Responsable pédagogique	AF	AM	PB	**PM**
Période	Sem1	Sem2	**Sem 3**	Sem4
Volume horaire	Cours/TD		TP	
	1		**5**	

PSYST
TP SOCK UniCast

Indicateur temporel (hors rédaction du compte-rendu) :

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

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

Le but du TP est de coder un serveur multiclients. Partir du source fourni *servUNICASTsuj.c* dans PNET(sur rimatara).

Cf Annexe 2 pour les exemples d'exécutions et en Annexe 1 pour le code à compléter.

1. Codage du serveur

Codez un serveur unicast en suivant les étapes ci-dessous. Le programme serveur permettant d'envoyer/recevoir vers tous les programmes clients. Cela exploite la diffusion naturelle des réseaux. La socket côté serveur aura le port 5000, et sera en mode connecté .

1.1. Q1 Enregistrement de la socket d'écoute dans la structure fdset

Codez l'enregistrement de la socket d'écoute sTCP dans le descripteur de nom fdset. Utilisez la macro FD_SET().

1.2. Q2 Test d'écoute par select()

Codez l'écoute de tout nouveau descripteur entrant par select(), de façon à détecter une nouvelle connexion où n sera le code de retour , et *readfds* la structure en lecture. maxfd est le nombre max de clients. Attention au dernier paramètre du select, mettre un timeout.

1.3. Q3 Test de nouvelles données

Codez le test de présence de nouvelles données en lecture par la fonction FD_ISSET() ; où le descripteur à scruter est celui de votre socket d'écoute new , et la structure associée de type fd_set *readfs*.

Rappel extrait du man sur l'interface liée au select() (TSVP)

```
    select(int nfds, fd_set *restrict readfds, fd_set *restrict writefds,
        fd_set *restrict errorfds, struct timeval *restrict timeout);

ESCRIPTION
    select() examines the I/O descriptor sets whose addresses are passed in
    readfds, writefds, and errorfds to see if some of their descriptors are
    ready for reading, are ready for writing, or have an exceptional condi-
    tion pending, respectively.  The first nfds descriptors are checked in
    each set; i.e., the descriptors from 0 through nfds-1 in the descriptor
    sets are examined.  (Example: If you have set two file descriptors "4"
    and "17", nfds should  not be "2", but rather "17 + 1" or "18".)  On
    return, select() replaces the given descriptor sets with subsets consist-
    ing of those descriptors that are ready for the requested operation.
    select() returns the total number of ready descriptors in all the sets.

    The descriptor sets are stored as bit fields in arrays of integers.  The
    following macros are provided for manipulating such descriptor sets:
    FD_ZERO(&fdset) initializes a descriptor set fdset to the null set.
    FD_SET(fd, &fdset) includes a particular descriptor fd in fdset.
    FD_CLR(fd, &fdset) removes fd from fdset.  FD_ISSET(fd, &fdset) is non-
    zero if fd is a member of fdset, zero otherwise.  FD_COPY(&fdset_orig,
    &fdset_copy) replaces an already allocated &fdset_copy file descriptor
    set with a copy of &fdset_orig.  The behavior of these macros is unde-
    fined if a descriptor value is less than zero or greater than or equal to
    FD_SETSIZE, which is normally at least equal to the maximum number of
    descriptors supported by the system.

    If timeout is a non-nil pointer, it specifies a maximum interval to wait
    for the selection to complete.  If timeout is a nil pointer, the select
    blocks indefinitely.  To effect a poll, the timeout argument should be
    non-nil, pointing to a zero-valued timeval structure.  Timeout is not
    changed by select(), and may be reused on subsequent calls, however it is
    good style to re-initialize it before each invocation of select().

    Any of readfds, writefds, and errorfds may be given as nil pointers if no
    descriptors are of interest.

ETURN VALUES
    select() returns the number of ready descriptors that are contained in
    the descriptor sets, or -1 if an error occurred.  If the time limit
    expires, select() returns 0.  If select() returns with an error, includ-
    ing one due to an interrupted call, the descriptor sets will be unmodi-
    fied and the global variable errno will be set to indicate the error.

-----
```

1.4. Q4 Acceptation de connexion

Codez la création de la nouvelle socket de service sur un accept.

Rappel : En 1er , le descripteur de la socket d'ecoute.Il faut mettre l'adresse de la structure socket distante en 2eme argument.

La socket de service courante sera new.

1.5. Q5 Récupération du nom de machine

Codez la récupération du nom de machine distante via la primitive gethostbyaddr().

```
struct hostent *
gethostbyaddr(const void *addr, socklen_t len, int type);
```

The addr argument passed to **gethostbyaddr**() should point to an address which is len bytes long, in binary form (i.e., not an IP address in human readable ASCII form). The type argument specifies the address family (e.g. AF_INET, AF_INET6, etc.) of this address.

Conseil : Pour le 1er argument, prendre le l'adresse du champ sin_addr de la structure socket

1.6. Q6 Boucle de lecture des descripteurs

Compléter la lecture des données présentes sur l'ensemble des sockets de services ouvertes.

On testera si il y a bien une machine (un nom présent dans le tableau Ahost_info[ ]), et auquel cas, on testera la présence de données par appel à ioctl(new, FIONREAD, &bytes) . La variable bytes contient le nombre effectifs d'octets à lire sur le descripteur new !

Pensez à renvoyer la donnée lue sur la socket, à destination du client !!

