Module ITC313 - Informatique

Partie C / C++

TP7 + TP8 programmation / communication socket

Benoît Darties - benoit.darties@u-bourgogne.fr
Université de Bourgogne

Année universitaire 2016-2017

---

1 Communication distante en utilisant l'outil `netcat`

Les exercice de cette question visent à étendre les outils de communication à des processus situés sur des machines distinctes. Nous verrons dans un premier temps un modèle de communication entre processus distants directement via un outil nommé ***`netcat`****, puis écrirons des programme en C qui utilisent les modes de communication vus en cours. Dans chacun des exercices, le processus à l'origine de la demande de communication est le client, tandis que le processus qui reçoit une demande de communication est le serveur. Ceci ne veut pas pour autant dire que la communication est forcément dans le sens client vers serveur. Cette communication peut s'effectuer de serveur vers client, ou alterner les deux. Il est important de bien distinguer ces notions*

prérequis : pour réaliser correctement les exercices de cette section, il sera nécessaire d'identifier les machines par leur adresse ip. Il est possible d'obtenir l'adresse ip d'une machine en exécutant la commande `ifconfig` , et en recherchant la ligne correspondante à une adresse ip. Dans l'exemple suivant :

```
1 $ ifconfig
2 ...
3 eth0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST>
4 inet 172.20.10.3 netmask 0xfffffff0 broadcast 172.20.10.15
5 ..
```

l'adresse ip de la machine est `172.20.10.3`

Exercice 1 : *Découverte de la commande* ***`nc : netcat`***

L'objectif de cet exercice est de vous familiariser avec une commande puissante appelée **`nc`***. Cette commande, inspirée de la commande* **`cat`***, permet d'afficher et de recevoir un flux d'octets, non pas à l'écran et depuis le clavier, mais sur ou depuis le réseau, et plus précisément vers un processus distant identifié par une adresse ip un numéro de port et un mode (connecté - tcp - ou non connecté -udp -)*

Nous allons manipuler les commandes de base de `netcat`. L'idée derrière cet exercice est de savoir manipuler des outils fiables qui nous permettront de tester par la suite les programmes que nous allons développer. En particulier, il permettra de déterminer plus rapidement qu'elle est l'origine d'un problème de communication : le processus client, le processus serveur ou les deux.

1. Ouvrez deux terminaux côte à côte. Pour des raisons de compréhension, le premier terminal sera appelé le *terminal serveur* et le second le terminal *client*. Changez les repertoires de travail pour que terminaux client et serveur travaillent dans des répertoires différents.
2. Etablissement d'une connexion en mode connecté - tcp -
   (a) Dans le terminal serveur, lancez la commande `nc -l 3000`. Cette commande crée un processus serveur qui se met en écoute de tout ce qui arrive sur le port passé en paramètre, ici 3000, en mode connecté - tcp -, et affiche à l'écran tout ce qu'il reçoit.
   (b) Dans le terminal client, lancez la commande `nc localhost 3000`.[1]. Cette commande crée un processus client qui se connecte à un processus serveur présent sur la machine ou adresse ip passée en premier paramètre, ici la machine locale - localhost- , associé au port passé en second paramètre, ici 3000, en mode connecté - tcp -. Si la connexion est établie, la commande `nc` se met en attente, écoute ce qui est saisi sur l'entrée standard - ici le clavier - et envoie tout au processus serveur. Si la commande échoue (par exemple, s'il n'y a pas de serveur associé à l'adresse ip et au port demandé), elle se termine directement. Tapez quelques messages dans le terminal client en validant sur entrée à chaque fois, et observez le résultat dans le terminal serveur. Terminez vote saisie avec `ctrl+d` pour envoyer le caractère `eof` et mettre fin aux deux processus.
   (c) Relancez la commande `nc -l 3000` sur le terminal serveur. Sur le terminal client, exécutez la commande `echo "bonjour" |nc localhost 3000`. Constatez que le message est transféré directement au serveur.
   (d) Relancez la commande `nc -l 3000` sur le terminal serveur, et sur le terminal client la commande `nc localhost 3000` , mais cette fois écrivez quelque chose dans le terminal du serveur et validez. Voyez que le message apparait sur la console du client. Tapez maintenant quelques caractères dans le terminal client en validant ; Voyez que communication, bien qu'initiée depuis le client, est bidirectionnelle.
3. Etablissement d'une connexion en mode non connecté - udp -
   (a) Refaites les manipulations précédentes en utilisant les mêmes commandes, mais en y rajoutant l'option `-u`. Cette option permet d'établir une communication non pas en mode connecté - tcp - mais en mode non connecté - udp -. Notez qu'ne différence de comportement existe lorsque le message est envoyé d'abord depuis le serveur vers le client. Pouvez-vous expliquer cette différence ? Comment expliquer alors que si le client envoie d'abord un message, alors le serveur peut lui répondre. Effectuez plusieurs tests, puis dresser le mode de fonctionnement général de `netcat` en mode non connecté en vous basant sur les éléments vus en cours.

