

# Examen 2021

## Contexte

On désire développer une architecture sous forme d'arbre binaire où chaque entité du réseau a au plus deux fils et exactement un père sauf l'entité racine qui elle n'a pas de père. A tout moment des entités peuvent s'ajouter au réseau en faisant la demande aux noeuds n'ayant pas deux fils déjà attribués. On peut de plus envoyer un message à un noeud qui le transmet alors à ses fils qui eux-mêmes le transmettront à leurs fils, etc. Nous ne traiterons pas dans ce sujet les demandes de déconnexion du réseau. Les noeuds écoutent les messages qui viennent de leur père sur le port UDP 1234. Les noeuds communiquent sur le port TCP 5678 pour la communication venant de l'extérieur de l'arbre. Un noeud peut également envoyer des messages vers l'extérieur en multi-diffusion en utilisant l'adresse de multi-diffusion 233.222.222.1 et le port 4242. Nous détaillons maintenant les différentes fonctionnalités de notre application réseau.

### La construction de l'arbre

Lorsqu'une entité souhaite devenir le fils d'un noeud n de l'arbre, elle se connecte en TCP sur le port 5678 de ce noeud et elle envoie un message [CONNECT\n]. Si le noeud n a déjà deux fils ('c'est à dire qu'il a déjà accepté deux demandes d'insertion), il répond par le message [FULL\n], sinon il répond [OKSON\n] pour signaler que cette entité est maintenant enregistrée comme l'un de ses deux fils. Après avoir reçu [FULL\n] ou [OKSON\n], l'entité ayant fait la requête ferme la connexion et est prête à recevoir les messages de son père sur son port UDP (si l'insertion a été acceptée).

Une entité extérieure peut demander également l'adresse d'un noeud ayant un fils libre. Pour cela, elle se connecte à la racine de l'arbre sur le port TCP et elle envoie le message [LEAF\n] et se déconnecte. Si la racine a un fils libre, elle attend une seconde (avec un sleep) et elle envoie par multi-diffusion (à l'IP 233.222.222.1 et au port 4242), le message [L ip] de taille au plus 21 octets où ip est une chaîne de caractères correspondant à son adresse IP. Si la racine n'a pas de fils libre, après un délai d'une seconde, elle envoie par UDP à ses deux fils le message [FREE], qui suivent alors le même comportement: selon si ils ont ou n'ont pas de fils libre, dans le premier cas ils multi-diffusent un message [L ip] (avec leur adresse IP) et dans le deuxième cas, ils envoient le message [FREE] à leurs fils, etc

### La diffusion de messages dans l'arbre

Pour diffuser un message dans l'arbre, un client extérieur peut se connecter au port TCP d'un noeud et il lui envoie un message de la forme [M mess\n] où mess est une chaîne de caractères à transmettre (qui ne contient pas de caractères \n et qui fait au plus 500 caractères). A la réception d'un tel message, le noeud de l'arbre le traite puis ferme la connexion TCP. Ensuite il transmet sur le port UDP 1234 de ses fils (si il en a) le message [T mess]. Lorsqu'un noeud de l'arbre reçoit sur son port UDP 1234 un message de la forme [T mess]. Il envoie à ces fils (si il en a) sur leur port UDP le message [T mess].

## Programmation en C

On commence par supposer qu'un tel arbre existe déjà et que sa racine se trouve sur la machine [machine1.progresseaux.fr](#) qui est accessible depuis votre machine.