Annexe 1 : code (à compléter à partir du source fourni servUNICASTsuj.c)

```
/*****************************************/
/* Prog serveur unicast multiclient TCP    */
/* PM v2                                   */
/* 2/10/18                                 */
/* gcc -Wall -o servUNICAST servUNICATsuj.c*/
/*****************************************/


#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>
#include <syslog.h>
#include <sys/stat.h>
#include <sys/ioctl.h>

#include <unistd.h>
#include <string.h>
#define DEBUG 11

int sTCP;
void arret(int sig)
{
    printf(" \nARRET PROG DE TEST SOCKET   ---> echo \n");
     close(sTCP);
    kill(getpid(),SIGKILL);

}

main()
{

    struct sockaddr_in  myaddr,from;
    struct hostent  *hp;    //pointeur de host
    char *Ahost_info[6];  //6 machines MAXIMUM

    char buf[100],res[100];
    int cc = 0, n, new, bytes = 0, maxfd = 0;
    int taille ;
    fd_set fdset, readfds ;

    /* init a NULL du nom de machine */
    for(new = 0; new <6 ; new ++)
        Ahost_info[new] = NULL;

    FD_ZERO(&fdset);
    FD_SET(sTCP,&fdset);

    /* Attache routine sortie correcte sur CTRL-C */
```

```
    signal(SIGINT, arret);

    /* init a zero des champs de la socket */
    bzero((char *)&myaddr,sizeof(myaddr));

//creation socket mode connecté
    if((sTCP =socket(PF_INET, SOCK_STREAM, 0)) < 0) {
           perror("serveur: Socket");
           exit(3);
    }

    myaddr.sin_family = PF_INET;
    myaddr.sin_addr.s_addr =  INADDR_ANY;

    myaddr.sin_port     = htons(5000); /* PORT personnalise */

    //Attachement
    if (bind(sTCP,(struct sockaddr*)&myaddr, sizeof(myaddr)) < 0) {

                printf("%8x",myaddr.sin_addr);
           perror("erreur de bind");
           exit(6);
    }

    /* attente connexions clientes 6  ici*/
    printf("attente de demande connexion");
    listen(sTCP,6);

    FD_ZERO( &fdset ) ;

    /* Q1 Enregistrement de la socket d'ecoute dans la structure fdset */
    FD_SET(             ) ;

    /* boucle ecoute des connexions clientes */
    while(1){
        bcopy(&fdset,&readfds,sizeof(fd_set));

        maxfd = 6 ;
        /* Q2 Test d'ecoute si nouvelle connexion */
          n = select(       ,         ,  NULL, NULL,       ) ;


             /*  Q3 TEST si nelle donnée entrante en lecture */
            if ( FD_ISSET(          , &readfds )) {
                taille = sizeof(myaddr);
                 /* Q4 accepte nouvelle connexion cliente */
          new = accept(          , (struct sockaddr *)&         ,
                        (socklen_t *)&taille);

                /* Q5 recupere info distante du nom de machine */
                   hp = gethostbyaddr(&            ,
                             sizeof(struct in_addr),PF_INET);
```

```
                    //recopie du nom de machine distante
                 strcpy(Ahost_info[new]=(char*)malloc(strlen(hp->h_name) + 1),
                        hp->h_name );
                   printf("Call using #%d from %s\n",new,Ahost_info[new]);


               }

            //boucle de parcours de toutes les connexions
            for(new = 0; new < maxfd; new ++)
               {
                  if(Ahost_info[new])   // il y a une machine au bout !!! du descripteur
                       {
                           //do{
                           //Q6 TEST si donnees arrivees
                                   ioctl(     ,         , &bytes ) ;
                           if(bytes)  //si il y a qqchose à lire !! sinon read bloquant
                           {
                               cc = read(                          );
                               if(cc>=1) {
                                   buf[cc]='\0';    /fin de chaine
                                    //erreur car machine deconnectee
                                   if(write(new,buf,cc) < 0)
                                   {
                                       free(Ahost_info[new]);
                                       Ahost_info[new] = NULL; // RAZ pteur
                                       close(new);
                                           //retire descripteur service de structure fdset
                                       FD_CLR(new,&fdset);
                                   }
                                    printf("SERVEUR CIBLE Echo de %s\n",buf);

                               }
                           }
                       }
               }
    }
    close(sTCP);
           exit(0);
}
```

Annexe 2 : Exemples d'exécution

```
maylaenderpatrick@ubuntu:~/Documents/TS2/socket/enC/correction$ telnet 192.168.1.20 5000
Trying 192.168.1.20...
Connected to 192.168.1.20.
Escape character is '^]'.
C3
C3
```

Figure 1 : un premier client telnet qui envoie C3 et récupère l'écho

```
patrickMAC:correction maylaenderpatrick$ ./cli
hostAddr :  100007f
adresse 127.0.0.1
N socket cliente 3
C2
client de pid 7107
 3 carac recus du serveur >> C2
```

Fig2 : un deuxième client (votre client TCP du TP1 est utilisable)

```
patrickMAC:correction maylaenderpatrick$ ./cli
hostAddr :  100007f
adresse 127.0.0.1
N socket cliente 3
C1
client de pid 7106
 3 carac recus du serveur >> C1
```

Fig 3 : un troisième client...

```
patrickMAC:correction maylaenderpatrick$ ./serv2bis
attente de demande connexionCall using #4 from localhost
SERVEUR CIBLE Echo de C1

Call using #5 from localhost
SERVEUR CIBLE Echo de C2

Call using #7 from ??
SERVEUR CIBLE Echo de C3
```

Fig 4 : Le serveur