# TCP : C

## Serveur

```
 #include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <time.h>
#include <fcntl.h>
#include <pthread.h>
#include <string.h>
#include <ctype.h>

#define MAX_NAME 10
#define MAX_MSG 20
#define SA struct sockaddr
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

// données relatives à l'entier max et son client associé
char *max_pseudo;
uint16_t max_int;
u_int32_t ip_addr_max = 0;

// structure comportant les information du client courant
typedef struct client {
    int *socket;
    uint32_t ip;
} client;


void *communication(void *arg) {
    /*
     * récupération des arguments de la structure donnée en argument
     * et initialisation des paramètres.
     */
    client *cli = (client*) arg;
    int *socket_com = cli -> socket;
    uint32_t ip_com = cli -> ip;

    // pseudo du client
    char name[MAX_NAME];
    int ret = recv(*socket_com, name, (MAX_NAME)*sizeof(char), 0);
    name[ret] = '\0';
    printf("Message reçu : %s\n", name);

    // réponse "Hello <pseudo>"
    char resp[MAX_NAME+7];
    memcpy(resp, "HELLO ", 6);
    memcpy(resp+6, name, MAX_NAME);
    resp[MAX_NAME+6] = '\0';
    printf("réponse : %s\n", resp);
    send(*socket_com, resp, strlen(resp)*sizeof(char), 0);

    // gestion message client
    char request[MAX_MSG];
    ret = recv(*socket_com, request, (MAX_MSG)*sizeof(char), 0);

    request[ret] = '\0';
```

```c
    // récupération du premier mot (pour connaitre la requête)
    char arg1[3+1];
    uint16_t arg2;

    memcpy(arg1, request, 3);
    arg1[4] = '\0';

    if (strcmp(arg1, "INT") == 0) {

        memcpy(&arg2, request+4, 2);
        printf("Message reçu : INT %u\n", (unsigned short)ntohs(arg2));

        if (ntohs(arg2) >= max_int) {

            printf("Mise à jour max INT\n");
            pthread_mutex_lock(&lock);

            max_int = ntohs(arg2);
            memcpy(max_pseudo, name, MAX_NAME);
            memcpy(&ip_addr_max,&ip_com,4);

            pthread_mutex_unlock(&lock);

        }

        send(*socket_com, "INTOK", strlen("INTOK")*sizeof(char), 0);
        goto end;

    }

    if (strcmp(arg1, "MAX") == 0) {
        printf("Message reçu : MAX\n");
        char resp_req[MAX_MSG]; // 3+10+4+2+1
        printf("max int : %u\n", (unsigned short)max_int);
        if (max_int == 0) {
            ret = send(*socket_com, "NOP" , 3, 0);
            assert(ret >= 0);
        } else {
            memcpy(resp_req, "REP", 3);
            memcpy(resp_req+3, max_pseudo, 10);
            memcpy(resp_req+13, &ip_addr_max, 4);
            uint16_t maxToSend = htons(max_int);
            memcpy(resp_req+17, &maxToSend, 2);
            ret = send(*socket_com, resp_req , 19, 0);
            assert(ret >= 0);
        }
    }

    end:
        close(*socket_com);
        return NULL;

}

int main(int argc, char **argv) {

    (void) argc;

    int sock = socket(PF_INET,SOCK_STREAM,0);

    struct sockaddr_in server_socket;
    server_socket.sin_family = AF_INET;
    server_socket.sin_port= htons(atoi(argv[1]));
    server_socket.sin_addr.s_addr=htonl(INADDR_ANY);
    int ret = bind(sock, (SA*) &server_socket, sizeof(server_socket));
    assert(ret >= 0);

    ret = listen(sock,0);
    assert(ret >= 0);

    struct sockaddr_in caller;
    socklen_t socket_size = sizeof(caller);

    max_int = 0;
    max_pseudo = (char *) malloc(sizeof(char));

    while(1) {

        int *server_socket_bis = (int *)malloc(sizeof(int));
        *server_socket_bis = accept(sock, (SA*)&caller, &socket_size);

        if(*server_socket_bis >=0) {

            client *cli = malloc(sizeof(client));
            cli -> socket = server_socket_bis;
            cli -> ip = caller.sin_addr.s_addr;

            pthread_t th;
            pthread_create(&th, NULL, communication, cli);
            pthread_join(th, NULL);

            free(cli);
            close(*server_socket_bis);

        }
    }

    close(sock);
    return EXIT_SUCCESS;

}
```