Exercice 2 : *Utilisation de la commande* `nc : netcat` *pour le transfert de fichier et l'évaluation de la bande passante*

La commande `netcat`*, souvent réduite au nom* `nc`*, est un utilitaire très puissant qui reprend le principe de la commande* `cat` *sur un support réseau. Les possibilités de cette commande sont énormes, et permettent de mettre en place très simplement un serveur en mode connecté ou non connecté pour transférer du texte ou un fichier. Cette commande peut également jouer le rôle de client.*

Nous avons vu la manipulations de la commande `nc` pour envoyer et recevoir un message en mode connecté - tcp - et non connecté - udp -. Nous allons exlpoiter ces fonctionnalités

1. Transfert d'un fichier via `netcat`
   (a) En réutilisant la commande `nc (netcat)` vue précédemment en TD, et la commande `cat`, et en les combinant avec les redirections entré-sortie vers et depuis fichiers et filtres, imaginez une solution permettant d'envoyer un fichier en mode connecté - tcp -

1. Le mot "localhost" désigne votre machine personnelle, vous pouvez remplacer ce mot-clé par une adresse ip d'une machine sur laquelle est lancé la commande faisant office de serveur si vous souhaitez étendre la manipulation sur deux machines distances. Cependant, il faudra alors veiller à utiliser des ports différents

Remarques / questions : benoit.darties@u-bourgogne.fr

depuis un terminal client vers un terminal serveur situé sur une autre machine. Testez votre solution d'abord lorsque le terminal client et le terminal serveur sont sur la même machine, puis sur des machines différentes.

(b) Récupérez sur internet un fichier de plusieurs centaines de méga-octets (par exemple une archive de Linux sur le site d'ubuntu). Notez sa taille. Puis transférez ce fichier au moyen de la solution décrite précédemment entre deux machines différentes.

(c) Lorsqu'une commande est exécutée, il est possible de connaitre sa durée d'utilisation exacte si l'on fait précéder la ligne de commande de la commande `time`. Dans l'exemple suivant, la commande `ls` est exécutée, et son temps réel et processeur est indiqué juste après :

```
1 Galactica-2:programmes benoit$ time ls
2 ...
3 real  0m0.011s
4 user  0m0.003s
5 sys 0m0.005s
```

Utilisez la commande `time` pour évaluer la durée de transfert d'un gros fichier depuis le client vers le serveur. Déduisez-en le débit applicatif que vous avez obtenu sur ce transfert. Réitérez l'opération une ou deux fois et constatez des écarts légers.

