Universidad Tecnológica Nacional – Facultad Regional Buenos Aires
Ingeniería en Sistemas de Información
Sistemas Operativos (95-2027)

Sistemas Operativos

Trabajo Práctico Nro. 2 - 1er Cuatrimestre 2009

Motor de Indexación y Búsqueda de Documentos

Revisión 0.7

Índice

1.-Introducción

El trabajo práctico de este cuatrimestre consiste en el desarrollo de un sistema distribuido el cual intenta emular el comportamiento de un buscador de páginas Web, así como los distintos servidores de páginas distribuidos en una gran red.

La idea principal se basa en diseñar una aproximación a un motor de búsqueda en menor escala el cual estará formado por varios componentes distribuidos que tendrán una función específica y que existirán en diferentes plataformas.

Por otro lado se desarrollará una versión modificada de un Web Server ó servidor de páginas Web que permita alojar tanto páginas HTML como cualquier otro tipo de documentos en un directorio para luego ser consultado ó descargado por un usuario que utilice un browser.

Se enumeran a continuación los conceptos teóricos y prácticos más significativos sobre sistemas operativos que cubre el trabajo práctico y que el alumno aprenderá:

- Creación y manipulación de procesos y threads (hilos) mediante la API del sistema.
- Sincronización de Procesos, IPC y manejo de señales.
- Administración de Memoria mediante algoritmos de selección de víctima.
- Manejo del Filesystem a través de la interfaz de sistema.
- Sistemas Distribuidos.
- Arquitecturas basadas en protocolos y mensajería.
- Diferencias entre los Sistemas Operativos más populares.

2.-Objetivos del Trabajo Práctico

- Que los alumnos adquieran los conocimientos prácticos del uso de un conjunto de herramientas que ofrecen los sistemas operativos modernos.
- Que entiendan la importancia de una norma o protocolo estándar en la comunicación entre procesos y diferentes plataformas.
- Que dominen los problemas específicos de este tipo de implementaciones.
- Que apliquen en forma práctica el uso de lenguaje C en implementaciones de bajo nivel.
- Que el grupo de alumnos aprendan el trabajo en equipo, las problemáticas y las responsabilidades que eso implica.

3.-Características del Sistema Distribuido

A continuación se hace una breve introducción a cada uno de los componentes que intervienen en el sistema. Ver diagrama de distribución física.

Motor de Búsqueda (Search Engine)

El buscador será el componente principal de este sistema y estará a cargo de varias funciones:

En primer lugar le deberá permitir a un usuario de nuestra red buscar algún tipo de recurso, ya sea una página Web, un archivo de imagen ó algún otro tipo de archivo.
Para llevar a cabo este proceso de búsqueda se le presentará al browser una página HTML con un formulario donde el usuario introduce una ó varias palabras clave a buscar, así como la opción del tipo de contenido. Estas palabras son interpretadas por un proceso llamado *Query Processor* el cual será encargado de consultar el directorio de datos del buscador. Aquí se almacenarán las relaciones entre palabras y direcciones de documentos donde aparecen esas palabras. Una vez recuperados los resultados, este proceso los transformará en código HTML y se los enviará al browser.

Al mismo tiempo, este motor de búsqueda tendrá una tarea muy importante; la de mantener actualizado la base de datos ó directorio con toda la información que exista en la red**.** Para llevar a cabo este complejo proceso, el buscador le delegará la tarea a un proceso llamado *Crawler* se irá duplicándose por cada uno de los servidores de contenido que existan en la red, analizando los recursos existentes y enviando la información procesada de nuevo al buscador.
Por último, existe un proceso que estará a la espera de la información que el Crawler genera. A este proceso se lo conoce como *Web Store.* Este será el que recibirá la lista de relaciones entre palabras clave y dirección asociada, para luego almacenarlas en el directorio.

Dado que el buscador tendrá una cantidad alta de usuarios consultando por documentos, este dispondrá de un balanceador de carga (Load Balancing) para alivianar la carga de trabajo del *Query Processor*.

Servidor Web (Web Server)

La función principal de este servidor es aceptar pedidos de clientes (en nuestro caso serán Web browsers) y servirles los archivos solicitados. Estos documentos que el cliente solicita no solo serán páginas Web sino que también serán imágenes ó cualquier otro tipo de archivo.
Este proceso crece en complejidad ya que tendrá la capacidad de manejar concurrentemente cierto número de clientes. Ya sea para consultar un documento HTML ó como para descargar contenido del servidor. Todo este contenido lo mantendrá almacenado en un directorio local el cual se podrá actualizar en cualquier momento por el administrador del servidor sin necesidad de ponerlo fuera de servicio.

Una diferencia importante de este proceso es que dispondrá un módulo especialmente diseñado para colaborar con el motor de búsqueda. Mediante un puerto especial, recibirá la orden de crear una instancia del llamado *Crawler* para realizar el análisis de los archivos.

Web Crawler

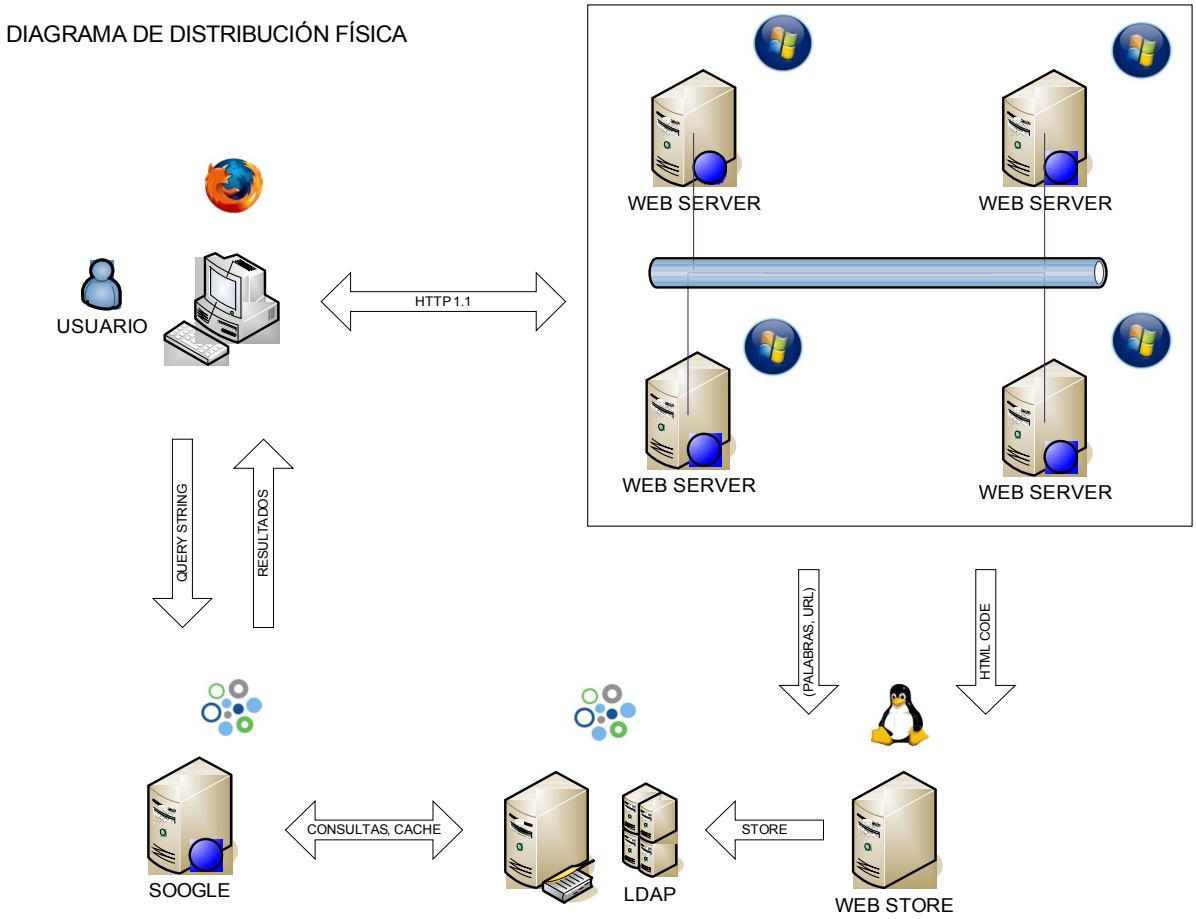
Su función consiste en la búsqueda de todas las páginas HTML y contenidos que existan en un Web Server para luego analizarlos y enviarlos al motor de búsqueda y que este pueda almacenar toda esa información en su directorio de datos. El Crawler se irá duplicando por todos los servidores que existan y estén relacionados entre sí.
Este proceso de indexación de la red comienza en el buscador y cada cierto tiempo envía un pedido de instanciación de *Crawler* a un Web Server conocido. Este, al recibir la orden, tendrá la capacidad de crear una instancia si cumple con determinadas condiciones.

