Université de Nice Sophia Antipolis
UFR Sciences - Département d'Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TP n° 1 : Introduction aux processus en Unix

1 Conversion d'adresses

Les deux programmes qui suivent vous montrent comment utiliser les fonctions de conversion ntohx(3) et htonx(3) et les fonctions gethostbyxxx(3). Essayez-les et expliquez de quelle façon ils fonctionnent (les sources sont sur la page web).

Pour connaitre quelles librairies vous devez eventuellement lier, il faut regarder le manuel **pour chaque** fonction de bibliothèque utilisée.*NB : Pour compiler ces fichiers en langage C, selon la machine utilisée vous pouvez avoir besoin de lier avec une librairie externe. Par exemple sur MacOS il faut ajouter* `-lc` *:*
`gcc prog1.c -lc -o prog1`

*Pour connaitre quelles librairies vous devez eventuellement lier, il faut regarder le manuel **pour chaque** fonction de bibliothèque utilisée.*

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>

int main(int argc,char **argv)
{
  unsigned long addr,net;

  if (argc != 2)
    {
      fprintf(stderr,"Usage: %s hostname\n",argv[0]);
      exit(1);
    }
  if ((net = inet_addr(argv[1])) == -1)
    {
      fprintf(stderr,"inet_addr: invalid host name %s !\n", argv[1]);
      exit(1);
    }

  printf("Converted value (network) is %lu (%u.%u.%u.%u)\n",
```

```
 net,
 (net >> 24)& 0xff,
 (net >> 16)& 0xff,
 (net >> 8)& 0xff,
 net & 0xff);

  addr = ntohl(net);
  printf("Converted value (host) is %lu (%u.%u.%u.%u)\n",
 addr,
 (addr >> 24)& 0xff,
 (addr >> 16)& 0xff,
 (addr >> 8)& 0xff,
 addr & 0xff);
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

extern int h_errno;

int main(int argc,char **argv)
{
  unsigned long addr,net;
  struct hostent *he;

  if (argc != 2)
    {
      fprintf(stderr,"Usage: %s (xx.yy.zz.tt|hostname)\n",argv[0]);
      exit(1);
    }

  if ((net = inet_addr(argv[1])) == -1)
    {
      if (! (he = gethostbyname(argv[1])))
        {
          fprintf(stderr,"Invalid adresse %s\n",argv[1]);
        }
    }
  else
    {

      if (! (he = gethostbyaddr((char *)&net,sizeof(addr),AF_INET)))
        {
          fprintf(stderr,"Invalid adresse %s\n",argv[1]);
        }
    }

  printf("CNAME: %s\n",he->h_name);
```

```
  while (*(he->h_aliases))
    printf("ALIAS: %s\n",*(he->h_aliases++));

  while (*(he->h_addr_list))
    {
      struct in_addr in;

      (void) memcpy(&in.s_addr, *(he->h_addr_list++), sizeof (in.s_addr));
      printf("ADDR : %s (len=%d)\n",inet_ntoa(in),he->h_length);
    }

  return 0;
}
```

2 Programmation sockets

Exercice 1 *Le but de cet exercice est de parvenir à faire communiquer deux processus qui s'executent sur deux machines différentes. Il vous faut pour cela écrire deux programmes en langage C :*

1. *un programme, appelé 'envoyeur_udp' dont l'unique tâche est d'envoyer la chaîne de caractère "Hello world" au second programme, au travers d'un socket UDP ;*
2. *un programme, appelé 'receveur_udp' dont l'unique tâche est de se mettre en attente d'un message (une chaîne de caractères) en provenance d'un socket UDP et de l'afficher à l'écran.*

Pour tester que vos programmes fonctionnent, ouvrez deux terminaux, l'un sur votre machine et l'autre sur la machine voisine (rlogin). Cela fait, lancez d'abord le programme receveur sur l'une des machines, puis le programme émetteur sur l'autre machine. Si tout se passe bien, le programme receveur doit afficher la chaîne "Hello world" dès que vous executer le programme emetteur ...

Dans un premier temps, vous pourrez vous contenter d'une version de ces programmes où le numéro de port et le nom de l'autre machine sont codés "en dur" dans le programme émetteur, à l'aide de macros `#define`*. En d'autres termes, votre programme émetteur ne saura envoyer son message que vers une seule machine, et pas une autre.*

Dans un deuxième temps, écrivez une version plus souple du programme émetteur, dans laquelle le nom de la machine sur laquelle s'exécute le programme receveur, ainsi que le numéro de port UDP pouront être recus en paramètres.

Exercice 2 *Cet exercice a pour but d'améliorer les deux programmes précédents de la facon suivante :*

1. *le programme émetteur doit lire des chaînes de caractères au clavier et transmettre chacune d'elles vers le programme receveur. Il s'arrete lorsque la chaîne "fin" est lue.*
2. *le programme receveur doit se mettre en attente des messages du programme émetteur, et afficher les chaînes de caractères correspondantes a l'écran. Le programme s'arrète dès que la chaîne de caractères "fin" est recue.*

Exercice 3 *Faites de nouvelles versions des programmes précédents utilisant des sockets TCP au liau de sockets UDP (attention, il faut dans ce cas établir une connexion avant d'envoyer un message).*

Exercice 4 *Le but de ce programme est de construire une petite application client/serveur TCP :*

1. *le serveur se met en attente d'un message du client. Ce message est supposé représenter un nom de fichier présent sur la machine où s'exécute le serveur (il faut que le serveur vérifie que le fichier existe). Si le fichier existe, le serveur transmet en retour le contenu de ce fichier au client.*
2. *le client envoie un nom de fichier au serveur puis se met en attente de la reception du contenu du fichier demandé et affiche ce dernier a l'écran. Le client recoit le nom du fichier, le nom de la machine sur laquelle s'exécute le serveur et le numéro du port sur lequel le serveur "écoute" en paramètre.*