TD 4 - Sockets et Client / Serveur

P.Esling, T. Lieu, P.Trebuchet

11 octobre 2013

Exercice 1 – Serveur d'echo

Question 1

Écrire en Java un serveur TCP d'`echo` (fichier `echoserver.java`) qui retourne aux clients ce que ces derniers lui émettent. Dans cette première version, le serveur écoute sur un port, crée un socket client. Puis á chaque ligne tapée par le client, le serveur renvoie exactement la même ligne. On supposera que le numéro du port sur lequel le serveur écoute est donné en paramètre sur le ligne de commande, *e.g.*, pour écouter sur le port 12345, on utilisera la commande `echoserver 12345`.

Solution:

```
import java.io.*;
import java.net.*;

public class                    echoserver
{
 public static void             main(String args[])
 {
  BufferedReader                inchan;
  DataOutputStream              outchan;
  ServerSocket                  serv;
  Socket                                client;

  try
  {
    int port = Integer.parseInt(args[0]);
    serv = new ServerSocket(port);
    while(true)
    {
      client = serv.accept();
      try
      {
        inchan = new BufferedReader(new InputStreamReader(client.getInputStream()));
        outchan = new DataOutputStream(client.getOutputStream());
        while (true)
        {
          String command = inchan.readLine();
          if(command.equals("")) { System.out.println("Fin de connexion."); break;}
          outchan.writeChars(command);
        }
      }
      catch(IOException e) { System.err.println("I/O Error"); e.printStackTrace()}
```

```
      client.close();
    }
   }
   catch(Throwable t) {t.printStackTrace(System.err); }
  }
}
```

Question 2

Quel probléme survient lors de l'utilisation de cette implémentation ? Comment régler un tel problème ?

Solution:

Pas de sous-processus, ni de thread séparé pour chaque client, le serveur ne peut donc avoir qu'un seul client á la fois et ne va donc pas répondre aux connexions simultanées.

Question 3

Écrire un autre serveur `echoserverThread.java` qui gère les connexions simultanées et délègue l'écho á un second processus. Ainsi plusieurs clients pourront profiter simultanément des services de ce serveur. Le probléme de l'utilisation de processus Unix pour sous-traiter des tâches est que ceux-ci consomment beaucoup de ressources (en particulier, il y a duplication des environnements), on va minimiser cet artefact en utilisant des *threads*. Écrire une variante du serveur en utilisant des *threads*.

Solution:

```
import java.io.*;
import java.net.*;

public class                    echoserverThread
{
 public static void             main(String args[])
 {
  ServerSocket                  serv;
  Socket                                client;

  try
  {
    int port = Integer.parseInt(args[0]);
    serv = new ServerSocket(port);
    while(true)
    {
      client = serv.accept();
          System.out.println("Nouvelle connexion.");
      echoClient clt = new echoClient(client);
      clt.start();
    }
   }
   catch(Throwable t) {t.printStackTrace(System.err); }
  }
}

class                                 echoClient extends Thread
{
  BufferedReader                inchan;
  DataOutputStream              outchan;
  Socket                                socket;
```

```
  echoClient(Socket s)
  {
        try
    {
      inchan = new BufferedReader(new InputStreamReader(s.getInputStream()));
      outchan = new DataOutputStream(s.getOutputStream());
        }
    catch(IOException e) { e.printStackTrace(); System.exit(1);}
        socket = s;
  }

  public void run()
  {
    try {
    while (true)
    {
      String command = inchan.readLine();
      if(command.equals("")) { System.out.println("Fin de connexion."); break;}
      outchan.writeChars(command + "\n");
    }
        socket.close();
    } catch(IOException e) { e.printStackTrace(); System.exit(1);}
  }
}
```

Question 4

Simuler une intéraction avec le serveur en utilisant la commande `telnet` ou la commande `nc` [1].

Question 5

Quel est le probléme restant avec la dernière implémentation du serveur ? Comment l'améliorer pour gérer cette situation ?