Durante su ejecución se encargará de analizar todos los recursos que existan en el directorio. Analizará esa información y la enviará al Web Store para su posterior almacenamiento. Durante el análisis de datos es posible que alguna página HTML contenga un enlace a otro documento ubicado en otro servidor. Al detectar

esto el *Crawler* se comunicará con ese servidor para intentar crear una instancia de si mismo en esa ubicación y asi continuar con el proceso.
El administrador del servidor Web podrá restringir que documentos pueden ser públicos o privados para evitar que sean publicados en la Web.

Directorio de Datos

En este sistema este directorio cumplirá la función de base de datos almacenando toda la información recolectada por los Crawlers de todos los recursos que se encuentran en la red. Este directorio usará el protocolo LDAP (*Lightweight Directory Access Protocol*) para poder acceder al contenido del directorio.
Aquí, más precisamente, se almacenarán las palabras clave y la información de los archivos, así como su ubicación en la red.



4.-Descripción detallada de las entregas

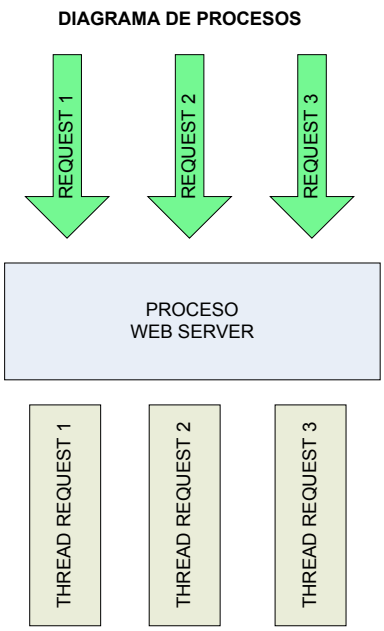
4.1.-Primer entrega

Descripción

Web Server

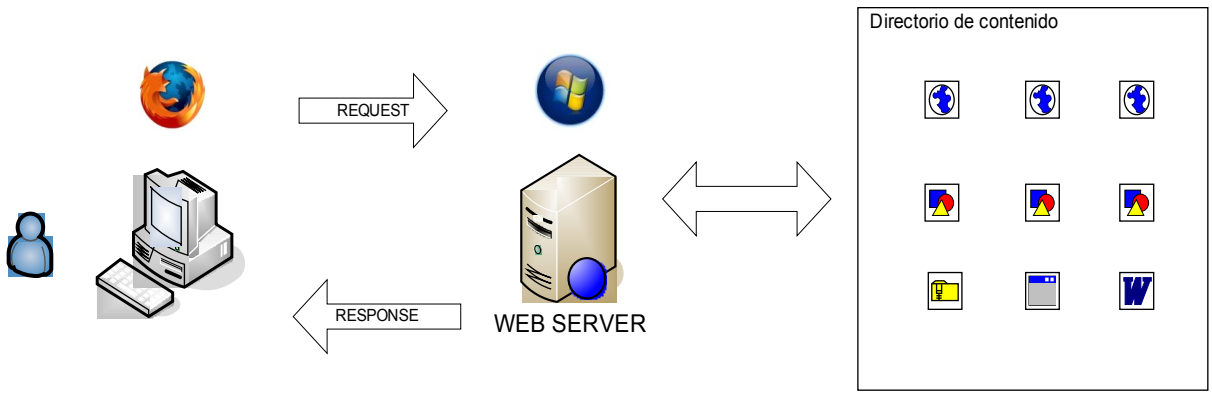
Esta entrega consiste en desarrollar la primera etapa de un Web Server en Windows. Se levantará un proceso por línea de comando que se pondrá a la escucha de requests realizados por el browser.

El cliente realizará un *HTTP GET* con el nombre del recurso a descargar y el servidor aceptará el pedido consultando si ese recurso existe en su repositorio del filesystem. Si el recurso existe se enviará un response con *HTTP 200 OK* y a continuación el flujo de bytes perteneciente al archivo. Si no existe deberá responder un *HTTP 404 NOT FOUND*. Al finalizar dicha transferencia el servidor deberá cerrar la conexión. El servidor deberá tener en cuenta que si el archivo es HTML o una Imagen deberá ser capaz de mostrarlo en el browser. Pero si es cualquier otro tipo de extension entonces el browser le dará la opción de iniciar su descarga.



Por cada request, el servidor generará un nuevo thread de ejecución (ver requerimientos técnicos) el cual realizará el proceso descrito en el párrafo anterior. De esta manera, tendrá la capacidad atender a distintos clientes al mismo tiempo. El grado de concurrencia del servidor, al igual que el port de escucha y el path al repositorio de recursos serán parametrizables por archivo de configuración.
Cuando se supera la cantidad de conexiones simultaneas, los nuevos pedidos serán encolados en una cola de pendientes y que será administrada bajo el algoritmo FIFO. No habrá límites de clientes que podrán estar encolados. El proceso informará el estado de la cola por consola en todo momento. Un cliente estará en la cola un tiempo (medido en milisegundos) configurable por el administrador. Pasado ese tiempo el servidor responderá por pantalla un *HTTP 408 REQUEST TIMEOUT* a dicho request encolado.

El servidor podrá ser puesto fuera de servicio por consola y a los usuarios que soliciten algún recurso se le informará mediante un mensaje que no se encuentra disponible.
El proceso seguirá ejecutando y podrá ser activado para que continúen sus funciones de upload. No obstante el Crawler seguirá ejecutando en cuyo caso lo esté haciendo en ese momento. También podrá ser finalizado por consola. Ante este evento el proceso deberá generar un archivo log con la cantidad de requests aceptados, la cantidad de bytes transferidos y el tiempo total de ejecución.



