Universidad Tecnológica Nacional – Facultad Regional Buenos Aires
Ingeniería en Sistemas de Información
Sistemas Operativos (08-2027)

Trabajo Práctico Nro. 2 – 1er Cuatrimestre 2010
Sistema de Redes de Noticias

Revisión 1.6

Índice

1.-Introducción

El trabajo práctico de este cuatrimestre consiste en el desarrollo de una arquitectura distribuida cliente servidor basada en 2 protocolos de nivel de aplicación: NNTP y HTTP. El objetivo del trabajo práctico es implementar un conjunto de herramientas que permitan lograr la integración de sistemas. Por tal motivo se propone una arquitectura que integra 2 conceptos bien definidos: Servidores web y Servidores de Noticias.

Dada la complejidad del mismo, se dividió el práctico en entregas parciales las cuales permiten diseñar la arquitectura definitiva de forma gradual, simplificando su desarrollo. Además consta de varios procesos divididos en tres categorías: Aplicación, Sistema Operativo y Almacenamiento, los cuales se van a comunicar entre sí mediante el paso de mensajes mediante sockets TCP/IP.

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

- Creación y manipulación de procesos y threads mediante la API del sistema.
- Sincronización de Procesos, IPC y manejo de señales.
- Administración de Memoria.
- Manejo del File System a través de la interfaz de sistema.
- Introducción a los Sistemas Distribuidos.
- Arquitecturas basadas en capas físicas, protocolos y mensajería.
- Diferencias entre las plataformas más populares del mercado.

2.-Objetivos del Trabajo Práctico

Desde el punto de vista académico el trabajo está diseñado para:

- 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 y 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 de Redes de Noticias.

A continuación se hace una breve introducción a cada uno de los componentes que intervienen en el sistema. El siguiente diagrama de distribución física muestra en forma resumida los principales componentes.

