Redes de Computadoras – Curso 2013

Capa de Aplicación – Obligatorio 1

Grupo 50
Leticia Arias – 1641989
Sebastián Piazza – 4504299
Santiago Máximo – 4647623

(2013/09/23)

Índice

1 Introducción

El presente documento incluye:

- decisiones de diseño
- máquinas de estado
- pseudocódigo
- documentación de la implementación

Junto con este documento se entrega:

- los archivos correspondientes a la implementación de la aplicación propuesta
- scripts para la construcción del ejecutable (makefile)

1.1 Objetivo del trabajo

- aplicar los conceptos teóricos de capa de aplicación, mediante el desarrollo de una aplicación que funciona en red
- familiarizarse con el uso de las API de sockets en C/C++, fundamental en el desarrollo de aplicaciones para redes de computadoras

1.2 Descripción general del problema

Se desea implementar un **SISTEMA P2P** para **COMPARTIR ARCHIVOS**.

Un mismo archivo puede estar disponible en varios clientes. Para identificar los archivos se usará hash MD5.

La solución utilizará un protocolo de transporte confiable, orientado a conexión (TCP).

tracker	**registra** la ubicación de los archivos compartidos	
client	**transferencia** de **archivos**	
	funciona **a la vez** como **uploader** y **downloader**	
	single-thread, atiende múltiples conexiones usando la llamada poll del sistema (atiende varias descargas a la vez)	
	uploader	coloca archivo en lista local de archivos compartidos
		registra su dirección y sus archivos compartidos en el tracker
		escucha por pedidos en una dirección dada (IP-puerto)
	downloader	consulta al tracker por el archivo y obtiene la o las direcciones de los uploaders de ese archivo
		realiza el pedido del archivo a un uploader
	uploader	inicia la transferencia del archivo al downloader

2 Decisiones de diseño

2.1 Sockets bloqueantes vs. NO bloqueantes

Los sockets son, por defecto, “bloqueantes”.

En general, al realizar operaciones de entrada/salida, el proceso queda en espera hasta que se satisfaga alguna condición que permita completar la operación.

Para poder realizar determinadas operaciones de forma no bloqueante, de manera que sea posible proseguir con otras tareas en vez de esperar que la condición se satisfaga, se puede utilizar sockets “NO bloqueantes”. Cuando el socket es NO bloqueante, al realizar una operación de lectura o escritura sobre el mismo, si ésta no se realizar en ese momento, la llamada devuelve un error determinado.

Al trabajar sobre múltiples sockets, para saber cuándo es posible llevar a cabo una tarea sobre cada uno de ellos, se puede utilizar la llamada poll del sistema, comprobando el estado de todos ellos al mismo tiempo; una vez comprobado este estado, se itera sobre todos los sockets y se realiza o no operaciones de acuerdo a este estado.

Conjugando sockets no bloqueantes con una llamada de tipo poll, se puede en un solo hilo, simular concurrencia sobre todos los sockets ya que se hace un uso más eficiente de los mismos.

La utilización de sockets bloqueantes es más sencilla de implementar pero puede ocasionar bloqueos en el proceso.

La utilización de sockets no bloqueantes implica más controles en la implementación, dado que en muchos casos, es necesario controlar los cambios de estado en el socket entre una invocación y la siguiente al poll.

Para nuestra implementación, decidimos hacer lo siguiente:

- utilizar **sockets NO bloqueantes** para la **transferencia de archivos entre pares**, dado que son las operaciones que insumen más tiempo y en las cuales el bloqueo sería más notorio
- utilizar **sockets bloqueantes** para el **resto de las conexiones** que no transmiten grandes cantidades de información.

Somos conscientes de que sería más conveniente que todos los sockets fueran no bloqueantes, tomamos estas decisiones para no hacer tan compleja la implementación en algunos casos.

Otro atributo que influye en el momento de enviar y recibir los diferentes archivos, es el tamaño del buffer con el cual se leen los archivos por parte de los uploaders (y que posteriormente se deposita en el socket de envío); un tamaño muy grande del buffer puede provocar que la aplicación envíe archivos de forma casi secuencial, en cambio un tamaño más pequeño hace que se itere más rápidamente entre los diferentes sockets.

2.2 client

- **Carpetas de archivos por defecto**
  - **share**: carpeta en la cual se encuentra los archivos a compartir (a publicar en el tracker)
  - **downloads**: carpeta en la cual quedan depositados los archivos bajados de un par

