

Programmation Réseau

Damien Gros

CEA

8 octobre 2018

Plan du cours

Programmation Réseau

- Introduction

- En mode connecté

- Serveur/Client basiques JAVA

- Thread JAVA

- Multi-Connexion

Introduction

- ▶ Tous les langages permettent de faire de la programmation ! ASM, C, Python, Java, etc...
- ▶ Plus le langage est bas niveau, plus il y a de **choses** à gérer ! (voir la suite du cours)

Introduction

- ▶ On ne se place jamais dans la couche 3 (Réseau) (qui est une couche hardware).
- ▶ On utilise :
 - ▶ Soit une application (couche 7) qui fait le nécessaire pour nous ;
 - ▶ Soit on se place dans la couche 4 : TCP et UDP ;
- ▶ Objectif du premier TD : faire un client/serveur "basique" complètement décorrélé de tout protocole tel que FTP, HTTP, etc.
- ▶ Par la suite, utiliser ces premiers éléments pour ajouter des services.
- ▶ On appelle serveur : un programme qui attend une connexion (ou plusieurs).
- ▶ On appelle client : un programme qui se connecte à un serveur.

Échange en mode connecté

On est donc sur du TCP !

- ▶ Connexion TCP entre les deux machines ;
- ▶ Principe du téléphone : 2 personnes (l'appelant et l'appelé)
- ▶ L'appelé (le serveur) :
 - ▶ Une première prise ;
 - ▶ Un numéro de téléphone (avec un téléphone) ;
 - ▶ Une seconde prise avec un second téléphone.
- ▶ Appelant (le client) :
 - ▶ Une prise ;
 - ▶ Un numéro de téléphone et un combiné de téléphone ;
 - ▶ Nécessité du numéro de téléphone **complet** de la personne que l'on veut appeler

Procédure de la communication

L'appelé (le serveur) :

- ▶ Installer le premier combiné sur la première prise (raccorder au numéro de téléphone) ;
- ▶ Mettre les appels en attente ;
- ▶ Lors d'un appel, transférer vers le second combiné pour décrocher automatiquement ;
- ▶ On peut écouter et parler en même temps ;
- ▶ Raccrocher.

L'appelant (le client) :

- ▶ Installer la prise et le combiné ;
- ▶ Composer le numéro de téléphone de la personne à appeler !
- ▶ On peut écouter et parler en même temps ;
- ▶ Raccrocher.

Les sockets

- ▶ "Interface de connexion"
- ▶ Permet aux devs d'exploiter/interagir/etc. les services des systèmes d'exploitation
- ▶ Présentes sur tous les OS !
- ▶ Couche 4 du modèle OSI ;
- ▶ 2 "grands" types de socket :
 - ▶ Réseau : permet de communiquer entre les machines ;
 - ▶ UNIX : IPC (Inter Process Communication)

Serveur JAVA : basique

```
// Classe principale
class HelloServer1 {
    public static void main(String argv[]) throws Exception
    {
        //On installe le combine sur le numero de telephone
        ServerSocket serversocket = new ServerSocket(1111);

        //On attend les appels entrants
        Socket socket = serversocket.accept();

        //On ouvre un tube pour envoyer un message
        PrintStream out = new PrintStream( socket.getOutputStream() );

        //On ecrit et on envoie le message
        out.println( "Hello World!" );

        //On raccroche
        socket.close();
    }
}
```


Client JAVA

```
class HelloClient1 {  
    public static void main(String argv[]) throws Exception  
    {  
        // On compose le numero de telephone de la personne a contacter  
        Socket socket = new Socket("localhost", 1111);  
        //Socket socket = new Socket("127.0.0.1", 1111);  
  
        // On ouvre un tube pour lire ce que nous envoie le serveur  
        InputStreamReader inputStream = new InputStreamReader( socket.  
            getInputStream() );  
        BufferedReader input = new BufferedReader(inputStream );  
  
        // On affiche a la console les elements envoyes par le serveur  
        System.out.println( input.readLine() );  
  
        //On raccroche  
        socket.close();  
    }  
}
```

Serveur JAVA plusieurs messages

```
// Classe principale
class HelloServer2 {
    public static void main(String argv[]) throws Exception
    {
        int i=0;
        //On installe le combine sur le numero de telephone
        ServerSocket serversocket = new ServerSocket(1111);

        //On attend les appels entrants
        Socket socket = serversocket.accept();
        System.out.println(socket.getLocalPort());
        //On ouvre un tube pour envoyer des donnees
        PrintStream out = new PrintStream( socket.getOutputStream() );

        // Une boucle envoyant plusieurs messages
        while(i != 5)
        {
            out.println( "Hello World!" );
            i++;
        }

        //On raccroche
        socket.close();
    }
}
```

Serveur C

```
#include<stdio.h>
#include<string.h> //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h> //write

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
}
```

Serveur C

```
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;

server.sin_port = htons( 1111 );

//Bind
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) <
    0)
{
    //print the error message
    perror("bind failed. Error");
    return 1;
}

//Listen
listen(socket_desc , 3);

//Accept and incoming connection
c = sizeof(struct sockaddr_in);
```

Serveur C

```
//accept connection from an incoming client
client_sock = accept(socket_desc, (struct sockaddr *)&client, (
    socklen_t*)&c);
if (client_sock < 0)
{
    perror("accept failed");
    return 1;
}
//Send the message back to client
write(client_sock , client_message , strlen(client_message));
return 0;
}
```

Echange en mode non connecté

On est en UDP !

