FICHE DE RÉVISIONS DE PR6

Exemples en C

Librairies à utiliser

Imports en vrac

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

Structures en vrac

```
struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};
struct in_addr {
    unsigned long s_addr;  // load with inet_aton()
};
struct sockaddr_in6 {
    sa_family_t     sin6_family;   /* AF_INET6 */
    in_port_t       sin6_port;     /* port number */
    uint32_t        sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr;     /* IPv6 address */
    uint32_t        sin6_scope_id; /* Scope ID (new in 2.4) */
};
struct in6_addr {
    unsigned char   s6_addr[16];   /* IPv6 address */
};
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    size_t           ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};
struct pollfd {
    int   fd;         /* file descriptor */
    short events;     /* requested events */
    short revents;    /* returned events */
};
```

Cas concrets TCP

Serveur “c’est celui qui dit qui est”

```
   int port = 4242;
   int sock = socket(PF_INET, SOCK_STREAM, 0);
```

```
struct sockaddr_in address_sock;
address_sock.sin_family = AF_INET;
address_sock.sin_port = htons(port);
address_sock.sin_addr.s_addr = htonl(INADDR_ANY);
int r = bind(sock, (struct sockaddr *)&address_sock, sizeof(struct sockaddr_in));
if (r == 0){
    r = listen(sock, 0);
    while (1){
        struct sockaddr_in caller;
        socklen_t size = sizeof(caller);
        int sock2 = accept(sock, (struct sockaddr *)&caller, &size);
        if (sock2 >= 0){
            printf("Port de l'appelant: %d\n", ntohs(caller.sin_port));
            printf("Adresse de l'appelant: %s\n", inet_ntoa(caller.sin_addr));
            char buff[100];
            int recu = recv(sock2, buff, 99 * sizeof(char), 0);
            buff[recu] = '\0';
            printf("Message recu : %s\n", buff);
            send(sock2, buff, strlen(buff) * sizeof(char), 0);
            close(sock2);
        }}}
```

Client en manque d'affection

```
int port = 4242;
struct sockaddr_in adress_sock;
adress_sock.sin_family = AF_INET;
adress_sock.sin_port = htons(port);
inet_aton("127.0.0.1", &adress_sock.sin_addr);
int descr = socket(PF_INET, SOCK_STREAM, 0);
int r = connect(descr, (struct sockaddr *)&adress_sock, sizeof(struct sockaddr_in));
if (r != -1){
    char *mess = "Est-ce que tu m'aimes ?\n";
    send(descr, mess, strlen(mess), 0);
    char buff[100];
    int size_rec = recv(descr, buff, 99 * sizeof(char), 0);
    buff[size_rec] = '\0';
    printf("Caracteres recus : %d\n", size_rec);
    printf("Message : %s\n", buff);
    close(descr);
}
```

Client cherche hostname chaud dans ta région

Ici on ne va pas utiliser struct hostent* gethostbyname(const char* name); en mode old-school

```
struct addrinfo *first_info;
struct addrinfo hints;
memset(&hints, 0, sizeof hints);
hints.ai_family = PF_UNSPEC;
int r = getaddrinfo("www.google.com", NULL, &hints, &first_info);
if (r == 0){
    struct addrinfo *info = first_info;
    while (info != NULL){
        struct sockaddr *saddr = info->ai_addr;
        if (saddr->sa_family == AF_INET){
            struct sockaddr_in *addressin = (struct sockaddr_in *)saddr;
```

```
            struct in_addr address = (struct in_addr)(addressin->sin_addr);
            printf("Address IPv4 : %s\n", inet_ntoa(address));
        }
        if (saddr->sa_family == AF_INET6){
            struct sockaddr_in6 *addressin = (struct sockaddr_in6 *)saddr;
            struct in6_addr address = (struct in6_addr)(addressin->sin6_addr);
            char *string_address = (char *)malloc(sizeof(char) * INET6_ADDRSTRLEN);
            inet_ntop(AF_INET6, &address, string_address, INET6_ADDRSTRLEN);
            printf("Address IPv6 : %s\n", string_address);
        }
        info = info->ai_next;
    }
}
freeaddrinfo(first_info);
```

