parte de las operaciones administrativas de: listado de lista, agregar o quitar palabra. Vale aclarar que se pudo haber realizado un algoritmo de Escritor-Lectores para el caso en que se acceda a la lista únicamente para listar, pero debido al tiempo adicional que nos demandaría implementar dicho algoritmo, se decidió no implementarlo.

mutexListDUW: Misma descripción que el semáforo *mutexListDW,* con la diferencia que se aplica sobre la lista *listaDUW*.

mutexDenyPOST: Se utiliza para realizar la mutua-exclusión entre los hilos de administrador y peticiones web tanto en el acceso a consulta (por parte del hilo de petición web) o modificación (por parte del hilo de administración) del valor de configuración de permitir o bloquear el método http de tipo *POST*.

mutexDenyGET: Misma descripción que el semáforo *mutexDenyPOST,* con la diferencia que se aplica sobre el método de tipo *GET*.

2.3 Librerías

sys/socket.h: Librería basada en C, para la implementación del tipo abstracto *socket.* De la cual se utilizaron las siguientes funciones y tipo de datos:

socket(): Función para la creación de un *socket*, especificando las configuraciones del mismo mediante banderas como parámetros en la función.

struct sockaddr: Estructura en la que se configuran los datos a utilizar por el *socket,* entre ellas, la dirección en la que realizará la escucha.

bind(): Función que realiza el enlace entre un *socket* y la estructura *sockaddr*.

listen(): Función que realiza la escucha de un socket, utilizando la información enlazada al mismo previamente.

accept(): Función que realiza la aceptación de pedido en el que un socket se encontraba realizando la escucha en esa dirección.

close(): Función que cierra el socket.

pthread.h: Librería basada en C, para el manejo de hilos en el sistema operativo. De la cual se utilizaron las siguientes funciones y tipo de datos:

pthread_t: Tipo de datos, representando un hilo.

pthread_attr_init: Función que inicializa los atributos iniciales de un hilo.

pthread_attr_setdetachstate: Funcion en la que se le indica al hilo que no pase a estado *zombie* una vez que se le indique su fin de ejecución.

pthread_create(): Función para la creación de un hilo, indicándosele la línea en la que comenzará a ejecutar.

pthread_exit(): Función en la que se le indica al hilo su fin de ejecución.

pthread_mutex_t: Tipo de datos, representando un semáforo.

pthread_mutex_lock: Función que realiza el bloqueo de un semáforo.

pthread_mutex_unlock: Función que realiza el desbloqueo de un semáforo.

string.h: Se utilizó esta librería para el manejo de comandos del administrador, las palabras restringidas por la administración, y para la implementación del archivo de log.

3. Diseño

Al realizar el diseño funcional, lo primero que destacamos como primordial a tener en cuenta, fue los 2 roles que cumpliría el servidor proxy, que se mencionan y describen a continuación.

Servidor: Servidor para el agente web (explorador web) y para el administrador. Para el agente web debido a que será el encargado de recibir las peticiones web creadas por el explorador, y devolver una respuesta, dependiendo la misma, de la configuración en la que se encuentre el servidor en ese momento. Y para el administrador, debido a que será el encargado de implementar la atención a las operaciones administrativas, y brindar respuesta al cliente (administrador).

Cliente: Actua como cliente, ya que pedirá la respuesta http al servidor web correspondiente, dicho servidor es el especificado en el destino del pedido que recibió el servidor proxy mandado por el agente web.

A continuación se específica las invocaciones que debe llevar a cabo un sistema el cuál será servidor, basado en el tipo abstracto *socket.*

- Crear un socket mediante el system call socket().
- Realizar bind() del *socket*, esto se realiza para que el sistema operativo asocie el *socket* a nuestro programa.
- Notificar al sistema operativo que comience a “escuchar” por conexiones en el *socket* mediante la función listen().

- Aceptar conexiones, se realiza mediante la función accept().
- Enviar y recibir datos mediante las funciones send() y recv ().
- Cerrar la conexion por medio de la funcion close()

También se detalla a continuación los pasos para realizar como un cliente web:

- Crear un *socket* con la system call socket().
- Solicitar conexión con el servidor mediante la función connect(), a la misma hay que proveerle el IP del servidor y el puerto al que nos queremos conectar.
- Enviar y recibir datos mediante la función send() y recv().
- Cerrar la conexión por medio de la función close().

Como ya se mencionó, el servidor proxy actuará de servidor y cliente web al mismo tiempo. La solución tomada en el diseño de la implementación para realizar lo mencionado, se explica a continuación:

El programa principal crea un hilo que funciona como servidor para los administradores. El mismo crea un *socket* y comienza a aceptar conexiones, si la conexión se creó con éxito el servidor crea un hilo para atender la misma, este hilo se encargará de recibir el mensaje enviado por el administrador, interpretarlo, y ejecutar la operación interpretada. En caso de éxito, notifica al administrador con un mensaje de éxito. Así como también si el mensaje es inválido, se le notifica al administrador del hecho.