Diagrama de distribución física

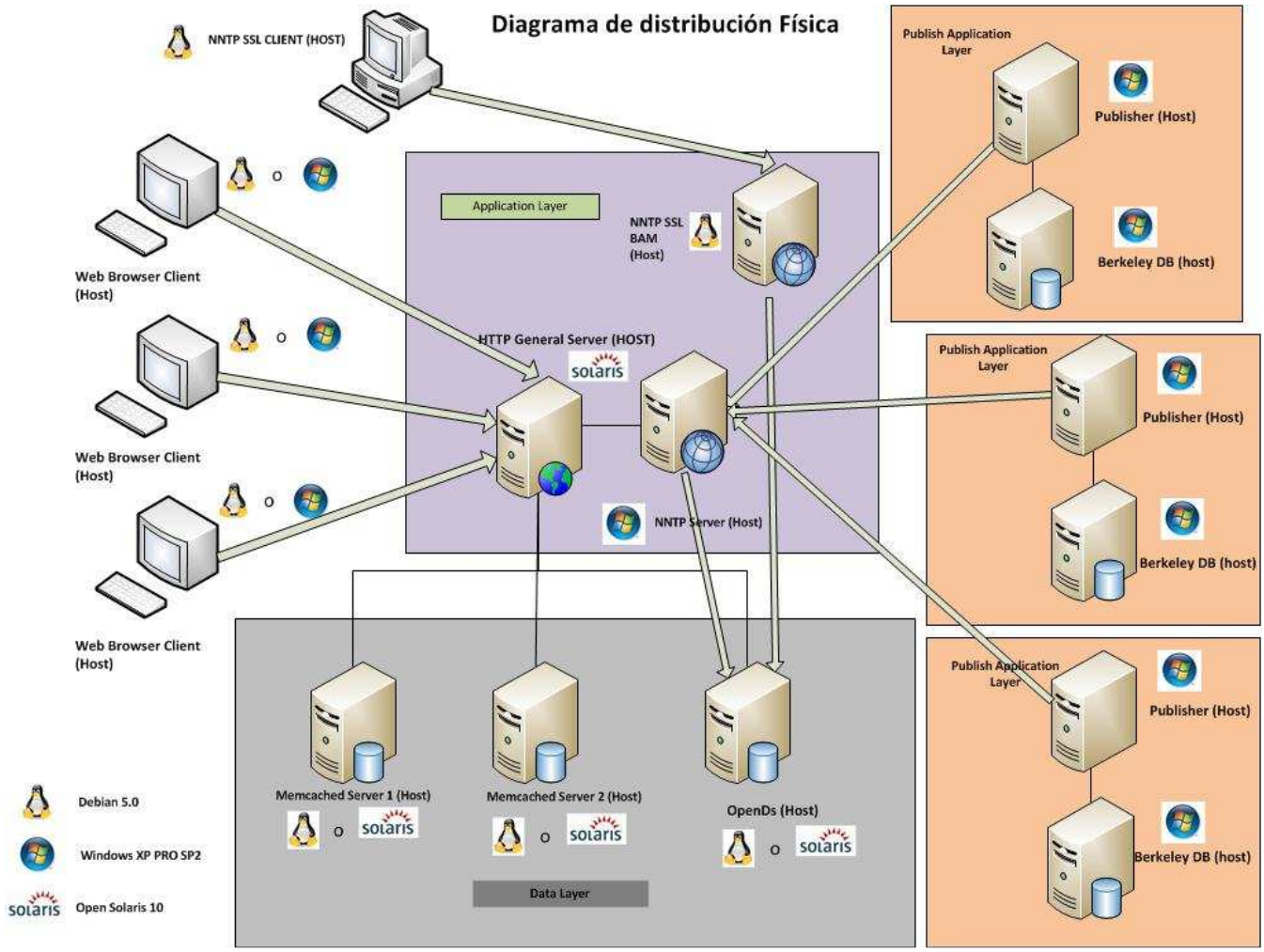
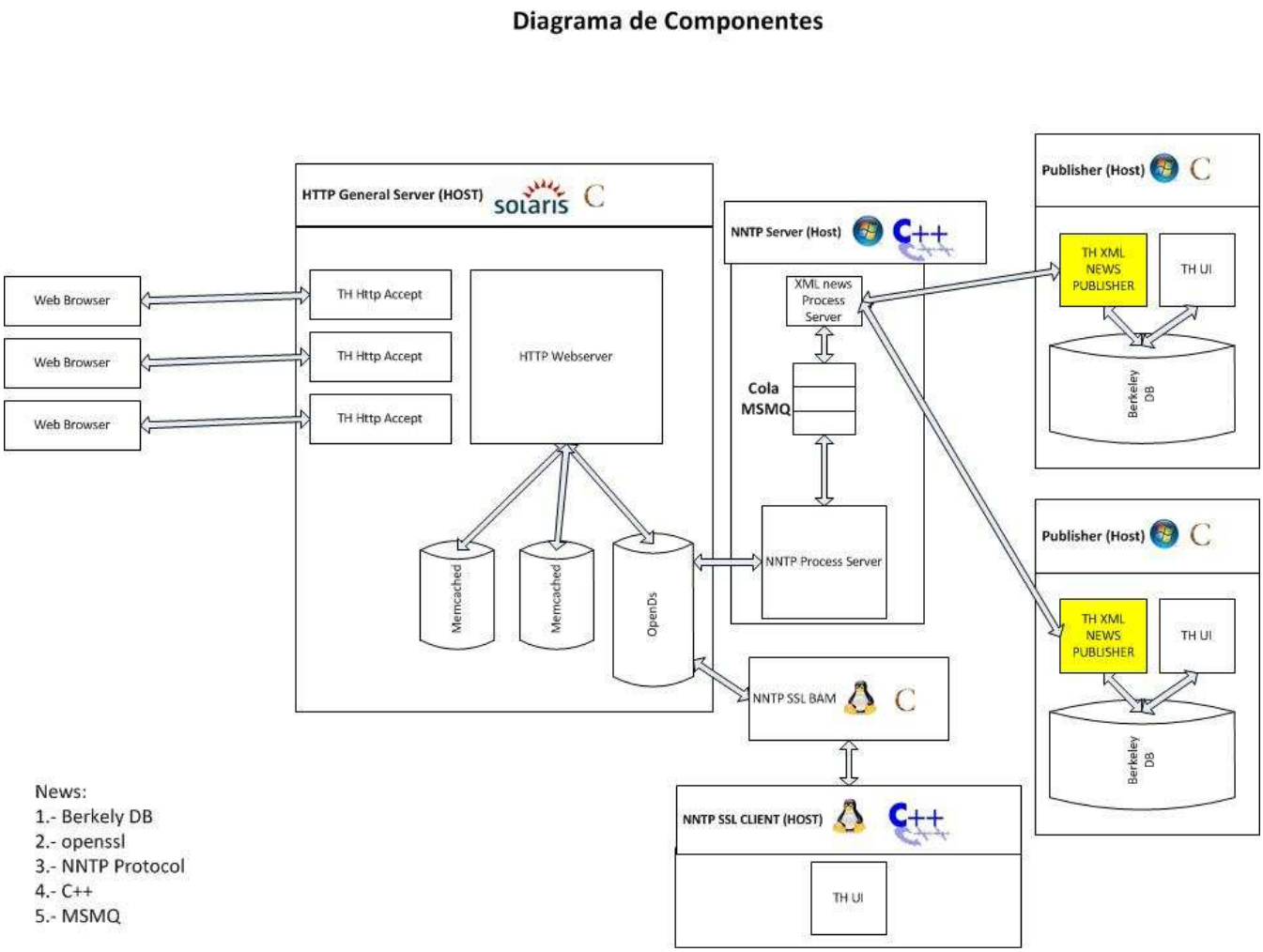


Diagrama de Componentes mínimos



Servidor HTTP - Solaris

Descripción:

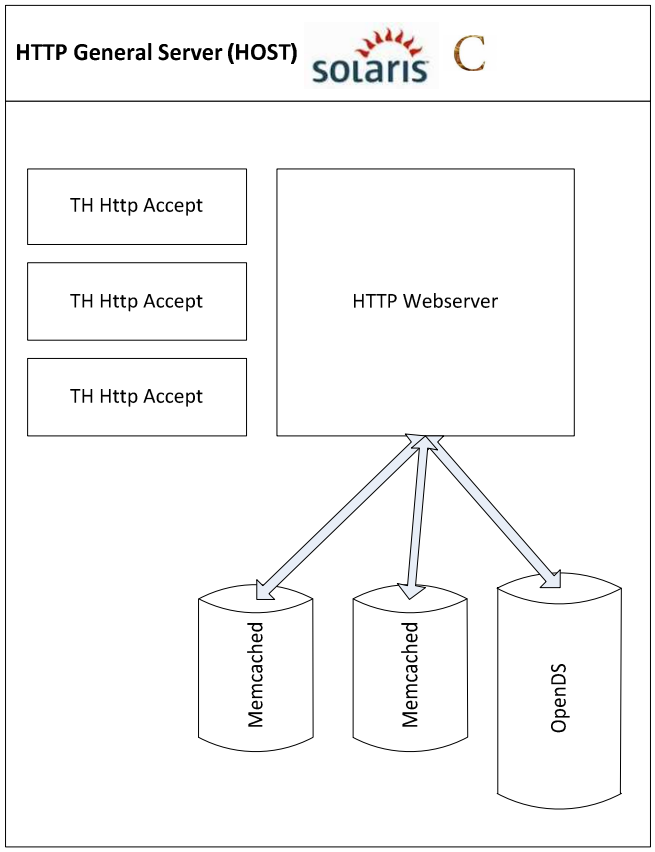
En este nodo se encuentra el servidor de protocolo de transferencia de texto ó HTTP por su sigla en inglés (HyperText Transfer Protocol - http://www.w3.org/Protocols/rfc2616/rfc2616.html). Proveerá servicios de transferencias de noticias usando el protocolo HTTP estándar hacia el exterior. Internamente, este proceso será el encargado de usar las noticias proporcionadas por el Servidor NNTP.

Aspectos de diseño:

La aplicación se desarrollará para la plataforma **Solaris,** con el lenguaje de programación **C** y será **multithread** para aprovechar al máximo las capacidades de los sistemas multiprocesadores actuales.

Este proceso obtendrá la información almacenada en la base de datos de las noticias implementada con **OpenDS.** Estas noticias están a disposición de los usuarios y las servirá a todos los clientes web que se conecten al servidor y soliciten el listado de newsgroups. Listará un resultado con links (estilo google) por cada uno de los newsgroups. Cada link de newsgroup llevará a un nuevo listado de las noticias almacenadas y las servirá al cliente en caso de ser solicitadas.

La aplicación contará con dos servidores de caché implementados con el servicio de **memcached**. Estos le ofrecerán al servidor HTTP un único nivel de caché de datos en la cual el servidor podrá almacenar y recuperar la información más solicitada por los clientes web sin volverla a solicitarla nuevamente a la base de datos.
La caché mantendrá los datos de acuerdo al algoritmo interno de selección de víctima con el que se configurará memcached.



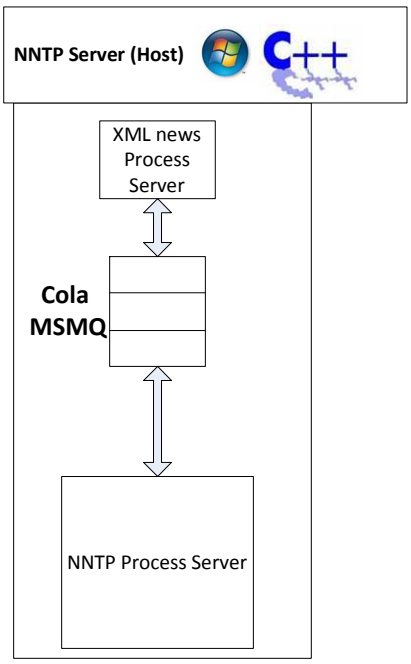
Servidor NNTP - Windows

Descripción:

En este nodo se encuentra el servidor de protocolo de transferencia de noticias ó NNTP por su sigla en inglés (Network News Transport Protocol - http://tools.ietf.org/html/rfc3977). Proveerá servicios de almacenamiento de noticias usando el protocolo NNTP hacia el exterior. Internamente, este proceso recibirá las noticias desde los procesos Publisher y los almacenará en la base de datos

Aspectos de diseño:

El servidor estará constituido por dos aplicaciones que se desarrollarán para la plataforma **Windows,** con el lenguaje de programación **C++.**
Se utilizará manejo de procesos y threads para aprovechar al máximo las capacidades de los sistemas multiprocesadores actuales.



Este servidor estará formado por 2 aplicaciones:

1. **Aplicación XML news Process Server:** Este proceso estará a la escucha de conexiones de Mensajes XML de noticias enviadas por los nodos Publisher a través del protocolo IPC/IRC (ver anexo protocolos). Cuando se le conecte un nuevo proceso publisher creará un nuevo thread para atender la solicitud y deberá almacenar la noticia en una cola de mensajes implementada con la tecnología MSMQ.

2. **Aplicación NNTP Server:** El proceso estará a cargo de monitorear la cola de mensajes de **MSMQ**. Levantará las noticias de la cola de mensajes y las guardará en el directorio de datos **OpenDS**.

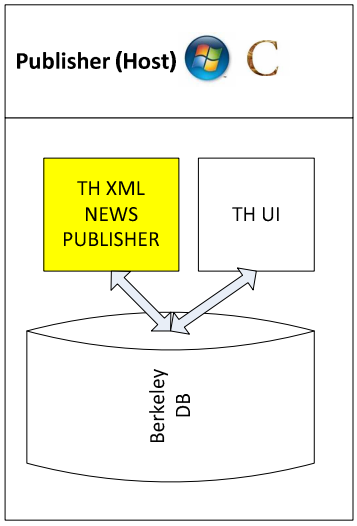
Publisher - Windows

Descripción:

En este nodo se encuentra el servidor Publisher. Será el encargado de proveer servicios de noticias que serán cargadas por los usuarios y trasmitidas al servidor NNTP Server para que este lo almacene.

