

INF3405 – Réseaux Informatiques Laboratoire n°3

Familiarisation avec les sockets

Cédric DELAUNAY – 1914647

Joël POULIN – 1750072

Poste L4708-11

École Polytechnique de Montréal

Département génie informatique et génie logiciel

6 octobre 2017

A. Fonction getaddrinfo

```
getaddrinfo a réussi!  
Adresse # 1  
  Flags: 0x0  
  Famille: AF_INET6 (IPv6)  
  Adresse IPv6: fe80::c80a:944:e91a:f975  
  Taille de cette adresse: 28 octets  
  Nom canonique: (null)  
Adresse # 2  
  Flags: 0x0  
  Famille: AF_INET6 (IPv6)  
  Adresse IPv6: fe80::f922:319a:272d:d7f0  
  Taille de cette adresse: 28 octets  
  Nom canonique: (null)  
Adresse # 3  
  Flags: 0x0  
  Famille: AF_INET6 (IPv6)  
  Adresse IPv6: fe80::40cf:2614:cd7e:599d  
  Taille de cette adresse: 28 octets  
  Nom canonique: (null)  
Adresse # 4  
  Flags: 0x0  
  Famille: AF_INET6 (IPv6)  
  Adresse IPv6: fe80::4cc6:eace:f145:71b1  
  Taille de cette adresse: 28 octets  
  Nom canonique: (null)  
Adresse # 5  
  Flags: 0x0  
  Famille: AF_INET (IPv4)  
  Adresse IPv4: 132.207.29.111  
  Taille de cette adresse: 16 octets  
  Nom canonique: (null)  
Adresse # 6  
  Flags: 0x0  
  Famille: AF_INET (IPv4)  
  Adresse IPv4: 192.168.80.1  
  Taille de cette adresse: 16 octets  
  Nom canonique: (null)  
Adresse # 7  
  Flags: 0x0  
  Famille: AF_INET (IPv4)  
  Adresse IPv4: 192.168.132.1  
  Taille de cette adresse: 16 octets  
  Nom canonique: (null)  
Adresse # 8  
  Flags: 0x0  
  Famille: AF_INET (IPv4)  
  Adresse IPv4: 169.254.113.177  
  Taille de cette adresse: 16 octets  
  Nom canonique: (null)  
Appuyer une touche pour finir...
```

1. Il y en a 8, dont 4 sont des IPv4 et les 4 autres sont des IPv6.
2. L'adresse IPv4 de l'interface active sur le réseau de l'école est 132.207.29.111. Elle débute donc en décimal par le nombre 132, converti en binaire par 10000100. Une adresse débutant par les bits 10 correspond à une adresse de classe C.

3. Un nœud connecté à plusieurs réseaux peut entreposer plusieurs adresses IP, puisque l'identification avec chacun des réseaux sera différent et requiert une configuration d'adresse IP propre à chaque réseau.
4. Il existe 5 classes d'adresses IPv4 : A, B, C D et E. La classe A se situe dans une plage d'adresses de 1.0.0.0 à 126.0.0.0 (16 777 214 adresses hôtes par réseau), le masque de sous-réseau par défaut est 255.0.0.0. La classe B est dans une plage d'adresses de 128.0.0.0 à 191.255.0.0 (65 534 adresses hôtes par réseau) et le masque de sous-réseau est 255.255.0.0. La classe C se situe entre les adresses 192.0.0.0 et 223.255.255.0 (254 adresses hôtes par réseau) et le masque de sous-réseau est 255.255.255.0. La classe D est entre les adresses 224.0.0.0 et 239.0.0.0 (268 435 456 groupes multicast) et le masque de sous-réseau 240.0.0.0. La classe E est présentement réservée à des fins de développement futur, mais n'est pas utilisée pour le moment.
5. A. La représentation binaire de 132.207.29.111 (classe C) est 10000100.11001111.00011101.01101111 et le masque de sous-réseau 255.255.255.0 en binaire est 11111111.11111111.11111111.00000000
B. L'adresse du réseau auquel appartient le poste de travail est 132.207.29.0, ce qui correspond en binaire à 10000100.11001111.00011101.00000000, car pour une adresse de classe C, les 3 premiers octets indiquent l'adresse du réseau et le dernier est réservé à la machine.

```
getaddrinfo a réussi!  
Adresse # 1  
  Flags: 0x0  
  Famille: AF_INET (IPv4)  
  Adresse IPv4: 132.207.29.103  
  Taille de cette adresse: 16 octets  
  Nom canonique: (null)  
Appuyer une touche pour finir...
```