(d) Vérifiez que le fichier envoyé et le fichier reçu sont identiques à l'octet près. Pour cela, nous utiliserons un `hash`. De manière simpliée, il s'agit d'une fonction, qui lorsqu'on lui donne une suite d'octets potentiellement très grande, retourne une *signature*, c'est à dire un ensemble d'octets de taille fixe et très réduite. Si les deux suites d'octets en entrée sont rigoureusement identiques, alors la signature retournée est la même. Si les suites d'octets divergent un tout petit peu, alors les signatures sont différentes. Pour calculer une signature sur un fichier, utilisez la commande `md5` suivie du nom du fichier. Calculez les signatures des deux fichiers et vérifiez qu'elles sont rigoureusement identiques.

(e) Etendez votre solution de transfert avec la contrainte de n'utiliser qu'un seul terminal : utilisez la commande `ssh` pour vous connecter à à terminal virtuel sur une autre machine qui fera office de serveur, lancez votre solution en arrière-plan, quittez la session ssh et lancez le client. Décrivez la démarche effectuée.

(f) Décrivez une démarche similaire permettant à un camarade situé sur une autre machine de vous envoyer un fichier sans avoir besoin de droits d'authentification.

(g) Effectuez désormais le transfert d'un gros fichier en mode non connecté - UDP - entre deux machines distinctes avec l'option -u. Calculez la signature de chaque fichier. Ces signatures correspondent-elles ? Répétez l'opération 2 ou 3 fois.

Exercice 3 : *Une histoire de serveurs concurrents ...*

Dans cet exercice, nous regardons quelles capacités de cohabitation existent entre des serveurs qui voudraient utiliser le même port de communication

Effectuez les tests nécessaires avec la commande `netcat` pour répondre aux questions suivantes :

1. Peut-on avoir 2 serveurs en mode connecté - TCP - opérant sur le même port de la même machine ? Lancez deux instances de `nc` sur le même port, et testez la communication avec une instance client lancée dans un autre terminal.
2. Même question avec deux serveurs en mode non connecté - UDP.
3. Même question avec un serveur en mode non connecté - UDP - et un connecté - TCP -.

Remarques / questions : benoit.darties@u-bourgogne.fr

Exercice 4 : *Comprendre une requête HTTP*

Dans cet exercice, nous regardons une utilisation simple et originale de ***`netcat`*** *: comprendre ce que votre navigateur internet envoie comme information lorsqu'il effectue une requête sur un site web*

Il est possible que cet exercice ne soit pas réalisable sur le port 80 si vous n'avez pas les droits nécessaire pour utiliser ce denier. Auquel cas, utilisez un autre port, par exemple 5000, et adaptez l'url en utilisant `http://localhost:5000`. Notez également qu'une fois une connexion établie avec `nc`, il faut fermer le serveur et le relancer, ce dernier n'étant pas multi-clients.

1. Initier un connexion http :
   (a) Un serveur web tourne généralement sur le port 80 en mode connecté - TCP -. Utilisez la commande `nc` pour créer un serveur tournant sur le port 80 en mode connecté. Désormais, tout ce qui est reçu par ce serveur sera affiché à l'écran.
   (b) Ouvrez votre navigateur internet, et essayez d'ouvrir la page `http://localhost/`. On rappelle que `localhost` est un raccourci pour désigner votre machine personnelle. La page ne se charge pas. Qu'obtenez-vous dans la console dans laquelle tourne `nc` ?
   (c) Fermez le serveur, relancez le, et refaites la même manipulation avec un autre navigateur. Que constatez-vous ? Quel paramètre a changé ? Quelles sont les informations que votre navigateur communique généralement au serveur et sous quel nom officiel ?
2. Répondre à un connexion http :
   (a) Fermez le serveur `nc`, relancez-le. Puis ouvrez une connexion vers `http://localhost/` avec votre navigateur Notez la demande de connexion. Dans la console du serveur `nc`, tapez un message, sans terminer avec le caractère de fin de fichier `ctrl^d`. Voyez que le statut dans le navigateur change, et que ce dernier est en mode "réception de données". terminez alors la saisie avec le caractère de fin de fichier `ctrl^d`, et voyez le résultat dans le navigateur.
   (b) Le formatage d'une page web est possible grace au langage HTML. Ce langage a balise est très simple, mais ne fait pas l'objet de ce TP. Néanmoins, un exemple rudimentaire est présenté ci-après dans le fichier `exempleHTML.html` : Reprenez la manipulation précédente avec `nc` mais en copiant et collant désormais le texte de ce fichier. Observez alors le fichier résultat dans votre navigateur.