Modo de testeo de la entrega
• Para el caso en que el cliente solicite un recurso que no sea un archivo HTML se deberá poder verificar la integridad del archivo ejecutando en ambos extremos el comando md5sum (en Windows deberá usarse un comando similar). • Se deberá soportar el protocolo HTTP versión 1.1. • El browser que se usará para las pruebas será Mozilla Firefox. Cabe aclarar que al respetar el estándar HTTP 1.1 deberá poder utilizarse con Internet Explorer entre otros. • Los nombres de los archivos no superarán los 30 caracteres.

Tiempo estimado	Dos semanas
Tipo de entrega	No Obligatoria
Fecha	25/04/2009

4.2.-Segunda entrega

Descripción

Motor de búsqueda

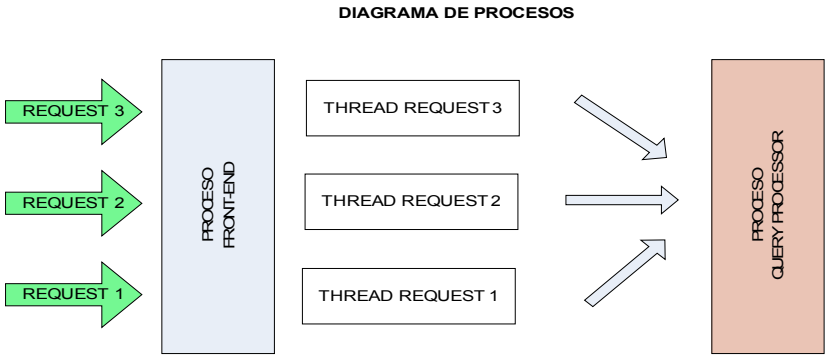
La primer parte consistirá en el desarrollo del buscador sobre la plataforma Solaris. Se levantarán dos procesos: el *Front-end* encargado de presentar la página del buscador y el *Query Processor* encargado de recuperar los datos del directorio.

Front-end

Proceso multithread que contiene el formulario que el usuario usará para especificar las palabras que quiere buscar. Este formulario HTML tendrá dos *INPUTS*: uno **txt** (que contiene las palabras a buscar) y otro **opt** (que tiene el tipo de documento a buscar). La forma de envío será utilizando el método ***GET***. La combinación de palabras permitidas está especificada en la sección *Página principal de búsqueda.*

Query Processor

El *Front-end*, ante un submit del formulario, recibirá los parámetros de la búsqueda y deberá interpretarlos para luego enviárselos al *Query Processor* que será el encargado de realizar la consulta al directorio de datos. Este, una vez obtenidos los resultados, le enviará al front-end dicho listado que será el encargado de transformarlos en una página HTML (como lo especifica la sección *Página principal de búsqueda*) para que el browser lo interprete. En esta primera etapa, el *WebStore* no formará parte del mismo.



Directorio de almacenamiento de datos

Consta de los siguientes directorios:

Directorio de palabras: este directorio es uno de los más importantes y el que más aumentará en tamaño ya que contiene todas las palabras indexadas de la red. Estará dividido en dos directorios principales:

El primer directorio estará dedicado a almacenar palabras sobre documentos HTML. Por cada una también almacenará la URL donde se encuentra dicha palabra. Además de información adicional contará con un UUID generado para esa URL.

Palabra	URL	TITULO	DESC	UUID
Pepe	http://10.10.10.2:8080/index.html	{titulo}	{desc}	9988200e-10bf-11de-afb2-00a0d1792348
Tito	http://10.10.10.2:8080/index.html	{titulo}	{desc}	9988200e-10bf-11de-afb2-00a0d1792348
Pepe	http://10.10.10.2:8080/portal.html	{titulo}	{desc}	9988200e-10be-22de-aab2-01a0d1792348
Juan	http://10.10.10.2:8080/portal.html	{titulo}	{desc}	9988200e-10be-22de-aab2-01a0d1792348

El segundo directorio de palabras será exclusivo para el resto de los archivos que existen en la red: archivos de imágenes ó cualquier otro tipo. Será igual al anterior a diferencia que se guarda el tamaño y el formato y un campo que indica si se trata de una imagen.

PALABRA	URL	TIPO	FORMATO	TAMAÑO
foto	http://10.10.10.56:8080/foto_vacas.jpg	1	JPG	23552
logo	http://10.10.10.87:8080/utn_logo.gif	1	GIF	10234
book	http://10.10.10.56:8080/so_book.pdf	2	PDF	1032455

Directorio de código: aquí se almacena una copia del código HTML de cada página indexada. El código es recuperable por UUID. Solo aplica a páginas HTML.

UUID	HTML CODE
9988200e-10bf-11de-afb2-00a0d1792348	Código de index.html
9988200e-10be-22de-aab2-01a0d1792348	Código de portal.html

Directorio de expiración de URL: contiene los hosts de las páginas que se indexaron más un Unix Time Stamp. No contiene hosts duplicados.

HOST	UTS
10.10.10.56:8080	432423423
10.10.10.2:8080	342523532
10.10.10.87:8080	213213414
10.10.10.56:8080	124314413

Modo de testeo de la entrega
• Se debe hacer un correcto uso de los recursos del sistema: consumo de CPU por proceso, manejo eficiente de la memoria dinámica, respetar las restricciones del enunciado. • Se deberá generar un pequeño proceso (parte de la funcionalidad la tendrá más tarde el *Crawler*) que permita insertar registros en el directorio de palabras con datos válidos para luego poder ser consultados accediendo al front-end del buscador. Los demás directorios no serán utilizados en esta entrega.

Tiempo estimado	Dos semanas
Tipo de entrega	No obligatoria
Fecha	09/05/2009

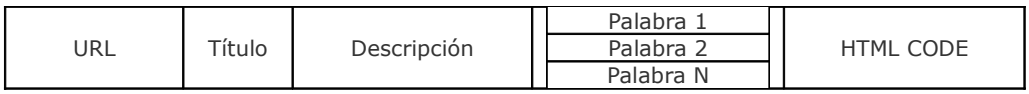
4.3.-Tercer entrega

Descripción

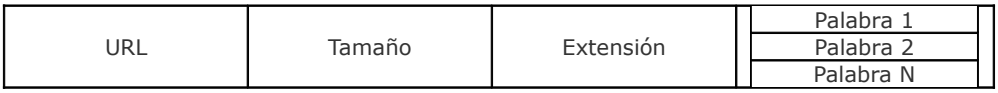
WebStore

Será un proceso desarrollado en Linux y su única responsabilidad será estar a la espera de la información que le envíe el *Crawler*. A partir de esta entrega ya pasa a formar parte del motor de búsqueda. Al recibir un paquete (ver mas abajo), con la información perteneciente a una página ó archivo, lo procesará y lo almacenará en el directorio de palabras para su posterior consulta. Los directorios comenzaran a incrementarse en tamaño a medida que los *Crawlers* vayan indexando páginas. En esta entrega el campo con el código HTML no es utilizado. También se actualizará el directorio de expiración generando un unix time stamp.