Solution:

- Une fois créés, on perd le contrôle sur les threads client, donc plus de possibilité d'interaction globale. - On ne peut pas contrôler le nombre maximum de threads qui tournent á un moment donné - Pas de possibilité de mettre les clients en queue pour faire du traitement de requêtes par paquets. Il faut implémenter une pool de threads.

Question 6

On se propose de gérer les problèmes précédents en utilisant une *pool* de threads. L'idée est de créer et lancer dès le démarrage du serveur un ensemble de threads que l'on conserve dans une liste (*Vector*). Plus aucun thread ne pourra être créé par la suite. Les threads clients se mettent en attente d'un signal du serveur. Lors d'une nouvelle connexion, le serveur ajoute le nouveau socket dans une liste d'attente et notifie un des threads. Le thread doit alors se charger d'aller récupérer le socket du nouveau client et d'effectuer le même comportement que précédemment.

Solution:

```
import java.util.Vector;
import java.io.*;
import java.net.*;

public class                    echoserverPool
```

1. Attention, `nc` (netcat) existe sous de nombreuses versions et leurs utilisations diffèrent.

```
{
 public static void             main(String args[])
 {
    int port = Integer.parseInt(args[0]);
        int capacity = Integer.parseInt(args[1]);
        echoServer server = new echoServer(port, capacity);
        server.run();
  }
}

class                                   echoServer
{
  Vector<echoClient>    clients;
  Vector<Socket>        sockets;
  ServerSocket                  serv;
  Socket                                client;
  int                                   capacity;
  int                                   nbConnectedClients;
  int                                   nbWaitingSocks;
  int                                   port;

  echoServer(int p, int c)
  {
          capacity = c;
          port = p;
          clients = new Vector<echoClient>(c);
          sockets = new Vector<Socket>();
          for (int i = 0; i < c; i++)
          {
                  echoClient tmpEcho = new echoClient(this);
                  clients.add(tmpEcho);
                  tmpEcho.start();
          }
          nbConnectedClients = 0;
          nbWaitingSocks = 0;
  }

  public Socket                 removeFirstSocket()
  {
          Socket ret = sockets.get(0);
          sockets.removeElementAt(0);
          return ret;
  }

  public void                   newConnect()
  {
    nbConnectedClients++;
    nbWaitingSocks--;
    System.out.println(" Thread handled connection.");
        System.out.println("   * " + nbConnectedClients + " connected.");
    System.out.println("   * " + nbWaitingSocks + " waiting.");
  }

  public void                   clientLeft()
  {
    nbConnectedClients--;
```

```
    System.out.println(" Client left.");
    System.out.println("   * " + nbConnectedClients + " connected.");
    System.out.println("   * " + nbWaitingSocks + " waiting.");
  }

  public int                    stillWaiting() { return nbWaitingSocks;}

  public void                   run()
  {
    try
    {
          serv = new ServerSocket(port);
      while(true)
      {
        client = serv.accept();
        System.out.println("New connexion at server.");
        synchronized (this)
        {
          sockets.add(client);
          nbWaitingSocks++;
          this.notify();
                }
        }
    }
    catch(Throwable t) {t.printStackTrace(System.err); }
  }
}

class                                   echoClient extends Thread
{
  BufferedReader                inchan;
  DataOutputStream              outchan;
  echoServer                    server;
  Socket                                socket;
  int                                   idC;

  echoClient(echoServer s) { server = s; }

  public void                   run()
  {
    Socket                              s;

        while (true)
        {
      synchronized (server)
      {
        if (server.stillWaiting() == 0)
          try { server.wait(); } catch(InterruptedException e) {e.printStackTrace();}
        s = server.removeFirstSocket();
        server.newConnect();
          }
      try
          {
        inchan = new BufferedReader(new InputStreamReader(s.getInputStream()));
        outchan = new DataOutputStream(s.getOutputStream());
        socket = s;
```

```
        while (true)
        {
              String command = inchan.readLine();
                  if(command == null || command.equals("")) { System.out.println("Fin de connexion."); b
                  outchan.writeChars(command + "\n");
                }
            socket.close();
                synchronized (server) { server.clientLeft(); }
          } catch(IOException e) { e.printStackTrace(); System.exit(1); }
        }
  }
}
```

