Universidad Tecnológica Nacional – Facultad Regional Buenos Aires
Ingeniería en Sistemas de Información
Sistemas Operativos (95-2027) – 2008

Sistemas Operativos – Trabajo Práctico Nro. 2 2C 2008

Sistemas de descarga de archivos

Revisión 0.9

Índice

1.-Introducción

El trabajo práctico de este cuatrimestre consiste en el desarrollo de un sistema distribuido el cual permite realizar búsquedas de archivos y descargarlos por la red. Está compuesto por cuatro componentes principales: un servidor de red, un servidor de caché, un servidor de archivos y un cliente de descarga. El cliente de descarga, se desarrollará sobre el sistema operativo Windows, por el contrario los otros 3 componentes se desarrollarán sobre el sistema operativo Linux.
Estos componentes deben tener la capacidad de instalarse en diferentes equipos y todos ellos se comunicarán mediante conexiones utilizando el protocolo TCP/IP.

Se enumeran a continuación los conceptos teóricos más significativos que cubre el trabajo práctico a saber:

- Creación y manipulación de procesos.
- Creación y manipulación de threads (hilos).
- Sincronización de Procesos e IPC.
- Administración de Memoria mediante algoritmos de selección de víctima.
- Administración de Archivos.
- Sistemas Distribuidos.
- Arquitecturas basadas en protocolos.

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 sobre Linux y Windows.
- Que el grupo de alumnos aprenda el trabajo en equipo y la responsabilidad 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.

Servidor de red

El servidor de red es el proceso que tiene como objetivo conectar el cliente de descarga de archivos con los servidores de archivos. A través del mismo, los clientes realizan las peticiones de los archivos a descargar y el servidor de red distribuye la consulta a todos los servidores de archivos conocidos en la red.
Cuando los servidores de archivos devuelven el resultado de la consulta el servidor de red recoge todas las respuestas, las unifica y envía un único mensaje con el resultado de la búsqueda.
Se encarga también de consultar y actualizar la información que le brinda el servidor de caché cuando está disponible.

Servidor de caché

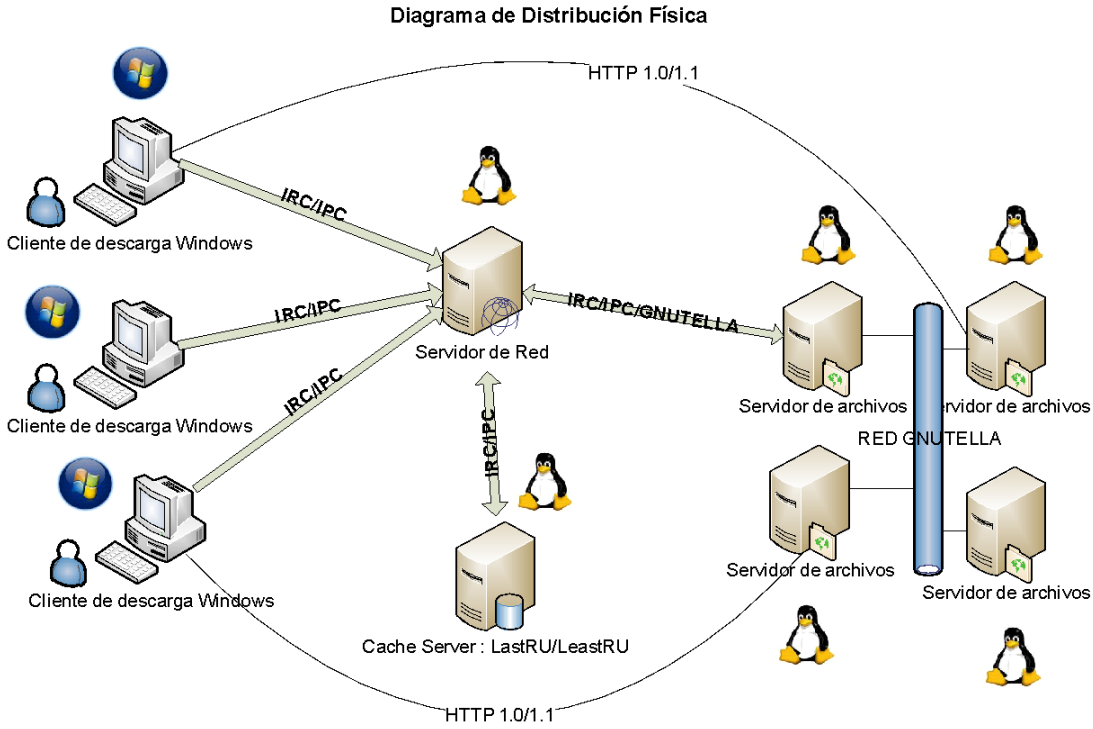
El servidor de red puede aceptar la conexión de un único servidor de caché. A partir de la existencia del mismo, cada resultado de búsqueda que el servidor de red obtiene de los servidores de archivos, el servidor de caché la debe almacenar en su memoria local con el objetivo de que la próxima búsqueda que tenga el mismo requerimiento sea consultada al servidor de caché en vez de ser consultada a los servidores de archivos, optimizando los tiempos de respuesta.
Está caché (en memoria) de información tendrá un tiempo de vida configurable por archivo de configuración. La arquitectura de la memoria caché estará definida por dos algoritmos de selección de víctima: Last Recently Used y Least Recently Used.

Servidores de archivos/Servidores Gnutella

El servidor de archivos consta, como mínimo, de dos procesos:
Un proceso encargado de: administrar la red gnutella, administrar los archivos compartidos en la red y encargado de aceptar conexiones para nuevas descargas. El segundo proceso se encarga de realizar el upload de un archivo.
El primer proceso en primer lugar debe informar por cada búsqueda si en su directorio de archivos compartidos se encuentra algún archivo que cumpla con el requerimiento de la búsqueda.
En segundo lugar debe aceptar conexiones de los diversos clientes de descarga y servírselos para que lo descarguen. Para lograrlo creará un nuevo proceso de upload por cada archivo solicitado.

Cliente de descarga de archivos

La idea principal de esta parte del trabajo práctico se basa en acoplar un cliente al servidor de archivos que se estará ejecutando sobre plataforma Linux. Este cliente consiste en una aplicación desarrollada en lenguaje C sobre Windows utilizando la API que este ofrece. Es el punto de entrada del usuario de este sistema. Contiene la única interfaz con la cual se va a manipular el sistema. A través de la misma, el usuario podrá realizar búsquedas de archivos y seleccionar los archivos que desea descargar. Los archivos se pueden descargar en forma simultánea.
Por cada solicitud de descarga de archivo, el proceso deberá crear un nuevo thread de descarga. Cuando el cliente de descarga finalice su ejecución, se informará en un reporte las métricas finales de procesamiento.