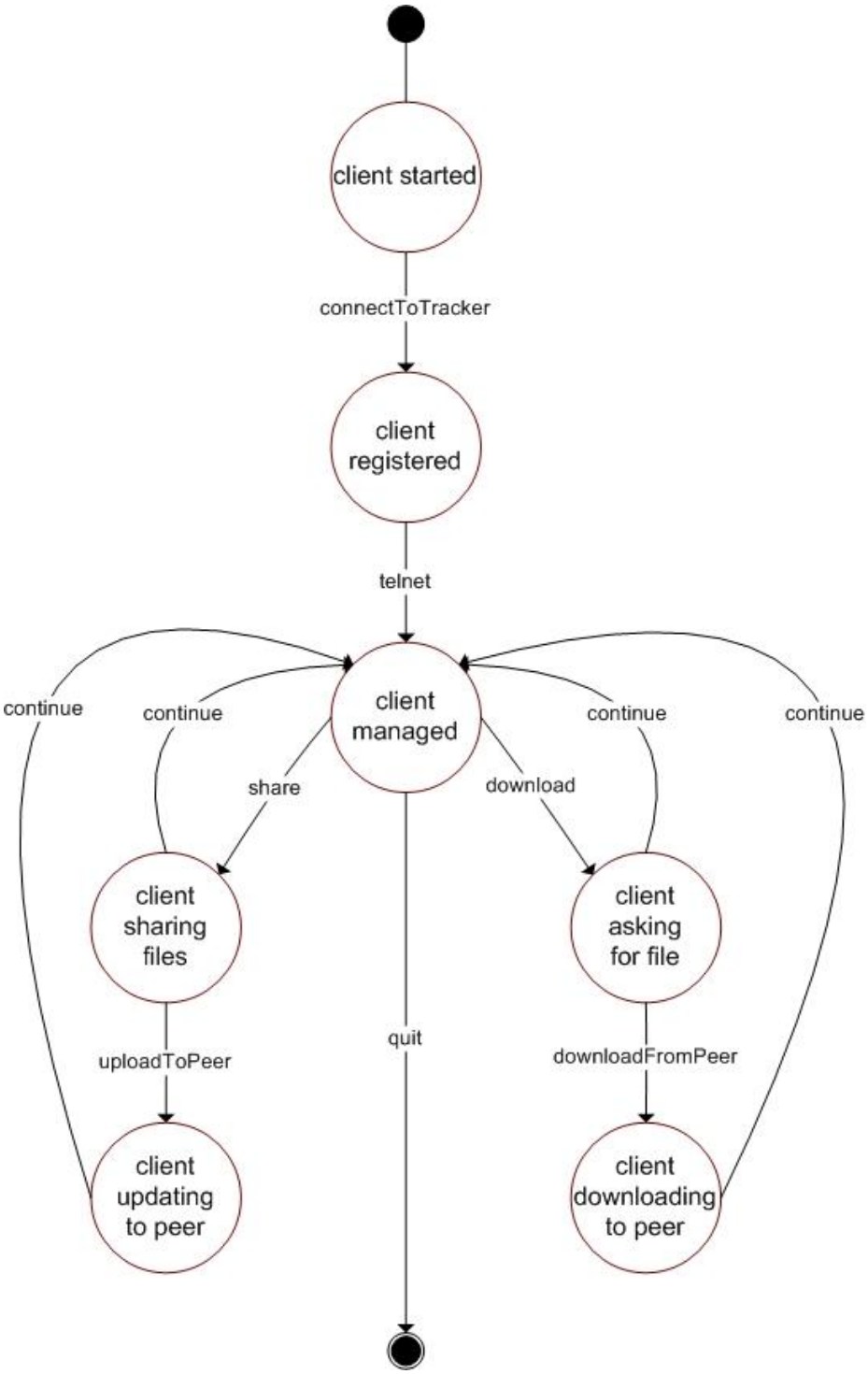
2.3 tracker

- **Registro de shared files**
  - **no se permite** registrar archivos que tengan el **mismo filename** y **diferente md5**