Aspectos de diseño:

La aplicación se desarrollará para la plataforma **Windows,** con el lenguaje de programación **ANSI C** y será **multithread** para aprovechar al máximo las capacidades de los sistemas multiprocesadores actuales.



El servidor Publisher será un proceso compuesto por dos threads de ejecución:

1. **GUI Thread:** Proveerá una interfáz de usuario por consola, la cual permitirá cargar noticias y almacenarlas en el servidor de base de datos intermedio **Berkeley DB**.

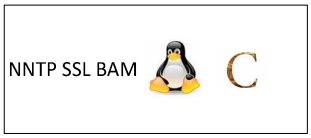
2. **XML News Publisher Thread:** Se encargará de establecer una conexión XML/RPC con el servidor NNTP. A partir de la misma tomará del almacenamiento intermedio las noticias existentes y las enviará al servidor utilizando el protocolo IPC/RPC para el transporte de los mensajes XML (ver anexo protocolos). Las noticias no deberán ser borradas del almacenamiento intermedio. Se marcarán como transmitidas una vez finalizado dicha transacción.

NNTP SSL BAM - Linux

Descripción:

En este nodo se encuentra el Servidor de protocolo de transferencia de noticias ó NNTP por su sigla en inglés (Network News Transport Protocol - http://tools.ietf.org/html/rfc3977). Proveerá servicios al cliente NNTP que le permitirá recuperar las noticias almacenadas en la base de datos utilizando el protocolo NNTP estándar.

Aspectos de diseño:



La aplicación se desarrollará para la plataforma **Linux** con el lenguaje de programación **ANSI C**.

Este Servidor estará formado por un proceso el cual se encargará de establecer la conexión con la base de datos **OpenDS** mediante el uso de la **API** desarrollada para C (http://www.openldap.org/).

A partir de esta conexión podrá consultar las noticias solicitadas por el cliente. El proceso aceptará la conexión desde un único cliente NNTP SSL estableciendo una conexión segura utilizando la **API** desarrollada en C (http://www.openssl.org/) y servirá las noticias solicitadas por el mismo.

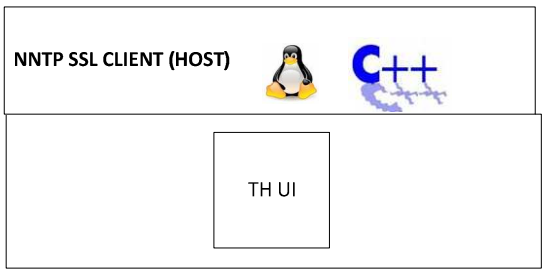
NNTP SSL CLIENT - Linux

Descripcion:

En este nodo se encuentra el cliente de protocolo de transferencia de noticias ó NNTP por su sigla en inglés (Network News Transport Protocol - http://tools.ietf.org/html/rfc3977). Le permitirá al usuario del sistema recuperar las noticias almacenadas en el servidor NNTP utilizando el protocolo NNTP estándar.

Aspectos de diseño:

La aplicación se desarrollará para la plataforma **Linux,** con el lenguaje de programación **C++** y será **multithread** para aprovechar al máximo las capacidades de los sistemas multiprocesadores actuales.



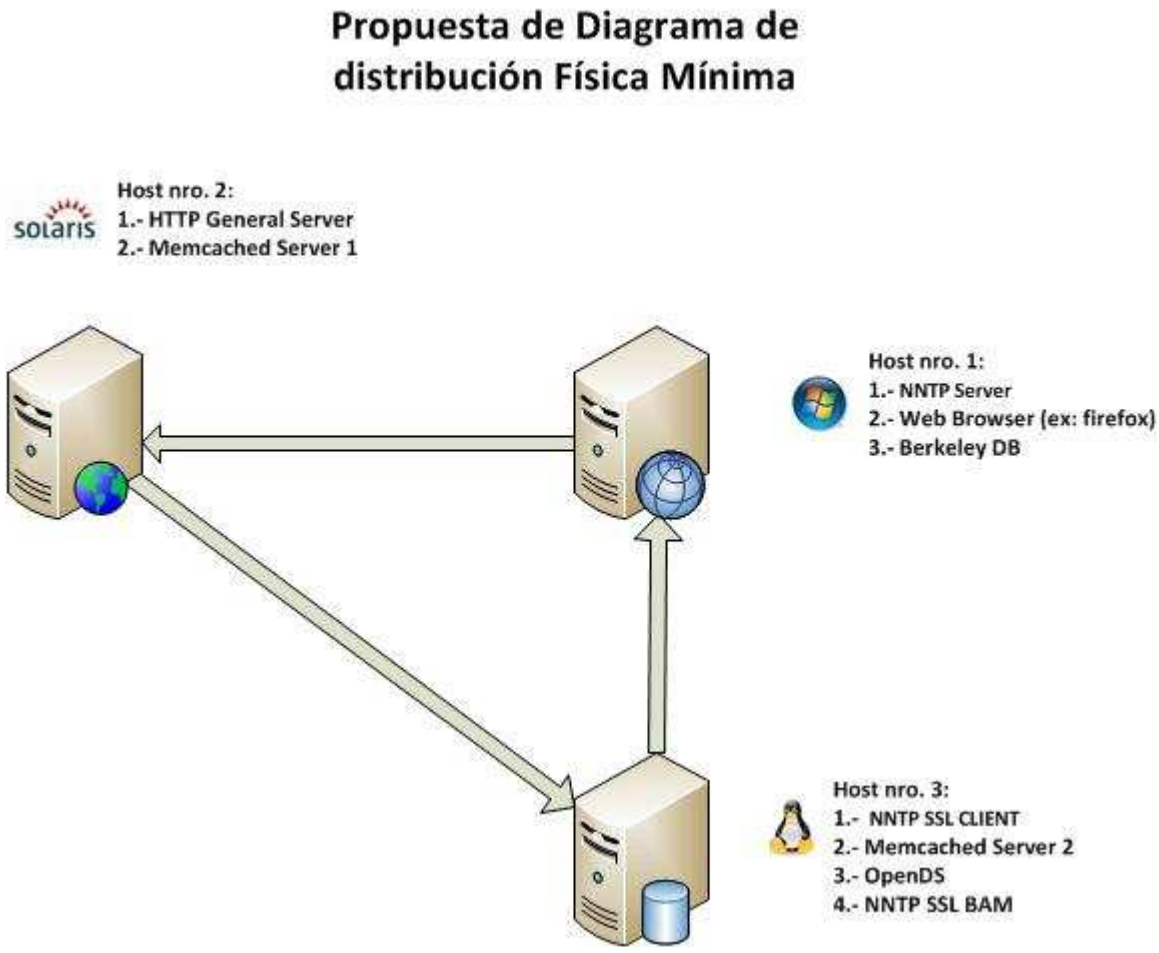
Este cliente de noticias será un proceso formado por dos threads:

1. El thread principal se encargará de establecer la conexión con el servidor NNTP mediante el uso de una conexión AF_INET SSL (http://www.openssl.org/).

2. El segundo thread será el encargado de obtener los pedidos de noticias por palabra clave por parte del usuario. El thread principal los enviará por una conexión segura, recuperando los datos. Por último, el segundo thread interpretará esos resultados para presentárselos al usuario por pantalla.

Diagrama de Distribución de Test

Este es un ejemplo de un despliegue mínimo de componentes que podemos realizar para probar la completitud del trabajo práctico.



4.-Descripción detallada de las entregas

Es importante recalcar que si bien las entregas tienen sus respectivas fechas y están numeradas y ordenadas existe mucho trabajo que correctamente dividido entre los integrantes del grupo se puede realizar de forma paralela.

4.1.-Primer entrega - Linux – C++

Descripción

En esta entrega se deberá desarrollar el **NNTP SSL CLIENT**. Será necesario implementar los comandos estándar del protocolo nntp (http://www.faqs.org/rfcs/rfc977.html).

Al momento de iniciarse el proceso, este generará un nuevo thread para la interface de usuario y se conectará mediante un canal seguro (Secure Socket Layer) a un servidor NNTP (por ejemplo nntpd). Para implementación de la conexión SSL se utilizará la biblioteca open source openSSL (http://www.openssl.org/).

La interface de usuario de la aplicación es manejada por un thread que le permite al usuario obtener e imprimir comandos NNTP desde el teclado y hacia la pantalla (stdout). A partir de la lectura de un comando NNTP, se procederá a enviarlo por la conexión socket SSL (AF_INET) y se aguardará la respuesta del servidor. Esta respuesta se imprimirá por pantalla para luego tomar por consola el siguiente comando. El programa finaliza con el comando NNTP QUIT, enviando el mensaje al servidor NNTP, cerrando la conexión y finalizando el programa.

Los comandos NNTP a implementar y soportar por el cliente son:

- **ARTICLE**
- **BODY**
- **GROUP**
- **LISTGROUP**
- **HEAD**
- **LIST NEWSGROUPS**
- **NEXT**
- **POST**
- **QUIT**

Modo de testeo de la entrega	
• Instalando el servidor de nntpd server process en Linux podemos configurarlo y disparar comandos de inserción de noticias para luego consumirlo desde nuestro cliente	
Tiempo estimado	una semana
Tipo de entrega	No Obligatoria
Fecha	24/04/2010

4.2.-Segunda entrega - Solaris - C

Descripción

En esta entrega deberán desarrollar por completo el Servidor HTTP. Interactuará con los browsers mas conocidos soportando algunos comandos de la versión 1.1 del protocolo HTTP (http://www.faqs.org/rfcs/rfc2616.html). El servidor contará con una **capa lógica de datos** de la cual obtendrá las noticias para servirlas a todos los clientes que las requieran.

La capa lógica de datos estará formada por un directorio de datos implementado con la tecnología OpenDS (http://www.opends.org/) y 1 cluster de 2 servidores de memoria compartida distribuída implementados con la tecnología memcached (http://memcached.org/).

Cada registro de noticia almacenado en directorio, será consultado previamente en la memoria distribuida (representado en este caso por un cluster memcached) utilizando la API libmemcached (http://tangent.org/552/libmemcached.html).
De no encontrar el registro, se procederá a consultar el directorio de datos. Una vez obtenido el registro y antes de ser enviado al cliente que lo solicita, se lo almacenará en la caché para que la próxima vez que se consulte, se encuentre, alivianando el acceso a disco por parte de la base de datos de OpenDS.
De encontrarse la caché llena, será el propio servicio de clustering de memcached, encargado de seleccionar la víctima para el reemplazo en la caché.