- 6.
7. La fonction est `getaddrinfo()`. Elle retourne notamment la famille à laquelle appartient l'adresse IP ainsi que l'adresse elle-même.

```
dwRetVal = getaddrinfo(host, NULL, NULL, &result);
```

8. Les NetID de notre poste et du poste L4708-03 sont égaux (132.207.29), ils appartiennent donc au même réseau.

B. Développement d'applications de réseau en utilisant les sockets

9. Le socket utilise la famille d'adresses IPv4. Dans le main du Lab302, on peut apercevoir que le socket est instancié avec AF_INET en paramètre, qui est une variable qui définit la famille d'adresses.

```
// On va creer le socket pour communiquer avec le serveur
leSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Dans le main du Labo301, qui utilise des dépendances communes comme AF_INET, on peut apercevoir que la variable correspond à une famille IPv4.

```
case AF_INET:
    printf("AF_INET (IPv4)\n");
    sockaddr_ipv4 = (struct sockaddr_in *) ptr->ai_addr;
    printf("\tAdresse IPv4: %s\n",
        inet_ntoa(sockaddr_ipv4->sin_addr) );
    break;
```

10. Le port utilisé est le port 5000. On peut l'apercevoir dans le code à cet endroit :

```
// On indique le nom et le port du serveur auquel on veut se connecter
//char *host = "L4708-XX";
//char *host = "L4708-XX.lerb.polymtl.ca";
//char *host = "add_IP locale";
char *host = "132.207.29.111";
char *port = "5000";
```

11. Le type du socket à utilisé est le SOCK_STREAM, ce qui correspond à un socket de type TCP.

- 12.

```
sockaddr_in service;
service.sin_family =
//service.sin_addr.s_
// service.sin_addr_
service.sin_addr.s_ad
service.sin_port = ht

if (bind(&serverSocket, (SOCKADDR*)&service, sizeof(service)) == SOCKET_ERROR) {
```

13. Nous constatons que le mot reçu par le serveur est le même que le mot envoyé. Néanmoins, lorsque le mot est retransmis au client, une modification de la casse apparaît.

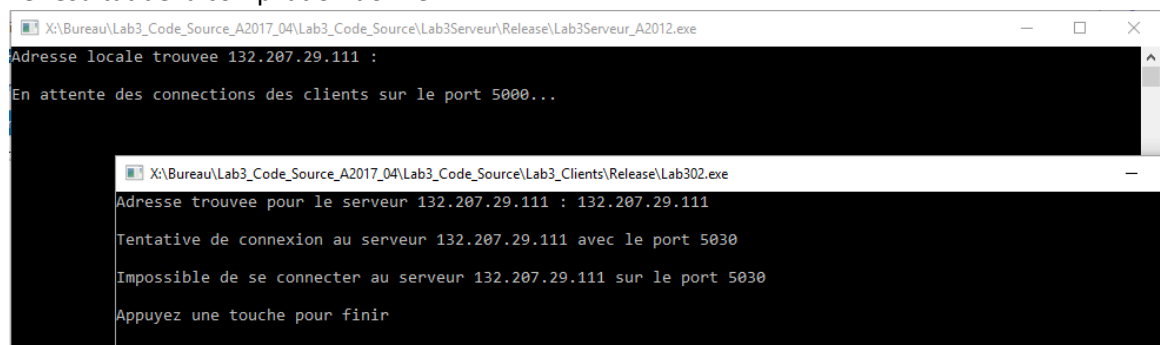
14. On peut envoyer des mots au serveur grâce à la méthode suivante :

