Responsable pédagogique	AF	AM	PB	**PM**
Période	Sem1	Sem2	**Sem3**	Sem4
Volume horaire	Cours/TD		TP	
			8	

PSYST
TP SOCK TCP & UDP

Indicateur temporel (hors rédaction du compte-rendu) :

questions	1h	2h	3h	4h	5h	6h	7h	8h
1								
2								
3								
4								

Documents à rendre : Compte-rendu contenant à minima les sources (avec entête standard) et les résultats obtenus.

Note : La fin des programmes client et serveur pourra être envisagée sur interception du signal SIGINT (lié à une séquence CTRL-C) Cf en Annexe 3 et 4. Les programmes client et serveur pourront dans un premier temps dialoguer par l'interface loopback (d'adresse ip 127.0.0.1); c'est à dire être sur la même machine. Le nom de l'hôte serveur (ex. localhost) pourra ensuite être saisi depuis la ligne de commande. Cf fig.1 Annexe 3.

1. Codage du serveur

Prendre le fichier serv_suj.c fourni ; et complétez-le en suivant les étapes en Annexe 1, le programme serveur , permettant de faire un écho de chaine reçue sur la socket de port 5000, en mode TCP. Le serveur accepte un seul client.

2. Codage du client

Prendre le fichier cli_suj.c fourni et suivre les étapes de l'annexe 2. Validez la communication effective entre votre serveur et client. Cf Annexe 3.

3. Gestion multi-clients

Modifier le code du serveur pour accepter un nombre de programmes clients supérieur à 1. (Annexe 5)

Indice : Utilisez la primitive *listen()* ; puis dupliquer le programme serveur pour suivre l'algorithme suivant :

DEBUT
Création de la socket ;
Attente multi clients sur la socket d'écoute;

REPETER
service : ENTIER ; { declarer une socket de service pour chaque demande cliente !}

accepter toute nouvelle demande ; { service = accept(....) ;}

Duplication du processus par code <- fork() ;

```
Si code  = 0 Alors
        /* code du fils */
        ferme socket ecoute ;
        Boucle de lecture et renvoi sur socket ;  /*  PARTIE déjà faite en Q1 */

        sortie de programmes ;

Sinon si code = -1 Alors
        /* cas d'erreur */
        sortie de programmes ;
Sinon           /* code du père */
        ferme socket service dupliquée ;   /* close(service) ; : Important car sinon double
                                              descripteur !! */
Finsi
 JUSQU'A Réception du signal SIGINT ;
FINPROG
```

4. Codage en UDP

Partez des sources fournis (si vous le souhaitez) cli_udp_suj.c et serv_udp_suj.c.
Complétez les codes précédents afin de travailler en mode non connecté.

Rappelez-vous que l'on travaille en mode non connecté, les comportements des programmes serveur et client sont identiques.

Pensez à utiliser recvfrom() et sendto() ! Cf cours. Le résultat attendu est le même qu'en TCP ; soit obtenir côté client, l'écho renvoyé par le serveur.
Cf annexe 6

Annexe 1 : étapes de création d'une socket serveur en TCP (en ipv4, et internet)

Etape 1 : La création d'une socket se fait à l'aide de la primitive *socket* dont le prototype est :
int socket(int domaine, int type, int protocole); **Cf man 7 socket** (page de manuel n°7)
Elle renvoie l'identifiant de la socket (sock_SERVEUR dans votre fichier). Les domaines possibles sont donnés dans le tableau : PRENDRE le nom symbolique pour le 1er paramètre domaine !

Nom symbolique	Domaine	Format des adresses
PF_UNIX	Local	sockaddr_un
PF_INET	Internet version 4	sockaddr_in
PF_INET6	Internet version 6	sockaddr_in6

Le 2ème paramètre type peut être entre autres : SOCK_STREAM (mode connecté), SOCK_DGRAM (mode non connecté).

Note : Pour le 3ème paramètre, mettre 0 dans protocole.

Etape 2 : Renseigner structure adresse locale de format struct sockaddr_in

Voici la définition du type sockaddr_in :

```
struct sockaddr_in
{
        short sin_family;  /* PF_INET , ...*/
        u_short sin_port;  /* numero de port supérieur à 1024 pour un service utilisateur*/
        struct in_addr sin_addr ;   /* Structure imbriquée => adresse IP machine distante */
        char sin_zero[8];  /* bourrage */
};

struct in_addr{
        u_long    s_addr;
};
```

Voici un appel typique en code C permettant de dire que le serveur accepte toutes adresses clients:

```
struct sockaddr_in myaddr ; /* format d'adresse locale du serveur de (type sockaddr_in) */
myaddr.sin_addr.s_addr = INADDR_ANY;
```

Renseigner ensuite le domaine (champ sin_family) puis le champ sin_port de votre structure locale.

Etape 3 : Attachement de la socket

Pour associer une adresse à cette socket, on fait appel à la primitive bind. Elle renvoie une valeur <0 sur un échec, et positive sur succès.

Prototype : int bind(int descripteur, struct sockaddr *ptr_adresse, int long_adresse);

Le premier argument est le descripteur de la socket, le 2ème est l' adresse(au sens du C) de la structure contenant les informations sur la socket. Le dernier paramètre est la taille de cette structure (pensez à **sizeof()**).

Etape 4 et 5 : Attente de connexions (primitive listen()) et acceptations (primitive accept()).