"Fichier exempleHTML.html"

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4      <H1> Liste des pokemon :</H1>
5
6  <P>Voici la liste des pokemon blablabla...</P>
7  <UL>
8      <LI>Salameche</LI>
9      <LI>Bulbizarre</LI>
10     <LI>Ronflex</LI>
11     <LI>Mew</LI>
12     <LI>...</LI>
13 </UL>
14
15 </body>
16 </html>
```


Remarques / questions : benoit.darties@u-bourgogne.fr

(c) Recopiez enfin ce texte dans un fichier `exempleHTML.html`, en le modifiant légèrement. Utilisez ensuite les redirections d'entrée -sortie de sorte que lorsqu'un client (navigateur, ou autre) se connecte à votre serveur `nc`, le contenu du fichier `exempleHTML.html` lui soit automatiquement envoyé.

2 Développement d'un client et d'un serveur en C

Exercice 5 : *Mise en place d'une communication en mode non connecte*

L'objectif de cet exercice est de découvrir les fonctions et structures de base en C permettant une communication en mode non connecté - udp.

Nous allons mettre en place deux processus, `clientUDP` et `serveurUDP`, qui vont échanger en mode non connecté. Le principe recherché est le suivant :

— Le serveur se met en écoute d'une communication en mode non connecté sur le port `5000`
— Le client envoie un message au serveur, et se termine.
— Lorsque le serveur reçoit un message, il l'inverse (par exemple le la phrase "vers l'infini et au dela" devient "aled ua te srev inifni'l srev", et l'affiche à l'écran, et se met en attente d'un nouveau message.

Nous utiliserons pour cela deux fichiers fournis `clientUDP.c` et `serveurUDP.c` écrits en C pour établir une communication udp en C. Ces fichiers squelettes sont construits à partir des fonctions et structures vues en cours et sont présentés ci-après. On vous demande ici de comprendre le contenu de ces fichiers, et de les compléter pour atteindre correctement l'objectif visé.

1. Identifiez l'adresse ip de votre machine grâce à la commande `ifconfig`
2. **Création du programme client par completion de** `clientUDP.c` :
   En reprenant les exemples vus en cours, et en complétant le fichier `clientUDP.c` :
   (a) Adaptez les macros présentées dans les instructions de pré-processing pour faire correspondre la macro `IP_DU_SERVEUR` avec l'adresse IP de la machine sur laquelle le serveur UDP sera exécuté ; Définissez un port de communication à utiliser sur le serveur et modifiez-la macro correspondante.
   (b) Ajoutez les instructions nécessaires à la création d'une socket en mode UDP.
   (c) Complétez la structure contenant les informations de contact du serveur, en réutilisant les macros de pré-processing précédemment modifiées. Veillez à adapter le format de présentation des données au format hôte (commandes `htonl`, `htons` ...).
   (d) Le message à envoyer est déjà défini dans un bufffer - il n'est pas nécessaire de modifier ce dernier -. Envoyez le contenu de ce buffer au serveur UDP en complétant l'appel à la fonction `sendto`.
   (e) A toute fin utile, ajoutez ensuite une instruction `printf()` affichant un message à l'écran indiquant que le message a été envoyé au serveur.
   (f) Utilisez la commande `netcat` lancée en serveur UDP sur le port que vous avez choisi pour tester votre client.
   (g) En utilisant le passage de paramètres à la fonction `main()`, modifiez votre programme de sorte que lors de l'exécution de ce dernier, le premier paramètre soit l'adresse IP utilisée pour la connexion, et le second le port d'application distant à utiliser.
3. **Création du programme serveur par completion de** `serveurUDP.c` :
   En reprenant les exemples vus en cours, et en complétant le fichier `serveurUDP.c` :
   (a) Ajoutez les instructions nécessaires à la création d'une socket en mode UDP.
   (b) Faites en sorte que cette socket soit attachée à toutes les adresses IP de votre machine- avec le mot-clé `INADDR_ANY` converti en format réseau avec la fonction `htonl()`, sur le port que vous aviez précédemment indiqué dans le programme du client(supérieur à 5000, et converti au format réseau avec la fonction `htons()`) .

Remarques / questions : benoit.darties@u-bourgogne.fr

(c) Positionnez le serveur en mode écoute, puis à chaque fois qu'un message est reçu, ce dernier le récupère dans un buffer, l'inverse au moyen de la fonction `inverser_buffer()`, ajoute le caractère '/0' après le dernier octet reçu (ceci est nécessaire pour des raisons d'affichage) et affiche le message à l'écran

(d) Utilisez le programme `clientUDP` précédemment défini pour tester votre serveur.