## Client 1

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <time.h>

#define MAX_NAME 10

int main(int argc, char *argv[]){
  (void) argc;
  struct sockaddr_in adress_sock;
  adress_sock.sin_family = AF_INET;
  adress_sock.sin_addr.s_addr = inet_addr("127.0.0.1");
  adress_sock.sin_port = htons(atoi(argv[1])); //Récupération du numéro de port
  inet_aton("lulu", &adress_sock.sin_addr);

  //Pour la générer un random uint32_t
  srand(time(NULL));
  int sock;
  // création de 5 clients
  for(int i = 0 ; i < 5 ; i++){
    sock = socket(PF_INET, SOCK_STREAM, 0);
    int ret = connect(sock, (struct sockaddr *)&adress_sock, sizeof(struct sockaddr_in));
    printf("\n#################### Nouveau client ####################\n");

    // taille des pseudo = 10 mais peut aussi être inférieur
    char *pseudo_list[5] = {"Louismmmmm", "Paulmmmmmm", "Lauriemmmm", "Alicemmmmm", "Hugommmmmm"};
    char name[MAX_NAME];
    strcpy(name, pseudo_list[i]);

    if(ret != -1){

      send(sock,name,strlen(name)*sizeof(char),0);
      printf("Envoie du pseudo : %s\n", name);

      //  attente de la réponse "HELLO <pseudo>"
      char reponse_hello_pseudo[MAX_NAME + 7];
      int size_rec = recv(sock, reponse_hello_pseudo, (MAX_NAME + 7)*sizeof(char), 0);
      reponse_hello_pseudo[size_rec] = '\0';
      printf("Réception : %s\n", reponse_hello_pseudo);


      /*
       *
       * Requête INT<val>
       *
       */

      // Génération du nombre aléatoire à envoyer
      // (sur 2 octets, soit 16 bits, soit 2^16 : un nombre compris [0 ; 65535])
      uint16_t val = rand() % 65535;
      val = htons(val);

      //Création du message + envoi de la requête
      char send_int_val[7];
      memcpy(send_int_val, "INT ", 4);
      memcpy(send_int_val+4, &val, 2);
      send_int_val[7] = '\0';
      printf("Envoie : INT%d\n", (unsigned short)val);
      send(sock,send_int_val,strlen(send_int_val), 0);

      // ******** Attente de confirmation de la reception  ******** //
      char reponse_intok[6];
      size_rec = recv(sock, reponse_intok, (6)*sizeof(char), 0);
      reponse_intok[size_rec] = '\0';
      printf("Confirmation reception de la requête INT<val> : %s\n", reponse_intok);

      // Après l'envoi du message le client se déconnecte
      printf("Fin de la connection avec %s, deconnexion\n", name);
      close(sock);

    }

    else{
      perror("Erreur de connexion du client 1");
      close(sock);
      exit(1);
    }