Appel typique :

Les variables déclarées en début de programme sont :

struct sockaddr_in myaddr , from; /* myaddr : format adresse locale , from : format adresse du client(distant)*/

```
/* connexions clientes en attente Max = 1  ici*/
        printf("attente de demande connexion");
        listen( sock_SERVEUR,1);

for(;;)    /* boucle infinie de traitement des connexions*/
{
        int new,len = sizeof(from);
        new = accept(sock_SERVEUR,(struct sockaddr *)&from,&len);
```

len contient la taille de l'adresse distante ; new est la nouvelle socket

Etape 6 : envoi / réception sur la socket nouvellement acceptée

Note : Prendre les fonctions read(), write par exemple et mettre cela dans la boucle de traitement des connexions

Annexe 2 : étapes de création d'une socket cliente en TCP

Etape 1 : Identique au serveur.

Etape 2 : Identique aussi **excepté pour l'initialisation champ adresse (sin_addr)** : il faut mettre ce champ à la valeur de l'adresse ip du serveur !!

Exemple de séquence d'appel en C :

```
unsigned long hostAddr = 0; /* adresse IP distante*/
struct sockaddr_in  sin; /* format adresse LOCALE */

bzero((char *) &sin,sizeof(sin));

hostAddr = (inet_addr("127.0.0.1"));  /* recup. adresse du serveur (ici localhost) */
/* adresse loopback à adapter si besoin !!*/

/* copie l'adresse serveur dans champ adresse (sin_addr) de votre structure locale */
bcopy(&hostAddr,sin.sin_addr,sizeof(hostAddr));
```

Macro retournant l'adresse ip
En int à partie d'une chaine

Copier l'adresse ip du
Prog. Distant dans le champ de l'adresse

Renseigner ensuite le domaine (champ sin_family) puis le champ sin_port de votre structure locale.

Etape 3 : demande de connexion au serveur par la primitive connect()

Prototype : int connect (int s , struct sockaddr *dest, int taille_dest) ;

Elle permet d'établir la connexion avec la socket serveur. Renvoie <0 si echec, >0 autrement (succès).

Le premier argument est l'identifiant de la socket cliente ; le 2ème est l'adresse (au sens du C) de la socket locale, et le 3ème sa taille (obtenue par sizeof(...)).

Appel typique :

```
struct sockaddr_in    sin;

if(connect(sock_CLIENT,(struct sockaddr *) &sin, sizeof(sin))<0)
        {
                perror("client:pb connexion au serveur par socket");
        exit(-1);
}
```

Etape 4 : Envoi / Réception sur la socket par write(), send() / read() , recv()

Annexe 3 : Exemples d'exécution

```
patrickMAC:correction maylaenderpatrick$ ./cli
hostAddr :  100007f
adresse 127.0.0.1
N socket cliente 3
test mono client
client de pid 17860
 17 carac recus du serveur >> test mono client
```

```
patrickMAC:correction maylaenderpatrick$ ./serv
attente de demande connexionnouveau client sur fd 4
DEBUT serveur
SERVEUR CIBLE Echo de test mono client
```

Fig. 1 : client et serveur en localhost

```
patrickMAC:c maylaenderpatrick$ ./cli 192.168.1.14
adresse 192.168.1.14
N socket cliente 3
exemple de saisie de l'adresse ip en 2eme argument de la ligne de commande
client de pid 17787
 75 carac recus du serveur >> exemple de saisie de l'adresse ip en 2eme argument
 de la ligne de commande
```

Fig.2 : client avec adresse ip (du serveur) saisie sur ligne de commande

```
nouveau client sur fd 4
DEBUT serveur
SERVEUR CIBLE Echo de exemple de saisie de l'adresse ip en 2eme argument de la l
igne de commande
```

Fig.3 : comportement inchangé du serveur

Annexe 4 : Rappel sur la gestion de signaux

Tout signal unix peut être associé à une routine de traitement (signal handler) que l'on installe en début de programme en général.

```
int sTCP;  //socket
void arret(int sig) {
        …  /* tout code utile : fermeture socket, libération ressources, etc ...*/
        kill(getpid(),SIGKILL);   /* prog se termine par envoi signal de destruction*/
}
….
main()
{
        /* Attache routine sortie correcte sur CTRL-C */
        signal(SIGINT, arret);
…}
```

Attache le Gestionnaire au signal

Annexe 5 : exemple d'exécution

```
patrickMAC:c maylaenderpatrick$ ./serv2
attente de demande connexionnouveau client sur fd 4
DEBUT serveur
SERVEUR CIBLE Echo de hello ici client 1

nouveau client sur fd 4
DEBUT serveur
SERVEUR CIBLE Echo de hello ici client 2
```

Fig.3 : serveur qui dialogue avec 2 clients (sur la même machine)

```
patrickMAC:c maylaenderpatrick$ ./cli
hostAddr :  100007f
adresse 127.0.0.1
N socket cliente 3
hello ici client 1
client de pid 17722
 19 carac recus du serveur >> hello ici client 1
```

```
patrickMAC:c maylaenderpatrick$ ./cli
hostAddr :  100007f
adresse 127.0.0.1
N socket cliente 3
hello ici client 2
client de pid 17726
 19 carac recus du serveur >> hello ici client 2
```

Fig.3 bis : client1 et 2

Annexe 6 :

```
maylaenderpatrick@ubuntu:~/Documents/TS2/socket/enC/correction$ ./cli_udp
hostAddr :  100007f
adresse 127.0.0.1

N socket cliente 3

client de pid 114933
 1 carac recus du serveur >>

hello ici le client udp
client de pid 114933
 24 carac recus du serveur >> hello ici le client udp
```

Fig. 4 Le client udp

```
maylaenderpatrick@ubuntu:~/Documents/TS2/socket/enC/correction$ ./serv_udp
hostAddr :  100007f
N socket serveur 3

serveur de pid 114932
 1 carac recus du client >>

serveur de pid 114932
 24 carac recus du client >> hello ici le client udp
```

Fig5. Le serveur udp