Serveur qui prend en charge des clients de manière concurrente (comme le fait ta mère)

```
void *communication(void *arg){
   int so = *((int *)arg);
   char *mess = "BONJOUR";
   send(so, mess, strlen(mess) * sizeof(char), 0);
   char buff[100];
   int recu = recv(so, buff, 99 * sizeof(char), 0);
   buff[recu] = '\0';
   printf("Recu : %s\n", buff);
   free(arg);
   close(so);
   return NULL;
}


int main(int argc, char **argv){
   int port = 4242;
   int sock = socket(PF_INET, SOCK_STREAM, 0);
   struct sockaddr_in address_sock;
   address_sock.sin_family = AF_INET;
   address_sock.sin_port = htons(port);
   address_sock.sin_addr.s_addr = htonl(INADDR_ANY);
   int r = bind(sock, (struct sockaddr *)&address_sock, sizeof(struct sockaddr_in));
   if (r == 0){
       r = listen(sock, 0);
       while (1){
           struct sockaddr_in caller;
           socklen_t size = sizeof(caller);
           int *sock2 = (int *)malloc(sizeof(int));
           *sock2 = accept(sock, (struct sockaddr *)&caller, &size);
           if (*sock2 >= 0){
               printf("Nouvelle connexion\n");
               pthread_t th;
               int r2 = pthread_create(&th, NULL, communication, sock2);
               if (r2 != 0){
                   printf("Probleme de création de thread");
                   exit(0);
               }}
       }}
   else{
       printf("Probleme de bind\n");
```

```
    }
    return 0;}
```

Exemple avec valeur de retour

```
void *affiche(void *arg){
    char *s = (char *)arg; //on peut le faire car on sait qu'on va lui filer une string
    printf("Message = %s\n", s);
    char *retour = (char *)malloc(100 * sizeof(char));
    sprintf(retour, "%s", "il a tapé un sprint... lol\n");
    return retour;
    //équivaut à : pthread_exit(retour)
}


//creation du thread
pthread_t th1;
char *s1 = "dasso?";
int r1 = pthread_create(&th1, NULL, affiche, s1);

//on attend la fin du thread et on récupère le résultat
char *res1;
pthread_join(th1, (void **)&res1);
```

Serveur qui fait des entrées / sorties non-bloquantes avec select & laisse les pool

Exemple avec select

```
    int sock1 = socket(PF_INET, SOCK_DGRAM, 0);
    struct sockaddr_in address_sock1;
    address_sock1.sin_family = AF_INET;
    address_sock1.sin_port = htons(5555);
    address_sock1.sin_addr.s_addr = htonl(INADDR_ANY);
    int sock2 = socket(PF_INET, SOCK_DGRAM, 0);
    struct sockaddr_in address_sock2;
    address_sock2.sin_family = AF_INET;
    address_sock2.sin_port = htons(5556);
    address_sock2.sin_addr.s_addr = htonl(INADDR_ANY);
    int r = bind(sock1, (struct sockaddr *)&address_sock1, sizeof(struct sockaddr_in));
    if (r == 0){
        int r2 = bind(sock2, (struct sockaddr *)&address_sock2, sizeof(struct sockaddr_in));
        if (r2 == 0){
            fcntl(sock1, F_SETFL, O_NONBLOCK);
            fcntl(sock2, F_SETFL, O_NONBLOCK);
            fd_set initial;
            int fd_max = 0;
            FD_ZERO(&initial);
            FD_SET(sock1, &initial);
            if (fd_max < sock1){
                fd_max = sock1; }
            FD_SET(sock2, &initial);
            if (fd_max < sock2){
                fd_max = sock2;}
            char tampon[100];
            int rec1 = 0;
            int rec2 = 0;
            while (1){
```

```
            fd_set rdfs;
            FD_COPY(&initial, &rdfs);
            int ret = select(fd_max + 1, &rdfs, NULL, NULL, NULL);
            while (ret > 0){
                if (FD_ISSET(sock1, &rdfs)){
                    rec1 = recv(sock1, tampon, 100, 0);
                    printf("Taille de données reçues %d\n", rec1);
                    if (rec1 >= 0){
                        tampon[rec1] = '\0';
                        printf("Message recu : %s\n", tampon);}
                    ret--;}
                if (FD_ISSET(sock2, &rdfs)) {
                    rec2 = recv(sock2, tampon, 100, 0);
                    printf("Taille de données reçues %d\n", rec2);
                    if (rec2 >= 0) {
                        tampon[rec2] = '\0';
                        printf("Message recu : %s\n", tampon);}
                    ret--;
} } } } }
```