Para un documento HTML el paquete enviado por el *Crawler* será:

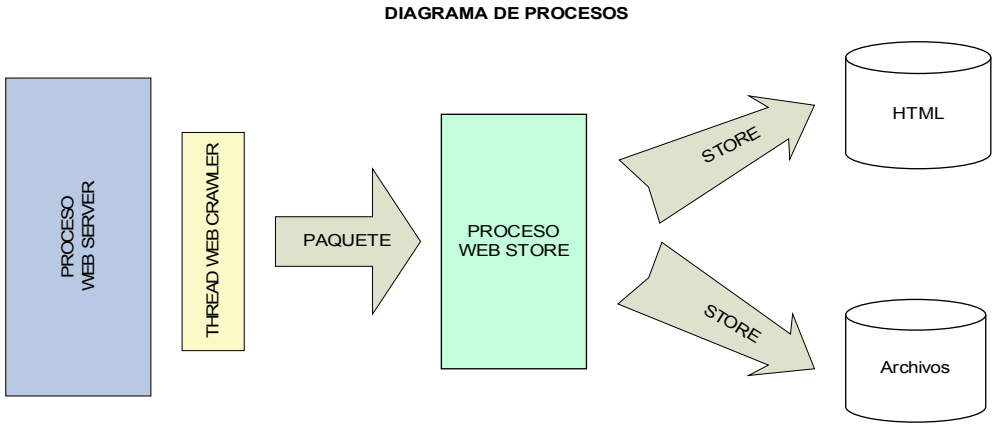


Para cualquier otro archivo:



El Web Store es un componente single-thread, esto quiere decir que solo tendrá un thread de ejecución pero será capaz de atender múltiples solicitudes de diferentes Crawlers. También será el encargado de comenzar el proceso de duplicación del Web Crawler por la red de servidores. Este proceso comienza a partir de la dirección de una única página Web conocida alojada en un servidor de la red.

El programa, ante el envío de la señal *SIGUSR1*, generará la primera solicitud al servidor remoto cuya dirección es obtenida del archivo de configuración. Esto lo logra creando un nuevo proceso en el sistema. Luego de ese primer envió, este proceso recibirá la señal cada cierto tiempo (configurable por el administrador del sistema) y consultará el directorio de expiración. Con esos hosts que se van acumulando, el Crawler los usará de punto de partida para iniciar nuevamente la migración del Crawler solo cuando la diferencia de tiempo entre el timestamp del registro haya superado un tiempo configurable por archivo. Ejemplo: si un campo fue insertado hace 10 minutos y el tiempo está configurado en 9 minutos entonces se debe enviar una solicitud al Web Server para crear un Crawler. Luego del pedido deberá actualizarse el tiempo para la próxima vuelta.



Creación de Crawlers en la red

El Web Server dispondrá de un puerto que estará en espera de un mensaje para instanciar un *Crawler.* Al recibir la solicitud el servidor creará un nuevo thread. Se deben cumplir las siguientes condiciones para recibir la migración de un Crawler y por ende crear el thread:

- ✓ Que no exista una instancia previa de un *Crawler* actualmente en ejecución.
- ✓ Que haya pasado un período de tiempo, configurable, en el cual el servidor no aceptará una migración.

Entonces, al establecer la conexión se realiza el siguiente handshake entre el Crawler y el WebServer: "SOOGLE CRAWLER CONNECT/0.1\n\n". El Server comprobará las condiciones de creación respondiendo "CRAWLER FAIL\n\n" en caso de no poder aceptar la solicitud y "CRAWLER OK\n\n" en caso contrario.

Al crearse este proceso, el thread deberá cambiar su *Thread Priority Level* a *THREAD_PRIORITY_HIGHEST* para aumentar su *Base Priority*. El proceso deberá mantener su *Process Priority Class*.

Proceso de indexación de documentos

El administrador de un servidor podrá restringir los archivos que el Crawler analizará y publicará en la Web. Esto lo hace para hacer público ó privado un documento. Uno ó varios archivos (dentro ó no de un subdirectorio) que se encuentren dentro del directorio principal de descarga con el atributo *FILE_ATTRIBUTE_READONLY* activo serán ignorados por el Crawler y no serán indexados. En caso contrario se podrán leer los archivos.

La primera vez que un Web Server es descubierto por el Crawler, este creará una tabla local con la que almacenará (por cada archivo público) un hash de 128 bits con el algoritmo MD5 utilizando la Windows API. En una próxima migración, el Crawler, comparará el hash actual del archivo con el almacenado en la tabla para detectar si ha sido alterado. En caso que haya sido alterado entonces los procesará y actualizará la tabla con los nuevos valores. Si existiesen nuevos archivos ó se hayan borrado en el directorio ó hayan cambiado el permiso se actualizará la tabla dejandola consistente. Dicha tabla es persistente ante caídas del proceso. A partir de este momento el Web Server podrá mostrar por pantalla el listado con los archivos disponibles con el hash de cada uno.
Entonces, para los archivos nuevos ó alterados el Crawler generará el paquete correspondiente (según el tipo de documento) para ser enviado al *Web Store*. En esta entrega, el Crawler no enviará el paquete en caso de que el archivo haya sido modificado.

Análisis de un documento

Una vez cumplidas las condiciones para que el Crawler pueda hacer el análisis, este hará el parseo del documento HTML y se concentrará en los siguientes TAG's:

- **<META description>**: contiene la descripción de la página y es lo que se presentará en los resultados.

- **<META keywords>**: palabras clave que describen los temas más relevantes del documento. Serán las palabras que se indexarán.
- **<A href>**: contiene enlaces a otras páginas de la Web. Al encontrarse con el TAG se desencadenará un evento asincrónico que hará que se genere un nuevo *Crawler* en el servidor indicado en el atributo *href*. Se ignora si se referencia a sí mismo.
- **<TITLE>**: Indica el título principal de la página.
- **<IMG SRC>**: contiene enlaces a recursos de imágenes para su indexación y visualización.

¿Y con los archivos que no son HTML como los analiza? Simple, los archivos siempre tendrán el formato "nombre_del_archivo.extension" (solo caracteres alfanuméricos punto extensión). Las palabras clave estarán en el nombre del archivo separadas por un **underscore** [_]. Por lo tanto la búsqueda de una palabra será por inclusión en el nombre real del archivo. No dispondrá de ninguna descripción pero obtendrá el tamaño y extensión del archivo.

Modo de testeo de la entrega
• Es importante comenzar a integrar los procesos con las entregas anteriores para que comienzan a ser consistentes. • Comprobar el correcto funcionamiento del Web Store haciendo uso de los recursos. • Chequear la integridad de los mensajes ya que los mismos comienzan a ser de tamaño variable.