4.-Descripción detallada de las entregas

4.1.-Primer entrega

Descripción

Cliente de descarga de archivos:

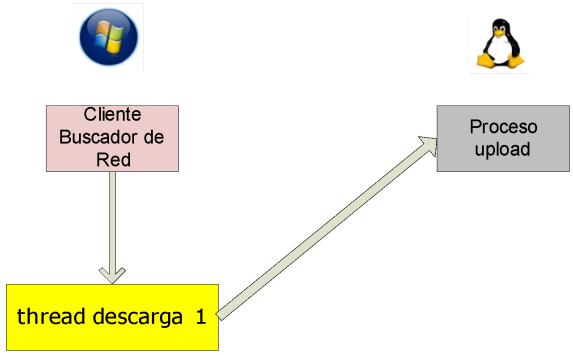
Se deberán crear 2 procesos: uno en Windows y otro en Linux. El proceso que se ejecuta en Windows será un programa comando que recibirá por parámetros el nombre de un archivo a descargar.
La IP y el puerto del proceso de upload desarrollado en Linux como el de download en Windows deberán ser obtenidos de los correspondientes archivos de configuración.

Una vez que este proceso se ejecute, creará un nuevo thread que establecerá la conexión con el servidor Linux en forma automática para descargar el archivo. A continuación enviará un mensaje *HTTP GET* al servidor de descarga con el nombre del archivo que desea descargar. El servidor Linux atenderá esta petición y buscará el archivo en su directorio de archivos compartidos. Si el archivo existe enviará un mensaje *HTTP 200 OK* y a continuación, por la misma conexión, enviará el archivo para que el cliente lo descargue. Una vez finalizada la transferencia cerrará la conexión.
Si el archivo no se encuentra en el directorio especificado se enviará un mensaje *HTTP 404 NOT FOUND* según corresponda.

Una vez realizada la descarga, logueará en el archivo Log el tiempo de descarga del archivo, el nombre y todo lo detallado en la sección "Cliente de Descarga".
El proceso servidor desarrollado en Linux se pondrá a la escucha de aceptar conexiones para realizar la descarga de un archivo, así como también se encargará de servir el archivo que se pida por cada conexión.

El trabajo práctico debe soportar ambos estándares del protocolo HTTP (1.0/1.1), para más información ver anexo protocolo nro. 1 HTTP.

Gráfico Representativo:



Modo de testeo de la entrega

- En primer lugar, una vez realizada la descarga, se puede probar si la integridad del archivo es correcta ejecutando en ambos extremos de la conexión el comando md5sum o algún programa que permita realizar checking de integridad.

- El servidor Linux de descargas debe ser capaz de enviar un archivo solicitado a través del browser (por ejemplo Firefox). Respetando el protocolo HTTP el navegador funciona como un cliente de descarga más. Por ejemplo: `http://127.0.0.1:8080/home/test.txt?v=20090101`

Nota: Los modos de testeo de entrega son recomendaciones que se indican con el objetivo de que el grupo pueda hacer algunos tests básicos sobre la aplicación. Bajo ningún aspecto se considera que cumpliendo estas pruebas la entrega estará aprobada.

Tiempo Estimado: 3 semanas
Tipo de entrega: obligatoria
Fecha: 13-09-08

4.2.-Segunda Entrega

Descripción

Servidor de red:

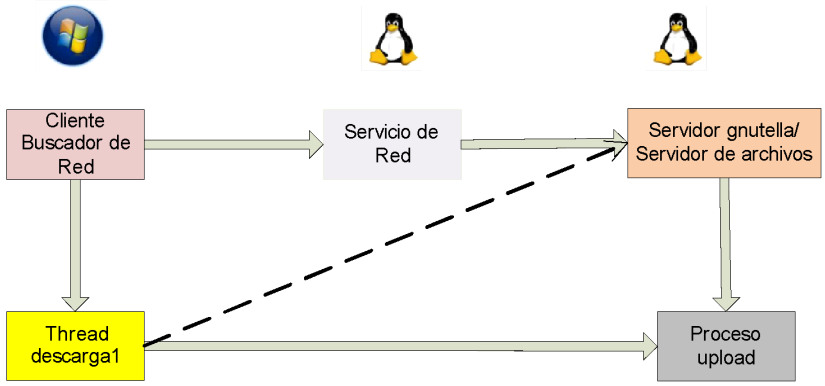
En esta entrega se deberá desarrollar el servidor de red en Linux. Para lo cual se deberá crear un nuevo proceso en Linux que atienda conexiones de los clientes de descarga de Windows y pueda establecer conexiones con los servidores de archivos (Linux).
Al inicio de ejecución el servidor de red se conectará a un servidor de archivos y realizará un handshake por protocolo IRC/IPC Standard (ver anexo protocolo nro. 2 IPC/IRC) identificándose como un Servidor de Red. A partir de ese momento, estará preparado para aceptar conexiones de los clientes de descargas de Windows.

El cliente de Windows solicitará un archivo por consola y enviará un mensaje de request al servidor de red a través del protocolo IPC/IRC Standard. El servidor de red tomará esta petición y la convertirá en una solicitud Query GNUTELLA standard, enviándola luego al servidor "gnutella/servidor de archivos" directo que está conectado. Este servidor buscará en su directorio de archivos todos los nombres de archivos que respondan a ese pedido y responderá con el correspondiente QUERY HIT al Servidor de Red con la IP y el PUERTO en el cual el servidor "gnutella/servidor de archivos" está escuchando para procesar las descargas de archivos. La IP y PUERTO de escucha, se deben poder configurar por archivo de configuración.

Ejemplo: si busco "pepe.txt" y el servidor tiene en su directorio "pepe.txt", "pepe.txt.p" y "pep.txt" devolverá los 2 primeros archivos.
El mapeo del nombre TIENE QUE SER POR INCLUSION.
(Para más información ver protocolo GNUTELLA en las especificaciones del mensaje QUERYHIT)

El Servidor de Red enviará una respuesta con todos los nombres de archivos y el servidor que tiene asociado (IPs y PUERTOs del servidor "gnutella/servidor de archivos"), al cliente Windows correspondiente. El cliente Windows listará por consola el listado de archivos y el identificador del servidor al que pertenece.
El usuario seleccionará el archivo que desea descargar y el cliente Windows creará un nuevo thread para la descarga. El thread se conectará e iniciará un HTTP GET. El servidor buscará en su directorio si el archivo existe y responderá un HTTP 200 OK y creará un nuevo proceso que enviará el archivo al cliente a través de la misma conexión. Si el archivo no existe, enviará un HTTP 404 Not Found y cerrará la conexión.
Se deben implementar los mensajes: ***PING/PONG/QUERY/QUERY HIT*** del protocolo gnutella (ver anexo correspondiente).