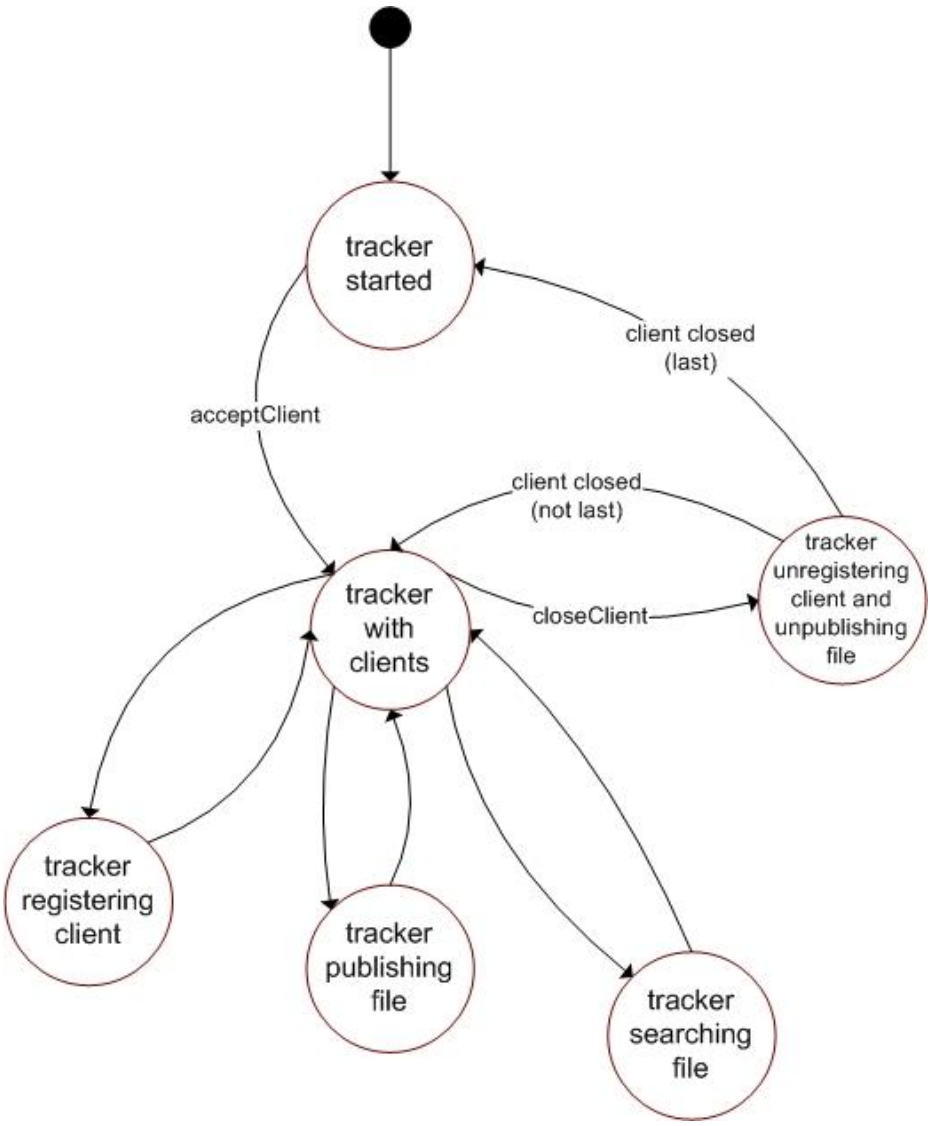
- **Búsqueda de shared files**
  - dado el **filename**, se obtiene el md5 que le corresponde en el tracker y se devuelve la lista de todos los clientes que comparten un archivo con ese **md5**
    (podría darse el caso de que dos archivos con diferente filename tuvieran el mismo md5, por lo cual, el cliente podría hacer finalmente el download de un archivo con un filename diferente al que solicitó al tracker, aunque finalmente lo guardará con el nombre que lo solicitó inicialmente)

3 Máquinas de estado

3.1 client – máquina de estados



3.2 tracker – máquina de estados



4 Pseudocódigos

4.1 client - pseudocódigo

client	
parámetro	**valor por defecto**
IP	127.0.0.1
PORT	5556
CONSOLE_PORT	6666
TRACKER_PORT	5555
TRACKER_IP	127.0.0.1

```
Inicio cliente
        Me conecto al tracker
                Agrego tracker_socket al array de sockets
        Escucho por pares
                Agrego pares_listen_socket al array de sockets
        Escucho por la consola
                Agrego consola_listen_socket al array de sockets
Loop
        Invoco Poll (array de sockets)
        Evento en pares_listen_socket
                Si tengo un pedido de conexión de un par
                        Agrego par_socket al array de sockets
        Evento en consola_listen_socket
                Si tengo un pedido de conexión de consola
                        Agrego consola_socket al array de sockets
        Evento en consola_socket
                Case    share filename
                        download filename
                        show share
                        show downloads
                        show uploads
                        quit
        Evento en par_socket
                Case    download
                        upload
Fin Loop
```

4.2 tracker - pseudocódigo

tracker	
Parámetro	**Valor por defecto**
ipTracker	127.0.0.1
puertoTracker	5556

Inicio tracker
    Escucho por clientes
        Agrego client_listen_socket al array de sockets
Loop
    Invoco Poll (array de sockets)
    Evento en client_listen_socket
        Si tengo un pedido de conexión de un cliente
            Agrego client_socket al array de sockets
    Evento en client_socket
        Case register
             publish
             search
             close
End Loop

5 Implementación

5.1 Detalle de archivos

	Descripción
`client.cc`	lógica principal del cliente implementa client - pseudocódigo
`clientItem.cc` `clientItem.hh`	modela cliente (archivos) estructura de cliente en sí mismo estructura de clientes representada en el tracker estructuras para almacenamiento de estadísticas
`connect_library.cc`	encapsula operaciones propias de la conexión errores pedidos de conexión escucha de conexiones seteo de socket bloqueante reordenamiento de array de sockets parseo de respuesta elección de par para hacer download
`fileHelper.cc`	encapsula funciones relacionadas con files (MD5)
`tracker.cc`	lógica principal del tracker implementa tracker - pseudocódigo
`util.cc` `util.hh`	utilidad de parseo

5.2 makefile