Tiempo estimado	Tres semanas
Tipo de entrega	Obligatoria
Fecha	30/05/2009

4.4.-Cuarta entrega

Descripción

Load Balancing & Query Manager

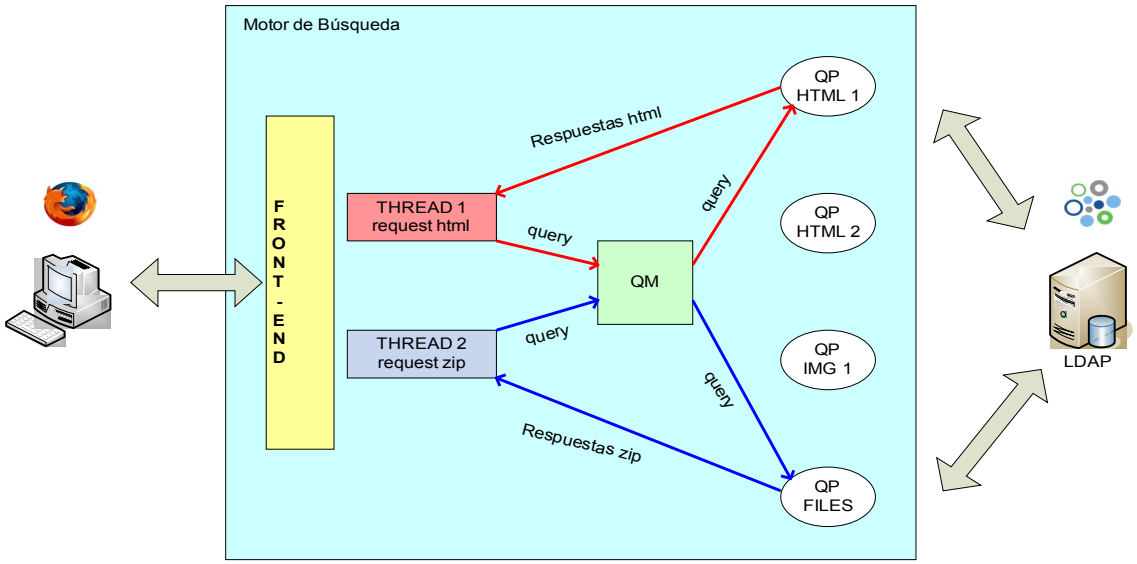
A partir de esta entrega el front-end del buscador estará asociado a un proceso que se llamará Query Manager (QM). Su función será la de balancear la carga de solicitudes entre uno o más Query Processors (QP).

Ahora un QP podrá ser configurado mediante un archivo con varias propiedades:

- ✓ La primera propiedad indica el tipo de recurso que atenderá. Significa que este proceso solo acepta pedidos de páginas HTML ó imágenes ó cualquier otro archivo ó una combinación de ambos. Si es el único deberá poder aceptar todo tipo de solicitudes.
- ✓ La cantidad de conexiones simultaneas que el proceso podrá atender.
- ✓ Tiempo en milisegundos en el que demora en retornar la respuesta al front-end. Esto se utiliza para realizar pruebas y seguimiento.

Antes de que el buscador esté operativo, el QM estará a la espera de que se le asignen QP's. Al recibir un nuevo QP este le enviará los valores de cada política. Con esta información el QM mantendrá una tabla con la que podrá encaminar las solicitudes entre los distintos QP. Se podrán adicionar nuevos QP en runtime sin necesidad de poner fuera de servicio el QM.

Entonces, el proceso de búsqueda con distribución de carga quedaría de la siguiente manera: El QM recibe el *query string* el cual lo derivará al QP que esté disponible según la cantidad de conexiones disponibles y según el tipo de recurso que maneja. Si ninguno está disponible lo mantendrá en una cola que sigue el algoritmo FIFO hasta que se libere un QP. La distribución para los QP se hará en forma equitativa, es decir si hay en un determinado instante 5 pedidos de un determinado tipo entonces la distribución para dos QP que manejen ese recurso quedaría: QP1, QP2. QP1, QP2 y el quinto pedido queda encolado.
El QP al recibir los parámetros seguirá el mismo proceso que fue descripto en la entrega pasada. Los resultados de la búsqueda serán enviados directamente el thread que atendió el request en el front-end.



Políticas de seguridad en un Web Server

El Web Server tendrá ciertas políticas que harán que quede fuera de servicio en el caso de no cumplirlas. Las mismas puede ser habilitadas o deshabilitadas por configuración. Al no cumplirse una de estas políticas se mostrará por pantalla que el servidor se encuentra fuera de servicio y el motivo de dicho problema. A continuación se enumeran tales políticas:

- *Control de flujo saliente*: el servidor dispondrá de un control de paquetes (medido en bytes) que son enviados al exterior. Al sobrepasar un valor máximo configurable el servidor queda fuera de servicio.
- *Cantidad de conexiones TCP aceptadas*: se establece una cantidad configurable de conexiones que pueden ser aceptadas. Al superar dicho límite, se le deberá informar al solicitante el motivo y denegar el pedido de conexión.

Modo de testeo de la entrega
• Configurar tiempos altos de espera para las respuestas de los QP de tal manera de generar pedidos en forma simultanea.

Tiempo estimado	Tres semanas
Tipo de entrega	Obligatoria
Fecha	20/06/2009

4.5.-Quinta entrega – Entrega Final

Descripción

A partir de ahora todos los componentes que se desarrollaron en las entregas pasadas comienzan a relacionarse entre sí. Habrá varias instancias de Web Server en la misma red con sus archivos en sus respectivos directorios de filesystem. La única relación que habrá entre ellos será a través de documentos HTML que contendrán enlaces a otras páginas. A partir de ahora el Web Server incorpora la generación de reportes que se describen en la sección *Reporte de ejecución de un Web Server.*
Por otro lado, se levantará el motor de búsqueda Soogle ya en su totalidad. El proceso *Front-End* estará a la espera del *Query Manager* para poder comenzar. Al mismo tiempo habrá una instancia del Web Store corriendo, en espera de la orden del usuario para comenzar el proceso de indexación de la red. Antes de estar operativos será necesario asignarle dos ó más Query Processor para balancear la carga. Una vez instanciados estos procesos Soogle estará operativo. En cualquier momento se podrán consultar los datos por consola que se especifican en la sección *Presentación de estadísticas del motor de Búsqueda*.

Caché de páginas

En esta entrega el buscador ofrecerá la opción de almacenar una copia de una página HTML en caché. De esta manera, al generar los resultados, se le mostrará al usuario que dicha página tiene una copia en la caché. Gracias a un código que la identifica y la relaciona con la URL real se podrá recuperar de la base de datos y mostrarla por pantalla. ¿Pero, qué pasa con los archivos que dejan de existir? Simple, el buscador podrá mantener en el tiempo un documento que en la ubicación original podría no existir.

Almacenamiento en caché

Para lograr esta magia el Crawler, luego de enviar las palabras de una página, también enviará el código HTML completo. Obviamente esto no aplica para el resto de los archivos. El Web Store además de almacenar las palabras, generará un uuid (*Universally Unique Identifier*) para dicha URL. Por lo tanto, se comenzará a almacenar esta relación en el *directorio de código*. Si dicha URL ya existe el contenido deberá ser actualizado con el nuevo código.

Ahora, el Query Processor (exclusivo de páginas Web) incorporará la siguiente lógica: cuando recupera la tupla [PALABRA, URL, UUID] dispondrá del uuid para luego generar una URL especial formada por la IP del front-end y como parámetro dicho uuid. La presentación al browser quedaría de la siguiente manera:

```
Resultados obtenidos:
Tipo de resultados: paginas html
Cantidad: 1
Tiempo de respuesta: 0,40 segundos
Palabras: "sistemas operativos"
-----------------------------------------------------------------------

    1. Título: Sistemas Operativos
       Descripción: Futura página oficial de la cátedra
       Link: http://192.168.2.80/index.html
       En cache: http://192.168.2.2/cache=550e8400-e29b-41d4-a716-446655440000

-----------------------------------------------------------------------
```

Actualización de documentos

¿Qué ocurre cuando el Crawler detecta que un archivo sufrió un cambio? ¿Las páginas en caché tampoco se actualizan?