Gráfico Representativo:



Modo de testeo de la entrega

- Es importante que el servidor de red soporte varios clientes de descarga Windows con lo cual se debería de poder solicitar al servidor de red varios pedidos, por el otro lado, el servidor "gnutella/servidor de archivos" debe soportar múltiples descargas. 1 POR CADA CLIENTE DE DESCARGA CONECTADO AL MISMO.
- Se debe hacer un correcto uso de los recursos del sistema: consumo de CPU por proceso, manejo eficiente de la memoria dinámica.

Nota: Investigar los comandos de linux: top, ps, free, uptime, entre otros.

Tiempo Estimado: 3 semanas
Tipo de entrega: No obligatoria.
Fecha: 04-10-08

4.3.-Tercer Entrega

Descripción

Red gnutella/Servidores de archivos:

En esta entrega se deberá poder levantar la red punto a punto gnutella completa. Para esto, los servidores de archivos deberán poder establecer reglas de ruteo para encontrar los diversos nodos en la red (ver anexo protocolo nro. 3 gnutella).

Cada vez que un servidor "gnutella/servidor de archivos" se levanta, se pondrá a la escucha de nuevas conexiones. Si se le conecta un nuevo servidor "gnutella/servidor de archivos", se realizará un HANDSHAKE GNUTELLA (ver anexo protocolo nro. 3 gnutella). Este nuevo servidor, le enviará un ping y el servidor que lo reciba disminuirá el campo TTL en 1 (valor que debe poder establecerse por archivo de configuración) e incrementará el campo Hops. El servidor responderá el PING con el correspondiente PONG y luego propagará el PING a todos sus servidores conectados, salvo que alguno de ellos sea el servidor de Red.

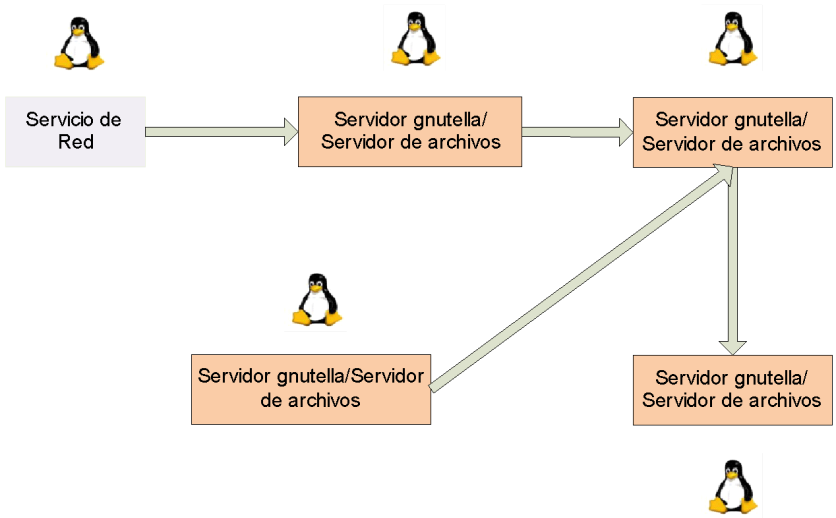
Cuando el servidor "gnutella/servidor de archivos" que propagó el PING reciba un PONG cuyo ID corresponda al PING de origen, enviará la respuesta por el mismo camino que recibió el PING al servidor "gnutella/servidor de archivos" correspondiente exclusivamente. Todas las reglas de ruteo que se ejecutan en cada uno de los servidores deben ser grabadas en el archivo log de ejecución.

- Los mensajes PING y PONG son generados por los servidores "gnutella/servidor de archivos" exclusivamente.
- Los mensajes QUERY son generados por los servidores de Red exclusivamente y aplican las mismas reglas de propagación que los mensajes PING.

- Los mensajes QUERY HIT son generados por los servidores "gnutella/servidor de archivos" y aplican las mismas reglas de ruteo que los mensajes PONG.

Todas las reglas de ruteo pueden ser leídas en el protocolo gnutella en el anexo. En el protocolo gnutella se nombra a los procesos con el nombre "SERVENT". Este proceso es para nosotros el proceso "gnutella/servidor de archivos".
Se deben implementar el mensaje ***HANDSHAKE*** del protocolo gnutella (ver anexo correspondiente).

Gráfico Representativo:



Modo de testeo de la entrega

- Lo más importante a testear es la red de ruteo por lo que se recomienda conectar la mayor cantidad de nodos posibles y desconectar servidores intermedios (sin generar islas) intentando que los mensajes PING puedan seguir encontrando los nodos más lejanos.

Tiempo Estimado: 3 semanas
Tipo de entrega: obligatoria.
Fecha: 25-10-08

4.4.-Cuarta Entrega

Descripción

Servidor de Cache:

En esta entrega se deberá desarrollar un nuevo proceso Linux denominado servidor de Caché. Este servidor cuando se conecte al servidor de red le proporcionará una caché con la cuál le permitirá evitar la sobrecarga de consultas en la red gnutella guardando en forma temporal los request sucesivos y las respuestas correspondientes.

Para lograrlo contará con una lista en memoria de request/response. Esta lista deberá ser de tamaño fijo (configurado por archivo de configuración).
Una vez que esté completa, frente a un nuevo request/response entregado por el servidor de red, el servidor de caché deberá reemplazar un elemento de la lista por este nuevo candidato. La selección de la víctima estará reglada por el algoritmo de selección de víctima ejecutando en ese momento. Los algoritmos pueden ser ***Last Recently Used*** y ***Least Recently Used***.

Cuando se inicia el servidor de caché, se leerá desde el archivo de configuración la IP y el PUERTO del servidor de red al cual se deberá de conectar. A su vez también leerá de ese archivo con cuál de los dos tipos de algoritmos iniciará su ejecución.

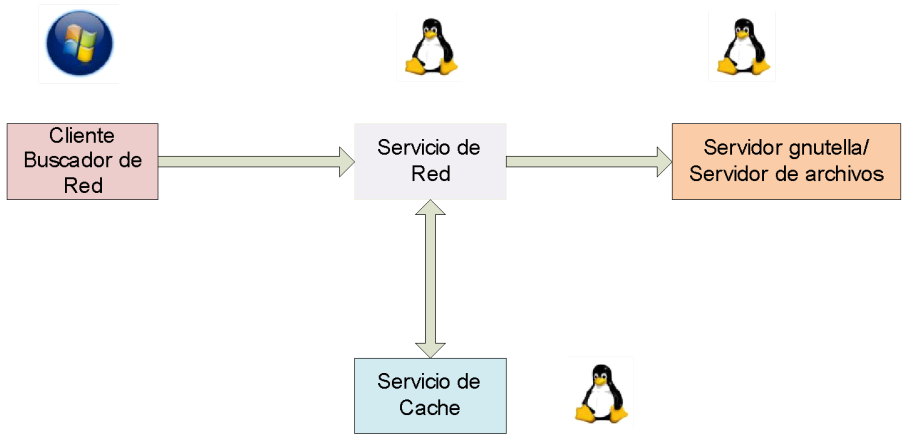
Para el cambio de algoritmo se deberá respetar los siguientes requerimientos:

- Si el servidor de caché recibe la señal ***SIGUSR1*** deberá, en tiempo de ejecución, cambiar como algoritmo de selección de víctima el algoritmo *Last Recently Used*.