  }
  close(sock);
  return EXIT_SUCCESS;
}
```

## Client 2

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <assert.h>

#define MAX_NAME 10
#define MAX 19

int main(int argc, char **argv){
    (void) argc;
    struct sockaddr_in adress_sock;
    adress_sock.sin_family = AF_INET;
    adress_sock.sin_addr.s_addr = inet_addr("127.0.0.1");
    adress_sock.sin_port =  htons(atoi(argv[1]));

    int ret = inet_aton("lulu", &adress_sock.sin_addr);
    assert(ret >= 0);

    //Connexion au serveur
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    ret = connect(sock, (struct sockaddr *)&adress_sock, sizeof(struct sockaddr_in));

    if (ret == -1) {
        perror("Erreur de connexion du client 2");
        goto end;
    }


    char name[MAX_NAME];
    // taille du pseudo égale à 10 mais peut aussi être inférieur
    memcpy(name, "Matthieumm", 10);

    ret = send(sock, name, 10, 0);
    assert(ret >= 0);

    //Attente de la réponse du serveur : "HELLO <pseudo>"
    char reponse_hello_pseudo[MAX_NAME + 6];
    ret = recv(sock, reponse_hello_pseudo, MAX_NAME+6, 0);
    assert(ret >= 0);

    ret = send(sock, "MAX", 3, 0);
    assert(ret >= 0);

    char resp_req[MAX];
    ret = recv(sock, resp_req, 19, 0);
    assert(ret >= 0);

    if (ret == 3) {
        printf("There is no max int in the server\n");
        goto end;
    }

    char pseudo[10+1];
    pseudo[10] = '\0';
    uint32_t ip;
    uint16_t max_int_val;
    memcpy(&pseudo, resp_req+3, 10);
    memcpy(&max_int_val, resp_req+17, 2);
    memcpy(&ip, resp_req+13, 4);

    struct in_addr struct_output;
    struct_output.s_addr = ip;
    char *ip_output = inet_ntoa(struct_output);

    printf("réponse serveur : REP%s%d%s\n", pseudo, ntohs(max_int_val), ip_output);
    printf("pseudo : %s\n", pseudo);
    printf("max int : %d\n", ntohs(max_int_val));
    printf("ip : %s\n", ip_output);

    end:
        close(sock);
        return 0;
}
```

# TCP : JAVA

## Client

```
public class ClientTCP implements Runnable {
    private Socket clientSocket;
    private BufferedReader in;
    private PrintWriter out;

    public ClientTCP(String ip, int port) {
        try {
            DebugLogger.print(DebugType.COM, "Création de la connection TCP avec le serveur...");
            this.clientSocket = new Socket(ip, port);
            this.out = new PrintWriter(clientSocket.getOutputStream());
            this.in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {

        // tant que le socket est connecté
        while(Client.isConnected) {

            try {

                    readVal = in.read();

                    // si le socket est déconnecté : arrêter de lire
                    if (readVal == -1) {
                        return;
                    }

                    if (/* something */) {
                        return;
                    }

            } catch(IOException e) {
                DebugLogger.print(DebugType.CONFIRM, "Socket closed : disconnected");
            }
        }
        clientSocket.close();
    }
```