Por otro lado el programa principal funciona como un servidor para los clientes web, creando un *socket*, y aceptando conexiones. Por cada conexión aceptada se crea un hilo que se encargará de atender el pedido.

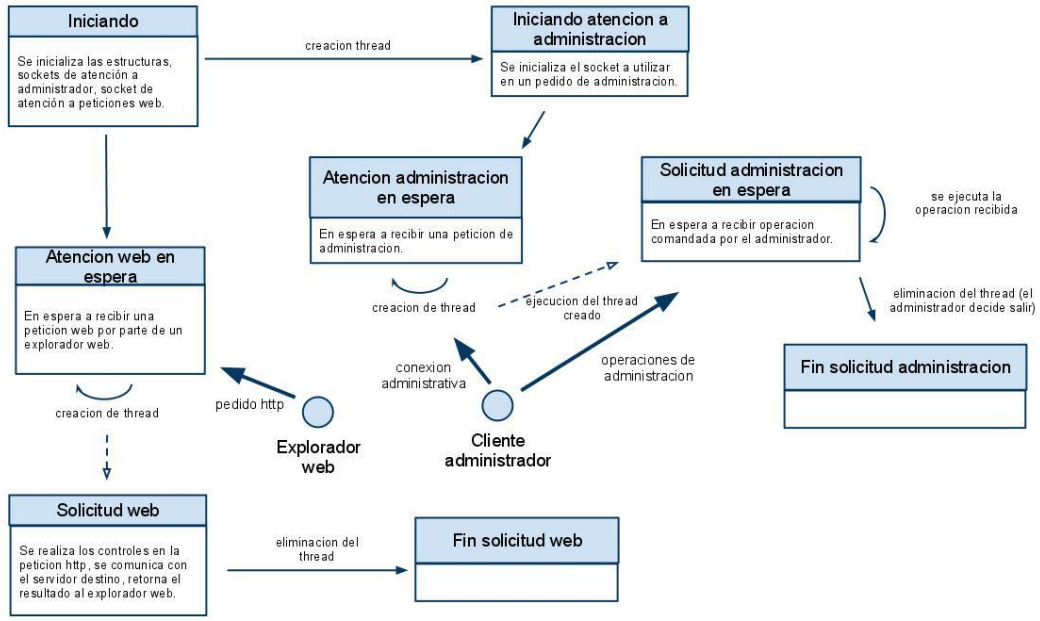
Este hilo recibe el pedido y lo procesa. En primer lugar verifica que el método y la url sean válidos, en caso de no serlo se le envía el mensaje de error correspondiente al cliente web. Si resultan validos extrae el nombre del host de la url, cambia el tipo de protocolo de HTTP/1.1 a HTTP/1.0 y elimina los encabezados que corresponden específicamente a HTTP/1.1.

Luego con el nombre del host realiza una búsqueda DNS para obtener el IP del servidor y crea un socket con el IP obtenido y el puerto 80 (puerto web) que configuramos por defecto ya que los pedidos web se atienden en este puerto. En el caso particular de que el puerto no sea el 80 el mismo debería ser extraído del host, ya que es un parámetro opcional del mismo.

El hilo se conecta al servidor y realiza el pedido. Nuestro diseño de implementación en esta parte consiste en esperar a recibir todo el pedido del servidor web para luego enviárselo al cliente, otra alternativa seria que a medida que se reciben datos enviarlos directamente al cliente. Lo realizamos de esta manera ya que el servidor web puede enviar la respuesta por partes y por lo tanto al recibir una parte de la respuesta puede quedar una palabra cortada en la misma y por lo tanto no podríamos saber si la misma es una de las palabras a filtrar de la página o no.

Una vez recibido por completo el mensaje, se filtran las palabras que estén prohibidas por la administración, cierra la conexión con el servidor web, le envía al cliente la respuesta y cierra la conexión con el cliente. Para eliminar las palabras decidimos sustituir cada carácter de la palabra por un *. Lo hicimos de esta forma de manera que el usuario visualice que se realizó el filtrado por parte del proxy de determinadas palabras, estando las mismas en la lista de palabras prohibidas a mostrar.