- Si el servidor de caché recibe la señal ***SIGUSR2*** deberá, en tiempo de ejecución, cambiar como algoritmo de selección de víctima, el algoritmo Least Recently Used.

Cuando el servidor de caché se conecte al servidor de red este le enviará un mensaje IPC/IRC comunicándole que está listo para servirlo. A partir de ese momento el servidor de red, antes de ejecutar un query a la red gnutella, preguntará por un mensaje IPC/IRC si ese request se encuentra en caché. De ser así, el servidor de caché responderá la petición con la respuesta correspondiente.
De no existir en la caché, se enviará un mensaje "Not found". A continuación, el servidor de red iniciará la petición al servidor de archivos al cuál se conectó, tomará la respuesta y se la enviará tanto al cliente de descarga Windows como al servidor de caché para que la cargue a su memoria. El servidor de caché se comunica con el servidor de Red mediante el protocolo IPC/IRC (ver anexo protocolos) sobre una conexión AF_INET.

Gráfico Representativo:



Modo de testeo de la entrega

- Es importante testear como dinámicamente (en runtime) se cambian los algoritmos. Por otro lado, es importante testear si los mensajes que se intercambian entre el servidor de caché y el servidor de red no se truncan.

Tiempo Estimado: 2 semanas
Tipo de entrega: no obligatoria.
Fecha: 08-11-08

4.5.-Quinta Entrega

Descripción

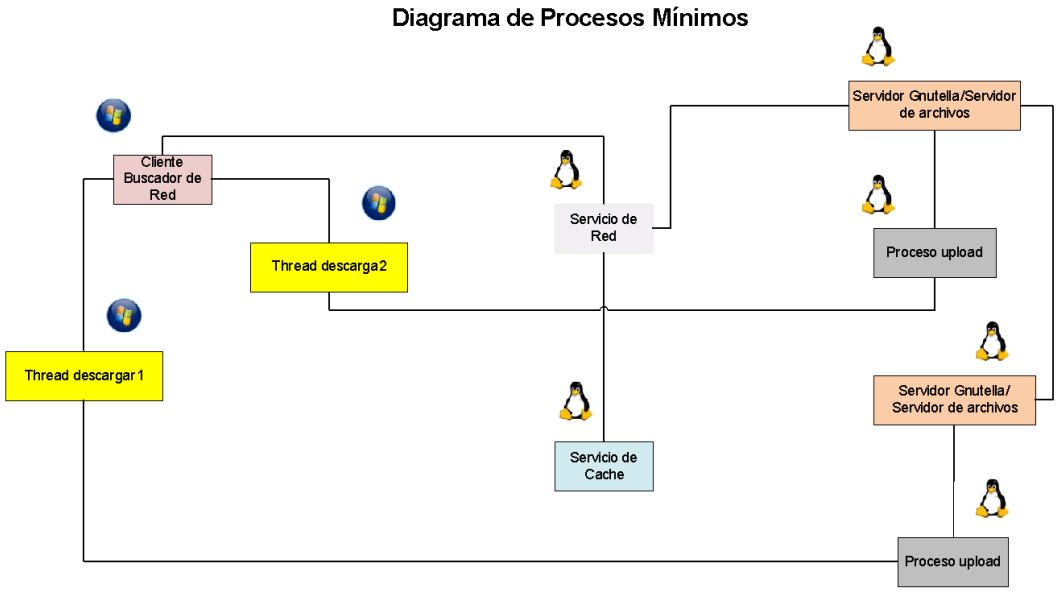
Entrega final integradora:

En esta entrega se deberá integrar todo el trabajo práctico y su comportamiento dinámico. El servidor de caché puede ingresar y salir de la red tantas veces como se quiera sin afectar los criterios de búsqueda.
El servidor de caché, cuando finaliza su ejecución en forma normal, deberá persistir en un archivo los datos de la caché. Cuando el proceso se vuelva a ejecutar deberá tomar los datos de este archivo y regenerar la caché en el estado en que se encontraba.

En otra instancia, se deberá poder admitir reconexión dentro de la red gnutella. Si un Servidor "gnutella/servidor de archivos" está conectado a un único servidor "gnutella/servidor de archivos" y esta conexión se pierde, deberá verificar en su lista de servidores conocidos e intentar de conectarse al primero que le responda. Seguirá intentando conectarse hasta agotar su lista de servidores. El objetivo es eliminar las islas (ver anexo protocolo gnutella para más información).

Cuando un cliente de descarga de archivos realiza una solicitud, el Servidor de Red la intenta de obtener de la caché, si no se encuentra disponible, realizará una consulta a la red gnutella de servidores de archivos, esperará por un tiempo las respuestas de todos los servidores (tiempo configurado por archivo de configuración) y luego el Servidor de Red enviará una respuesta con todos los nombres de archivos y servidores que los contienen (IPs y PUERTOs de cada servidor "gnutella/servidor de archivos"), al cliente Windows correspondiente y al servidor de caché (siempre y cuando esté disponible) para que lo almacene en su memoria. Finalizado el tiempo de espera, si ninguna respuesta es obtenida, se le enviará un mensaje de "Not Found" al cliente para que le avise al usuario que debe realizar una nueva búsqueda.

Se presenta a continuación un diagrama ejemplificador de procesos mínimos que debe conformar esta aplicación.



Modo de testeo de la entrega

- Se recomienda el testing de stress (Prueba diseñada para determinar la respuesta de un sistema bajo condiciones de carga) en la aplicación 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. Es indispensable para la

aprobación del trabajo práctico la entrega completa de la funcionalidad y su correcto funcionamiento

Tiempo Estimado: 3 semanas
Tipo de entrega: obligatoria y de evaluación.
Fecha: 29-11-08

5.-Cliente de Descarga para Windows - Introducción

5.1.-Cliente de Descarga para Windows – Requerimientos funcionales

El cliente de Windows permitirá la descarga de archivos de que se encuentran alojados en distintos nodos de una red de servidores.

