BitTorrent Client
Taller de programacion I
Primer Cuatrimestre 2022

July 4, 2022

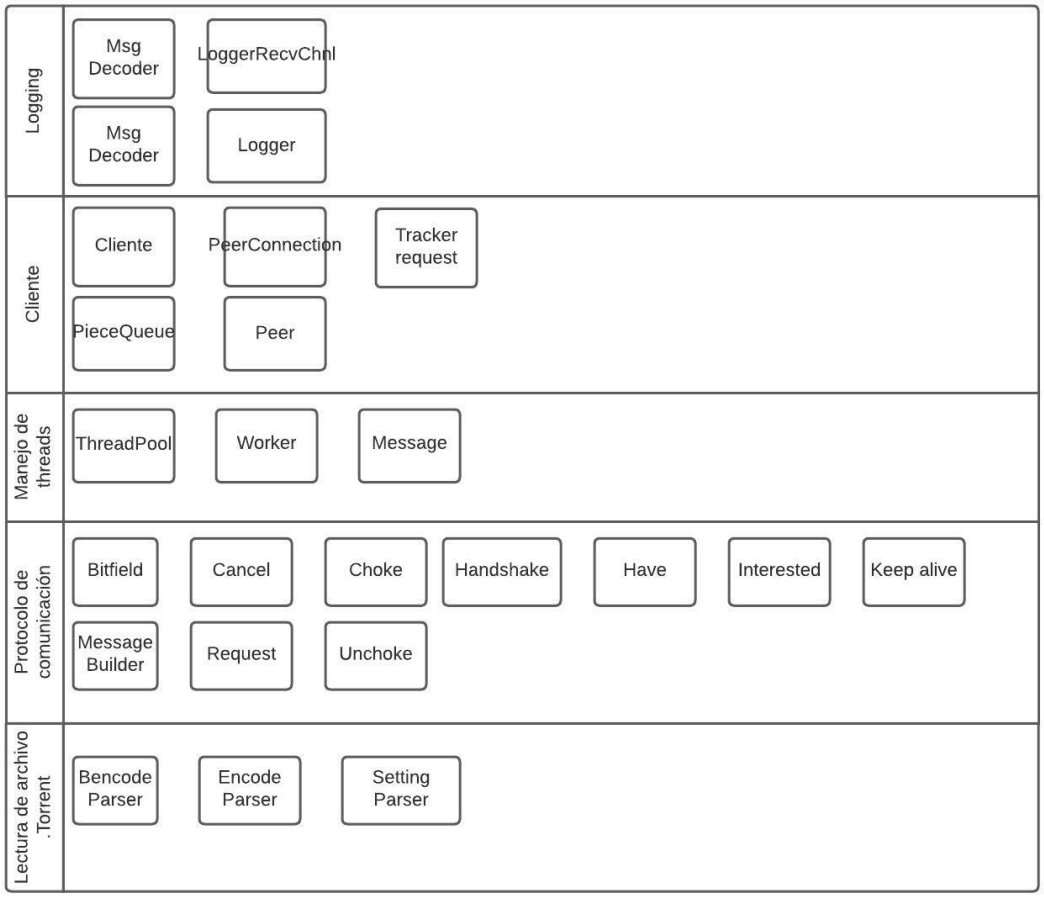
Integrante	Padrón	Correo electrónico
Marczewski Santiago	106404	smarczewski@fi.uba.ar
Montecalvo Ignacio	105555	imontecalvo@fi.uba.ar
Arillo Luis Anibal	56078	larillo@fi.uba.ar
Sabao Tomás	99437	tsabao@fi.uba.ar

Contents

1 Introduction

El objetivo del presente informe es presentar en detalle el desarrollo del proyecto de BitTorrent Client

2 Módulos del programa



Los módulos son:

- Módulo logging: Encargado de la creación de archivos de logs, de mensajes a ser logeados y del control de recepción de los mensajes.
- Módulo cliente: Encargado de las operaciónes vinculadas a la descarga de

piezas por parte de otros peers

- Módulo manejo de threads: Encargado de la funcionalidad multi-thread del proyecto
- Módulo protocolo de comunicación: Encargado de la correcta generación de lo mensajes con los que se comunica con otros peers
- Módulo lectura de archivo .Torrent Módulo encargado de obtener la información necesaria sobre el archivo para poder llevar a cabo la descarga.