Exemple avec poll (quand on l'utilise on parle de poll-emploi)

```
int sock1 = socket(PF_INET, SOCK_DGRAM, 0);
struct sockaddr_in address_sock1;
address_sock1.sin_family = AF_INET;
address_sock1.sin_port = htons(5555);
address_sock1.sin_addr.s_addr = htonl(INADDR_ANY);
int sock2 = socket(PF_INET, SOCK_DGRAM, 0);
struct sockaddr_in address_sock2;
address_sock2.sin_family = AF_INET;
address_sock2.sin_port = htons(5556);
address_sock2.sin_addr.s_addr = htonl(INADDR_ANY);
int r = bind(sock1, (struct sockaddr *)&address_sock1, sizeof(struct sockaddr_in));
if (r == 0){
    int r2 = bind(sock2, (struct sockaddr *)&address_sock2, sizeof(struct sockaddr_in));
    if (r2 == 0) {
        fcntl(sock1, F_SETFL, O_NONBLOCK);
        fcntl(sock2, F_SETFL, O_NONBLOCK);
        struct pollfd p[2];
        p[0].fd = sock1;
        p[0].events = POLLIN;
        p[1].fd = sock2;
        p[1].events = POLLIN;
        char tampon[100];
        int rec1 = 0;
        int i;
        while (1)  {
            int ret = poll(p, 2, -1);
            if (ret > 0){
                for (i = 0; i < 2; i++){
                    if (p[i].revents == POLLIN){
                        rec1 = recv(p[i].fd, tampon, 100, 0);
                        printf("Taille de données reçues %d\n", rec1);
                        if (rec1 >= 0){
                            tampon[rec1] = '\0';
                            printf("Message recu : %s\n", tampon);
} } } } } } }
```

Cas concret UDP

Couple Client / Serveur qui s'envoient des paquets (sans doute des nudes…)

- Serveur

```
int sock = socket(PF_INET, SOCK_DGRAM, 0);
sock = socket(PF_INET, SOCK_DGRAM, 0);
struct sockaddr_in address_sock;
address_sock.sin_family = AF_INET;
address_sock.sin_port = htons(5555);
address_sock.sin_addr.s_addr = htonl(INADDR_ANY);
int r = bind(sock, (struct sockaddr *)&address_sock, sizeof(struct sockaddr_in));
struct sockaddr_in emet;
socklen_t a = sizeof(emet);
if (r == 0){
    char tampon[100];
    while (1){
        int rec = recvfrom(sock, tampon, 100, 0, (struct sockaddr *)&emet, &a);
        tampon[rec] = '\0';
        printf("Message recu : %s\n", tampon);
        printf("Port de l'emetteur: %d\n", ntohs(emet.sin_port));
        printf("Adresse de l'emetteur: %s\n", inet_ntoa(emet.sin_addr));
    }}
```

- Client

```
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("localhost", "5555", &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));
           }}}
```

Alors petite intro avant cette partie Broadcast / Multicast : qu'est-ce que c'est ?

La diffusion intégrale ou "broadcast", s'effectue en direction de toutes les machines d'un réseau donné. De son côté la multi-diffusion "multicast", est dirigée vers un groupe de machines qui se sont abonnées au préalable.

Pour le broadcast, on désigne un alias : "on envoie sur la dernière adresse possible du réseau", qui correspond donc à l'adresse IP 255.255.255.255. Il n'y a pas de routage pour ce mode, et l'envoi est limité au réseau local uniquement. On peut quand même envoyer un message en broadcast à un AUTRE réseau local, mais il faut alors connaître son adresse de broadcast. Comment cela se fesse ?

Et bien c'est simple : en réalité, les réseaux locaux viennent avec un mask, qui précise les bits des adresses de ce réseau correspondant à l'adresse du réseau. Par exemple, 127.50.24.0/24 veut dire que les 24 premiers bits correspondent à l'adresse du réseau. Quel rapport ? Et bien adresse broadcast = adresse du réseau + 1 sur les bits qui restent !

Exemple : pour 127.50.24.0/24, c'est 127.50.24.255 et si c'était 127.50.24.0/23 car serait 127.50.25.255 :)

Pour envoyer, on précise un numéro de port aussi, et du coup pour recevoir on a juste à écouter sur ce port.