4. Diagrama de estados



5. Pseudocódigo

Pseudocódigo del servidor proxy:

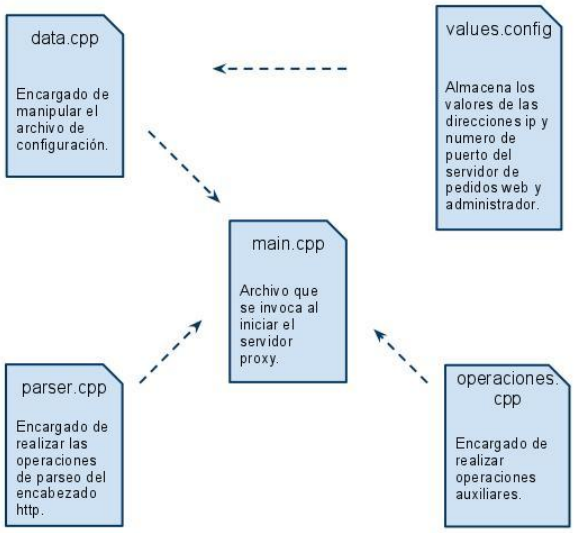
```
Main()
Begin
        obtenerPuertoIpServidorConfigurado()      //se obtiene desde el archivo de configuracion
        socket_web := crearSocketPeticionWeb
        crearHilo_AtencionAdministracion()
        while (true)
        Begin
                aceptarSolicitudWeb(socket_web)
                crearHilo_Web(socket_web)
        End
End

AtencionAdministracion()
Begin
        obtenerPuertoIpAdministracionConfigurado()//se obtiene desde el archivo de configuracion
```

```
        socket_admin:= crearSocketAdministracion();
        while(true)
        Begin
                aceptarSolicitudAdministracion(socket_admin);
                crearHilo_Administracion();
        End
End

Administracion()
Begin
        while(comando!=salir)
        Begin
                consultoComando();
                comienzoMutuaExclusion();
                ejecutoAccionComando();
                finalizoMutuaExclusion();
                notificoAdministrador();
        End
End

Web()
Begin
        recibirMensajeCompleto()          //bifurcación para obtener el mensaje completo
        método:=obtenerMetodoHttp()
        si (esValidoMetodo(metodo)) //valido si es GET,POST o HEAD, y está permitido desde admin.
        Begin
                modificarMensajeAHttp1.0()
                url:=obtenerUrl ()
                if (verificarUrlValida(url)) //valida si las palabras contenidas en la url no están
                bloqueadas por admin.
                Begin
                        mandoPedidoAServidor()
                        modificoRespuestaConPalabrasRestringidas()//palabras en el contenido html
                        restringidas desde admin.
                        mandarRespuestaAExploradorWeb
                End
                        Else
                        Begin
                                mandarAvisoUrlNoValidaAExploradorWeb()
                        End
        End
        Else
        Begin
                mandarAvisoMetodoNoValidoAExploradorWeb();
        End
End
```

6. Extras

6.1 Estructura de archivos

A continuación se muestra un diagrama de la estructura en conjunto de los archivos del sistema.



6.2 Funcionalidades extras

Manejador Ctrl +C: para soportar el fin de ejecución del servidor proxy mediante Ctr +C, se debió implementar la captura de tal evento, con el fin de cerrar los *sockets* abiertos por el sistema, debido a que de lo contrario, el sistema terminaría su ejecución pero permanecerían abiertos los *sockets* en los que el sistema realizaba la escucha, o se encontraba transmitiendo datos.

Archivo de log: Se cuenta con un archivo de información (log) en el cuál se van almacenando mensajes realizados en el archivo principal del sistema (main.cpp) junto con la hora actual. De esta manera se puede realizar un rastreo de lo ejecutado por el servidor, tanto para el mantenimiento o corrección de problemas.

6.3 dificultades encontradas

Durante el desarrollo del obligatorio nos encontramos con diversas dificultades. Dentro de las mas complejas estuvieron el manejo de memoria y los threads. Dado que se es un lenguaje de bajo nivel, tuvimos problemas para el manejo de la memoria, sobre todo con los char * y los strings ya que los mismos terminan en ¨\0¨ y al recibir un pedido, si el mismo es una imagen, la misma puede contener ese caracter y por lo tanto la imagen puede mandarse incompleta. Para solucionar este problema tuvimos que utilizar la función memcpy que copia bytes de una dirección a otra.

Por otro lado uno de los problemas que nos llevo mas tiempo resolver estaba relacionado con los threads y los socket descriptors. El sistema operativo setea por defecto

un numero de file descriptors y threads que un programa puede crear, y nuestro problema consistía en que después de navegar por varias paginas y utilizar algunos administradores, nos dejaba crear nuevas conexiones (en el wireshark aparecia el 3-way.handshake) pero no se creaban los hilos de atención de los mismos. Después de investigar bastante sobre el tema encontramos que los threads por defecto se crean ´´joinable´´, lo cual quiere decir que una vez que se mata el thread el mismo queda en estado zombie esperando a que el programa que lo invoco termine su ejecución. Al quedar en estado zombie, no se eliminaba el thread y por lo tanto en algún momento alcanzábamos el máximo de threads permitidos por el sistema operativo, por lo tanto la rutina de atención del pedido no se creaba y quedaba ¨colgado¨ el navegador.