```
// Envoyer le mot au serveur  
iResult = send(leSocket, motEnvoye, 7, 0 );
```

15. La fonction permettant de recevoir les informations provenant du serveur est :

```
// Maintenant, on va recevoir l' information envoyée par le serveur  
iResult = recv(leSocket, motRecu, 7, 0);
```

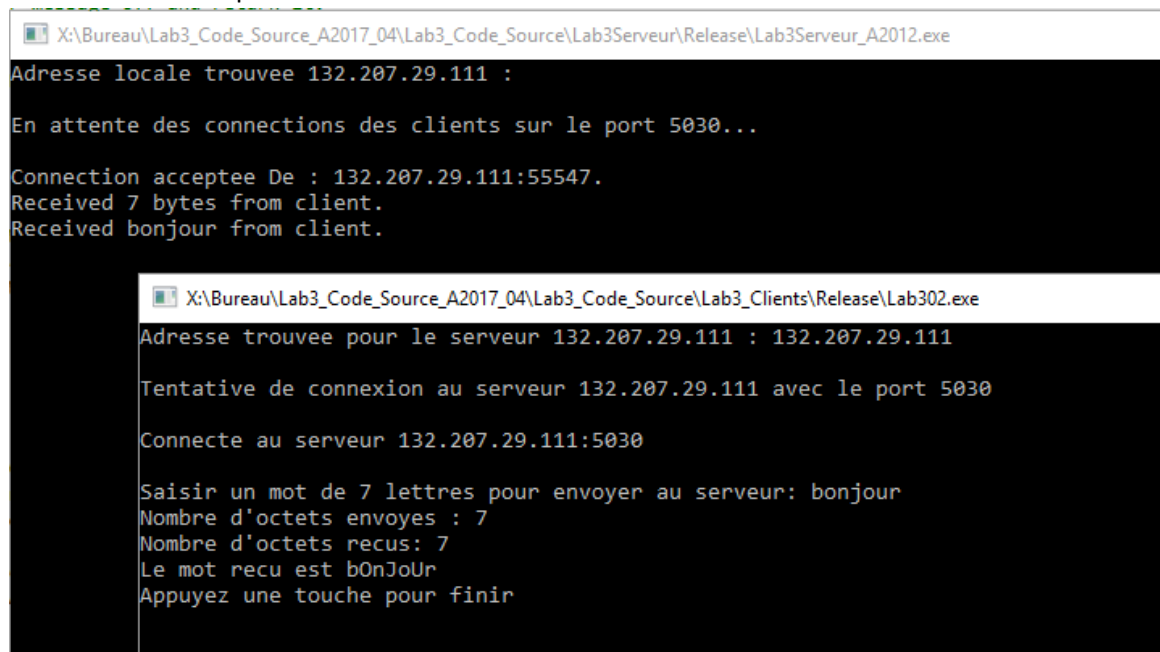
16. Le résultat de la compilation donne :



```
X:\Bureau\Lab3_Code_Source_A2017_04\Lab3_Code_Source\Lab3Serveur\Release\Lab3Serveur_A2012.exe  
Adresse locale trouvee 132.207.29.111 :  
En attente des connections des clients sur le port 5000...  
  
X:\Bureau\Lab3_Code_Source_A2017_04\Lab3_Code_Source\Lab3_Clients\Release\Lab302.exe  
Adresse trouvee pour le serveur 132.207.29.111 : 132.207.29.111  
Tentative de connexion au serveur 132.207.29.111 avec le port 5030  
Impossible de se connecter au serveur 132.207.29.111 sur le port 5030  
Appuyez une touche pour finir
```

Ce résultat était prévisible étant donné que nous n'avons pas modifié le port dans le code source du serveur.

17. Résultat de la compilation :



```
X:\Bureau\Lab3_Code_Source_A2017_04\Lab3_Code_Source\Lab3Serveur\Release\Lab3Serveur_A2012.exe  
Adresse locale trouvee 132.207.29.111 :  
En attente des connections des clients sur le port 5030...  
Connection acceptee De : 132.207.29.111:55547.  
Received 7 bytes from client.  
Received bonjour from client.  
  