Pour le multicast, on va devoir s'intéresser aux différentes classes d'adresse IP, car c'est la classe D qui est utilisée. Il s'agit des adresses allant de 224.0.0.0 à 239.255.255.255 : les 4 bits de gauche décrivent le réseau (= 1110) et les bits restants décrivent les machines de ce réseau. Attention, certaines adresses sont réservées et inutilisables : ce sont celles commençant par 224, 232, 233 ou 239 (pas touche donc).

Voilà voilà, paye l'intro assez fatou mais bon.

Serveur qui broadcast des trucs (sans doute des nudes...)

Les envois broadcast

```
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", NULL, &first_info);
  if (r == 0){
      if (first_info != NULL){
          struct sockaddr *saddr = first_info->ai_addr;
          char tmp[100];
          int i = 0;
          for (i = 0; i <= 10; i++){
              strcpy(tmp, "MESSAGE ");
              char entier[3];
              sprintf(entier, "%d", i);
              strcat(tmp, entier);
              sendto(sock, tmp, strlen(tmp), 0, saddr, (socklen_t)sizeof(struct sockaddr_in));
            }
        }}}
```

Les réceptions broadcast

```
int sock = socket(PF_INET, SOCK_DGRAM, 0);
sock = socket(PF_INET, SOCK_DGRAM, 0);
struct sockaddr_in address_sock;
address_sock.sin_family = AF_INET;
address_sock.sin_port = htons(8888);
address_sock.sin_addr.s_addr = htonl(INADDR_ANY);
int r = bind(sock, (struct sockaddr *)&address_sock, sizeof(struct sockaddr_in));
if (r == 0){
    char tampon[100];
    while (1){
        int rec = recv(sock, tampon, 100, 0);
        tampon[rec] = '\0';
        printf("Message recu : %s\n", tampon);}}
```

Les envois multi-cast

```
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", NULL, &first_info);
if (r == ){
    if (first_info != NULL){
        struct sockaddr *saddr = first_info->ai_addr;
        char tmp[100];
        int i = 0;
        for (i = 0; i <= 10; i++){
            strcpy(tmp, "MESSAGE ");
            char entier[3];
            sprintf(entier, "%d", i);
            strcat(tmp, entier);
            sendto(sock, tmp, strlen(tmp), 0, saddr, (socklen_t)sizeof(struct sockaddr_in));
        }}}
```

Les réceptions multi-cast (ne pas oublier de s'abonner, lâcher un pouce bleu, tout ça tout ça)

```
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);
}
```

Exemples en Java

Communication TCP : exemple typique

Server

```
public class ServeurHi {
   public static void main(String[] args) {
       try {
           ServerSocket server = new ServerSocket(4242);
           while (true) {
               Socket socket = server.accept();
               BufferedReader br =
                         new BufferedReader(
                               new InputStreamReader(socket.getInputStream()));
               PrintWriter pw =
                         new PrintWriter(
                               new OutputStreamWriter(socket.getOutputStream()));
```

```
                pw.print("HI\n");
                pw.flush();
                String mess = br.readLine();
                System.out.println("Message recu :" + mess);
                br.close();
                pw.close();
                socket.close();
            }
        } catch (Exception e) {
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Serveur concurrent

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

    public ServiceHi(Socket s) {
        this.socket = s;
    }
    public void run() {
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
           PrintWriter pw = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
            pw.print("HI\n");
            pw.flush();
            String mess = br.readLine();
            System.out.println("Message recu :" + mess);
            br.close();
            pw.close();
            socket.close();
        } catch (Exception e) {
            System.out.println(e);
            e.printStackTrace();}}}
public class ServeurHiConcur {
    public static void main(String[] args) {
        try {
            ServerSocket server = new ServerSocket(4242);
            while (true) {
                Socket socket = server.accept();
                ServiceHi serv = new ServiceHi(socket);
                Thread t = new Thread(serv);
                t.start();
            }
        } catch (Exception e) {
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Client

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

```
        try {
            Socket socket = new Socket("lulu", 4242);
            BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter pw = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
            String mess = br.readLine();
            System.out.println("Message recu du serveur :" + mess);
            pw.print("HALLO\n");
            pw.flush();
            pw.close();
            br.close();
            socket.close();
        } catch (Exception e) {
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Communication UDP : exemple typique

