FONCTIONS DE BASE - MANIPULATION ADRESSES ET NOMS MACHINES

```
/*  compilation sous linux :   gcc <nom_du_fichier> * /
/*   execution : a.out <nom_de_la_machine>  */
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

main (int argc, char **argv) {
int retour ;
/* definitions pour partie 1                                                    */
char    nom1[255] ;                 /* longueur théorique  max 255 octets = 63*4 +3   */
char * nom2 ;

struct in_addr    adr1;
struct in_addr * adr2;

/* allocation memoire, si on veut faire de l'allocation dynamique               */
adr2 = (struct in_addr *) malloc (sizeof (struct in_addr));
nom2     = (char *) malloc (255);


/*****************************************************************/
/*        Partie 1 : fonctions de manipulation des adresses                    */
/*****************************************************************/

strcpy (nom1, "193.52.16.20");

/* appel de la fonction inet_aton : traduction @ pointée en @ numerique         */
retour = inet_aton(nom1, adr1) ;
if ( retour == 1 )
          { printf ("adresse sur 32 bits (avec u): %u\n", adr1->s_addr);
            printf ("adresse sur 32 bits (avec d): %d\n", adr1->s_addr);
          }
else      { printf("erreur dans inet_aton\n");
          }

/* recopie d'argument pour traduction en sens inverse                           */
bcopy(adr1, &adr2, sizeof (adr1));

/* appel de la fonction  inet_ntoa  */
strcpy (nom2, inet_ntoa(adr2));

printf ("@ IP de la machine : %s\n\n", nom2);

/* résultats obtenus :                                                          */
/*        adresse sur 32 bits (avec u): 3241414676                              */
/*        adresse sur 32 bits (avec d): -1053552620                             */
/*                                                                              */
/*        @ IP de la machine : 193.52.16.20                                     */
/*                                                                              */
/*        -1053552620 correspond à +3241414676 en notation binaire     */
/*       11000001 00110100 00010000 00010100 = 193.52.16.20          */
```

```
/* ajout de definitions pour la partie 2                                    */
char nommachine[255];
struct hostent * statichostent;
struct in_addr adresse;

/*************************************************************/
/*        Partie 2 : fonctions gethostname/gethostbyname/gethostbyaddr   */
/*************************************************************/

/* gethostname : appel de la fonction et affichage resultat                 */
retour = gethostname (nommachine, 255);
printf("Nom de la machine : %s \n", nommachine);

/* gethostbyname : appel de fonction                                        */
statichostent = gethostbyname("corsen.univ-brest.fr");

/* gethostbyname : recopie dans adresse                                     */
bcopy(statichostent->h_addr, &adresse, statichostent->h_length);

/* gethostbyname : affichage des résultats                                  */
printf ("La machine %s a pour adresse %u \n", statichostent->h_name, adresse.s_addr);
printf ("La machine %s a pour adresse %s \n", statichostent->h_name, inet_ntoa(adresse));

/* gethostbyaddr : appel de fonction                                        */
statichostent = gethostbyaddr((const char *) &adresse.s_addr, sizeof(long), AF_INET);

/* gethostbyaddr : affichage des résultats                                  */
printf ("La machine a pour nom : %s \n", statichostent->h_name);

/* resultats obtenus :                                                      */
/*        Nom de la machine : lepouldu                                      */
/*        La machine corsen a pour adresse 3241414757                       */
/*        La machine corsen a pour adresse 193.52.16.101                    */
/*        La machine a pour nom : corsen                                    */
}
```