Question 7

On souhaite modifier le serveur pour le transformer en serveur de clavardage (*chat*). Pour ce faire, on va utiliser notre systéme de pool de threads en l'augmentant par une liste de flux que le serveur va s'occuper de redistribuer. De plus, on veut maintenant que l'utilisateur ne recoive pas ses propres messages en duplicatas (comme dans un vrai salon)

Modifier le serveur pour qu'il surveille tous ses clients et envoie ce que l'un écrit á tous les autres.

Solution:

```
import java.util.Vector;
import java.io.*;
import java.net.*;

public class                    echoserverPoolChat
{
 public static void             main(String args[])
 {
    int port = Integer.parseInt(args[0]);
        int capacity = Integer.parseInt(args[1]);
        echoServer server = new echoServer(port, capacity);
        server.run();
  }
}

class                                   echoServer
{
  Vector<echoClient>            clients;
  Vector<Socket>                        sockets;
  Vector<DataOutputStream>      streams;
  ServerSocket                          serv;
  Socket                                        client;
  int                                           capacity;
  int                                           nbConnectedClients;
  int                                           nbWaitingSocks;
  int                                           port;

  echoServer(int p, int c)
  {
          capacity = c;
          port = p;
          clients = new Vector<echoClient>(c);
          sockets = new Vector<Socket>();
          streams = new Vector<DataOutputStream>();
```

```
        for (int i = 0; i < c; i++)
        {
                echoClient tmpEcho = new echoClient(this);
                clients.add(tmpEcho);
                tmpEcho.start();
        }
        nbConnectedClients = 0;
        nbWaitingSocks = 0;
}

public Socket                   removeFirstSocket()
{
        Socket ret = sockets.get(0);
        sockets.removeElementAt(0);
        return ret;
}

public void                     newConnect(DataOutputStream out)
{
  nbConnectedClients++;
  nbWaitingSocks--;
  System.out.println(" Thread handled connection.");
      System.out.println("   * " + nbConnectedClients + " connected.");
  System.out.println("   * " + nbWaitingSocks + " waiting.");
      streams.add(out);
      writeAllButMe("*** New user on chat ***", out);
}

public void                     clientLeft(DataOutputStream out)
{
  nbConnectedClients--;
  System.out.println(" Client left.");
  System.out.println("   * " + nbConnectedClients + " connected.");
      System.out.println("   * " + nbWaitingSocks + " waiting.");
      writeAllButMe("*** A user has left ***", out);
  streams.remove(out);
}

public void                     writeAllButMe(String s, DataOutputStream out)
{
        try
        {
        for (int i = 0; i < nbConnectedClients; i++)
              if (streams.elementAt(i) != out)
                streams.elementAt(i).writeChars(s);
        }
        catch (IOException e) {}
}

public int                      stillWaiting() { return nbWaitingSocks;}

public void                     run()
{
  try
  {
        serv = new ServerSocket(port);
```

```
      while(true)
      {
        client = serv.accept();
        System.out.println("New connexion at server.");
        synchronized (this)
        {
          sockets.add(client);
          nbWaitingSocks++;
          this.notify();
                }
        }
    }
    catch(Throwable t) {t.printStackTrace(System.err); }
  }
}

class                                   echoClient extends Thread
{
  BufferedReader                inchan;
  DataOutputStream              outchan;
  echoServer                    server;
  Socket                                socket;
  int                                   idC;

  echoClient(echoServer s) { server = s; }

  public void                   run()
  {
    Socket                              s;

        while (true)
        {
      synchronized (server)
      {
        if (server.stillWaiting() == 0)
          try { server.wait(); } catch(InterruptedException e) {e.printStackTrace();}
        s = server.removeFirstSocket();
          }
      try
                {
                inchan = new BufferedReader(new InputStreamReader(s.getInputStream()));
                outchan = new DataOutputStream(s.getOutputStream());
                socket = s;
                synchronized (server) { server.newConnect(outchan); }
        while (true)
        {
              String command = inchan.readLine();
                  if(command == null || command.equals("")) { System.out.println("Fin de connexion."); b
                  synchronized (server) { server.writeAllButMe(command + "\n", outchan); }
                }
                synchronized (server) { server.clientLeft(outchan); }
            socket.close();
        } catch(IOException e) { e.printStackTrace(); System.exit(1); }
      }
  }
}
```