3 Ejecución del programa

3.1 Archivos requeridos

Para la ejecución del programa se debe contar con:

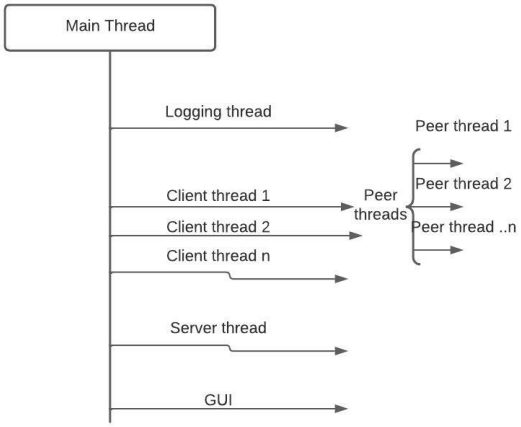
- Un archivo .torrent o un directorio con archivos .torrent
- Un archivo de configuracion especificando:
  - Puerto utilizado para la conexion
  - Path valido donde se desean guardar los logs
  - Path valido donde se desean guardar los archivos descargados

3.2 Comando de ejecución

Comando de ejecución

```
 cargo run [ruta al archivo .torrent ] [ruta al archivo de
configuracion]
```

3.3 Threads involucrados en la ejecucion



3.4 Pasos de la ejecución

El programa obedece el siguiente orden de ejecución:

- Parseo de los archivos de configuracion y de .torrent
- Parseo de los archivos presentes en la carpeta de descargas, determinando que piezas ya se poseen.
- Generación de los archivos de log para la funcionalidad cliente/servidor
- Generación de los hilos de ejecución para el lado cliente y para el lado del servidor
- **Lado Cliente:** Descarga el archivo especificado en el archivo .torrent. Una vez finalizada la descarga el hilo finaliza su ejecución
- **Lado Servidor:** Conforme piezas son descargadas por el cliente, son puestas a disposición por el servidor para ser enviadas a peers que la requieran. Continuara sirviendo piezas hasta que el usuario decida terminar el proceso

4 Descarga de archivos .torrent

4.1 Escaneo de archivos previo a descarga

Al momento de la inicialización del programa, se parsea el contenido del directorio en el cual se van a almacenar los archivos descargados. En base a dichos archivos se actualiza un bitfield que se genera para llevar cuenta de cuales piezas se van a descargar. De esa manera se puede reanudar la descarga en el caso de

que esta sea interrumpida en el futuro, ya que el cliente va a buscar solo aquellas piezas faltantes en el bitfield

4.2 Descarga de múltiples torrents

Por cada torrent que se descarga, se genera un cliente encargado de descargarlo en un thread. El número de archivos que se pueden bajar en simultaneo no se encuentra limitado, pero si el numero de piezas en descarga que cada uno puede realizar.

5 Descarga de Piezas

5.1 Descarga de piezas por multiples hilos

Para la descarga de las distintas piezas se decidió utilizar un esquema de cola con hilos "worker". La cola se encuentra compuesta por las piezas que se quieren descargar. Cada hilo se encarga de tomar una pieza de la cola y realizar la descarga de la misma

5.2 Vinculación hilo - peer

Se decidió que cada hilo que maneja la descarga se encuentr vinculado a un peer en particular proporcionado por el tracker. De esta manera el establecimiento de la conexion con cada peer se realiza una única vez, y en el caso de que el peer no posea la pieza que se obtuvo de la cola, est puede ser devuelta y el hilo procede con una nueva pieza extraida de la cola

5.3 Comunicacion entre cliente y los threads de descarga

Al tomar los threads piezas para descargar, puede darse la situación de que un thread intenta tomar una pieza de la cola y esta se encuentra vacia. Que la cola se encuentre vacia no es indicador de que todas las piezas hayan sido descargadas, dado que puede ocurrir que algun thread que tomo una de las piezas para descargar no pueda hacerlo por el peer asignado no tenerla. Debido a eso es necesario un mecanismo de control para saber que piezas fueron correctamente descargadas. Para ello cada cliente que descarga un torrent tiene un bitfield representativo de las piezas que lo conforman, y genera un canal que se encarga de recibir mensajes de los threads de descargas, avisandole de cuando una pieza fue correctamente descargada y validada. De esa manera se evita que se descarte un posible peer para descargar, finalizando la conexion con el peer del thread solo una vez que la descarga fue finalizada además de estar la cola de piezas vacia.

5.4 Preguntas de interes generadas durante el desarrollo

- **¿Qué pasa si la respuesta del peer no es entendible?**
  En el caso de que la respuesta del peer no sea conforme a alguno de los mensajes que pueden llegar a ser enviados, se considera al peer como invalido y se termina la ejecución del thread de descarga asociadado al mismo

- **¿Si cada tread se encuentra vinculado a un peer, que pasa si se trata de bajar un archivo con una gran cantidad de peers?**
  Mediante la utilización de una threadpool, se limita el número de peers a los cuales el cliente se encuentra conectado en todo momento.