La aplicación será configurable a través de un archivo de configuración el cual será cargado al comienzo de la ejecución. Estos parámetros le permitirán al usuario entre otras cosas, especificar la dirección del servidor de la red al que se deberá conectarse, los directorios de descarga, y cualquier otro valor que se considere variable.

Cuando la aplicación se encuentre conectada al servidor de archivos, el usuario ingresará por teclado el nombre del archivo que desee buscar e inmediatamente se realizará la búsqueda por toda la red siguiendo las especificaciones del servidor.
Los resultados se presentarán en pantalla y el usuario podrá seleccionar uno ó más archivos que cumplan con el criterio de búsqueda. Las descargas serán en forma simultánea y el usuario podrá ver los archivos descargados en la ubicación que indicó.

Una vez finalizada la búsqueda y descarga podrá finalizar la aplicación. En ese instante se le presentará al usuario el informe de las estadísticas tomadas durante la ejecución.

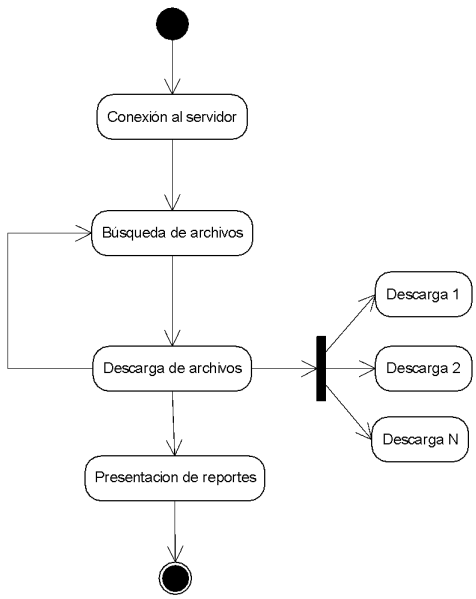


Diagrama de actividad del Cliente

Búsqueda de archivos

Una vez establecida la conexión con el servidor de archivos, se le ofrecerá al usuario la opción de ingresar por teclado el nombre del archivo ó una palabra. La aplicación inmediatamente deberá comenzar la búsqueda para que luego retorne un listado con los resultados.

Un ejemplo de esto sería: si en la red existiesen dos archivos ***"nombrearchivo1.mp3"*** y ***"archivonombre1234.mp3"*** donde la palabra a buscar fuese ***"nombre"*** entonces los dos archivos serían resultados válidos. En cambio, el resultado de ***"nombrearchivo1.mp3"*** solo debería retornar resultados para ese archivo.

```
----------------------------------------------------------------------
CLIENTE DE DESCARGA
Estado: Conectado
IP local: 192.168.1.2
IP Servidor: 192.168.1.10
PUERTO Servidor: 9001

Buscar archivo: unArchivo
----------------------------------------------------------------------
```

Al ingresar el nombre se enviará el pedido al servidor el cual efectuará la búsqueda del mismo a través de toda la red. En ese instante el cliente cambiará su estado indicando que se encuentra efectuando la búsqueda.

```
----------------------------------------------------------------------
CLIENTE DE DESCARGA
Estado: Buscando...
IP local: 192.168.1.2
IP Servidor: 192.168.1.10

Buscando archivo: unArchivo
----------------------------------------------------------------------
```

Descarga de archivos

Completada la búsqueda se informará por pantalla los archivos encontrados, el tamaño del archivo y la dirección donde se encuentra almacenado. Debe existir una opción que le permita al usuario poder realizar la descarga o bien cancelar y volver a buscar. Por ejemplo, la pantalla que se muestra a continuación describe este caso:

```
----------------------------------------------------------------------
CLIENTE DE DESCARGA
Estado: Resultados
IP local: 192.168.1.2
IP Servidor: 192.168.1.10

Resultados del archivo: unArchivo

NúmeroNombre      Fuente  PUERTO SERVERID Tamaño

1 -  unArchivo-musica.mp3     192.168.1.4   2830   ABC13 MB
2 -  unArchivoDeWord.doc192.168.1.11  3540   LKJ120 MB
3 -  backupUnArchivo.dat192.168.1.34  3890   RHG9MB
4 -  unArchivo-OLD1.dat       192.168.1.65  4230   PQR1 MB
5 -  unArchivo-OLD2.dat       192.168.1.11  4628   TTT245 KB
6 -  unArchivo.dat      192.168.1.34  4699   SISOPS   2 GB

Descargar número: 1,3,5,6     Para Cancelar ingrese -1
```

_____

Para comenzar a descargar archivos del servidor, la aplicación aceptará el ingreso del número del archivo por teclado. En caso de realizar una descarga múltiple, se ingresarán varios números separados por una coma con lo cual el sistema se encargará de analizar la entrada. De igual forma, tal como se dijo previamente, cada descarga tendrá su único thread asociado. Una vez interpretados los números, se iniciarán las descargas de los archivos al disco. Inmediatamente, se le presentará al usuario el menú de búsqueda para poder realizar otra consulta si así lo desease.

Reportes de ejecución

Esta es la etapa final de la ejecución de la aplicación y tiene por objetivo presentar un informe en un archivo de texto con las estadísticas tomadas durante toda la ejecución. Los tiempos de Kernel Time y User time se obtendrán de acuerdo a lo indicado en la sección de requerimientos tecnológicos.

Las estadísticas son las siguientes:

- Cantidad de descargas realizadas.
- Cantidad de datos transferidos (medido en bytes) discriminado por archivo.
- Tiempo utilizado por descarga discriminado por archivo.
- Tiempo en que el cliente FTP se ejecutó en modo kernel (Kernel time).
- Tiempo en que el cliente FTP se ejecutó en modo usuario (User time).
- Tiempo total de ejecución del cliente FTP en el sistema.

Algunas aclaraciones:

Toda la información de reporte se loguea cuando se finaliza la operación (por ejemplo una descarga de un archivo). Si la aplicación por algún evento ajeno al sistema deja de funcionar, NO debe loguear ningún suceso ni volcar los estados parciales de los procesos en ejecución.

Configuración de la aplicación

Los parámetros obligatorios configurables por archivo serán el directorio de descarga y la dirección y puerto de conexión del servidor. Cualquier otro parámetro que el alumno considere puede ser añadido al archivo, por ejemplo directorio en donde se deposita el log file.

Protocolo de comunicación: Servidor de Red – Cliente de Descarga

Ver anexo Protocolos del Trabajo Práctico (protocolo nro. 2 IPC/IRC).

5.2.-Cliente de Descarga para Windows – Requerimientos tecnológicos

Para poder realizar el trabajo práctico será necesario disponer, como mínimo, de los siguientes requerimientos:

- Sistema Operativo: Windows XP Professional.
- IDE: Visual C++ 2005.
- Windows SDK (No incluido en la versión Express).