Question 8

Réécrire le code du serveur `echoserverThread.java` en OCaml, permettant d'utiliser un mécanisme de threads lors d'une connection client.

Solution:

```
let creer_serveur port max_con =
  let sock = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0
  and addr = Unix.inet_addr_of_string "127.0.0.1" in
    Unix.setsockopt sock Unix.SO_REUSEADDR true;
    Unix.bind sock (Unix.ADDR_INET(addr, port));
    Unix.listen sock max_con;
    sock;;

let serveur_process sock service =
  while true do
    let (s, caller) = Unix.accept sock     in
      match Unix.fork() with
        | 0 -> (* code du fils *)
            if Unix.fork() <> 0 then exit 0 ;
            let inchan = Unix.in_channel_of_descr s
            and outchan = Unix.out_channel_of_descr s
            in
              service inchan outchan ;
              close_in inchan ;
              close_out outchan ;
              exit 0
        | id -> (* code du pere *)
            Unix.close s;
            ignore(Unix.waitpid [] id)
  done;;

let echo_service inchan outchan =
  while true do
    let line = input_line inchan in
      output_string outchan (line^"\n");
      flush outchan
  done;;

let main () =
  let port = int_of_string Sys.argv.(1)in
  let sock = creer_serveur port 4 in
    serveur_process sock echo_service;;

let _ = main();;
```

Question 9

Lorsque le serveur meurt inopinément (*e.g.*, on lui envoie un signal via Ctrl-C), le port sur lequel celui-ci écoutait est mal fermé. Dans le cas de l'utilisation des appels Unix (á contrario de la librairie Socket de Java), il n'y a pas réutilisation systématique de son adresse, c'est-á-dire, que si vous voulez relancer immédiatement un serveur sur le même port vous ne pouvez vous y "binder". Comment peut on éviter ce comportement ?

Solution:

On peut demander a Linux (>2.0) de réutiliser le numéro de port si aucune *socket* n'est en écoute dessus. Il faut utiliser la fonction `setsockopt` et positionner l'option `SO_REUSEADDR` du serveur. Ajouter la ligne : Unix.setsockopt sock Unix.SO_REUSEADDR true ;

Question 10

Pour finir, on peut se dire que l'on a pas réellement besoin de maintenir un état connecté pour ne traiter que quelques petits messages, il pourrait être plus futé d'utiliser un mode non connecté. Écrire un client et un serveur utilisant une *socket* en mode *datagram* pour faire le même genre de chose qu'au dessus.

Solution:

Pour le serveur

```
let creer_serveur port max_con =
  let sock = Unix.socket Unix.PF_INET Unix.SOCK_DGRAM 0
  and addr = Unix.inet_addr_of_string "127.0.0.1" in
    Unix.setsockopt sock Unix.SO_REUSEADDR true;
    Unix.bind sock (Unix.ADDR_INET(addr, port));
    sock ;;

let serveur_process sock=
  let message = String.create 10000 in
    while true do
      match Unix.recvfrom sock message 0 10000 [] with
        | (longueur, Unix.ADDR_INET (addr, port)) ->
            Printf.printf "Client %s said ``%s''\n%!"
              (Unix.gethostbyaddr addr).Unix.h_name
              (String.sub message 0 longueur);
            ignore (Unix.sendto sock message 0 longueur [] (Unix.ADDR_INET(addr,port)));
        | _ -> failwith "Cas non gr dans echoserver3.ml"
    done ;;

let main () =
  let port = int_of_string Sys.argv.(1)    in
  let sock = creer_serveur port 4 in
    serveur_process sock;;

main();;
```