(e) Complétez votre programme pour que le serveur affiche à chaque message reçu l'identité du client qui a envoyé le message, ainsi que le port que le client a utilisé. Ceci est réalisable en utilisant une autre structure (déjà présente dans le code), qui est automatiquement remplie avec les informations du client à chaque fois que le serveur reçoit un message avec `recvfrom`. Il faudra cependant veiller à convertir le nom du client présent dans cette structure au format ascii avec la fonction `inet_ntoa()`, et la valeur du port au format hôte avec la fonction `ntohs` .

(f) Ce serveur est-il multi-clients ? lancez plusieurs clients UDP sur plusieurs machines et effectuez quelques tests pour vérifier le bon fonctionnement de votre serveur.

4. Comment adapter votre code dans chacun des programmes, pour que le client, une fois qu'il a envoyé le message, se mettre en attente d'une réponse du serveur et l'affiche à l'écran sitôt cette réponse reçue, tandis que le serveur envoie également le message inversé au client ?

Exercice 6 : *Création d'une architecture (client UDP) - (relai UDP-TCP)- (serveur TCP)*

Cet exercice difficile termine la partie communication distante. Il consiste à manipuler les deux mode connecté et non connecté, en faisant communiquer un serveur tournant en TCP avec un client tournant en UDP. Cette communication n'est pas directement possible et nécessite un processus intermédiaire qui fera le relai entre le client et serveur.

L'objectif de cet exercice est d'écrire d'un programme `relai` permettant une communication entre un serveur s'exécutant en mode connecté (TCP) et un client fonctionnant en mode non connecté (UDP). Le relai recoit un message du client, le transfère au serveur. Ce serveur traite le message, et le renvoie au relai. Le relai transfère alors le message au client qui l'affiche à l'écran. L'objectif est que client et serveur puissent communiquer, de façon bidirectionnelle.

Les différents éléments à mettre en place sont détaillés ci-après. Le schéma récapitulatif ci après donne des indications sur les communications à établir.

Pour simplifier l'exercice, nous donnons les éléments suivants :
— Le serveur `serveurTCP` fonctionne de la façon suivante : A chaque réception d'un message, il inverse ce message, et le renvoie à son destinataire via la socket de travail.
— Le serveur sera mono-client (ne gèrent pas plusieurs connexions entrantes simultanées)
— Le code du client `clientUDP` n'est pas donné. Néanmoins, il est très proche de ce qui a été réalisé dans l'exercice 13 du TD. Manquent simplement les instructions suivantes : une fois le message envoyé, le client se met en écoute d'une réponse, la récupère et l'affiche à l'écran.
— Le relai `relaisUDPTCP` débute en attendant une requête du client lorsque le relai reçoit le message du client, il le renvoie immédiatement vers le serveur, et se met en attente de la réponse du serveur. Dès qu'il reçoit la réponse du serveur, il la renvoie immédiatement au client et se met en attente d'un nouveau message d'un autre client.

1. En vous basant sur les éléments précisés ci-après, complétez le schéma précédent reprenant les différentes communications entre les entités présentes, en précisant les socket à créer et le mode de communication à utiliser pour chacune d'entre elle. Rajoutez quelques principes généraux de fonctionnement de chacun de ces programmes.

Remarques / questions : benoit.darties@u-bourgogne.fr

2. **Développement du client `clientUDP` :**
   (a) Le processus client possède les caractéristiques suivantes :
   — Modèle de communication en UDP
   — L'adresse IP de la passerelle et le port de communication sur lequel contacter le processus sont passés en paramètres lors de l'appel à `clientUDP`.
   — Le processus crée une socket de communication distante permettant de dialoguer avec la passerelle.
   — Le processus demande ensuite à l'utilisateur de saisir une ligne. Cette partie n'est pas triviale en C pur, surtout lorsque l'on veut récupérer une ligne contenant des espaces et évaluer sa taille utile (qui est différente de la taille du buffer utilisé pour stocker cette ligne. On vous donne ci-après les instructions qui permettent de récupérer une ligne entière au clavier, de la stocker dans un buffer, et de récupérer sa taille effective (voir "récupérer une ligne au clavier et l'afficher")

"récupérer une ligne au clavier et l'afficher"

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main () {
5    char buffer[1000];
6    int taille;
7    scanf("%[^\n]", buffer);
8    taille =  strlen(buffer);
9
10   printf("ligne entree :  %s\n", buffer);
11   printf("taille de la ligne entree : %i\n", taille);
12 }
```