## Server

```
class ServeurTcp {

    public void servir(int noPort) {
        try {
            ServerSocket serveur = new ServerSocket(noPort);
            /*
            * La variable "enService" contrôle la boucle principale du serveur
            * qui se termine avec l'application. À vous de trouver une façon
            * d'en changer la valeur.
            */
            boolean enService = true;

            while (enService) {
                /*
                 * Le programme est ici bloqué dans l'attente d'une connexion. La
                 * connexion aura pour effet de créer un socket (variable "client")
                 * branché à l'application distante.
                 */
                 */
                Socket client = serveur.accept();
                /*
                 * Création des flux d'entrée-sortie pour échanger de l'information
                 * avec le client en format binaire. On utilisera (mais il peut y
                 * en avoir d'autres plus appropriées) les méthodes "readByte" pour
                 * lire et "writeByte" pour écrire.
                 */
                DataInputStream in = new DataInputStream(client.getInputStream());
                DataOutputStream out = new DataOutputStream(client.getOutputStream());
                /*
                 * Création des flux d'entrée-sortie pour échanger de l'information
                 * avec le client en format texte. On utilisera les méthodes
                 * "readLine" pour lire et "println" pour écrire.
                 */
                BufferedReader reader = new BufferedReader(new InputStreamReader(client.getInputStream()));
                PrintWriter writer = new PrintWriter(new OutputStreamWriter(client.getOutputStream()));
                /*
                 * La variable "fini" contrôle la boucle secondaire, celle qui se
                 * se termine avec la déconnexion d'un client. À vous de trouver
                 * une façon d'en changer la valeur.
                 */
                boolean fini = false;
                String ligneMessage = "";
                String mot = "";
                char[] charBuffer = new char[100];
               /*
                * Traitement (lecture des requêtes du client et écriture des
                * réponses du serveur).
                */
                while (!fini) {

                    // BufferedReader
                    // lit toute la chaine de caractère sur le buffer jusqu'à la fin de ligne \n
                    ligneMessage = reader.readLine();
                    // lit le premier caractère sur le buffer (sa valeur entière : -1 si buffer nul)
                    // puis cast en char pour avoir sa valeur en caractère
                    mot += (char) reader.read().byteValue();
                    // lit le nombre de caractère demandé et les mets dans un tableau de char
                    // public int read(char[] cbuf, int off, int len);
                    read(charBuffer, 0, 99);

                    // PrintWriter
                    // write(49) prints "1", print(49) prints "49"
                    writer.write(49); // write "1"
                    writer.write("hello world"); // write "hello world"
                    writer.write("hello world", 0, 4); // write "hello"
                    writer.write(new char[] { 'h', 'e', 'l', 'l', 'o' });
                    writer.write(new char[] { 'h', 'e', 'l', 'l', 'o' }, 0, 1); // write "he"
                    writer.print(49) // print "49"
                    writer.print("hello"); // print "hello"
                    writer.print('h'); // print "h"
                    writer.print(new char[] { 'h', 'e', 'l', 'l', 'o' }); // print "hello"

                }

                /*
                 * Libération des ressources à la fin du traitement d'un client.
                 */
                reader.close();
                writer.close();
                client.close();
            }

            serveur.close();
        } catch (IOException ioe) {
             /*
              * Ici on choisit de quitter l'application, mais il y aurait peut-être
              * moyen de traiter l'exception.
              */
             System.err.println("Erreur d'entre-sortie");
             System.exit(1);
        }
    }

   /*
    * Point d'entrée de l'application, cette méthode fait trois choses :
    * 1) valide les paramètres de la ligne de commande;
    * 2) crée une instance de la classe courante;
    * 3) appelle une méthode de l'objet pour démarrer le traitement.
    */
    public static void main(String[] args) {
        switch (args.length) {
            case 0:
                System.err.println("Vous devez specifier un numero de port");
                break;
            case 1:
                try {
                    int noPort = Integer.parseInt(args[0]);

                    if ((noPort >= 0) && (noPort <= 65537)) {
                        new ServeurTcp().servir(noPort);
                    } else  {
                        System.err.println("Le numero de port est hors intervale");
                    }
                } catch (NumberFormatException nfe) {
                    System.err.println("Le numero de port doit etre un nombre entier");
                }
                break;
            default:
                System.err.println("Il y a trop de parametres");
                break;
        }
    }
}
```

# UDP : C

## Server

```
 #include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <time.h>
#include <fcntl.h>
#include <pthread.h>
#include <string.h>
#include <ctype.h>

int main() {

    int sock = socket(PF_INET, SOCK_DGRAM, 0);

    struct sockaddr_in server_socket;
    server_socket.sin_family = AF_INET;
    server_socket.sin_port = htons(1717);
    server_socket.sin_addr.s_addr = htonl(INADDR_ANY);

    /*
     * OU
     * struct addrinfo hints;
     * bzero(&hints, sizeof(struct addrinfo));
     * hints.ai_family = AF_INET;
     * hints.ai_socktype=SOCK_DGRAM;
     * struct addrinfo *first_info;
     * int r=getaddrinfo("localhost","1717",&hints,&first_info);
     */

    int ret = bind(sock,(struct sockaddr*) &server_socket,sizeof(server_socket));
    assert(ret >= 0);

    struct sockaddr_in emet;
    socklen_t a=sizeof(emet);
    char msg_rcv[100];
    char msg_snd[22];
    memcpy(msg_snd, "ça va toi ? Moi ouais\0", 22);

    struct sockaddr_in client_socket;

    while(1){
        int rec = recvfrom(sock,msg_rcv,100,0,(struct sockaddr *)&emet,&a);
        msg_rcv[rec] = '\0';
        printf("Message recu : %s\n", msg_rcv);
        client_socket.sin_port = emet.sin_port;
        client_socket.sin_addr = emet.sin_addr;
        printf("%s\n", msg_snd);
        if (strcmp(msg_rcv, "close") == 0) {
            break;
        }
        sendto(sock, msg_snd, 22, 0,
                (struct sockaddr *)&client_socket,
                (socklen_t) sizeof(struct sockaddr_in));
    }
    close(sock);

}
```