Comunicaciones – Sockets

Para la comunicación cliente/servidor se utilizarán los protocolos especificados en el trabajo práctico y se implementarán utilizando sockets (modelo Berkeley, el mismo utilizado del lado Linux) orientados a la conexión del tipo AF_INET sobre TCP.

Para la programación básica de sockets se va a utilizar la biblioteca Winsock API declarada en *winsock2.h*. Se requiere linkear con la dependecia *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 bilioteca 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.

Referencia a WSAData

Dirección de memoria a una estructura de datos del tipo WSADATA que recibirá los detalles de la implementación de la biblioteca.

Manejo de Threads

Dado que nuestra aplicación es multithread cada solicitud de descarga que el usuario genere el proceso principal la delegará a un nuevo thread quien será el encargado de realizar el proceso.
La aplicación también obtendrá información de tiempos por cada thread. Para esto utilizará la API más adecuada de la biblioteca de *manejos de procesos y theads* y que le permita obtener por ejemplo, tiempos de creación ó cuanto tiempo el thread se ejecutó en kernel mode entre otros.

A continuación se describen los parámetros de creación de un thread.

Nota

Para la creación de threads se utilizará la función *_beginthreadex* perteneciente a la C/C++ Runtime library. De esta manera evitaremos posibles *memory leaks.* (Para investigar: ¿cuales son las causas de este comportamiento al usar la API *CreateThread*?).

Parámetros

start_address (dirección de comienzo)

Dirección de la rutina a ejecutar.

stack_size (tamaño del stack)

Deberá usarse el tamaño del stack por defecto.

arglist (lista de argumentos)

Ver según el caso.

security (atributos de seguridad)

No se deberá usar ningún atributo de seguridad.

initflag (opciones de inicio)

Ver según el caso.

thrdaddr (identificador del thread)

Referencia a una variable que tendrá el identificador del thread. Ver según el caso.

Importante

Cuando se especifica "*"Ver según el caso"* se quiere decir que el parámetro es variable según la necesidad de alumno y se utilizará según se crea más conveniente. Para el resto de los parámetros se indica la opción que se recomienda utilizar ó la que se utiliza por defecto en la llamada. Esto será valido también para las próximas secciones.

Thread Synchronization con Kernel Objects

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. Dado que la aplicación es multithread el alumno podrá hacer uso de estos recursos si lo considerase necesario.

A continuación se dará una descripción de los mecanismos que ofrece la API de Windows 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*

Wait functions

Se podrán utilizar estas funciones para lograr que un determinado thread bloquee su propia ejecución al ser utilizado con un objeto "sincronizable".

Wait functions de un solo objeto

- *WaitForSingleObject(params)*

Wait functions con múltiples objetos

- *WaitForMultipleObjects(params)*

Si se considera necesario el uso de otra función que no está en la lista, podrá ser utilizada justificando su decisión.

Manejo de Heaps

Cada thread que la aplicación genere, creará su propio bloque de páginas adicional en el espacio de dirección del proceso 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.
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 se 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 librería estándar de C para manejo de memoria (*Stdlib.h: malloc, free, etc*).

Para la creación de cada heap se utilizarán los siguientes parámetros de creación.

Parámetros

Opciones

Valor	Significado
HEAP_NO_SERIALIZE 0x00000001	No se usará el acceso serializado en las múltiples llamadas al heap. El uso de esta opción no garantiza la exclusión mutua entre dos o más threads. Definido en Windows.h

Valor inicial

La cantidad de memoria que deberán ocupar las páginas al ser inicializadas es de 1024 Kb.

Tamaño máximo

El heap no tendrá restricciones en cuanto a tamaño. El tamaño máximo estará limitado por la cantidad de memoria disponible en el sistema.

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. Esto se aplica solo a errores generados por las llamadas a las API de Windows. Para los errores pertinentes a la aplicación de deberá respetar las normas de logueo del trabajo práctico.

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*.

Como el *message string* correspondiente a dicho error será obtenido de la tabla de errores del sistema, los parámetros que se utilizarán para las funciones son los siguientes:

Parámetros

Flags

Valor	Significado
FORMAT_MESSAGE_ALLOCATE_BUFFER 0x00000100	El parámetro es un puntero a un puntero **PVOID**. Se alloca suficiente memoria para el mensaje con formato obtenido. La función establece el puntero del nuevo buffer en la dirección del puntero pasado por parámetro.
FORMAT_MESSAGE_FROM_SYSTEM 0x00001000	El mensaje se busca de la tabla de errores del sistema.
FORMAT_MESSAGE_IGNORE_INSERTS 0x00000200	Se ignoran secuencias de escape en el mensaje.

Aclaración: estas opciones son recomendables pero nada impide que se adicionen parámetros extra si el alumno lo considera conveniente.

Identificador del lenguaje

Nuestra aplicación no usará la localización de textos para los mensajes, por lo tanto, si la API de Windows solicita un identificador se puede optar por: dejar que la API utilice el lenguaje por defecto y generar uno por defecto. Por ejemplo, la función definida en Windows.h obtiene el lenguaje por defecto del sistema.

```
WORD MAKELANGID (
  USHORT usPrimaryLanguage,
  USHORT usSubLanguage
)


//Lenguaje por defecto del sistema
dwLangId = MAKELANGID(LANG_NEUTRAL, SUBLANG_SYS_DEFAULT);
```

Structured Exception Handling

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 como se explica en la sección de **manejo de errores**.

Filesystem

Toda operación de I/O que involucre manejo de archivos en Windows deberá ser usando la Win32 API correspondiente en su versión *ANSI,* en caso de existir *UNICODE*. No está permitido el uso de la librería estándar de C para manejo archivos (*Stdlib.h*).
En el caso de que alguna de las operaciones falle deberá ser loguedo la causa del problema tal como se indica en sección sobre Manejo de Errores.

Se deberán tener en cuenta las siguientes consideraciones en la creación de un archivo:

Parámetros

Tipo de acceso

Los tipos mas comunes que se usarán son GENERIC_READ, GENERIC_WRITE, ambos (GENERIC_READ | GENERIC_WRITE) según se crea conveniente.

Modo compartido

El archivo no será compartido y no podrá ser abierto nuevamente hasta que el handle al archivo halla sido cerrado.

Security (atributos de seguridad)

No se usará ningún atributo en especial.

Acción de creación

Acción que se lleva a cabo en el momento de la creación del archivo

Flags And Attributes

Los archivos creados tendrán los siguientes atributos de creación.

Atributo	Significado
FILE_ATTRIBUTE_NORMAL 128 0x80	El archivo no posee otros atributos.

TemplateFile

Este parámetro debe ser NULL.

