Laboratory Session #03

Distributed Systems Programming

Daniele Bringhenti, Riccardo Sisto

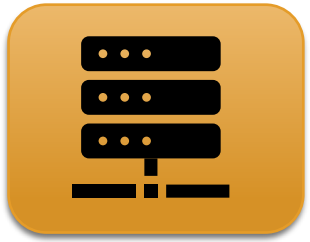
Socket API

- The **Socket API** represents the de-facto standard API for **accessing network** services provided by Internet layers 4-3-2.
- The main features of the Socket API are:
  1) **procedural** API
  2) **general** procedures rather than specific for a single protocol stack
     - TCP/IP is a particular case
  3) **designed** with **TCP/IP** in mind
     - asymmetric connection model
     - stream-oriented connections without message delimiters

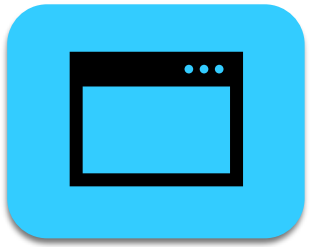
Sockets used for TCP/IP are called **stream** sockets.

Topics of the Laboratory Session

Laboratory Session #03 covers the following activities:



Implementation of a TCP/IP socket **server** application (in Java), which can perform **image conversion**



Implementation of a TCP/IP socket **client** application (in Java), which requests an image conversion

Converter Service

Main characteristics of the *Converter* service:

- **concurrent** TCP server, listening to the TCP port number **2001**;
- the server can establish TCP connections with multiple clients, and each request is managed by a different thread;
- it performs the media type **conversion** of image files;
- the media types supported are **three**: PNG, JPEG, and GIF.

Implementation tips:

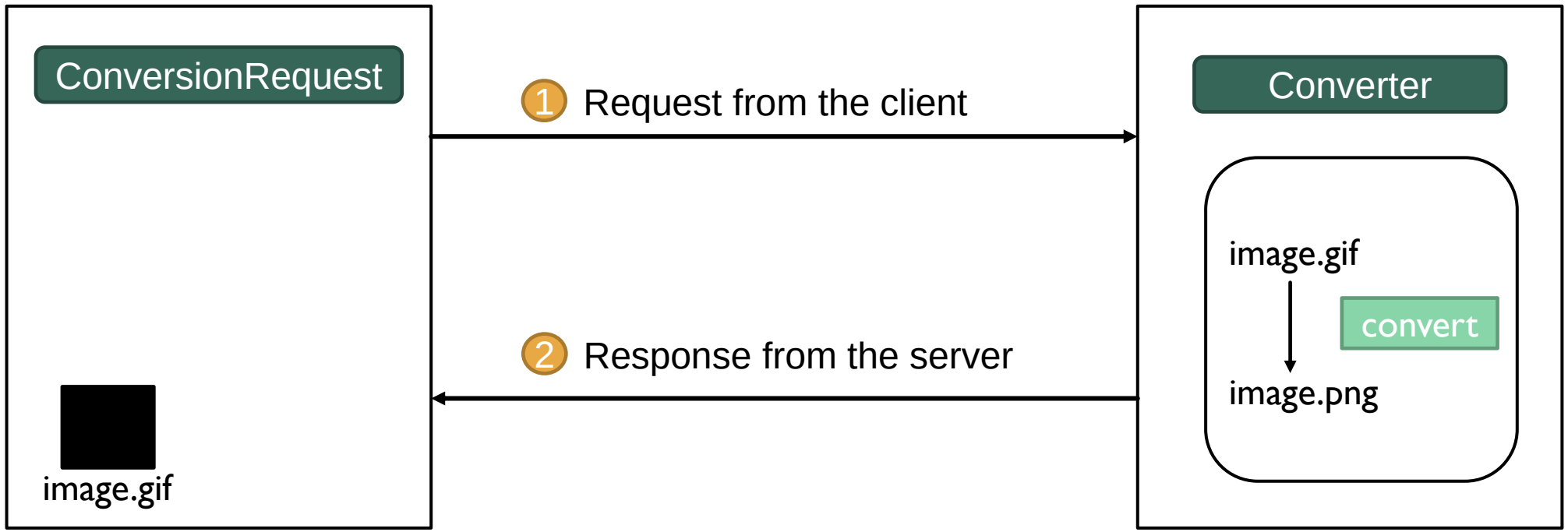
- reuse the code for image conversion from Lab02;
- the *ExecutorService* framework provided by the JDK may simplify the execution of tasks in asynchronous mode.