## Client

```
 #include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

int main(int argc, char **argv) {

    if(argc != 3) {
        printf("Wrong usage !\n");
        return EXIT_FAILURE;
    }

    int sock = socket(AF_INET, SOCK_DGRAM, 0);

    struct sockaddr_in remote = {
        .sin_family = AF_INET,
        .sin_port = htons(1717)
        .sin_addr.s_addr = INADDR_ANY; // .sin_addr.s_addr = inet_addr(argv[1]); sans inet_aton
    };

    int ret = inet_aton(argv[2],&remote.sin_addr);
    assert(ret > 0);

    for(int i = 0 ; i < atoi(argv[1]) ; i++) {
        sendto(sock, NULL, 0, 0, (struct sockaddr *) &remote, sizeof(remote));

        char buf[100];
        ret = recv(sock,buf,100,0);
        buf[ret] = '\0';

        printf("%s\n",buf);
    }
    close(sock);

    return EXIT_SUCCESS;
}
```

# UDP : JAVA

## Serveur

```
// Java program to illustrate Server side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class ServerUdp {
    public static void main(String[] args) {

        while(true) {
            try {
                byte[] receiveBuf = new byte[256];
                DatagramPacket packetToReceive = new DatagramPacket(receiveBuf, receiveBuf.length);
                socket.receive(packetToReceive);

                String resp = "hello world";
                byte[] respBuf = resp.getBytes();

                InetAddress adress = packetToReceive.getAddress();
                int port = request.getPort();

                DatagramPacket packetToSend = new DatagramPacket(respBuf, respBuf.length, adress, port);
                socket.send(packetToSend);

                String message = new String(packetToReceive.getData(), 0, packetToReceive.getLength());

                String[] messageArgs = message.split(" ");
            } catch (IOException e) {
                System.out.println("[ClientTCP/ERREUR] : erreur lors de la réception d'un message UDP");
            }
        }
    }
}
```

## Client

```
public class ClientUDP implements Runnable {
    private DatagramSocket socket;
    private InetAddress address;
    private int port;

    public ClientUDP(String ip) {

        boolean success = false;
        Random r = new Random();
        int port = r.nextInt(9999 - 1000) + 1000;

        int attempt = 1;

        System.out.println("Création du client UDP...");

        while(!success) {
            try {

                System.out.println("Tentative numéro " + attempt + "...");

                socket = new DatagramSocket(port);
                address = InetAddress.getByName("localhost");

                System.out.println("...succès");
                this = port;

                success = true;

            } catch (SocketException e) {
                System.out.println("[ClientUDP/ERREUR] : port UDP déjà utilisé");
                attempt++;
                port = r.nextInt(9999 - 1000) + 1000;
            }
            catch (UnknownHostException e) {
                System.out.println("[ClientUDP/ERREUR] : : l'adresse IP de l'hôte ne peut être déterminée");
                break;
            }
        }
    }

    @Override
    public void run() {

        while(Client.isConnected) {
            try {
                String messageClient = "hello world";
                byte[] byteBuffer = messageClient.getBytes();
                DatagramPacket packet = new DatagramPacket(byteBuffer, byteBuffer.length, adress, port);
                DatagramPacket packetToReceive = new DatagramPacket(byteBuffer, byteBuffer.length);
                socket.send(packet);
                socket.receive(packetToReceive);

                String messageServeur = new String(packetToReceive.getData(), 0, packetToReceive.getLength());

                String[] messageArgs = messageServeur.split(" ");
            } catch (IOException e) {
                System.out.println("[ClientTCP/ERREUR] : erreur lors de la réception d'un message UDP");
            }
        }
    }
```

## Multicast : JAVA

## Envoi

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

public class EnvoiMulticast {

    public static void main(String[] args){

        try{

            DatagramSocket dso = new DatagramSocket();
            byte[] data;

            for(int i=0 ; i <= 10 ; i++){
                String s = "MESSAGE " + i + " \n";
                data = s.getBytes();
                InetSocketAddress ia = new InetSocketAddress("225.1.2.4",9999);
                DatagramPacket paquet = new DatagramPacket(data,data.length,ia);
                dso.send(paquet);
            }