Time Functions

Estas funciones de la API de Windows se utilizarán para el manejo de tiempos, obtención de información y formatos de fecha. El alumno podrá hacer uso de las funciones cuando lo considere apropiado.

6.-Anexo – Archivo Log

El archivo Log deberá respetar el siguiente formato de presentación tanto en los procesos de Linux como en el de Windows.

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

Descripción

Fecha

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

NombreProceso

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 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 el sitio Web MSDN de Microsoft *http://msdn .microsoft.com*.

- **Threads y Procesos**

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

También se recomienda el siguiente material bibliográfico:

- Programming Applications for Microsoft Windows, 4th edición de Jeffrey Richter.
- Microsoft Windows Internals, 4th edición de Mark E. Russinovich y David A. Solomon.

Con respecto a la documentación de Linux, 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

8.-Anexo – Protocolos de comunicación

A continuación se presentará la especificación de los protocolos que se utilizarán en el trabajo práctico para la comunicación, tanto entre procesos locales como entre las diferentes plataformas.

8.1.-File Tranfer Protocol 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
Content-type: application/octet-stream\n\n
```

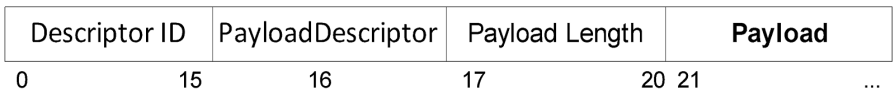
C.2) FILE FOUND WITH 200 OK RESPONSE -- STANDARD

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

Las headers Content-Disposition y Content-type son opcionales pero recomendados para el testing del protocolo.

8.2.-Inter. Process Comunications (IPCs) – Protocol IRC/IPC standard

La comunicación puede establecerse con diferentes tecnologías (Memoria Compartida, PIPES, Sockets, etc.)
Se utilizara un mensaje de protocolo interno, similar a la estructura del GNUTELLA y otros protocolos: 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 (cualquier valor que no coincida con la lista del protocolo Gnutella:0x00, 0x01, 0x80, 0x81).

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 (cualquier valor que no coincida con la lista del protocolo Gnutella:0x00, 0x01, 0x80, 0x81).

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.

8.3.-File Sharing Protocol – GNUTELLA 0.4

Se extiende a continuación la especificación del GNUTELLA 0.4 Parcial. Es limitada a los mensajes que se reutilizan de este protocolo para el TP. Sólo se utilizaran los siguientes Mensajes:

- GNUTELLA HANDSHAKE
- PING
- PONG
- QUERY
- QUERYHIT

El protocolo completo de Gnutella especifica:

1. Mensajes del protocolo File Sharing.
2. Esquema de Ruteo.
3. File Transfer Protocol y modo de descarga.

Para el trabajo práctico sólo se utilizará algunos de los mensajes del punto 1 y las reglas del punto 2. Se describen a continuación:

The Gnutella Protocol Specification v0.4[1] Document Revision 1.2
Clip2 http://www.clip2.com protocols@clip2.com

Gnutella[2] is a protocol for distributed search. Although the Gnutella protocol supports a traditional client/centralized server search paradigm, Gnutella's distinction is its peer-to-peer, decentralized model. In this model, every client is a server, and vice versa. These so-called Gnutella *servents* perform tasks normally associated with both clients and servers. They provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other servents, check for matches against their local data set, and respond with applicable results. Due to its distributed nature, a network of servents that implements the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servents goes offline.

Protocol Definition
The Gnutella protocol defines the way in which servents communicate over the network. It consists of a set of descriptors used for communicating data between servents and a set of rules governing the inter-servent exchange of descriptors. Currently, the following descriptors are defined:

Descriptor	Description
Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.
Query	The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set.
QueryHit	The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.

A Gnutella servent connects itself to the network by establishing a connection with another servent currently on the network. The acquisition of another servent's address is not part of the protocol definition and will not be described here (Host caché services are currently the predominant way of automating the acquisition of Gnutella servent addresses). Once

the address of another servent on the network is obtained, a TCP/IP connection to the servent is created, and the following Gnutella connection request string (ASCII encoded) may be sent:

GNUTELLA CONNECT/<protocol version string>\n\n

where <protocol version string> is defined to be the ASCII string "0.4" (or, equivalently, "\x30\x2e\x34") in this version of the specification.

1

This document represents the de facto standard Gnutella 0.4 protocol. However, several implementations have extended the descriptors that comprise the protocol, and have imposed additional rules on the transmission of these descriptors through the Gnutella network. Known extensions to the protocol are provided in an Appendix at the end of this document, but some variations not documented here may be encountered in practice.

2

Typically pronounced "new -tella" or, less commonly, "guh-new -tella".

A servent wishing to accept the connection request must respond with

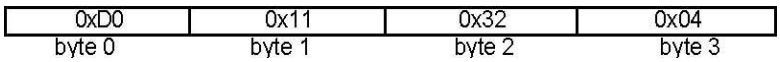
GNUTELLA OK\n\n

Any other response indicates the servent's unwillingness to accept the connection. A servent may reject an incoming connection request for a variety of reasons - a servent's pool of incoming connection slots may be exhausted, or it may not support the same version of the protocol as the requesting servent, for example.

Once a servent has connected successfully to the network, it communicates with other servents by sending and receiving Gnutella protocol descriptors. Each descriptor is preceded by a Descriptor Header with the byte structure given below.

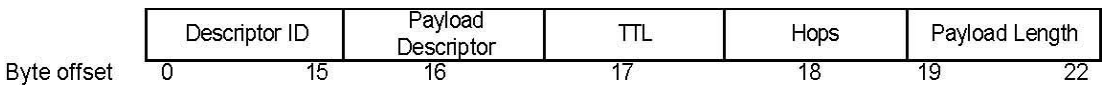
Note 1: All fields in the following structures are in little-endian byte order unless otherwise specified.

Note 2: All IP addresses in the following structures are in IPv4 format. For example, the IPv4 byte array



represents the dotted address 208.17.50.4.

Descriptor Header



Descriptor ID *A 16-byte string uniquely identifying the descriptor on the network*

Payload Descriptor *0x00 = Ping*
0x01 = Pong

0x80 = Query
0x81 = QueryHit

TTL *Time To Live.The number of times the descriptor will be forwarded by Gnutella servents before it is removed from the network. Each servent will decrement the TTL before passing it on to another servent. When the TTL eaches 0, the descriptor will no longer be forwarded.*