   — Le processus envoie ensuite la ligne entrée à la passerelle sur la socket créée pour l'occasion
   — Puis il attend la réponse de la passerelle sur la même socket que celle utilisée pour l'envoi, et affichera la réponse lue sur de cette dernière à l'écran.
   — Enfin il se termine.
   En vous aidant des éléments précédents, développez le code du programme `clientUDP`.

3. **Développement du serveur `serveurTCP` :**
   (a) Le serveur, que l'on nommera `serveurTCP`, s'exécutera idéalement sur une autre machine, et utilise le port 4444 en mode connecté. Son fonctionnement se décompose de la façon suivante :
   — **Le serveur** se met en attente d'une connexion sur sa socket principale ;
   — lorsqu'il reçoit une demande de connexion, il accepte cette dernière crée une socket de travail et écoute sur cette dernière ;
   — le serveur lit sur la socket de travail un message qui ne dépassera pas 1023 caractères et le stocke dans un buffer ;
   — il applique alors la fonction `inverserBuffer()` sur le buffer, pour en inverser l'ordre des caractères. Cette fonction et son utilisation sont données ci après
   — il émet ensuite le message inversé sur la socket de travail, et ferme cette dernière ;
   — il retourne à l'écoute de la socket principale en attente de connexion.
   La fonction `inverserBuffer()` est donnée ci-après :
   En vous aidant des éléments précédents, développez le code du programme `serveurTCP`.

4. **Développement du relai :**

5. le programme `relai-UDP-TCP` se charge de relayer les messages du client vers le serveur et réciproquement, en dialoguant avec chacune de ces entités avec le mode de communication correspondant (connecté ou non-connecté). On peut ainsi voir le relai comme jouant le rôle