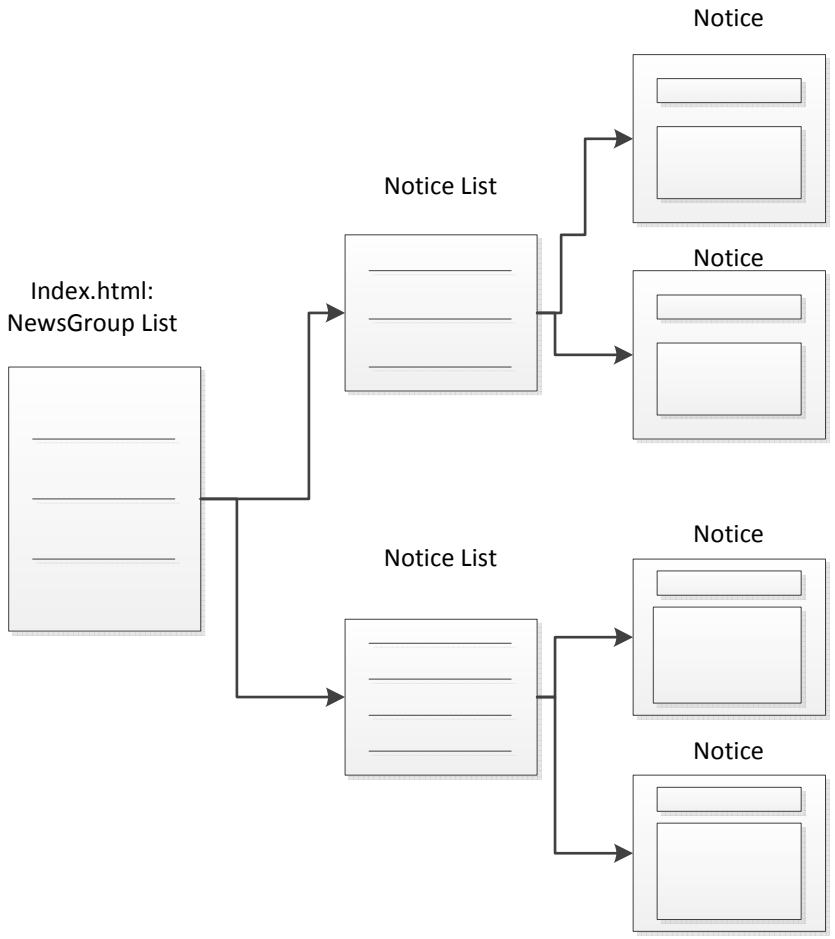
La información de noticias estará guardada en registros con el siguiente formato:

Newsgroup	idNoticia	HEAD	Body
Minuto.com	12345	El tp de operativos no deja dormir a los alumnos!!!	"..Esta vez los ayudantes de operativos no tienen perdón con el tp que se mandaron…."

La información será servida a través del protocolo HTTP 1.1 estandar reducido (ver anexo protocolos).
Cada vez que un browser realice una conexión para el envío de un mensaje http, se creará un nuevo thread para atender esa solicitud. Al finalizar la respuesta, se cerrará la conexión y el thread terminará su ejecución.

AL iniciar la conexión el cliente realizará una petición HTTP GET a la URL de inicio del servidor solicitando el archivo "index.html". El cliente recibirá en forma instantánea el resultado con la lista de newsgroups conocidos con un enlace asociado a cada uno de los mismos.
El usuario, al realizar click sobre uno de estos links, generará una nueva petición HTTP GET y el servidor devolverá un listado con los títulos de las noticias conocidas por el servidor de ese newsgroup y un enlace asociado a cada una de las mismas. Este último enlace realizará una última petición http GET que devolverá una página html con los elementos <TITLE> y <BODY> que contendrá los elementos HEAD y BODY de la noticia seleccionada.

Diagrama de Navegación de páginas HTML



Modo de testeo de la entrega	
• Agregar varias noticias a mano en la base de Datos de OpenDS. • Utilizar un browser como Firefox y realizar las peticiones correspondientes al servidor HTTP implementado.	
Tiempo estimado	Tres semanas
Tipo de entrega	No Obligatoria
Fecha	15/05/2010

4.3.-Tercer entrega – Linux – C

Descripción

En esta entrega se desarrollará el proceso NNTP Server BAM (http://www.faqs.org/rfcs/rfc977.html).

Se implementará un proceso NNTP server el cuál quedará a la escucha del cliente NNTP SSL implementado en la primer entrega. Este proceso denominado NNTP SSL BAM aceptará una única conexión de un cliente SSL NNTP (http://www.openssl.org/) que permitirá recibir los comandos del protocolo NNTP. El proceso buscará los datos necesarios en el directorio de datos OpenDS a través de la **API** desarrollada para **C** (http://www.openldap.org/). Enviará las respuestas en el formato estándar definido en el protocolo.

Modo de testeo de la entrega[o]	
• A este nivel tenemos un 60% del proyecto desarrollado e integrado. La forma de probarlo es validar que la información servida por el BAM desde nuestro cliente coincide con la información presentada a los web browser por nuestro servidor HTTP.	
Tiempo estimado	dos semanas
Tipo de entrega	Obligatoria
Fecha	05/06/2010

4.4.-Cuarta entrega – NNTP Server – C++

Descripción

En esta entrega se deberá desarrollar el Servidor NNTP. Este servidor estará constituido por 2 procesos bien definidos.