Pour le client

```
let socket =
  Unix.socket Unix.PF_INET Unix.SOCK_DGRAM
    (Unix.getprotobyname "udp").Unix.p_proto;;

(* Send a UDP message. *)
let ipaddr = (Unix.gethostbyname Sys.argv.(1)).Unix.h_addr_list.(0);;
let portaddr = Unix.ADDR_INET (ipaddr, (int_of_string Sys.argv.(2)));;
let len = Unix.sendto socket Sys.argv.(3) 0 (String.length Sys.argv.(3)) [] portaddr;;

(* Receive a UDP message. *)
let msg = String.create 10000;;
let len, portaddr = Unix.recvfrom socket msg 0 10000 [];;

print_endline (String.sub msg 0 len);;
```

Question 11

Écrire un client `Java` interagissant avec le serveur d'écho en mode *datagram*. Ce client lira sur sa ligne de commande une `adresse` et un `message` et enverra le `message` á l'adresse indiquée et affichera la réponse. La ligne de commande sera donc quelque chose comme : `java echoclient addresse_serveur port_serveur message` *On fera attention au fait que les chaînes Java sont Unicode.*

Solution:

```
import java.io.*;
import java.net.*;

public class echoclient {
  public static void main(String args[]) {
    try {
        String adresse = args[0];
        int port = Integer.parseInt(args[1]);
        String message = args[2];
        Socket sock = new Socket(adresse,port);
        BufferedReader inchan = new BufferedReader(new InputStreamReader(sock.getInputStream()));
        DataOutputStream outchan = new DataOutputStream(sock.getOutputStream());
        outchan.writeChars(message+"\n");
        String answer = inchan.readLine();
        System.out.println("Echo : " + answer);
    } catch(Throwable t) { t.printStackTrace(System.err); }
  } // end main
} // end class
```

Question 12

Écrire un *bot de spam* `OCaml` qui : émet 1 fois la chaîne ”Tro” émet 3 fois la chaîne `"Lo"` puis émet `"Exit"` et quitte en fermant proprement la connexion.

Solution:

```
(* compilation: ocamlc unix.cma pongclient.ml -o pongclient *)

let inet_addr_of_hostname serveur =
  try Unix.inet_addr_of_string serveur
  with Failure("inet_addr_of_string") ->
    try (Unix.gethostbyname serveur).Unix.h_addr_list.(0)
    with Not_found -> print_string serveur ;
      print_endline "Not found";
      exit 1;;

let connecter_client client_fun adresse port =
  let sock = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0
  and addr = inet_addr_of_hostname adresse in
    Unix.connect sock (Unix.ADDR_INET(addr,port)) ;
  let inchan = Unix.in_channel_of_descr sock
      and outchan = Unix.out_channel_of_descr sock in
      client_fun sock inchan outchan ;
      close_in inchan ;
      close_out outchan ;
      exit 0;;

let pongclient sock inchan outchan =
  Printf.fprintf outchan "Ping\n%!"; (* %! sert  flusher *)
  Printf.printf "Rponse = %s\n%!" (input_line inchan);
  Printf.fprintf outchan "Ping\n%!";

  Printf.printf "Rponse = %s\n%!" (input_line inchan);
```

```
  Printf.fprintf outchan "Ping\n%!";

  Printf.printf "Rponse = %s\n%!" (input_line inchan);
  Printf.fprintf outchan "Toto\n%!";

  Printf.fprintf outchan "Exit\n%!";
  Unix.shutdown sock Unix.SHUTDOWN_ALL;;

let main () =
  let adresse = Sys.argv.(1)
  and port = int_of_string Sys.argv.(2) in
    connecter_client pongclient adresse port;;

let _ = main ();;
```