6 Almacenamiento de las piezas

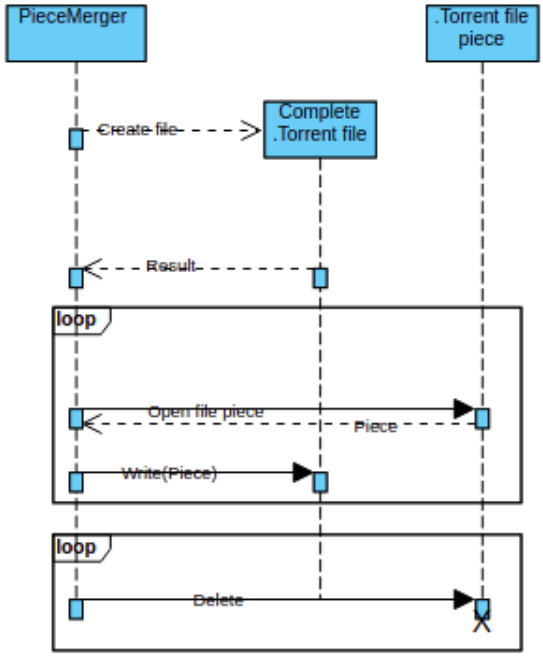
Se decidió mantener cada pieza en un buffer, al cual se van agregando cada uno de los bloques que envia el peer. Una vez que se completa la pieza, se valida la misma contra el valor de sha1 que debería producir. En el caso de que sea valida, esta se escribe a un archivo y se libera el buffer. Puede garantizarse entonces que el programa consumira una cantidad de memoria máxima igual a:

$$Memoria\ max = (\#Torrents) \times (\#of\ peers\ ) \times (Size\ biggest\ piece) \quad (1)$$

7 Joineo de piezas

La combinación de las piezas para la generación del archivo se realiza una vez todas las piezas fueron descargadas. Se genera un archivo vacio, al cual se le va appendeando el contenido de cada pieza en orden secuencial de piezas. Al tener solo el contenido de una sola pieza en memoria, no es intensiva computacionalmente Una vez que las piezas son unidas, se eliminan las piezas descargadas

7.1 Diagrama: joineo de piezas



8 Logging de mensajes

8.1 Generacion de archivos

Se generan dos archivos de log por ejecución del archivo, uno para logs del lado del servidor y uno para el lado cliente

8.2 Formato de logs

Los logs son generados con el siguiente formato:

Formato de logs

```
[Tipo de log] [Momento que se realizo el log] Mensaje
```

8.3 Manejo de multiples hilos

Dado que múltiples hilos pueden bajar archivos en simultaneo, se requiere de un mecanismo para que los logs se escriban en un mismo archivo. Se decidió utilizar channels para transmitir los mensajes que deben ser logeados al logger.

El uso de channels evita el cuello de botella que se puede generar al compartir elementos entre varios hilos, ya que cada hilo manda el mensaje para loggear y no detiene su ejecución, siendo el encargado del otro extremo del canal el encargado de manejar dichos mensajes enviados

9 GUI

9.1 Version requerida

La version requerida es GTK3.

9.2 Transmisión de la información a la vista

Se utilizan canales (channels). Se pueden enviar mensajes desde multiples lugares a un único receptor en el thread principal. Cuando el receiver recibe el mensaje, el main loop ejecuta la closure para manejar el evento.

9.3 Estimación de la velocidad de transmisión de datos

La GUI recibe mensajes enviados por el modelo, con los datos de la actualización de la descarga. Dado que los mensajes son enviados por múltiples threads, no se puede garantizar el orden en el que son recibidos por la vista, por lo que usar el tiempo entre la llegada de los mensajes no sería una buena estimacion del tiempo que toma en bajar un archivo. Una opción es que el cliente use un contador que se inicia previo al pedido de un bloque, y la finalización de dicho contador temporal una vez que el bloque es bajado. La velocidad de descarga puede ser enviada por cada mensaje. En tal caso se podria usar el promedio de las velocidades recibidas por los distintos mensajes, como una estimacion, usando las ultimas n recibidas, y siendo conservadas estas por el elemento de la gui que guarda esto

10 Conclusion

La realización del proyecto nos enseñó la importancia de mantener un orden de versiones al momento de desarrollar . Se sufrieron numerosos retrasos producto de trabajar con versiones desactualizadas en diversas ramas que poseian cada una funcionalidades distintas.