- ▶ Principe de la poste ;
- ▶ Appelant :
 - ▶ Un numéro de boîte aux lettres ;
 - ▶ Une boîte aux lettres bidirectionnelle ;
 - ▶ Une adresse personnelle ;
 - ▶ Connaître l'adresse distante.
- ▶ Appelé :
 - ▶ Un numéro de boîte aux lettres ;
 - ▶ Une boîte aux lettres bidirectionnelle ;
 - ▶ Une adresse personnelle.

Echange en mode non connecté

Appelant :

- ▶ Installer une boîte aux lettres
- ▶ Associer sa boîte aux lettres avec son adresse ;
- ▶ Déposer ou retirer des messages.

Appelé :

- ▶ Installer une boîte aux lettres ;
- ▶ Associer sa boîte aux lettres avec son adresse ;
- ▶ Déposer ou retirer des messages.

Echange en mode non connecté : serveur

```
class ServeurEcho
{
    final static int port = 8532;
    final static int taille = 1024;
    final static byte buffer[] = new byte[taille];

    public static void main(String argv[]) throws Exception
    {
        DatagramSocket socket = new DatagramSocket(port);
        DatagramPacket data = new DatagramPacket(buffer,buffer.length
        );
        socket.receive(data);
        System.out.println(data.getAddress());
        socket.send(data);
    }
}
```


Echange en mode non connecté : client

```
public class ClientUDP
{
    final static int taille = 1024;
    final static byte buffer[] = new byte[taille];

    public static void main(String argv[]) throws Exception
    {
        InetAddress serveurur = InetAddress.getByName(argv[0]);
        int length = argv[1].length();
        byte buffer[] = argv[1].getBytes();
        DatagramPacket dataSent = new DatagramPacket(buffer, length,
            serveurur, 8532);
        DatagramSocket socket = new DatagramSocket();

        socket.send(dataSent);

        DatagramPacket dataRecieved = new DatagramPacket(new byte[
            length], length);
        socket.receive(dataRecieved);
        System.out.println("Data recieved : " + new String(
            dataRecieved.getData()));
        System.out.println("From : " + dataRecieved.getAddress() + ":"
            + dataRecieved.getPort());
    }
}
```

Rappel sur les Threads

Thread vs Processus ?

- ▶ Processus :
 - ▶ A son propre espace mémoire ;
 - ▶ Peut utiliser toute la mémoire qui lui est alloué par le système ;
 - ▶ Possède un **père** : on parle de `fork()` → nouvel environnement, nouvelles variables, etc.
- ▶ Thread :
 - ▶ "Processus léger" ;
 - ▶ Partage l'espace mémoire du processus qui l'héberge ;
 - ▶ Communication simplifiée entre les différents threads d'un même processus.

Thread en Java

```
import java.io.*;
public class ThreadExemple extends Thread {

    public static void main(String argv[]) {
        new ThreadExemple();
    }

    public ThreadExemple() {
        start();
        while(true)
            System.out.println("toto");
    }

    public void run(){
        while(true)
            System.out.println("tata");
    }
}
```

Serveur mutli-threadé

```
class HelloServer3 {  
    public static void main(String argv[]) throws Exception  
    {  
  
        ServerSocket serversocket = new ServerSocket(1111);  
        Socket socket = serversocket.accept();  
        ServiceThread service = new ServiceThread(socket);  
    }  
}
```

Serveur mutli-threadé

```
class ServiceThread extends Thread {
    public Socket socket_thread;

    public ServiceThread (Socket socket){
        this.socket_thread = socket;
        start();
    }

    public void run()
    {
        try{
            PrintStream out = new PrintStream( socket_thread.
                getOutputStream() );
            out.println( "Hello World!" );
            socket_thread.close();
        }
        catch (Exception e)
        {}
    }
}
```

Serveur mutli-threadé

```
class HelloServer4 {  
    public static void main(String argv[]) throws Exception  
    {  
        ServerSocket serversocket = new ServerSocket(1111);  
        int i=0;  
        while(i != 5)  
        {  
            Socket socket = serversocket.accept();  
            ServiceThread service = new ServiceThread(socket);  
            i++;  
        }  
    }  
}
```