Bueno, este proceso se llevará a cabo en el Web Store antes de insertar nuevas palabras al directorio y generar información redundante. Cuando se recibe un nuevo paquete con información perteneciente a un documento se debe controlar en primer lugar si esa URL presenta resultados en el directorio de palabras. Si no presenta entonces el proceso es el descrito en la entrega pasada: se genera un UUID, se insertan las palabras y (según corresponda) se genera la caché.

El proceso es más complejo cuando la URL presenta ya resultados. Primero se deben insertar las palabras que no existen, caso se actualiza la información del campo en caso de a ver cambiado. Si además, las palabras corresponden a una página HTML se debe obtener el UUID correspondiente y actualizar el directorio de caché.

Modo de testeo de la entrega
• Lo recomendable en esta etapa de integración es realizar el testing de stress (*Prueba diseñada para determinar la respuesta de un sistema bajo condiciones de carga*) en la aplicación ya completa. Es importante tratar de desarrollar toda la funcionalidad con una semana (como mínimo) de anticipación a la entrega final a fin de realizar el testing de la aplicación por completo y reducir al máximo posible la cantidad de bugs. Recordar que es indispensable, para la aprobación del trabajo práctico, la entrega completa de la funcionalidad y su correcto funcionamiento.

Tiempo estimado	Dos semanas
Tipo de entrega	No Obligatoria
Fecha	04/07/2009

5.-Especificación de una página HTML

El código HTML con el que crearán las páginas Web será una simplificación del que se puede encontrar en una página real. Con este fin se facilita su manipulación. A continuación se muestra un ejemplo completo con los tags que se deben utilizar. Para su manipulación referirse a la sección de *Requerimientos tecnológicos*.

```
<!-- root element -->

<html>

    <head>

        <!-- información sobre esta página -->

        <title> Pagina de sistemas Operativos </title>

        <META name="keywords" content="Operativos, SO, ejemplo"/>

        <META name="description" content="Esta es una página de ejemplo"/>

    </head>

    <body>

       <!-- contenido de la página -->
       Puede haber texto <br/>

       <a href.=”http://192.168.2.123:8080/otraPagina.html”> o un link a otra pagina </a>

       <br/> Ó simplemente mostrar una imagen <br/>

       <img src=” http://192.168.2.123:8080/imagen_ejemplo.jpg” />

    </body>

</html>
```

6.-Página principal de Búsqueda

La página HTML del buscador será un simple formulario con dos inputs, el texto a buscar y el tipo de documento sobre el cual realizar la búsqueda.

Query String

El campo para ingresar texto deberá aceptar los siguientes criterios de búsqueda. Pueden ser cualquiera de las siguientes combinaciones:

- *Palabra*: se buscan los documentos que respondan a esa palabra.
- *Palabra1* + *palabra2* + ... + *palabraN*: que busque la primera, la segunda y ambas. Si se ingresan dos palabras seguidas del la forma “*palabra* <*espacio*> *palabra*” entonces tendrá el mismo comportamiento. Las páginas donde se encuentran ambas palabras tendrán mayor prioridad que la páginas que solo contengan una.
- *Palabra1* – *palabra2*: Se listan las páginas que contienen la primer palabra pero no las que contienen ambas.

Opciones de búsqueda

La pantalla permitirá que el usuario seleccione el tipo de documento a buscar. Las opciones disponibles son:

- Páginas Web: solo resultados que contengan páginas html.
- Imágenes: solo resultados de archivos de imágenes.
- Documentos: cualquier otro formato de archivos que no sean los anteriores. Por ejemplo archivos zip, documentos pdf, etc.

Resultados de una búsqueda

La página de resultados será un archivo HTML que el motor de búsqueda generará a partir de los resultados obtenidos. Las URL duplicadas se desechan, es decir, si dos palabras diferentes hacen referencia a la misma página entonces se tomará como un resultado que incluye las dos palabras.
A continuación se muestra un ejemplo de la información mínima que deberá mostrar:

```
Resultados obtenidos:
Tipo de resultados: paginas html
Cantidad: 3
Tiempo de respuesta: 0,40 segundos
Palabras: "sistemas operativos"
----------------------------------------------------------------------

    2. Título: Sistemas Operativos
       Descripción: Futura página oficial de la cátedra
       Link: http://192.168.2.80/index.html
       En cache:http://192.168.2.2/cache=9988200e-10bf-11de-afb2-00a0d1792348

    3. Título: Foro de Sistemas Operativos
       Descripción: La página el que viento se llevó
       Link: http://192.168.2.82:9090/foro.html
       En cache:http://192.168.2.2/cache=9a170bca-10bf-11de-8ee6-00a0d1792348

    4. Título: Sistemas y Organizaciones
       Descripción: Clases particulares
       Link: http://192.168.2.123:8080/sistemas.html
       En cache:http://192.168.2.2/cache=9693454a-10bf-11de-bf50-00a0d1792348

----------------------------------------------------------------------
```

Para el caso en que el usuario quiera buscar imágenes los resultados se presentarán con el siguiente formato:

```
Resultados obtenidos:
Tipo de resultados: imágenes
Cantidad: 1
Tiempo de respuesta: 546 milisegundos
Palabras: "logo"
----------------------------------------------------------------------

    1. Nombre: logo_utn
       Formato: BMP
       Tamaño: 23.6 Kb
       Link: http://192.168.2.80/logo_utn.bmp

----------------------------------------------------------------------
```

En cambio, si se solicitan otros documentos la presentación se hará de la siguiente manera:

```
Resultados obtenidos:
Tipo de resultados: documentos
Cantidad: 2
Tiempo de respuesta: 1,3 segundos
Palabras: "silberschatz"
----------------------------------------------------------------------
```

1. Nombre: operating_system_7th_silberschatz_emule
   Formato: PDF
   Tamaño: 63 MB
   Fecha de creación: 15/5/2008 12:40 PM
   Link: http://192.168.2.80/operating_system_7th_ed-silberschatz_emule.pdf

2. Nombre: silberschatz_CV
   Formato: DOC
   Tamaño: 230 KB
   Fecha de creación: 23/1/2009 09:30 AM
   Link: http://192.168.2.80:7345/silberschatz_CV.doc

7.-Reporte de ejecución de un Web Server

Esta es la etapa final de la ejecución del servidor Web y tiene por objetivo presentar un informe en un archivo de texto con las estadísticas tomadas durante toda su ejecución. Los tiempos de Kernel Time y User time se obtendrán de acuerdo a lo indicado en la sección de requerimientos tecnológicos.
Si la aplicación, por algún evento ajeno al sistema, deja de funcionar, NO debe registrar ningún suceso ni volcar los estados parciales de los procesos en ejecución.

Para generar el reporte se tendrán en cuenta las siguientes estadísticas:

- Cantidad de clientes que solicitaron recursos en el servidor (registrar la dirección IP y el user agent).
- Registro de tiempo del ingreso de un Crawler en el sistema. (Time Stamp de llegada y duración del proceso de analisis de documentos)
- Cantidad de datos transferidos medido en bytes durante toda su ejecución.
- Tiempo en que el servidor Web se ejecutó en modo kernel (Kernel time).
- Tiempo en que el servidor Web se ejecutó en modo usuario (User time).
- Tiempo total de ejecución del servidor en el sistema.