El primer proceso/aplicación será el encargado de exponer en todo momento un socket de escucha que le permitirá obtener datos de los procesos publisher. Tomará los datos en formato XML utilizando la librería libxml2 para su manipulación (http://xmlsoft.org/) y almacenará los datos en una cola **MSMQ** a la cuál se accederá a través de la API Message Queuing Object Libraries 3.0 (http://msdn.microsoft.com/en-us/library/ms703182(VS.85).aspx). Al aceptar la conexión de un proceso Publisher (dummy), realizará un handshake IPC/IRC (ver anexo protocolo) y se pondrá a la escucha de mensajes XML de noticias transportados por el protocolo IPC/RPC.

El segundo proceso/aplicación cada cierto tiempo (configurable por archivo de configuración) monitoreará la cola MSMQ para levantar los mensajes de noticias pendientes y los almacenará en el directorio de noticias de OpenDS (http://www.opends.org/).

Modo de testeo de la entrega
• Este proceso es bastante difícil de probar, lo que se recomienda en este punto es crear una mini aplicación de prueba que de inicio a la siguiente entrega, se conecte al socket AF_INET y envíe una noticia "hardcoded" al server para ver si llega a la cola y se levante por el proceso que toma de MSMQ y lo almacena en la opends. Monitorear el estado de la base utilizando el cliente de ldapadmin. • Realizar muchos logueos de control en el código grabando en el archivo log para ver que cada paso haga lo que tiene que hacer. • Se recomienda un buen manejo y control de errores.

Tiempo estimado	tres semanas
Tipo de entrega	No Obligatoria
Fecha	26/06/2010

4.5.- Quinta entrega – Entrega Final - Publisher - Windows - C

Descripción

En esta entrega se desarrollarán los nodos Publisher. Cada proceso Publisher representa un newsgroup asociado. El nombre del newsgroup se configurará por archivo de configuración.

El Proceso, al iniciarse, tomará el nombre del newsgroup asociado y lo utilizará para almacenar las noticias en el servidor NNTP. El proceso contará con 2 threads y una base de datos Berkeley DB asociada.
El thread encargado de la interface de usuario se creará al inicio y por línea de comandos permitirá el grabado de noticias en la base de datos Berkeley DB.

Por otro lado, el thread principal deberá crear cada cierto tiempo (configurable por archivo de configuración) un thread encargado de monitorear si existen entradas en la base de datos Berkeley DB (http://www.oracle.com/technology/documentation/berkeley-db/db/gsg/C/index.html). Este thread es un servicio que se crea y lee de la base de datos las entradas existentes, convirtiendo esas entradas en mensajes XML (http://xmlsoft.org/) con el formato definido y enviando los mensajes XML correspondientes a través de una conexión socket AF_INET que establece realizando un handshake IPC/IRC y utilizando el protocolo IPC/IRC (ver anexo protocolo) para empaquetar los mensajes XML, que le permite enviar información a través de la red al servidor.

Luego de enviar los datos, actualizará el registro de la base "transmitted" en 1 y terminará su ejecución (o sea que el thread debe dejar de existir).

