C

TCP

Serveur générique

```
int main(int argc, char* argv[]){
    int sock=socket(PF_INET, SOCK_STREAM, 0); //err==-1
    struct sockaddr_in adress;
    adress.sin_family=AF_INET; //ou AF_INET6 pour IPv6
    adress.sin_port=htons(atoi(argv[1])); //port
    --> adress.sin_addr.s_addr=htonl(INADDR_ANY);

    --> int r=bind(sock, (struct sockaddr *)&adress, sizeof(struct sockaddr_in));
//err==-1
    --> r=listen(sock, 0);  //err==-1
    while(1){
        struct sockaddr_in client;
        socklen_t size=sizeof(client);
        int sock2=accept(sock, (struct sockaddr *)&client, &size); //err==-1

        //lecture
        char buff[100];
        uint16_t answer, number=6666;
        int len=recv(sock2, &answer, sizeof(uint16_t), 0); //err==-1

        //écriture
        if(htons(answer)==number){
            sprintf(buff, "GAGNE\n");
            send(sock2, buff, strlen(buff), 0);
        }
        close(sock2);
    }
}
```

Serveur avec fork

```
while(1){
    int sock2=accept(sock, (struct sockaddr *)&client, &size); //err==-1
    int idfork=fork();
    if(idfork==0){
        close(sock);
        /*Code de communication*/
        close(sock2);
        exit(0);
    }
    else close(sock2);
}
```

Serveur avec Thread

```c
pthread_mutex_t verrou=PTHREAD_MUTEX_INITIALIZER;
int a=0; //variable globale

void* fonction(void* arg){ //avec deux sockets
    int player1=*((int*)arg);
    int player2=*((int*)arg+sizeof(int));
    free(arg);
    /*Code de communication*/
    close(player1);
    close(player2);

    //exemple pour le verrou
    pthread_mutex_lock(&verrou);
    a++;
    pthread_mutex_unlock(&verrou);
    return NULL;
}

while(1){
    int* sock2=(int*)malloc(sizeof(int));
    *sock2=accept(sock, (struct sockaddr*)&client, &size); //err==-1
    pthread_t th;
    pthread_create(&th, NULL, fonction, sock2);
    pthread_join(th1, NULL); //pour que les threads s'attendent
}
```

Client

```c
int main(int argc, char* argv[]){
    int sock=socket(PF_INET, SOCK_STREAM, 0); //err==-1
    struct sockaddr_in adress;
    adress.sin_family=AF_INET;
    adress.sin_port=htons(atoi(argv[2])); //port
    --> inet_aton(argv[1], &adress.sin_addr); //adresse (ex : "127.0.0.1")

    --> int r=connect(sock, (struct sockaddr *)&adress, sizeof(struct sockaddr_in));
//err==-1

    //écriture
    uint16_t answer=htons(666); //nombre random
    r=send(sock, &answer, sizeof(uint16_t), 0); //err==-1

    //lecture
    char buff[100];
    r=recv(sock, buff, 99, 0); //err==-1
    --> buff[r]='\0';

    if(strcmp(buff, "GAGNE\n")==0) printf("Victory");
    close(sock);
}
```

UDP

Serveur

```c
int main(int argc, char* argv[]){
    int sock=socket(PF_INET, SOCK_DGRAM, 0); //err==-1
    struct sockaddr_in address;
    address.sin_family=AF_INET;
    address.sin_port=htons(atoi(argv[1])); //port
    address.sin_addr.s_addr=htonl(INADDR_ANY);

    int r=bind(sock, (struct sockaddr *)&address_sock, sizeof(struct sockaddr_in));
//err==-1
    --> //pas de listen en UDP, ni d'accept

    struct sockaddr_in emet;
    socklen_t size=sizeof(emet);
    char tampon[100];
    while(1){
        int rec=recvfrom(sock, tampon, 100, 0, (struct sockaddr*)&emet, &size);
//err==-1
        tampon[rec]='\0';

        char fortune[512]; //initialisé par un memset quelque part
        sendto(sock, fortune, strlen(fortune), 0, (struct sockaddr*)&emet, size);
    }
}
```