ConversionRequest client

Main characteristics of the *ConversionRequest* client :

- it **interacts** with the *Converter* service to perform the conversion of an image file;
- it **receives** three parameters from the **command line**:
  1) the **original** media type of the image file;
  2) the **target** media type to which the image file must be converted;
  3) the **path** of the image file in the local file system of the *ConversionRequest* client.
     - the path is relative to the ./image folder (where . is the current client directory)

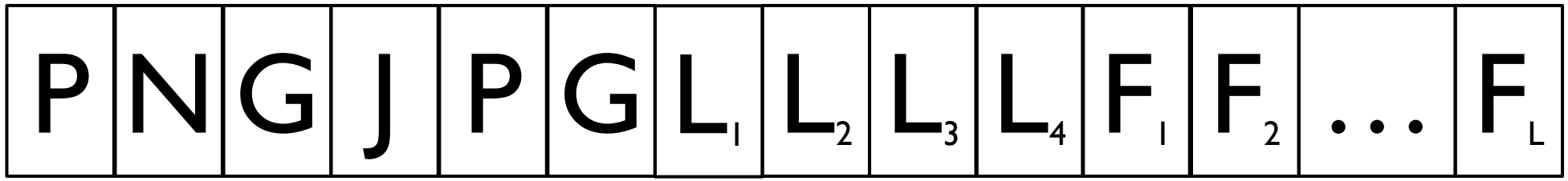
The Protocol: Messages exchanged on the TCP Connection



The Protocol: Request from the client

The client sends to the server:

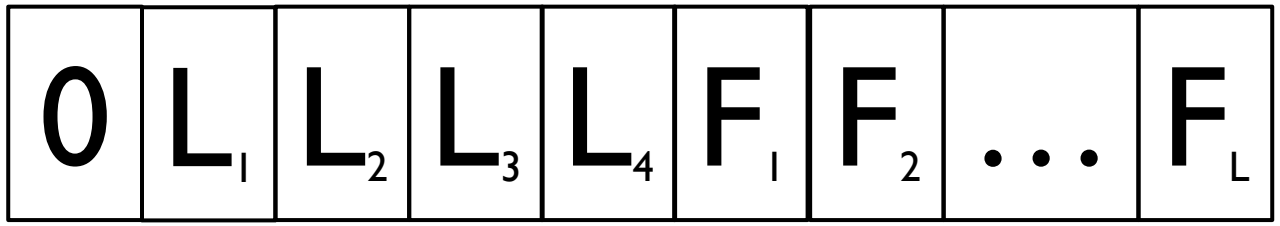
- three ASCII characters representing the **original media type** of the image file that must be converted (“PNG”, “JPG” or “GIF”);
- three ASCII characters representing the **target media type** for the conversion;
- a 4-byte 2’s complement integer number in network byte order, representing the **length** in bytes of the image file;
- the **bytes** of the image file.



The Protocol: Response from the server (I)

If there are no issues in receiving the client's message or in converting the image file, the server sends to the client:

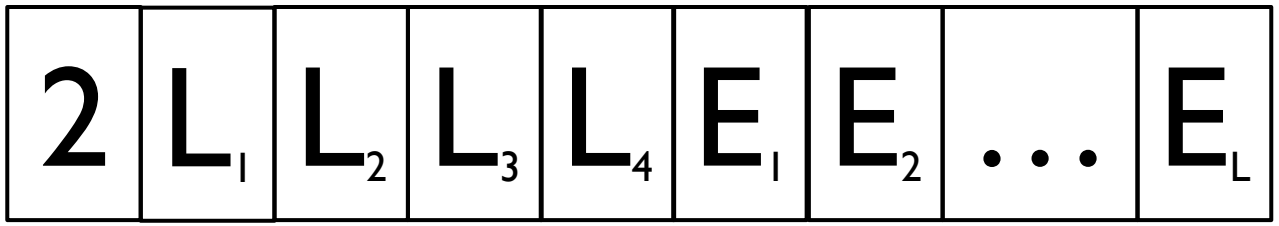
- the ASCII character '0' representing the **successful outcome** of the operations;
- a 4-byte 2's complement integer number in network byte order, representing the **length** in bytes of the converted image file;
- the **bytes** of the converted image file.



The Protocol: Response from the server (II)

If there are **issues** in receiving the client's message or in converting the image file, the server sends to the client:

- an ASCII character representing a number greater than 0 in case of **unsuccessful outcome** ('**1**' for a wrong request, '**2**' for internal error of the server);
- a 4-byte integer number in network byte order, representing the **length** in bytes of a string describing the error that occurred;
- the **bytes** of an ASCII string describing the error that occurred.



Final tips

- Both the client and the server should use **timeouts** when waiting for input from the peer, in order to avoid deadlocks.

- Tips for testing the implementation:
  - try the conversion of files of **different types**;
  - try a conversion with a file of **large size** (e.g., 10MB) and check the **efficiency**;
  - run more clients **concurrently** and check how execution time varies;
  - test **interoperability** against the reference client and server;
  - try your client and server under **erroneous conditions** and fix any **robustness** problems.

Thanks for your attention!

D. Bringhenti, R.Sisto

daniele.bringhenti@polito.it

riccardo.sisto@polito.it

netgroup