Listing 1 – Fonction `inverserBuffer`

```
1  void inverserBuffer(char buffer[], int taille) {
2  /*   Donnees :
3         char buffer[] : buffer contenant le message a inverser
4         int taille : taille du message contenu dans le buffer
5       Resultat :
6         inverse les caracteres du message contenu dans le buffer.
7  */
8    int i;
9    char tmp;
10   for (i=0; i < taille/2; i++){
11     tmp = buffer[i];
12     buffer[i] = buffer[taille -1 -i ];
13     buffer[taille - 1 - i] = buffer[i];
14   }
15 }
```

d'un *client en mode connecté* pour le programme `serveurTCP`, et comme jouant le role un serveur en mode non connecté pour le programme `clientUDP`. De ce fait, il utilise une socket en mode non-connecté en écoute des programmes `clientUDP` sur le port 2222, et fonctionne de la manière suivante :

- Le programme `relai UDP-TCP` écoute sur une socket UDP en port 2222 l'arrivée de messages de la part du client ;
- lorsqu'un message arrive, il est stocké dans un buffer, et l'identité du client est copiée dans la structure adéquate ;
- `relai UDP-TCP` établit alors une connexion TCP en direction du serveur, et lui transmet le message du client ; l'identité du serveur peut-être fixe, et passée en paramètre (ip + port) lors du lancement du programme `relai UDP-TCP`.
- il attend ensuite la réponse du serveur qu'il stocke dans un buffer, et ferme cette connexion ;
- il retransmet enfin le message vers le client en mode non connecté, dont l'identité a été stockée précédemment.

En vous aidant des éléments précédents, développez le code du programme `relai-UDP-TCP`. Sachez que le relai devra disposer de 2 descripteurs de sockets, l'un en mode connecté pour la communication avec le serveur, et l'autre en mode non connecté pour la communication avec le client. Nous aurons également besoin de trois structures d'adresses de type `struct sockaddr_in` :

(a) La première structure permettra de donner un nom à la socket dédiée à la communication avec le client ;

(b) la seconde structure doit contenir l'adresse de la socket du serveur ;

(c) la troisième structure permet de stocker l'adresse du client UDP afin de lui transférer par la suite la réponse du serveur.

2.1 Exercices bonus

Ces exercices sont facultatifs et réservés aux personnes qui souhaitent approfondir leurs connaissances.

Exercice 7 : *Résolution de noms*

Cet exercice a pour objectif de manipuler la fonction `gethostbyname()` . Cette fonction permet de transformer des noms de domaines en adresse ip, en interrogeant un serveur DNS.

1. A l'aide du manuel et des exemples disponibles sur Internet, consultez la documentation de la fonction `gethostbyname()` permettant la translation d'un nom de domaine (ex : www.yahoo.fr) vers une adresse IP.
2. Créez alors un programme qui affiche les adresses IP des noms de domaine www.yahoo.fr, www.gmail.com et www.u-bourgogne.fr

Exercice 8 : *Serveur multi-client en mode connecte*

Jusqu'à présent, une ruse toute simple nous a permis d'éluder le problème des serveur multi-clients : le fait que lorsqu'un client se connecte à un serveur en mode connecté, ce dernier est tout de suite pris en charge, un message lui est envoyé ou reçu et la connexion est tout de suite fermée. S'il fallait maintenir la connexion avec le client, on ne pourrait a priori pas pouvoir traiter l'arrivée d'un second client. L'objectif de cet exercice est d'outrepasser cette limite en utilisant l'appel système `fork()`.

1. En utilisant vos connaissances sur l'appel système `fork()` proposez une méthode pour gérer plusieurs clients
2. La méthode `fork()` s'avère lourde car nécessite plusieurs processus. Une alternative est de passer pas une autre approche, qui consiste à établir une liste des descripteurs à écouter simultanément. Lorsqu'un descripteur a une donnée à lire, il se manifeste et peut alors être traité. Ceci est rendu possible par l'utilisation d'une fonction nommée `select()`. Consultez le manuel de la fonction `select()` ainsi que la documentation présente sur internet afin de comprendre son mécanisme d'utilisation. Essayer de mettre en place cette solution sur un serveur en mode connecté. L'étude de cette fonction `select()` pourrait se révéler particulièrement utile pour l'option SE.

Remarques / questions : benoit.darties@u-bourgogne.fr