UTILISATION DES SOCKETS - SERVEUR TCP (simplifié – pas de création de fils)

```
/*  compilation sous linux :   gcc <nom_du_fichier> -o serveur_tcp * /
/* Lancement d'un serveur :  serveur_tcp port */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define TRUE 1

main (int argc, char **argv)
{
int socket_RV, socket_service;
char buf[256];
int entier_envoye;
struct sockaddr_in adresseRV;
int lgadresseRV;
struct sockaddr_in adresseClient;
int lgadresseClient;
struct hostent *hote;
unsigned short port;

/* creation de la socket de RV */
if ((socket_RV = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
        perror("socket");
        exit(1);
        }

/* preparation de l'adresse locale */
port = (unsigned short) atoi(argv[1]);

adresseRV.sin_family = AF_INET;
adresseRV.sin_port = htons(port);
adresseRV.sin_addr.s_addr = htonl(INADDR_ANY);   /* toutes les addresses de la machine locale  */
lgadresseRV = sizeof(adresseRV);

/* attachement de la socket a l'adresse locale */
if ((bind(socket_RV, (struct sockaddr *) &adresseRV, lgadresseRV)) == -1)
        {
        perror("bind");
        exit(3);
        }


/* declaration d'ouverture du service */
if (listen(socket_RV, 10)==-1)
        {
         perror("listen");
        exit(4);
        }
```

```
/* boucle d'attente de connexion */
  while (TRUE)                    /* !!! boucle infinie !!! */
        {
        printf("Debut de boucle\n");
        fflush(stdout);

        /* attente d'un client */
         lgadresseClient = sizeof(adresseClient);
         socket_service = accept(socket_RV, (struct sockaddr *) &adresseClient, &lgadresseClient);

        if (socket_service == -1)
                {
                perror("accept");
                exit(5);
                }
        /* un client est arrive */
        printf("connexion acceptee\n");
        fflush(stdout);

        /* Le serveur fait le traitement sans créer de fils – cas particulier de traitements courts   */
         /* lecture dans la socket d'une chaine de caractères */
         if (read(socket_service, buf, 256) < 0)
                {
                perror("read");
                exit(6);
                }
        printf("Chaine recue : %s\n", buf);
        fflush(stdout);

        /* ecriture dans la socket d'un entier */
        entier_envoye = 2222;
        if (write(socket_service, &entier_envoye, sizeof(int)) != sizeof(int))
                {
                perror("write");
                exit(7);
        }
        printf("Entier envoye : %d\n", entier_envoye);

        close(socket_service);

        }  /* fin du while – le serveur peut attendre un nouveau client   */
}
```

UTILISATION DES SOCKETS - CLIENT TCP

```
/*  compilation sous linux :   gcc <nom_du_fichier> -o client_tcp * /
/* Lancement d'un client : client_tcp port machine_serveur */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

main (int argc, char **argv)
{
int sock, entier_recu;
char buf[256];
struct sockaddr_in adresse_serveur;
struct hostent *hote;
unsigned short port;

/* creation de la socket locale */
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
        perror("socket");
        exit(1);
        }

/* recuperation de l'adresse IP du serveur (a partir de son nom) */
if ((hote = gethostbyname(argv[2])) == NULL)
        {
        perror("gethostbyname");
        exit(2);
        }

/* preparation de l'adresse du serveur */
port = (unsigned short) atoi(argv[1]);

adresse_serveur.sin_family = AF_INET;
adresse_serveur.sin_port = htons(port);
bcopy(hote->h_addr, &adresse_serveur.sin_addr, hote->h_length);    /* l'adresse du serveur */
printf("L'adresse du serveur en notation pointee %s\n", inet_ntoa(adresse_serveur.sin_addr));
fflush(stdout);

/* demande de connexion au serveur */
if (connect(sock, (struct sockaddr *) &adresse_serveur, sizeof(adresse_serveur))==-1)
        {
        perror("connect");
        exit(3);
        }

/* le serveur a accepté la connexion */
printf("connexion acceptee\n");
fflush(stdout);
```

```
/* ecriture dans la socket d'une chaine */
strcpy(buf, "Bonne année");
if (write(sock, buf, strlen(buf)+1) != strlen(buf)+1)
        {
        perror("write");
        exit(4);
        }
printf("Chaine envoyée : %s\n", buf);
fflush(stdout);


/* lecture dans la socket d'un entier */
if (read(sock, &entier_recu, sizeof(int)) < 0)
        {
        perror("read");
        exit(5);
        }
printf("Entier reçu : %d \n", entier_recu);
fflush(stdout);

} /* fin du client  */
```