X:\Bureau\Lab3_Code_Source_A2017_04\Lab3_Code_Source\Lab3_Clients\Release\Lab302.exe  
Adresse trouvee pour le serveur 132.207.29.111 : 132.207.29.111  
Tentative de connexion au serveur 132.207.29.111 avec le port 5030  
Connecte au serveur 132.207.29.111:5030  
Saisir un mot de 7 lettres pour envoyer au serveur: bonjour  
Nombre d'octets envoyes : 7  
Nombre d'octets recus: 7  
Le mot reçu est bOnJoUr  
Appuyez une touche pour finir
```

Un port permet l'échange de données entre deux nœuds (ici un client et un serveur). Autrement dit, même si les deux nœuds connaissent leurs adresses respectives, il est nécessaire d'ouvrir un port pour pouvoir échanger des données. Cela permet donc d'assurer une certaine sécurité puisqu'un tiers ne pourra échanger des données avec une machine si certains ports sont fermés.

18. Notre idée de départ était de récolter l'entrée de l'utilisateur sous forme de texte. Ensuite, de modifier le texte afin d'obtenir que des chiffres pour pouvoir après convertir la série de chiffre en un nombre entier. Ce nombre est ensuite testé, afin de savoir si l'adresse correspondante se situe dans le laboratoire. Sinon, on redemande l'adresse à l'utilisateur. Nous avons obtenu une erreur, car lors de la conversion de string vers un int, le nombre maximal de l'ordinateur est dépassé. Nous n'avons pas su résoudre ce problème dans les délais.

```
string hosttt;
cout << "Veuillez entrer l'adresse du serveur que vous voulez joindre: ";
cin >> hosttt;
string hostBis = hosttt;
hostBis.erase(std::remove(hostBis.begin(), hostBis.end(), '.'), hostBis.end());
int conv = stoi(hostBis);

while (conv < 13220729101 && conv > 13220729127) {
    cout << "Veuillez entrer l'adresse du serveur que vous voulez joindre: ";
    cin >> hosttt;
    hostBis = hosttt;
    hostBis.erase(std::remove(hostBis.begin(), hostBis.end(), '.'), hostBis.end());
    conv = stoi(hostBis);
}
const char *host = hosttt.c_str();
char *port = "5030";
```

19. Les méthodes `accept(ServerSocket, (sockaddr*)&sinRemote, &nAddrSize)` et `CreateThread(0, 0, EchoHandler, (void*)sd, 0, &nThreadID)` permettent d'implémenter la gestion de plusieurs connexions

20. Dans la partie de code ci-dessous, on récupère dans `nAddrSize` la taille de `sinRemote`. Puis, on crée un SOCKET avec `accept()`, qui prend en paramètre la variable `nAddrSize`, qui possède le nombre de connexions disponibles. Ensuite, on fait une vérification des erreurs. S'il n'y en a pas, on accepte la connexion et crée un nouveau *thread* qui servira de canal de communication pour cette connexion. En cas d'erreur, on retourne un message d'erreur. Le mécanisme roule constamment, tant que le processus n'est pas mis hors tension, afin de pouvoir accepter toutes nouvelles connexions.

```
while (true) {  
  
    sockaddr_in sinRemote;  
    int nAddrSize = sizeof(sinRemote);  
    // Create a SOCKET for accepting incoming requests.  
    // Accept the connection.  
    SOCKET sd = accept(ServerSocket, (sockaddr*)&sinRemote, &nAddrSize);  
    if (sd != INVALID_SOCKET) {  
        cout << "Connection acceptee De : " <<  
             inet_ntoa(sinRemote.sin_addr) << ":" <<  
             ntohs(sinRemote.sin_port) << "." <<  
             endl;  
  
        DWORD nThreadID;  
        CreateThread(0, 0, EchoHandler, (void*)sd, 0, &nThreadID);  
    }  
    else {  
        cerr << WSAGetLastError("Echec d'une connection.") <<  
             endl;  
        // return 1;  
    }  
}
```

21.

Protocole	Port(s) par défaut
FTP	21 (écoute), 20 (données par défaut)
HTTP	80
HTTPS	443
SSH	22