Client

```
public class ReceiveUDPPlus3 {
    public static void main(String[] args) {
        try {
            DatagramSocket dso = new DatagramSocket(5555);
            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 reÃ§u :" + st);
                InetSocketAddress ia = (InetSocketAddress) paquet.getSocketAddress();
                System.out.println("De la machine " + ia.getHostName());
               System.out.println("Depuis le port " + ia.getPort());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}


public class EnvoiUDP2 {
    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("localhost", 5555);
                DatagramPacket paquet = new DatagramPacket(data, data.length, ia);
                dso.send(paquet);
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

Broadcast et Multicast, aka le piaf qui fait chier

Envoi Broadcast

```
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();} } }
```

Réception Broadcast

```
public class RecoitBroadcast {
    public  static void main(String[] args){
        try{
            DatagramSocket dso=new DatagramSocket(8888);
            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);
            }
        } catch(Exception e){
            e.printStackTrace();} } }
```

Envoi Multicast

```
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 Multicast

```
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 reÃ§u :"+st);
            }
        } catch(Exception e){
            e.printStackTrace();} } }
```

Autres trucs qu'on espère ne vont pas tomber à l'examen mais on sait jamais

Sérialisation :

Comment envoyer une entité à travers le réseau de manière fiable et standardisée ? La céréale y sait
L'objet doit implémenter l'interface Serializable, et après on peut la convertir en XML / JSON / YAML…
La linéarisation n'est pas forcément que réflexive d'ailleurs. Les champs d'un objet sérialisable doivent aussi l'être, on ne peut pas tout sérialiser (exemple les objets dynamiques) et il doit il y avoir un constructeur vide + les getters / setters adaptés.

Pour encoder et envoyer :

```
Joueur j1 = new Joueur("Alice",12);
Joueur j2 = new Joueur("Bob",14);
FileOutputStream fo = new FileOutputStream("Joueurs.xml");
XMLEncoder xe = new XMLEncoder(fo);
xe.writeObject(j1);
xe.writeObject(j2);
xe.close();
fo.close();
//(le tout dans un try-catch)
```

Pour récupérer et traduire :

```
Joueur j = null;
FileInputStream fi = new FileInputStream("Joueurs.xml");
XMLDecoder xd = new XMLDecoder(fi);
try{
        j = (Joueur)xd.readObject();
        while(true){
                System.out.println(j.toString());
                j=(Joueur)xd.readObject();
        }
}catch(ArrayIndexOutOfBoundsException aie){
        ouille
}
xd.close();
```

```
fi.close();
//(le tout dans un try-catch)
```

ET EN FAIT NON : on va utiliser ça plutôt. D'ailleurs, mot clé `transient` pour snober l'attribut (ce dernier sera mis à null à la lecture).

```
ObjectOutputStream (void writeObject(Object o));
ObjectInputStream (Object readObject());
```

Les channels 5, ou "les entrées-sorties non-bloquantes en JAVA :

On va utiliser ici la librairie "non blocking in-outs" java.nio, et on va utiliser des canaux. Le principe est d'attendre en même temps sur un ensemble de ressources -> gain en efficacité. On utilise 2 méthodes isOpen() et close().

Socket remplacée par SocketChannel, DatagramSocket par DatagramChannel et ServerSocket par SSChannel.

On utilise les canaux comme un ordonnanceur (je trouve) : on crée un canal non-bloquant pour chaque chose que l'on veut faire, puis on les enregistre dans un sélecteur en indiquant les opérations que l'on veut surveiller. Ce dernier nous prévient lorsqu'une opération est disponible puis on les réalise. En fait en bloque, mais en même temps ? xD Bref

On va utiliser la classe Selector(), avec la méthode register avec (canal, opération).