UTILISATION DES SOCKETS - SERVEUR TCP COMPLET

```
/*  compilation sous linux :   gcc <nom_du_fichier> -o serveur_tcp * /
/* Lancement d'un serveur :  serveur_tcp port                          */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define TRUE 1

int creer_socket(int type, int *ptr_port, struct sockaddr_in *ptr_adresse)
{
struct sockaddr_in adresse;       /* utilisé en local */
int desc;
int longueur = sizeof(struct sockaddr_in);

if ((desc = socket(AF_INET, type, 0)) == -1)
        {
        perror("Creation socket impossible\n");
        return -1;
        }

adresse.sin_family=AF_INET;
adresse.sin_addr.s_addr=htonl(INADDR_ANY);
adresse.sin_port=htons(*ptr_port);

if (bind(desc, &adresse, longueur) == -1)
        {
        perror("Attachement de la socket impossible\n");
        close(desc);
        return -1;
        }

if (ptr_adresse != NULL)
        {
        getsockname(desc, ptr_adresse, &longueur);  /* permet de remplir ptr_adresse */
        }

return desc;
}

main(int argc, char *argv[])
{
struct sockaddr_in adresse;
int lg_adresse;
int port;
int socketRV, socket_service;
if (argc!=2)
        {
        fprintf(stderr, "Nombre de parametres incorrect\n");
        exit(2);
        }
```

```
/* creation et attachement de la socket d'ecoute */
port=atoi(argv[1]);
if ((socketRV=creer_socket(SOCK_STREAM, &port, &adresse))==-1)
        {
        fprintf(stderr, "Creation socket ecoute impossible\n");
        exit(2);
        }

/* declaration d'ouverture du service */
if (listen(socketRV, 10)==-1)
        {
        perror("listen");
        exit(2);
        }

/* boucle d'attente de connexion */
while (TRUE) {
        lg_adresse=sizeof(adresse);
        socket_service=accept(socketRV, &adresse, &lg_adresse);

        /* reception d'un signal qui a interrompu l'attente sur l'accept */
        if (socket_service==-1 && errno==EINTR)
                {
                continue;
                }

        /* erreur plus grave */
        if (socket_service==-1)
                {
                perror("accept");
                exit(2);
                }

        printf("connexion acceptee\n");


        if (fork()==0)
                {
                /* lancement du processus de service = le fils */
                /* il n'utilise plus la socket d'ecoute */
                close(socketRV);

                /* suite à compléter en fonction de l'application*/
                …
                …
                …
                }  /* fin du fils */

        /* le serveur principal = le père, n'utilise pas socket_service */
        close(socket_service);
        }
}
```

UTILISATION DES SOCKETS - CLIENT TCP COMPLET

```
/* Lancement d'un client : client_tcp serveur port */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int creer_socket(int type, int *ptr_port, struct sockaddr_in *ptr_adresse)
{
struct sockaddr_in adresse;
int desc;
int longueur = sizeof(struct sockaddr_in);

if ((desc = socket(AF_INET, type, 0)) == -1)
        {
        perror("Creation socket impossible\n");
        return -1;
        }

adresse.sin_family=AF_INET;
adresse.sin_addr.s_addr=htonl(INADDR_ANY);
adresse.sin_port=htons(*ptr_port);

if (bind(desc, &adresse, longueur) == -1)
        {
        perror("Attachement de la socket impossible\n");
        close(desc);
        return -1;
        }

if (ptr_adresse != NULL)
        {
        getsockname(desc, ptr_adresse, &longueur);
        }
return desc;
}



main(int argc, char *argv[]){
int port;
int socket_client;
struct hostent *hp;
struct sockaddr_in adresse_serveur, adresse_client;

if (argc!=2)
        {
        fprintf(stderr, "Nombre de parametres incorrect\n");
        exit(2);
        }
```

```
/* test d'existence du serveur */
if ((hp=gethostbyname(argv[1]))==NULL)
        {
        fprintf(stderr, "Serveur %s inconnu\n", argv[1]);
        exit(2);
        }

/* creation et attachement de la socket du client (port quelconque) */
if ((socket_client=creer_socket(SOCK_STREAM, &port, &adresse_client))==-1)
        {
        fprintf(stderr, "Creation socket client impossible\n");
        exit(2);
        }
printf("client sur le port %d\n", ntohs(adresse_client.sin_port));

/* preparation de l'adresse du serveur */
adresse_serveur.sin_family=AF_INET;
adresse_serveur.sin_port=htons(atoi(argv[2]));
bcopy(hp->h_addr, &adresse_serveur.sin_addr, hp->h_length);


/* demande de connexion au serveur */
if (connect(socket_client, &adresse_serveur, sizeof(adresse_serveur))==-1)
        {
        perror("connect");
        exit(2);
        }
printf("connexion acceptee\n");

/* à compléter en fonction de l'application */
…
…
…
}
```