1. Écrire en C un client qui transmet à l'arbre le message Hello Worldl.
2. Écrire en C une fonction requestLeaf qui se connecte en TCP à la racine de l'arbre et envoie le message [LEAF\n] et se déconnecte.
3. Écrire en C une fonction getLeaf qui attend un message [L ip] émis par multi-diffusion sur l'IP 233.222.222.1 et sur le port 4242 et, quand elle l'a reçu, renvoie la chaîne de caractères contenant l'adresse IP ip.
4. Écrire une fonction nextLeaf qui se connecte en TCP à une adresse IP (donnée sous-forme de chaîne de caractères en arguments de la fonction) sur le port 5678 et envoie le message [CONNECT\n]. Elle attend ensuite une réponse et renvoie l'entier 0 si elle reçoit un message [OKSON\n] et -1 si elle reçoit une autre réponse. Ensuite la fonction se déconnecte.
5. Écrire un programme C (i.e. une fonction main) qui se rajoute comme feuille de l'arbre dont la racine se trouve sur la machine [machine1.progresseaux.fr](#) et affiche un message sur le terminal de la forme l'am leaf quand cela est fait. Attention : certains programmes peuvent faire la même demande en parallèle et donc il n'est pas garanti, que entre le moment où l'on demande un noeud de l'arbre avec un fils libre (avec le message [LEAF\n]) et le moment où l'on essaie de se connecter à ce noeud le noeud renvoie [OKSON\n]. Entre temps, les noeuds avec un fils de libre peuvent avoir changé. Il faut donc traiter ce cas quitte à demander plusieurs fois de connaître les noeuds avec un fils de libre. En revanche, on ne vout demande pas de traiter les messages UDP qui arrivent de l'arbre une fois le programme ajouté comme feuille.

```
//Q1
int main() {
    struct addrinfo *first_info;
    struct addrinfo hints;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET;
    int r=getaddrinfo("machine1.progresseaux.fr", "5678", &hints, &first_info);
    if(r==0){
        struct addrinfo *info=first_info;
        int found=0;
        struct sockaddr_in *addressin;
        while(info!=NULL && found==0){
            addressin=(struct sockaddr_in *)info->ai_addr;
            found=1;
            info=info->ai_next;
        }
        if(found==1){
            int descr=socket(PF_INET, SOCK_STREAM, 0);
            int r2=connect(descr, (struct sockaddr *)addressin,
                           sizeof(struct sockaddr_in));
            if(r2!=-1){
                char *mess="M Hello Worldl\n";
                send(descr, mess, strlen(mess), 0);
                close(descr);
            }
        }
        return 0;
    }
}

//Q2.
int requestLeaf() {
    struct addrinfo *first_info;
    struct addrinfo hints;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET;
    int r=getaddrinfo("machine1.progresseaux.fr", "5678", &hints, &first_info);
    if(r==0){
        struct addrinfo *info=first_info;
        int found=0;
        struct sockaddr *saddr;
        struct sockaddr_in *addressin;
        while(info!=NULL && found==0){
            addressin=(struct sockaddr_in *)info->ai_addr;
            found=1;
            info=info->ai_next;
        }
        if(found==1){
            int descr=socket(PF_INET, SOCK_STREAM, 0);
            int r2=connect(descr, (struct sockaddr *)addressin,
                           sizeof(struct sockaddr_in));
            if(r2!=-1){
                char *mess="LEAF\n";
                send(descr, mess, strlen(mess), 0);
                close(descr);
            }
        }
        return 0;
    }
}

//Q3.
char* getLeaf() {
    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(4242);
```

1. Écrire le constructeur de la classe Root qui remplit le champ myIp avec l'adresse IP de la machine sur laquelle on exécute cette méthode et met les adresses IP des deux fils à null.
2. Écrire une méthode getIP qui prend en argument une socket TCP et renvoie la chaîne de caractères correspondant à l'IP de la machine connectée à l'autre bout de la socket.
3. Écrire une méthode setSon qui prend en argument une socket TCP et qui met l'adresse IP de la machine connectée à l'autre bout dans ipSon1 ou ipSon2 si l'un des deux vaut null et renvoie true si c'est le cas, et false si les deux champs ipSon1 et ipSon2 sont différents de null.
4. Écrire une méthode transMess qui prend comme arguments une chaîne de caractères mess et envoie en UDP le message [T mess] aux fils de cette racine (si elle en a).
5. Écrire une méthode treatLeaf qui ne prend pas d'argument et qui envoie en multidiffusion le message [L ip] sur l'IP et le port donné de multi-diffusion de l'arbre où ip est l'adresse IP du noeud si la racine a encore un fils libre et sinon elle envoie en UDP le message [FREE] aux fils de cette racine.
6. Écrire une méthode treatCPmess qui prend en arguments une socket TCP et une chaîne de caractères contenant un message venant de la communication TCP et traite le message selon si il s'agit d'un message de la forme [M mess\n] ou [CONNECT\n] ou [LEAF\n]. Cette méthode ne renverra rien

```
address_sock.sin_addr.s_addr=htonl(INADDR_ANY);
rbind(sock, (struct sockaddr *)&address_sock,sizeof(struct sockaddr_in));
struct ip_mreq mreq;
mreq.imr_multiaddr.s_addr=inet_addr("233.222.222.1");
mreq.imr_interface.s_addr=htonl(INADDR_ANY);
r=setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
char tampon[100];
int rec=recv(sock, tampon, 100, 0);
tampon[rec]='\0';
return rec+2;
}
return 0;
}

//Q4.
int nextLeaf(char *ip){
    struct sockaddr_in address_sock;
    address_sock.sin_family = AF_INET;
    address_sock.sin_port = htons(5678);
    inet_aton(ip, &address_sock.sin_addr);
    int descr=socket(PF_INET, SOCK_STREAM, 0);
    int r=connect(descr, (struct sockaddr *)&address_sock,
                sizeof(struct sockaddr_in));
    if(r!=-1){
        char *mess="CONNECT\n";
        send(descr, mess, strlen(mess), 0);
        char buff[100];
        int size_rec=recv(descr, buff, 99*sizeof(char), 0);
        buff[size_rec]='\0';
        if(strcmp(rec, "OKSON\n")==0){
            close(descr);
            return 0;
        }
        else{
            close(descr);
            return -1;
        }
        close(descr);
    }
    return -1;
}

//Q5.
int main() {
    int found=0;
    while(found==0){
        requestLeaf();
        char *ip=getLeaf();
        if(nextLeaf(ip)==0){
            found=1;
        }
    }
    printf("I am leaf\n");
}
```