Client

```c
int main(int argc, char** argv){
    int sock=socket(PF_INET, SOCK_DGRAM, 0); //err==-1
    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(argv[1], argv[2], &hints, &first_info); //adresse, port
    //err==-1 et first_info!=NULL, pas de connect
    struct sockaddr* saddr=first_info->ai_addr;
    char tampon[100];
    sendto(sock, NULL, 0, 0, saddr, (socklen_t)sizeof(struct sockaddr_in));

    int rec=recv(sock, tampon, 100, 0); //err==-1
    tampon[rec]='\0';
    printf("%s", tampon);
}
```

Broadcast

```
int main(){
    //Envoie
    int sock=socket(PF_INET, SOCK_DGRAM, 0); //err==-1
    int ok=1;
    --> int r=setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &ok, sizeof(ok)); //err==-1
    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", argv[1], &hints, &first_info); //port
    //err==-1 et first_info!=NULL
    struct sockaddr* saddr=first_info->ai_addr;
    char tampon[100];
    sprintf(tampon, "Message %d", 1);
    sendto(sock, tampon, strlen(tampon), 0, saddr, (socklen_t)sizeof(struct
sockaddr_in));

    //Réception
    int sock=socket(PF_INET, SOCK_DGRAM, 0); //err==-1
    struct sockaddr_in address_sock;
    address_sock.sin_family=AF_INET;
    address_sock.sin_port=htons(argv[1]); //port
    address_sock.sin_addr.s_addr=htonl(INADDR_ANY);

    int r=bind(sock, (struct sockaddr*)&address_sock, sizeof(struct sockaddr_in));
//err==-1
    char tampon[100];
    while(1){
        int rec=recv(sock, tampon, 100, 0);
        tampon[rec]='\0';
        printf("Message recu : %s\n", tampon);
    }
}
```

Multicast

```
int main(){
    //Envoie
    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(argv[1], argv[2], &hints, &first_info); //adresse comprise
entre 224.0.0.0 et 238.255.255.255, port
    //err==-1 et first_info!=NULL
    struct sockaddr* saddr=first_info->ai_addr;
    char tampon[100];
    sprintf(tampon, "Message %d", 1);
    sendto(sock, tampon, strlen(tampon), 0, saddr, (socklen_t)sizeof(struct
sockaddr_in));

    //Réception
    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(argv[2]); //port
    address_sock.sin_addr.s_addr=htonl(INADDR_ANY);

    r=bind(sock, (struct sockaddr*)&address_sock, sizeof(struct sockaddr_in));
//err==-1
    struct ip_mreq mreq;
    mreq.imr_multiaddr.s_addr=inet_addr(argv[1]); //adresse
    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);
    }
}
```

Java

TCP

Serveur générique

```
public static void main(String[] args){
    try{
        ServerSocket server=new ServerSocket(Integer.parseInt(args[0])); //port
        while(true){
            Socket socket=server.accept();

            //lecture
            BufferedReader br=new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String clientMessage=br.readLine();
            System.out.println(clientMessage);

            //écriture
            PrintWriter pw=new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
            pw.print("message du serveur");
            pw.flush();

            br.close();
            pw.close();
            socket.close();
        }
    }
}
```

Serveur avec Thread

```
public class Service implements Runnable{
    Socket socket;

    Service(Socket s){
        socket=s;
    }

    public void run(){
        try{
            /*Communication*/
            socket.close();
        }
    }
}

public class Serveur{
    public static void main(String[] args){
        try{
            ServerSocket server=new ServerSocket(Integer.parseInt(args[0]));
            while(true){
                Socket socket=server.accept();
                new Thread(new Service()).start();
            }
        }
    }
}
```

Client

```
public static void main(String[] args){
    try{
        Socket socket=new Socket(args[0], Integer.parseInt(args[1])); //adresse, port

        //écriture
        PrintWriter pw=new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
        pw.print("message\n");
        pw.flush();

        //lecture
        BufferedReader br=new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String resServer=br.readLine();
        System.out.println(resServer);

        br.close();
        pw.close();
        socket.close();
    }
}
```