El formato propuesto para los mensajes XML es el siguiente:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
      <news>
            <newsgroup>Minuto.com</newsgroup>
            <idNoticia>12345</idNoticia>
            <HEAD>El tp de operativos no deja dormir a los alumnos!!!</HEAD>
            <BODY>UFFF tengo 32 tecnologías en la cabeza</BODY>
      </news>
```

La estructura de la base de datos son registros simples que tendrán conceptualmente el siguiente formato:

Newsgroup	idNoticia	HEAD	Body	transmitted
Minuto.com	12345	El tp de operativos no deja dormir a los alumnos!!!	"UFFF tengo 32 tecnologías en la cabeza"	0

Modo de testeo de la entrega	
• Terminado este servidor el trabajo práctico está completo, la recomendación es realizar el test de stress (*Prueba diseñada para determinar la respuesta de un sistema bajo condiciones de carga*) cargándolo de varias noticias y grupos (10 publishers por ejemplo) y enviar varias noticas al servidor y consultarlas desde varios web browsers al mismo tiempo. • 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. • Si bien es redundante la implementación del campo newsgroup, simplifica el deploy para el testeo del proyecto, permitiendo que una única instancia de Berkeley DB sea utilizada por todos los procesos Publisher al mismo tiempo.	
Tiempo estimado	dos semanas

Tipo de entrega	Obligatoria presentación del práctico
Fecha	10/07/2010

5.- Requerimientos técnicos y limitaciones

Programación en C++

Para el desarrollo en la plataforma Windows solo estará permitido el uso de la Standard C++ Library (Ver Anexo Documentación). Todo biblioteca externa, no especificada en esta sección queda prohibida. Esto garantiza que los desarrollos sean reproducibles en todos los ambientes de trabajo. Un ejemplo de bibliotecas no permitidas son: MFC (Microsoft Foundation Class Library) y .NET Framework.

Dynamic Link Libraries

La creación de la Dll se realizará desde un proyecto nuevo en Visual Studio. El tipo de proyecto será para la creación biblioteca de enlace dinámico y estático. Como opción adicional no se exportarán símbolos.

Propiedades del proyecto

El proyecto del VS deberá ser configurado con las siguientes propiedades:

Propiedades C/C++:

- Runtime Library: Multi-Thread (/MT)
- Calling Convention: __cdecl (/Gd)
- Compile As: Compile as C Code (/TC)

Exportación de funciones

La forma de exportación se hará utilizando el modificador *__declspec(export)* por lo que no será necesario utilizar un module definition file (.def). La Dll posee una tabla de exportación con los nombres de todas las funciones que la misma exporta a otros ejecutables. Esta tabla se podrá consultar desde la herramienta DUMPBIN con la opción /EXPORT que nos brinda el Visual Studio.

Linking

El método de enlace que se usará en la aplicación será el enlace explicito. Por esto no será necesario enlazar la aplicación con ninguna import library (.lib). El enlace será hecho por la aplicación en run-time y para lograrlo se utilizarán las llamadas: LoadLibrary, GetProcAddress, FreeLibrary.

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 y no debe ser inicializada desde la Dll.

Los parámetros inicialización serán los siguientes:

Parámetros

Versión solicitada

Se utilizará la versión 2.2.

Process Management y Multithreading

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*). Para C++ está permitido el uso de los operadores *new* y *delete* para la alocación de objetos.

Solaris y Linux

No existen restricciones en cuanto al uso de memoria.

File system

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 ó de C++ para manejo archivos (*Stdio.h: fread, fwrite, fopen, etc*).

Para Linux y Solaris no existe tal restricción.

Structured Exception Handling

Windows, Solaris y Linux (sólo código C)

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) en Windows.

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**.

Código C++ en Visual Studio

Los mecanismos de *exception handling* (try, catch, throw)serán permitidos cuando el código compilado sea C++. El lenguaje ya trae mecanismos propios para manejar excepciones basados en ANSI C++ standard.

LDAP y OpenDS

Como implementación de LDAP se utilizará OpenDS de SUN. Se encuentra disponible en la Virtual Machine de Solaris de la cátedra y se debe utilizar esta versión. 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.

La cátedra da soporte para OpenLDAP brindando un wrapper de la biblioteca original. Se puede obtener del grupo oficial de la cátedra en la sección de archivos.

6.-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 la plataforma 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*).

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.

7.-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 Processes**

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

- **Dynamic Link Libraries**

  http://msdn.microsoft.com/en-us/library/ms682589(VS.85).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

- **Standard C++ Library Reference**

  http://msdn.microsoft.com/en-us/library/cscc687y(VS.80).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 cómo utilizar ***OpenLDAP SDK*** se encuentra disponible en la siguiente dirección:

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

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

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

Información sobre cómo utilizar ***memcached*** puede encontrase en los siguientes enlaces:

http://www.danga.com/memcached/

Las API clients de ***memcached*** las pueden encontrar en:

http://code.google.com/p/memcached/wiki/Clients

Open SSL se puede bajar desde:

http://www.openssl.org

Ejemplos clientes/servidores SSL desarrollados para Linux:

http://www.rtfm.com/openssl-examples/

Ejemplos de Berkeley DB con la API de C:

http://www.oracle.com/technology/documentation/berkeley-db/db/gsg/C/index.html

Ejemplo de MSMQ utilizando COM en C:

http://msdn.microsoft.com/en-us/library/ms811055.aspx

Un ejemplo simple (en VB) de cómo crear una Queue privada:

http://msdn.microsoft.com/en-us/library/ms704021(VS.85).aspx

MSMQ Send Message using COM
http://msdn.microsoft.com/en-us/library/ms706212(VS.85).aspx

MSMQ receive Message using COM
http://msdn.microsoft.com/en-us/library/ms705040(VS.85).aspx

Ejemplos de como enviar y recibir mensajes usando el tipo de datos Variant en COM
http://www.codersource.net/mfc_msmq_program_sample.html

libxml2 se puede descargar de:
http://xmlsoft.org/

8.- Anexo – Protocolos de comunicación

8.1. - NNTP SAMPLES

http://www.w3.org/Library/

http://www.w3.org/Library/src/

http://www.faqs.org/rfcs/rfc977.html

8.2. - Protocolo HTTP 1.0/1.1

A) MENSAJES GET

A.1) 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/1.1 404 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/1.1 200 OK\n
Content-type: text/html\n
```

```
Content-Disposition: attachment; filename="<Filename>"\n\n
```

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

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

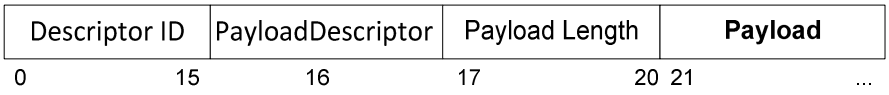
D) MENSAJES TIME OUT

D.1) TIME OUT RESPONSE -- STANDARD

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

8.3. - 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.

8.4. - Handshake

Para realizar los handshake entre los procesos se deberá enviar un mensaje de request del protocolo IPC/IRC con los siguientes valores:

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

PayloadDescriptor:
Nro. identificando el proceso que se está conectando, reservado para iniciar la conexión.

***PayLoad Lenght*:** 0, indicando que no hay payload.

El response por su parte tendrá el siguiente formato:

Descriptor ID:
El identificador del request para determinar a qué mensaje pertenece.

PayloadDescriptor:
Nro. reservado con 2 valores posibles: ok y fail (a continuación de un response fail se deberá cerrar la conexión).

PayLoad Lenght: 0, indicando que no hay payload.