Hops *The number of times the descriptor has been forwarded.as a descriptor is passed from servent to servent, the TTL and Hops fields of the header must satisfy the following condition:*

TTL(0) = TTL(i) + Hops(i)
Where TTL(*i*) and Hops(*i*) are the value of the TTL and Hops fields of the header at the descriptor's *i*-th hop, for *i* >= 0.

Payload Length *The length of the descriptor immediately following this header The next descriptor header is located exactly Payload_Length bytes from the end of this header i.e. there are no gaps or pad bytes in the Gnutella data stream.*

The TTL is the only mechanism for expiring descriptors on the network. Servents should carefully scrutinize the TTL field of received descriptors and lower them as necessary. Abuse of the TTL field will lead to an unnecessary amount of network traffic and poor network performance.
The Payload Length field is the only reliable way for a servent to find the beginning of the next descriptor in the input stream. The Gnutella protocol does not provide an "eye-catcher" string or any other descriptor synchronization method. Therefore, servents should rigorously validate the Payload Length field for each descriptor received (at least for fixed-length descriptors). If a servent becomes out of synch with its input stream, it should drop the connection associated with the stream since the upstream servent is either generating, or forwarding, invalid descriptors.

Immediately following the descriptor header, is a payload consisting of one of the following descriptors:

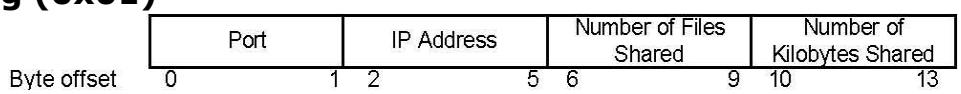
Ping (0x00)

Ping descriptors have no associated payload and are of zero length. A Ping is simply represented by a Descriptor Header whose Payload_Descriptor field is 0x00 and whose Payload_ Length field is 0x00000000.

A servent uses Ping descriptors to actively probe the network for other servents. A servent receiving a Ping descriptor may elect to respond with a Pong descriptor, which contains the address of an active Gnutella servent (possibly the one sending the Pong descriptor) and the amount of data it's sharing on the network.

This specification makes no recommendations as to the frequency at which a servent should send Ping descriptors, although servent implementers should make every attempt to minimize Ping traffic on the network

Pong (0x01)



Port *The port number on which the responding host can accept incoming*

connections

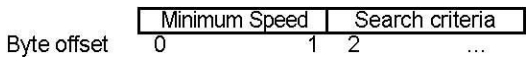
IP Address *The IP address of the responding host. This field is in big-endian format.*

Number of File shared *The number of files that the servent with the given IP address and port is sharing on the network*

Number of Kilobytes Shared *The number of kilobytes of data that the servent with the given IP address and port is sharing on the network.*

Pong descriptors are only sent in response to an incoming Ping descriptor. It is valid for more than one Pong descriptor to be sent in response to a single Ping descriptor. This enables host caches to send cached servent address information in response to a Ping request.

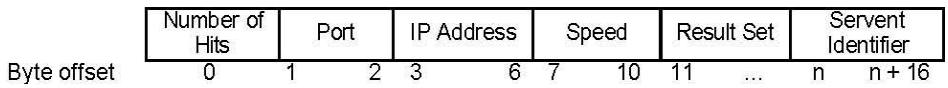
Query (0x80)



Minimum Speed The minimum speed (in kb/second) of servents that should respond to this message. A servent receiving a Query descriptor with a Minimum Speed field of n kb/s should only respond with a QueryHitif it is able to communicate at a speed >= n kb/s

Search Criteria A nul (i.e. 0x00) terminated search string. The maximum length of this string is bounded by the Payload_Length field of the descriptor header.

QueryHit (0x81)



Number of Hits *The number of query hits in the result set (see below).*

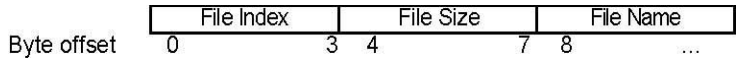
Port *The port number on which the responding host can accept incoming connections.*

IP Address *The IP address of the responding host.*

This field is in big-endian format.

Speed *The speed (in kb/second) of the responding host.*

Result Set A set of responses to the corresponding Query. This set contains Number_of_Hits elements, each with the following structure:



File Index: A number, assigned by the responding host, which is used to uniquely identify the file matching the corresponding query.

File Size: *The size (in bytes) of the file whose index is File_Index.*

File Name The double-nul (i.e. 0x0000) terminated name of the file whose index is File_Index.

The size of the result set is bounded by the size of the Payload_Length field in the Descriptor Header.

Servent Identifier *A 16-byte string uniquely identifying the responding servent on the network. This is typically some function of the servent's network address.*

QueryHit descriptors are only sent in response to an incoming Query descriptor. A servent should only reply to a Query with a QueryHit if it contains data that strictly meets the Query Search Criteria.

The Descriptor_Id field in the Descriptor Header of the QueryHit should contain the same value as that of the associated Query descriptor. This allows a servent to identify the QueryHit descriptors associated with Query descriptors it generated.

Descriptor Routing

The peer-to-peer nature of the Gnutella network requires servents to route network traffic (queries, query replies, etc.) appropriately. A well-behaved Gnutella servent will route protocol descriptors according to the following rules:

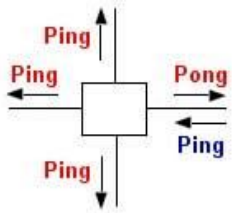
1. Pong descriptors may only be sent along the same path that carried the incoming Ping descriptor. This ensures that only those servents that routed the Ping descriptor will see the Pong descriptor in response. A servent that receives a Pong descriptor with Descriptor ID = *n*, but has not seen a Ping descriptor with Descriptor ID = *n* should remove the Pong descriptor from the network.

2. QueryHit descriptors may only be sent along the same path that carried the incoming Query descriptor. This ensures that only those servents that routed the Query descriptor will see the QueryHit descriptor in response. A servent that receives a QueryHit descriptor with Descriptor ID = *n*, but has not seen a Query descriptor with Descriptor ID = *n* should remove the QueryHit descriptor from the network.

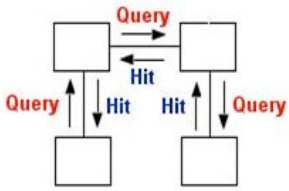
3. A servent will forward incoming Ping and Query descriptors to all of its directly connected servents, except the one that delivered the incoming Ping or Query.

4. A servent will decrement a descriptor header's TTL field, and increment its Hops field, before it forwards the descriptor to any directly connected servent. If, after decrementing the header's TTL field, the TTL field is found to be zero, the descriptor is not forwarded along any connection.

5. A servent receiving a descriptor with the same Payload Descriptor and Descriptor ID as one it has received before, should attempt to avoid forwarding the descriptor to any connected servent. Its intended recipients have already received such a descriptor, and sending it again merely wastes network bandwidth.



Example 1. Ping/Pong Routing



Example 2. Query/QueryHit Routing