## Programmation en java

Nous créons d'abord la classe Root qui correspond au noeud racine d'un arbre. Cette classe a un champ myIp pour stocker son adresse IP, deux champs ipSon1 et ipSon2 correspondant aux IP des deux fils du noeud. Nous rappelons qu'un noeud racine ne reçoit pas de messages UDP car il n'a pas de père

```
class Root {
    public String myIp ;
    public String ipSon1 ;
    public String ipSon2 ;
    // A completer
}
```

```

//Q6
public Root(){
    ipSon1=null;
    ipSon2=null;
    myIp=InetAddress.getLocalHost().getHostAddress();
}

//Q7
public String getIp(Socket s){
    return s.getInetAddress().getHostAddress();
}

//Q8
public boolean setSon(Socket s){
    if(ipSon1==null){
        ipSon1=getIp(s);
        return true;
    } else if(ipSon2==null){
        ipSon2=getIp(s);
        return true;
    }else{
        return false;
    }
}

//Q9
public void transMess(String mess){
    try{
        DatagramSocket dso=new DatagramSocket();
        byte[]data;
        String s="T "+mess;
        data=s.getBytes();
        if(ipSon1==null){
            InetAddress ia=new InetAddress(ipSon1,1234);
            DatagramPacket paquet=
                new DatagramPacket(data,data.length,ia);
            dso.send(paquet);
        }
        if(ipSon2==null){
            InetAddress ia=new InetAddress(ipSon2,1234);
            DatagramPacket paquet=
                new DatagramPacket(data,data.length,ia);
            dso.send(paquet);
        }
    } catch(Exception e){
        e.printStackTrace();
    }
}

//Q10
public void treatLeaf(){
    if(ipSon1==null || ipSon2==null){
        DatagramSocket dso=new DatagramSocket();
        byte[]data;
        String s="L "+myIp;
        data=s.getBytes();
        InetAddress ia=new InetAddress("233.222.222.1",4242);
        DatagramPacket paquet=new
            DatagramPacket(data,data.length,ia);
        dso.send(paquet);
    } else{
        DatagramSocket dso=new DatagramSocket();
        byte[]data;
        String s="FREE";
        InetAddress ia=new InetAddress(ipSon1,1234);
        DatagramPacket paquet=new DatagramPacket(data,data.length,ia);
        dso.send(paquet);
        ia=new InetAddress(ipSon2,1234);
        paquet=new DatagramPacket(data,data.length,ia);
        dso.send(paquet);
    }
}

```

```

    }
}

//Q11
public void treatCPmess(Socket s,String mess){
    if(mess.equals("LEAF")){
        Thread.sleep(1000);
        treatLeaf();
    }else if(mess.equals("CONNECT\n")){
        PrintWriter pw=new PrintWriter(new OutputStream(s.getOutputStream()));
        if(setSon(s)){
            pw.print("OKSON\n");
            pw.flush();
        }else{
            pw.print("FULL\n");
            pw.flush();
        }
    }else{
        String m=mess.substring(2,mess.length());
        transMess(m);
    }
}

```

Nous allons maintenant créer la classe TCPService qui contiendra la méthode en charge de la communication TCP du noeud racine. Cette classe a deux champs, un champ ro contenant une référence vers l'objet Root associé au service et un champ sock contenant une référence vers la socket TCP.

```

class TCPService implements Runnable {
    public Root ro ;
    public Socket sock ;
    public TCPService ( Root _ro , Socket _so ){
        this . ro = _ro ;
        this . sock = _so ;
    }
    // A completer
}

```

1. Écrire la méthode run de la classe TCPService. Cette méthode est appelée après une connexion à la racine et elle est responsable de la communication TCP entre l'entité extérieure et le noeud racine.
2. Écrire une classe ProgRoot qui contient un programme principal (c'est-à-dire une méthode main) pour un noeud racine . Ce programme crée tout d'abord un objet Root et attend ensuite en boucle des connexions sur le port TCP et pour chaque connexion lance un thread pour gérer la communication.
3. Quel problème pourrait-il se passer si deux clients demandent à être successeur du noeud racine « en même temps » ? Comment éviter ce problème en modifiant votre code ?

```
//Q12.
public void run(){
    BufferedReader br=new BufferedReader(
        new InputStreamReader(sock.getInputStream()));
    String mess=br.readLine();
    ro.treatTCPmess(mess+"\n");
    br.close();
    sock.close();
}

//Q13
public static void main(String[] argv){
    Root ro=new Root();
    try{
        ServerSocket server=new ServerSocket(5678);
        while(true){
            Socket socket=server.accept();
            TCPService serv=new TCPService(ro,socket);
            Thread t=new Thread(serv);
            t.start();
        }
    } catch(Exception e){
        System.out.println(e);
        e.printStackTrace();
    }
}

//Q14

/*Le nœud racine pourrait leur répondre aux deux qu'ils sont fils1
À cause de problème de concurrence (si par exemple la méthode setSon est appelée en même temps).
Une solution est de mettre toutes les méthodes accédant à ipSon1 et ipSon2 en synchronized.**/
```