Question 13

En utilisant l'analogie qui existe avec ce que vous connaissez en OCaml, dire ce que fait le code C suivant :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include <netinet/ip.h>
#include<time.h>
#include <netdb.h>

int main(int argc, char** argv)
{
  char buff[20];
  struct sockaddr_in servaddress;
  struct hostent *resbyname;
  int sock = socket(PF_INET,SOCK_STREAM,0),i;
  int err;
  if (argc < 2) {
      fprintf(stderr,"il me faut 2 arguments!\n");
      exit(1);
  }
  printf("argv 1 %s\n",argv[1]);
  resbyname = gethostbyname(argv[1]);
  servaddress.sin_family = AF_INET;
  servaddress.sin_port = htons(atoi(argv[2]));
  servaddress.sin_addr.s_addr = atoi(resbyname->h_addr_list[0]);
  resbyname->h_addr_list[0][0] = '\0';
  err = connect(sock, (struct sockaddr*) &servaddress, sizeof(struct sockaddr_in));
  printf("err %d\n",err);fflush(stdout);
  for(i=0; i<3; i++) {
    write(sock, "Ping", 5); //pour le '\0'
    read(sock, buff, sizeof(buff));
    printf("J'ai lu %s\n", buff);
   }
  write(sock, "Toto", 5);//pour le '\0'
  write(sock, "Exit", 5);//pour le '\0'
  close(sock);
  free(resbyname);
```

```
}
```

Solution:
Il fait la même chose que le client OCaml.

Exercice 2

Dans cet exercice on se restreindra á utiliser des *sockets* fonctionnant en IPv4. La commande `nc` est une commande dite *couteau suisse de l'administrateur réseaux*. C'est un petit utilitaire qui met en place une connexion TCP/IP soit comme client (connecte une *socket*) soit comme serveur (*socket* en attente acceptant les connexions).

Question 1

Donner une implantation OCaml de la commande `nc` lorsqu'elle se comporte en client.
Solution:

```
let main() =
  let host = Unix.gethostbyname (Sys.argv.(1)) in
  let host_addr  =  host.Unix.h_addr_list.(1) in
  let sock_descr = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0 in
  let outchan =  Unix.out_channel_of_descr sock_descr in
    Unix.connect sock_descr (Unix.ADDR_INET(host_addr, 12345));
    while true do
      let ligne = read_line () in
        try
          output_string outchan (ligne ^ "\n");
          flush outchan;
        with
          | End_of_file -> Printf.printf "Fin de connexion \n";flush stdout;
          | exn->
              print_endline (Printexc.to_string exn)
    done;;