UTILISATION DES SOCKETS - EMETTEUR UDP

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

main(int argc, char **argv)
{
 int sock, envoye, recu;
 char buf[256];
 struct sockaddr_in adr;
 int lgadr;
 struct hostent *hote;

/* cr'eation de la socket */
 if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
        perror("socket"); exit(1);
        }

/* recherche de l'@ IP de la machine distante */
 if ((hote = gethostbyname(argv[2])) == NULL)
        {
        perror("gethostbyname");
        exit(2);
        }

/* pr'eparation de l'adresse distante : port + la premier @ IP */
 adr.sin_family = AF_INET;
 adr.sin_port = htons(atoi(argv[1]));
 bcopy(hote->h_addr, &adr.sin_addr, hote->h_length);
 printf("L'adresse en notation pointee %s\n", inet_ntoa(adr.sin_addr));

/* echange de datagrammes */
 strcpy(buf, "salut");
 lgadr = sizeof(adr);
 if ((envoye = sendto(sock, buf, strlen(buf)+1, 0, &adr, lgadr)) != strlen(buf)+1)
        {
        perror("sendto");
        exit(1);
        }

 printf("salut envoye\n");
 lgadr = sizeof(adr);
 if ((recu = recvfrom(sock, buf, 256, 0, &adr, &lgadr)) == -1)
        {
        perror("recvfrom"); exit(1);}
        printf("j'ai recu %s\n", buf);
        }
}
```

UTILISATION DES SOCKETS - RECEPTEUR UDP

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

main (int argc, char **argv)
{
 int sock,recu,envoye;
 char buf[256], nomh[50];
 struct sockaddr_in adr;
 int lgadr;
 struct hostent *hote;

/* cr'eation de la socket */
 if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
          {
           perror("socket"); exit(1);
          }

/* r'ecup'eration du nom de la machine pr'esente */
 if (gethostname(nomh, 50) == -1)
          {
          perror("gethostname");
          exit(1);
          }
 printf("Je m'execute sur %s\n", nomh);

/* pr'eparation de l'adresse locale : port + toutes les @ IP */
 adr.sin_family = AF_INET;
 adr.sin_port = htons(atoi(argv[1]));
 adr.sin_addr.s_addr = htonl(INADDR_ANY);

/* attachement de la socket a` l'adresse locale */
 lgadr = sizeof(adr);
 if ((bind(sock, &adr, lgadr)) == -1)
          {
          perror("bind");
          exit(1);
          }


/* 'echange de datagrammes */
 lgadr = sizeof(adr);
 if ((recu = recvfrom(sock, buf, 256, 0, &adr, &lgadr)) == -1)
          {
          perror("recvfrom");
          exit(1);
          }
 printf("j'ai recu %s\n", buf);

 strcpy(buf, "tchao");
```

```
  if ((envoye = sendto(sock, buf, strlen(buf)+1, 0, &adr, lgadr)) != strlen(buf)+1)
          {
          perror("sendto");
          exit(1);
          }
  printf("reponse envoyee ...adios\n");
}
```

SERVEUR AVEC SELECT

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define TRUE 1
#define BUFSIZE 512

#define NB_CLIENTS_MAX 10

main ( int argc, char **argv)
{
int socket_RV, socket_service;
char buf[BUFSIZE];
struct sockaddr_in adr;
int lgadr;
struct sockaddr_in adresse;
int lg_adresse;
struct hostent *hote;
u_short port;
int n, max, i;
fd_set fdset;
int clients[NB_CLIENTS_MAX];           /* tableau des sockets des clients   */
int nbclients;

if (argc != 2)
 {
 printf("Usage : executable port \n");
 exit(2);
 }

lgadr = sizeof(adr);
lg_adresse=sizeof(adresse);