UDP

Serveur

```
public static void main(String[] args){
    try{
        DatagramSocket dso=new DatagramSocket(Integer.parseInt(args[0])); //port
        byte[] tampon=new byte[100];
        while(true){
            //réception
            DatagramPacket receive=new DatagramPacket(tampon, tampon.length);
            dso.receive(receive);
            System.out.println("receive : "+new String(receive.getData()));

            //envoi
            tampon=new String("message").getBytes();
            DatagramPacket toSend=new DatagramPacket(tampon, tampon.length,
                new InetSocketAddress(receive.getAddress(), receive.getPort()));
            dso.send(toSend);
            System.out.println("send : "+new String(toSend.getData()));
        }
    }
}
```

Client

```java
public static void main(String[] args){
    try{
        DatagramSocket dso=new DatagramSocket();
        byte[] data=new byte[100];
        for(int i=0; i<5; i++){
            //envoie
            InetSocketAddress ia=new InetSocketAddress(args[0],
Integer.parseInt(args[1])); //adresse, port
            DatagramPacket toSend=new DatagramPacket(new byte[0], 0, ia);
            dso.send(toSend); //envoie des paquets vides

            //réception
            DatagramPacket receive=new DatagramPacket(data, data.length);
            dso.receive(receive);
            System.out.println(new String(receive.getData(), 0, receive.getLength()));
        }
    }
}
```

Broadcast

```java
public static void main(String[] args){
    //Envoie
    try{
        DatagramSocket dso=new DatagramSocket();
        String s="MESSAGE\n";
        byte[] data=s.getBytes();
        InetSocketAddress ia=new InetSocketAddress("255.255.255.255",
Integer.parseInt(args[0])); //port et broadcast toujours à 255.255.255.255
        DatagramPacket paquet=new DatagramPacket(data, data.length, ia);
        dso.send(paquet);
    }

    //Réception
    try{
        DatagramSocket dso=new DatagramSocket(Integer.parseInt(args[0])); //port
        byte[] data=new byte[100];
        DatagramPacket paquet=new DatagramPacket(data, data.length);
        while(true){
            dso.receive(paquet);
            String st=new String(paquet.getData(), 0, paquet.getLength());
            System.out.println("J'ai reçu :"+st);
        }
    }
}
```

Multicast

```
public static void main(String[] args){
    //Envoie
    try{
        DatagramSocket dso=new DatagramSocket();
        String s="MESSAGE \n";
        byte[] data=s.getBytes();
        InetSocketAddress ia=new InetSocketAddress(args[0],
Integer.parseInt(args[1])); //adresse comprise entre 224.0.0.0 et 238.255.255.255,
port
        DatagramPacket paquet=new DatagramPacket(data, data.length, ia);
        dso.send(paquet);
    }

    //Réception
    try{
        MulticastSocket mso=new MulticastSocket(Integer.parseInt(args[1])); //port
        mso.joinGroup(InetAddress.getByName(args[0])); //adresse
        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);
        }
    }
}
```

Non bloquant

```java
public static void main(String[] args){
    try{
        Selector sel=Selector.open();
        DatagramChannel dsc1=DatagramChannel.open();
        DatagramChannel dsc2=DatagramChannel.open();
        dsc1.configureBlocking(false);
        dsc2.configureBlocking(false);
        dsc1.bind(new InetSocketAddress(4344));
        dsc2.bind(new InetSocketAddress(4343));
        dsc1.register(sel, SelectionKey.OP_READ);
        dsc2.register(sel, SelectionKey.OP_READ);
        while(true){
            System.out.println("Waiting for messages");
            sel.select();
            Iterator<SelectionKey> it=sel.selectedKeys().iterator();
            while(it.hasNext()){
                SelectionKey sk=it.next();
                it.remove();
                if(sk.isReadable() && sk.channel()==dsc1){
                    ByteBuffer buff=ByteBuffer.allocate(100);
                    System.out.println("Message UDP 4344 recu");
                    dsc1.receive(buff);
                    String st=new String(buff.array(), 0, buff.array().length);
                    buff.clear();
                    System.out.println("Message :"+st);
                }
                else if(sk.isReadable() && sk.channel()==dsc2){
                    ByteBuffer buff=ByteBuffer.allocate(100);
                    System.out.println("Message UDP 4343 recu");
                    dsc2.receive(buff);
                    String st=new String(buff.array(), 0, buff.array().length);
                    System.out.println("Message :"+st);
                }
                else System.out.println("Que s'est il passe");
            }
        }
    }
}
```