let _ = main();;
```

Question 2

Donner une implantation de la commande `nc` lorsqu'elle se comporte en serveur.

Question 3

Donner une implantation équivalente en Java.

Exercice 3 – (Bonus) Ordonnanceur concurrent en Java

Nous allons réaliser nous-même un *ordonnanceur* de tâches en essayant de simuler les fonctions de bases du scheduler de la librairie `FThreads`. Pour ce faire, nous utiliserons uniquement les Threads Java. Complétez le programme suivant en ajoutant un mécanisme de synchronisation par condition entre le thread ordonnanceur et les threads correspondants à ces tâches.

TP - Collections réparties

Exercice 4 – Collections réparties (TP)

Question 1

Écrire un serveur `DistributedAssoctable` en Java qui est une table d'associations disponible sur le réseau (serveur TCP). La table d'associations peut recevoir les commandes :

- `START` : un client demande l'accés á la table (ne fait rien côté serveur)
- `PUT key value` : ajouter une entrée dans la table
- `GET key` : récupérer une valeur dans la table
- `QUIT` la connexion est terminée.

Question 2

Écrire un client en `OCaml` ou `C` permettant d'interagir avec la table d'associations. Ce client devra lire les commandes sur son entrée standard vérifier qu'elles sont bien formées, ensuite il devra les envoyer au serveur et afficher la réponse de celui-ci.

Question 3

On souhaite enrichir le serveur pour enregistrer les statistiques d'accès á la table. Pour ce faire on modifie quelque peu les commandes du serveur : la commande `START` retourne maintenant un identifiant unique `id` de client et les autres commandes du serveur prennent un argument supplmentaire `id` pour savoir quel client accède á la table.

Afin de ne pas trop modifier le serveur d'associations, on va déléguer la gestion des statistique á un serveur tiers.

Écrire un serveur `OCaml` ou `C` proposant deux services :

1) un service `stats` acceptant les commandes suivantes :
   - `PUT id` augmente les statistiques d'accés pour le client `id`
   - `GET id` augmente les statistiques d'accés pour le serveur `id`
2) un service `bilan` acceptant la commande `GET` et retournant les statistiques par client selon le format : `client: ID Nombre d'ajouts = XXXX , Nombre d'accés = YYYY`

On pourra par exemple écouter sur deux port distincts pour parvenir á cet objectif.

Question 4

Écrire les interactions entre le serveur Java et le serveur OCaml

Tester le serveur de statistiques avec la commande `telnet`, puis mettre tout en marche.

Exercice 5 – Approfondissment

Question 1

En lisant la page de manuel de la section 3 de la fonction `socket`. Et en vous inspirant de ce que vous avez codé en OCaml produire une implantation de la commande `nc` en langage C.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include <netinet/ip.h>
#include<time.h>
#include <netdb.h>

int main(int argc, char** argv)
{
  char buff[20];
  struct sockaddr_in servaddress;
  struct hostent *resbyname;
  int sock = socket(PF_INET,SOCK_STREAM,0),i;
  int err;
  if (argc < 2) {
      fprintf(stderr,"il me faut 2 arguments!\n");
      exit(1);
  }
  printf("argv 1 %s\n",argv[1]);
  resbyname = gethostbyname(argv[1]);
  servaddress.sin_family = AF_INET;
  servaddress.sin_port = htons(atoi(argv[2]));
  servaddress.sin_addr.s_addr = atoi(resbyname->h_addr_list[0]);
  resbyname->h_addr_list[0][0] = '\0';
  err = connect(sock, (struct sockaddr*) &servaddress, sizeof(struct sockaddr_in));
  printf("err %d\n",err);fflush(stdout);
  for(i=0; i<3; i++) {
    write(sock, "Ping", 5); //pour le '\0'
    read(sock, buff, sizeof(buff));
    printf("J'ai lu %s\n", buff);
   }
  write(sock, "Toto", 5);//pour le '\0'
  write(sock, "Exit", 5);//pour le '\0'
  close(sock);
  free(resbyname);
}
```

Question 2

Que signifie ouvrir une socket en mode RAW ? Quels paramètres faut-il modifier dans la création de la *socket* pour cela ?

Solution:

Il suffit de déclarer une `SOCK_RAW` au lieu de déclarer une `SOCK_STREAM`.

Question 3

Que fait la fonction `setsockopt` ? Pourquoi l'option `IP_HDRINCL` est-elle utile pour implanter des *scans* évolués tels que les `NULL` scan, Christmas scan, etc. ?

Solution:

N.B. *In information technology, a Christmas tree packet is a packet with every single option set for whatever protocol is in use.* [2]

2. `http://en.wikipedia.org/wiki/Christmas_tree_packet`

Elle indique au systeme que le header IP est fourni par l'utilisateur et non par lui, ce qui permet de positionner les *flags* á la main et notamment de les grouper en paquets normalement illégaux pour faire ces *scans*.

Question 4

Écrire un *proxy* pour les `echoserver`, i.e. un serveur qui accepte les connexion sur le port 23456 récupère en premier lieu un nom d'hôte et un port, puis effectue une connection a cet hôte sur ce port et se comporte alors comme un tunnel entre le client et l'hôte.