8.-Presentación de estadísticas del Motor de Búsqueda

Durante la ejecución de la aplicación, el usuario administrador de sistema podrá consultar los siguientes datos que serán presentados por pantalla del Query Manager:

- Impresión del ranking de las páginas ú otros recursos más solicitados de la red.
- Ranking de las palabras más buscadas por los usuarios.
- Estado actual de los *Query Processors*. Discriminando por QP se deberá mostrar el tipo de documento que maneja, cantidad de consultas realizadas que fueron exitosas y consultas que no arrojaron resultados).

9.-Requerimientos técnicos y limitaciones

Comunicaciones – Sockets

Para la comunicación cliente/servidor se utilizarán los protocolos especificados en el trabajo práctico y se implementarán utilizando sockets (modelo Berkeley) orientados a la conexión del tipo *AF_INET* para IPv4 (en las tres plataformas).

Para la programación de sockets en Windows, se va a utilizar la biblioteca Winsock declarada en *winsock2.h*. Se requiere linkear con la dependencia *Ws2_32.lib*.

Inicialización de la biblioteca Winsock

Antes de llamar a cualquier función de la biblioteca Winsocks, se deberá inicializar dicha librería indicando la versión de Window Sockets y obteniendo los detalles de la implementación.
Finalizada la utilización de la biblioteca ó ante algún error en la inicialización de la misma, se deberá liberar los recursos y finalizar el uso de la biblioteca Winsock.
Esta inicialización se realiza **solamente una vez** por ejecución de la aplicación.

Los parámetros inicialización serán los siguientes:

Parámetros

Versión solicitada

Se utilizará la versión 2.2.

Process Management y Multithreading

En las tres plataformas no se contempla un sistema con múltiples procesadores (multiprocessor ó multicore) con kernel SMP. Por lo tanto, no se controlará el *Thread Affinity* a un determinado procesador en caso de que la API del sistema provea dicha funcionalidad.

Windows

Para la creación de threads en Windows se utilizará la función *_beginthreadex* perteneciente a la *C/C++ Runtime library*. De esta manera se evitarán posibles *memory leaks.* (Para investigar: ¿cuales son las causas de este comportamiento al usar la API *CreateThread*?).
Cada thread tendrá por defecto un *stack space* de 1 megabyte.

Solaris y Linux

En Linux, en caso de ser requerido, solo se podrá utilizar la API de la biblioteca Pthreads que forma parte del estándar POSIX.
Para la plataforma Solaris se utilizará la API de Solaris Threads para la creación y manipulación de hilos de ejecución. Se puede dar el caso en que exista funcionalidad que no esté presente en Solaris Threads y si exista en POSIX. En este caso es posible utilizar ambas para adquirir funcionalidad extra.
Cada Thread que la aplicación genere deberá tener por defecto el *stack space* en 1024 kilobytes por arriba del *PTHREAD_STACK_MIN*.
Para la creación de procesos, en caso de ser requerido, se utilizará el modelo *Fork-one Model*.

Thread Synchronization

En Windows se utilizarán Kernel Objects para la sincronización. Estos objetos serán los utilizados por las llamadas *wait functions* para coordinar la ejecución de múltiples threads al pasar de un estado señalizado a uno no señalizado.
A continuación se dará una descripción de los mecanismos que ofrece la Windows API y que estarán permitidos utilizar para la sincronización de threads en *user-mode*.

Los siguientes kernel objects podrán ser utilizados para la sincronización y queda a criterio del alumno cual utilizar según crea conveniente:

- *Events Kernel Objects*
- *Semaphore Kernel Objects*
- *Mutex*
- *Threads*
- *Waitable timers*

En Solaris y Linux también se podrán utilizar cualquier tipo de *synchronization objects*:

- *Mutex Locks*
- *Semaphores*
- *Condition variables*
- *Read-Write Locks*

Memory Management

Windows

Cada thread que la aplicación genere, creará su propio bloque de páginas adicional (su propio heap) en el espacio de dirección del proceso y al que solo él tendrá acceso. Esto lo hará para tener un manejo más eficiente de memoria y evitar el overhead generado por la sincronización entre threads. La cantidad de memoria que deberán ocupar las páginas al ser inicializadas es de 1024 Kb. No habrá restricciones en cuanto a tamaño. El tamaño máximo estará limitado por la cantidad de memoria disponible en el sistema.

Dado que el tiempo de vida de cada thread será corto, este no se ocupará de la fragmentación que se genere en el heap. Cuando el thread ya no necesite el heap este lo deberá eliminar para liberar los recursos.

En ningún momento un thread que no sea el principal utilizará la memoria del heap creado por defecto en el proceso ó heap global. Para esto se utilizarán las funciones de manejo de memoria proporcionadas por la API de Windows. No está permitido el uso de la biblioteca estándar de C para manejo de memoria (*Stdlib.h: malloc, free, etc*).

Solaris y Linux

No existen restricciones en cuanto al uso de memoria.

Filesystem

Toda operación de I/O que involucre manejo de archivos en Windows deberá ser usando la Windows API correspondiente en su versión *ANSI* (en caso de existir *UNICODE*). No está permitido el uso de la biblioteca estándar de C para manejo archivos (*Stdlib.h: fread, fwrite, fopen, etc*).

Para Linux y Solaris no existe tal restricción.

Manipulación de XML y HTML

Para la creación y manipulación de documentos XML y HTML estará permitido el uso de la biblioteca Libxml2. Toda la información y ejemplos se encuentran disponibles en http://xmlsoft.org/. Se descargar para las tres plataformas que se utilizan en este trabajo practico.
Recordar que el proyecto será testeado en otro entorno por lo tanto los header como las bibliotecas dinámicas (includes y libraries) pertenecientes a Libxml2 deberán formar parte de este proyecto ya que no se encontraran instaladas en el entorno del laboratorio.

LDAP y OpenDS

Como implementación de LDAP se utilizará OpenDS de SUN. Se encuentra disponible en la versión Solaris ya que el proyecto está escrito en Java y es multiplataforma. Requiere que el entorno tenga instalado un Java Runtime Environment. Se puede descargar gratuitamente de http://www.opends.org/.

Para utilizar el protocolo LDAP está permitido utilizar la API de OpenLDAP disponible en http://www.openldap.org/. Se encuentra para descargar tanto para Linux como para Solaris. También es posible la descarga con el administrador de paquetes de ambos sistemas.

9.-Anexo – Archivo Log y Debugging

Manejo de Errores y Excepciones

La aplicación mostrará por consola y generará en el correspondiente archivo Log el código de error y la descripción de dicho código obtenido del sistema tanto para las llamadas a la API de Windows como para las System Calls de Linux y Solaris. Para los errores pertinentes a la aplicación de deberá respetar las normas de logueo del trabajo práctico.

Algunas aclaraciones para Windows

Dichos códigos se encuentran definidos en el header *WinError.h*. Para la obtención del *message string* a partir cierto código generado, se utilizará la función más apropiada de la API de Windows en su correspondiente versión *ANSI* (en caso de existir *UNICODE*).