        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

## Réception

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

public class RecoitMulticast {

    public static void main(String[] args){

        try {

            MulticastSocket mso = new MulticastSocket(9999);
            mso.joinGroup(InetAddress.getByName("225.1.2.4"));
            byte[] data = new byte[100];
            DatagramPacket paquet = new DatagramPacket(data,data.length);

            while(true) {
                mso.receive(paquet);
                String st = new String(paquet.getData(),0,paquet.getLength());
                System.out.println("J'ai reçu :"+st);
            }

        } catch(Exception e){
            e.printStackTrace();
        }

    }
}
```

# Multicast : C

## Envoi

```
int main() {

    int sock=socket(PF_INET,SOCK_DGRAM,0);

    struct addrinfo *first_info;
    struct addrinfo hints;

    memset(&hints, 0, sizeof(struct addrinfo));

    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;

    int r=getaddrinfo("225.1.2.4", "9999", &hints,&first_info);

    if(r == 0){

        if(first_info != NULL){

            struct sockaddr *saddr = first_info -> ai_addr;
            char tampon[100];
            int i = 0;

            for(i = 0 ; i <= 10 ; i++){
                strcpy(tampon,"MESSAGE ");
                char entier[3];
                sprintf(entier, "%d", i);
                strcat(tampon, entier);
                sendto(sock, tampon, strlen(tampon), 0, saddr, (socklen_t)sizeof(struct sockaddr_in));
            }
        }
    }
    return 0;
}
```

## Réception

```
 int main() {

    int sock = socket(PF_INET,SOCK_DGRAM,0);
    int ok = 1;
    int r = setsockopt(sock,SOL_SOCKET,SO_REUSEPORT,&ok,sizeof(ok));

    struct sockaddr_in address_sock;

    address_sock.sin_family = AF_INET;
    address_sock.sin_port = htons(9999);
    address_sock.sin_addr.s_addr = htonl(INADDR_ANY);

    r = bind(sock, (struct sockaddr *)&address_sock, sizeof(struct sockaddr_in));

    struct ip_mreq mreq;

    mreq.imr_multiaddr.s_addr = inet_addr("225.1.2.4");
    mreq.imr_interface.s_addr = htonl(INADDR_ANY);

    r=setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));

    char tampon[100];

    while(1){
        int rec = recv(sock, tampon, 100, 0);
        tampon[rec] = '\0';
        printf("Message recu : %s\n", tampon);
    }
    return 0;
}
```

# Broadcast :

## JAVA :

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

public class EnvoiBroadcast {

    public static void main(String[] args){

        try{

            DatagramSocket dso = new DatagramSocket();
            byte[]data;

            for(int i = 0 ; i <= 10; i++){

                Thread.sleep(1000);
                String s = "MESSAGE " + i + " \n";
                data = s.getBytes();
                InetSocketAddress ia = new InetSocketAddress("255.255.255.255", 8888);
                DatagramPacket paquet = new DatagramPacket(data, data.length, ia);
                dso.send(paquet);
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

# C :

```
int main() {

    int sock = socket(PF_INET,SOCK_DGRAM,0);
    int ok = 1;
    int r = setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &ok, sizeof(ok));

    if(r == 0) {
        struct addrinfo *first_info;
        struct addrinfo hints;
        memset(&hints, 0, sizeof(struct addrinfo));
        hints.ai_family = AF_INET;
        hints.ai_socktype = SOCK_DGRAM;
        r = getaddrinfo("255.255.255.255", "8888", &hints,&first_info);

        if(r == 0){

            if(first_info != NULL){

                struct sockaddr *saddr = first_info -> ai_addr;
                char tampon[100];
                int i = 0;

                for(i = 0; i<=10 ; i++){
                    strcpy(tampon, "MESSAGE ");
                    char entier[3];
                    sprintf(entier, "%d", i);
                    strcat(tampon, entier);
                    sendto(sock, tampon, strlen(tampon), 0, saddr, (socklen_t) sizeof(struct sockaddr_in));
                }
            }
        }
    }
    return 0;
}
```