Cette dernière renvoit un clef appartenant à SelectionKey :

`OP_ACCEPT` `OP_CONNECT` `OP_READ` `OP_WRITE` (combinables grace à l'addition :)

On attend ensuite avec select() bloquant ou int selectNow() qui renvoit le nombre d'op possibles à la sortie de la sélection, on peut récupérer les clefs correspondantes grâce à :

```
abstract Set<SelectionKey> selectedKeys()
```

il faut penser à RETIRER LA CLEF AVEC REMOVE quand on la traite ensuite, on teste quelle opération est possible avec is[OPERATION] genre isReadable() pour finir, on récupère le channel correspondant avec channel().

```
import java.nio.channels.*; import java.util.Iterator; import java.nio.ByteBuffer; import java.net.InetSocketAddress;
public class DoubleRead{
    public static void main(String[]args){
        try{
            //on commence par créer un selector
            Selector selector = Selector.open();
            //ensuite, on crée nos deux channels UPD
            DatagramChannel datachan1 = DatagramChannel.open();
            DatagramChannel datachan2 = DatagramChannel.open();
            //on précise qu'elles sont non-bloquantes
            datachan1.configureBlocking(false);
            datachan2.configureBlocking(false);
            //on les bind à des ports
            datachan1.bind(new InetSocketAddress(4233));
            datachan2.bind(new InetSocketAddress(4234));
            //on les enregistre auprès du selecteur
            datachan1.register(selector, SelectionKey.OP_READ);
            datachan2.register(selector, SelectionKey.OP_READ);
            //on déclare un buffer d'octets pour lecture/écriture
            ByteBuffer buffer = ByteBuffer.allocate(100);
            //ensuite, on fait une boucle
            while(true){
                System.out.println("Waiting for messages");
                //on fait un select bloquant (pour attendre qu'on reçoive nos paquets)
                selector.select();
                //on a besoin de notre itérateur pour les clefs
```

```
                Iterator<SelectionKey> iterator = selector.selectedKeys().iterator();
                //on itère ensuite dessus
                while(iterator.hasNext()){
                    SelectionKey selectedKey = it.next();
                    iterator.remove();
                    //on a ensuite 2 options + 1 erreur xD
                    if(selectedKey.isReadable() && selectedKey.channel()==datachan1){
                        System.out.println("Message UPD reçu sur le port 4233");
                        datachan1.receive(buff);
                        String mess = new String(buff.array(),0,buff.array().length);
                        buff.clear();
                        System.out.println("Message reçu = "+mess);
                    }
                    else if (selectedKey.isReadable() && selectedKey.channel()==datachan2){
                        System.out.println("Message UPD reçu sur le port 4234");
                        datachan2.receive(buff);
                        String mess = new String(buff.array(),0,buff.array().length);
                        buff.clear();
                        System.out.println("Message reçu = "+mess);
                    }
                    else{
                        System.out.println("What the hell ?");
                    }
                }
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Notes et tips :

Rappels :

- la représentation des entiers pouvant varier d'une machine à une autre, on standardise au big endian pour la représentation réseau (htons pour host-to-network, et ntohs sinon)
- pour la gestion de la concurrence en C, on demande le verrou ALAMANO (terme technique).

```
pthread_mutex_t lock;
lock = PTHREAD_MUTEX_INITIALIZER ;
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- précisions sur la fonction select :

```
int select(
int numfds,  // max fd we wanna track + 1
fd_set *readfds,   // the set of fd we wanna read from
fd_set *writefds, // the set of fd we wanna write on
fd_set *exceptfds, // following the exceptionnal conditions
struct timeval *timeout // timeout, NULL for none
);
struct timeval {
int tv_sec;  // seconds
int tv_usec; // microseconds
};
FD_SET(int fd, fd_set *set)    // ajoute fd à l'ensemble
FD_CLR(int fd, fd_set *set)    // enlève fd de l'ensemble
```

```
FD_ISSET(int fd, fd_set *set)  // renvoie vraie si fd est dans l'ensemble.
FD_ZERO(fd_set *set)           // efface tous les éléments de l'ensemble
FD_COPY(fd_set *orig, fd_set *copy) // copie orig dans copy
```

warning : les fd_sets sont modifiés, on doit ré-initialiser avant un prochain appel à select.

- précisions sur la fonction poll :

```
// événements possibles
// POLLIN : lecture ou accept
// POLLOUT : écriture
// POLLPRI : lecture prioritaire
// POLLHUP : déconnexion
// POLLERR : erreur
```

- la concurrence en java, c'était comment déjà ?

1) code synchronized : `truc synchronized retour nom(trucs){}`
2) demander le verrou juste pour un piti block : `synchronized(this){}`
3) relâcher le verrou et attendre d'être notifyé : `wait()`
4) notifyé le/les threads en attente : `notify() || notifyAll()`

Signé : Un mec qui *s'en fiche* :)
(mentions spéciales : nutnut pour les rappels)