Structured Exception Handling

Windows, Solaris y Linux

Queda prohibido el uso de cualquier mecanismo de exception handling ya sea para sentencias del tipo *exception handler* (__try __except) ó *termination handlers* (__try __finally).

Si existiese una API de Windows que permita el control de errores mediante este mecanismo, generando una excepción ante un error, se deberá forzar a que retorne un código de error para ser logueado tal como se explica en la sección de **Formato del Archivo Log**.

Formato del archivo Log

El archivo Log deberá respetar el siguiente formato de presentación en todos los procesos que se ejecuten en cualquiera de los tres Sistemas Operativos.

En caso de utilizar distintos niveles detalle, el cambio entre uno u otro debe ser configurable por el usuario.

Se le recomienda al alumno registrar en este archivo los eventos más importantes de la ejecución de la aplicación, así como los valores necesarios para conocer el estado del sistema en un determinado momento. Esto es muy importante ya que en instancias finales de evaluación es probable que se haga uso de este archivo en situaciones donde la aplicación falla.

Fecha NombreProceso [PIDproceso][ThreadID]: TipoLog: Dato

Descripción

Fecha

Fecha del sistema. Deberá respetar el siguiente formato [HH:mm:ss.SSS].

Nombre Proceso

Nombre del proceso que está escribiendo en el Log.

PID Proceso

Process ID del proceso que está escribiendo en el Log.

Thread ID

ID del thread que escribe en el archivo. Opcional para el thread principal del proceso.

TipoLog

INFO, WARN, ERROR ó DEBUG nivel de detalle según lo que consideren apropiado.

Data

Descripción del evento ó cualquier información que se considere apropiada.

10.-Anexo – Documentación

El material de soporte para poder llevar a cabo este trabajo práctico se encuentra en su mayoría en la Web. A continuación se enumeran enlaces a páginas con la información necesaria para cada plataforma.
MSDN de Microsoft *http://msdn .microsoft.com*.
Sun Microsystems Documentation http://docs.sun.com/

- **Threads y Procesos**

  http://msdn.microsoft.com/en-us/library/ms686937%28VS.85%29.aspx

- **Scheduling**

  http://msdn.microsoft.com/en-us/library/ms685096(VS.85).aspx

- **Winsocks**

  http://msdn.microsoft.com/en-us/library/ms740673%28VS.85%29.aspx

- **Memory Management**

  http://msdn.microsoft.com/en-us/library/aa366779%28VS.85%29.aspx

- **Thread Syncronization**

  http://msdn.microsoft.com/en-us/library/ms682584%28VS.85%29.aspx

- **Time Functions**

  http://msdn.microsoft.com/en-us/library/ms724962(VS.85).aspx

- **Cryptography**

  http://msdn.microsoft.com/en-us/library/aa380255(VS.85).aspx

- **Security**

  http://msdn.microsoft.com/en-us/library/cc527452.aspx

También se recomienda el siguiente material bibliográfico como soporte teórico:

- Programming Applications for Microsoft Windows, 4th edición de Jeffrey Richter.
- Microsoft Windows Internals, 4th Edición de Mark E. Russinovich y David A. Solomon.
- Solaris Internals: Solaris 10 and Open Solaris Kernel Architecture, Segunda Edición de Richard McDougall
- Understanding the Linux Kernel, 3rd Edición de Daniel P. Bovet, Marco Cesati

También pueden encontrar toda la documentación recomendada por nosotros, en nuestro grupo, accediendo a:

http://groups.google.com.ar/group/tp-so-frba-utn/web/links-a-tutoriales?hl=es

Información sobre como utilizar ***OpenLDAP SDK*** se encuentra disponible en la siguiente dirección:

http://www.openldap.org/software/man.cgi?query=ldap

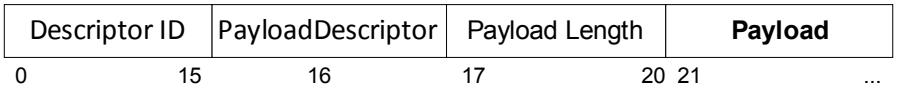
Información sobre como utilizar ***OpenDS*** puede encontrarse en los siguientes enlaces:

https://www.opends.org/wiki/page/OpenDSUserDocumentation

11.-Anexo – Protocolos de comunicación

11.1.-Inter Process Comunications (IPCs) – Protocolo IRC/IPC standard

Se utilizara un mensaje de protocolo interno. Estos son los campos mínimos que todo mensaje interno debe utilizar.



Request:

Descriptor ID:
Identificador de 16 bytes único descriptor en la red.

PayloadDescriptor:
Identificador de nro de protocolo.
PayLoad Lenght:
La longitud del descriptor inmediatamente seguido del header.

Payload:
La carga de datos que se necesite pasar. Queda libre al Usuario del protocolo.

Response:

Descriptor ID:
Identificador de 16 bytes correspondiente al Request.

PayloadDescriptor:
Identificador de nro de protocolo.

PayLoad Lenght:
La longitud del descriptor inmediatamente seguido del header.

Payload:
La carga de datos que se necesite pasar. Queda libre al Usuario del protocolo.

11.2.- Protocolo HTTP 1.0/1.1

A) MENSAJES GET

A.1) GET EXAMPLE HTTP 1.0 -- STANDARD

```
GET <Request-URI>?query_string HTTP/1.0\r\n\r\n
```

A.2) GET HTTP 1.1 -- STANDARD

```
GET <Request-URI>?query_string HTTP/1.1\r\n
Host: <hostname or IP   address of host>\r\n\r\n
```

Ejemplo:

En un browser sería algo como:

http://127.0.0.1:8080/home/test.txt?v=20090101

Donde:

- <Request-URI>?query_string =
  Ruta del archivo: /home/test.txt?v=20090101

- <hostname or IP address of host> =
  Host de destino: 127.0.0.1:8080

B) MENSAJES FILES NOT FOUND

B.1) FILE NOT FOUND WITH 404 NOT FOUND RESPONSE -- STANDARD

```
HTTP/1.0 404 Not Found\n\n
<b>ERROR</b>: File <Filename> was not found\n
```

B.2) FILE NOT FOUND WITH 404 NOT FOUND RESPONSE -- STANDARD

```
HTTP/1.1 404 Not Found\n\n
<b>ERROR</b>: File <Filename> was not found\n
```

Donde en el ejemplo:

- <Filename>: test.txt

C) MENSAJES FILES FOUND

C.1) FILE FOUND WITH 200 OK RESPONSE -- STANDARD

```
HTTP/1.0 200 OK\n
Content-type: text/html\n
Content-Disposition: attachment; filename="<Filename>"\n
```

En el caso de de cualquier otro tipo de archivo el content-type es:

```
Content-type: application/octet-stream\n\n
```

C.2) FILE FOUND WITH 200 OK RESPONSE -- STANDARD

```
HTTP/1.1 200 OK\n
Content-type: text/html\n
Content-Disposition: attachment; filename="<Filename>"\n
```

En el caso de de cualquier otro tipo de archivo el content-type es:

```
Content-type: application/octet-stream\n\n
```

D) MENSAJES TIME OUT

D.1) TIME OUT RESPONSE -- STANDARD

```
HTTP/1.0 408 Request Timeout\n\n
```

D.2) TIME OUT RESPONSE -- STANDARD

```
HTTP/1.1 408 Request Timeout\n\n
```