/* creation de la socket */
if ((socket_RV = socket(AF_INET, SOCK_STREAM, 0)) == -1)
          {
          perror("socket");
          exit(1);
          }
/* conversion du port passe en argument */
port = atoi(argv[1]);

/* attachement de la socket de RV au port */
adr.sin_family = AF_INET;
adr.sin_port = htons(port);
adr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
if ((bind(socket_RV, (struct sockaddr *) &adr, lgadr)) == -1)
        {
        perror("bind");
        exit(1);
        }

/* declaration d'ouverture du service */
if (listen(socket_RV, 10)==-1)
        {
        perror("listen");
        exit(2);
        }

/* On doit toujours scruter la socket de rendez-vous, on la met dans la case 0 */

clients[0] = socket_RV;
nbclients = 1;

while (1)                                   //MacDo ouvert 24h/24
        {
        /* construction de l'ens a surveiller */
        FD_ZERO(&fdset);

        /* calcul du maximum  et construction du FD_SET */
        max = 0;
        for (i=0; i<nbclients; i++)
                {
                FD_SET(clients[i], &fdset);
                 if (max < clients[i])
                        {
                        max = clients[i];
                        }
                }

        /* select avec attente infinie */
        select(max+1, &fdset, NULL, NULL, NULL);

        /* retour du select */
        if (FD_ISSET(clients[0], &fdset))
                { /* chouette, un nouveau client  : acceptation du nouveau client */
                socket_service=accept(socket_RV, (struct sockaddr *)  &adresse, &lg_adresse);
                if (socket_service==-1)
                        {
                        perror("accept");
                        exit(2);
                        }
                printf("connexion acceptee\n");
                fflush(stdout);
                clients[nbclients]=socket_service;
                nbclients++;
                }
```

```
        /* test des clients actuels */
        for (i=1; i<nbclients; i++)
                {
                if (FD_ISSET(clients[i], &fdset))
                        {
                        /* c'est ce client qui veut qqch => on lit sa requete */
                        if (read(clients[i], buf, BUFSIZE) == -1)
                                {
                                 perror("read");
                                exit(1);
                                }
                printf("le client %d veut un %s\n", i, buf);
                fflush(stdout);

                /* reponse a la requete */
                if (write(clients[i], buf, strlen(buf)+1) == -1)
                        {
                         perror("write");
                        exit(1);
                        }
                printf("j'ai envoye un %s\n", buf);
                fflush(stdout);
                }
  } // MacDo never stops
}
```

CLIENT AVEC SELECT

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define BUFSIZE 512

main (argc, argv)
int argc;
char **argv;
{
int sock;
char buf[BUFSIZE];
int n, continuer;
struct sockaddr_in adresse_serveur;
struct hostent *hote;
u_short port;

/* test des paramètres d'entrée            */
if (argc != 3)
        {
        printf("Usage : executable port machine \n");
        exit(2);
        }

/* creation de la socket */
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
        perror("socket");
        exit(1);
        }

/* recherche des informations sur le destinataire */
hote = gethostbyname(argv[2]);

/* conversion du port passe en argument */
port = (u_short) atoi(argv[1]);


/* adresse du destinataire */
adresse_serveur.sin_family = AF_INET;
adresse_serveur.sin_port = htons(port);
bcopy(hote->h_addr, &adresse_serveur.sin_addr.s_addr, hote->h_length);
```

```
/* demande de connexion au serveur */
if (connect(sock, &adresse_serveur, sizeof(adresse_serveur))==-1)
        {
        perror("connect");
        exit(2);
        }
printf("connexion acceptee\n");
fflush(stdout);

continuer = 1;

 while(continuer)
        {
        /* requete au serveur */
        strcpy(buf, "un Big Mac");
        if (write(sock,buf,strlen(buf)+1) == -1)
                {
                perror("write");
                exit(1);
                }
        printf("j'ai commande un Big Mac\n");
        fflush(stdout);

        /* attente de la reponse */
        if (read(sock,buf,BUFSIZE) == -1)
                {
                perror("read");
                exit(1);
                }

        printf("j'ai recu un %s\n", buf);
        fflush(stdout);
        printf("Voulez-vous continuer, oui (1), non (0)\n");
        scanf("%d", &continuer);
        }


/* fermeture de la socket */
close(sock);
}
```