Compte rendu tp3 seconde partie

1. Quelles sont les solutions que vous connaissez permettant de gérer plusieurs connexions simultanément?

Pour gérer plusieurs connexions simultanées, on peut appelé la fonction echo dans un fork ou un thread pour créer plusieurs processus et que le processus principal revienne à la fonction accept pour écouter d'éventuelle nouveaux clients.
On peut aussi utiliser select qui va avertir quand on peut lire quelque chose de nouveau dans un socket. Ainsi quand un nouveau client se connecte, le socket du serveur se met a jour et on utilise la fonction accept pour ajouté le socket du client nouvellement créer a la fonction select. Quand un client envoi des données sur ce socket, le serveur se charge de lire le socket client avec recv et de lui renvoyer le message avec send.

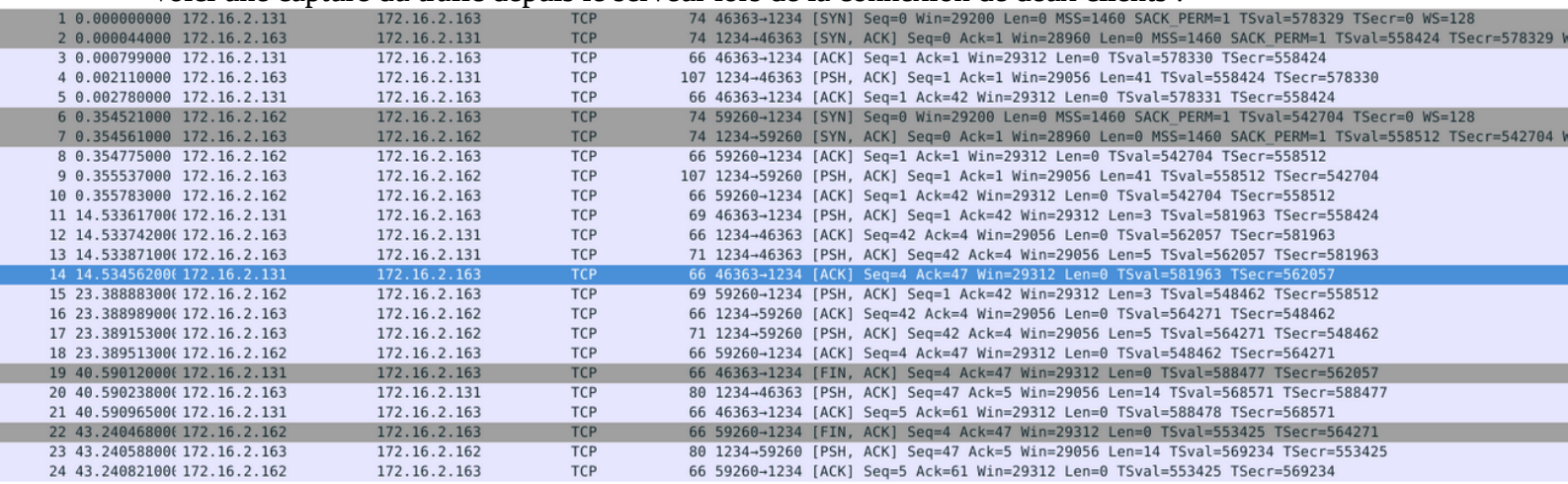
2. Modifier le code du serveur pour qu'il accepte plusieurs connexions simultanées.

Pour que le serveur accepte plusieurs connexions simultanée, j'ai ajouté un fork pour que le processus fils exécute la fonction echo et que le processus père retourne à la fonction accept :

```
74   while(1) {
75     /* attendre et gérer indéfiniment les connexions entrantes */
76     len=sizeof(struct sockaddr_in);
77     if( (n=accept(s,(struct sockaddr *)&client,(socklen_t*)&len)) < 0 ) {
78       perror("accept");
79       exit(7);
80     }
81     /* Nom réseau du client */
82     char hotec[NI_MAXHOST];  char portc[NI_MAXSERV];
83     err = getnameinfo((struct sockaddr*)&client,len,hotec,NI_MAXHOST,portc,NI_MAXSERV,0);
84     if (err < 0 ){
85       fprintf(stderr,"résolution client (%i): %s\n",n,gai_strerror(err));
86     }else{
87       fprintf(stderr,"accept! (%i) ip=%s port=%s\n",n,hotec,portc);
88     }
89     /* traitement */
90     if(fork() == 0)
91     {
92          echo(n,hotec,portc);
93          break;
94     }
95   }
96   return EXIT_SUCCESS;
97 }
```

3. Faire et commenter une capture de la connexion simultanée de deux clients au serveur.

Voici une capture du trafic depuis le serveur lors de la connexion de deux clients :



1	0.000000000	172.16.2.131	172.16.2.163	TCP	74	46363→1234 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=578329 TSecr=0 WS=128
2	0.000044000	172.16.2.163	172.16.2.131	TCP	74	1234→46363 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=558424 TSecr=578329 W
3	0.000799000	172.16.2.131	172.16.2.163	TCP	66	46363→1234 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=578330 TSecr=558424
4	0.002110000	172.16.2.163	172.16.2.131	TCP	107	1234→46363 [PSH, ACK] Seq=1 Ack=1 Win=29056 Len=41 TSval=558424 TSecr=578330
5	0.002780000	172.16.2.131	172.16.2.163	TCP	66	46363→1234 [ACK] Seq=1 Ack=42 Win=29312 Len=0 TSval=578331 TSecr=558424
6	0.354521000	172.16.2.162	172.16.2.163	TCP	74	59260→1234 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=542704 TSecr=0 WS=128
7	0.354561000	172.16.2.163	172.16.2.162	TCP	74	1234→59260 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=558512 TSecr=542704 W
8	0.354775000	172.16.2.162	172.16.2.163	TCP	66	59260→1234 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=542704 TSecr=558512
9	0.355537000	172.16.2.163	172.16.2.162	TCP	107	1234→59260 [PSH, ACK] Seq=1 Ack=1 Win=29056 Len=41 TSval=558512 TSecr=542704
10	0.355783000	172.16.2.162	172.16.2.163	TCP	66	59260→1234 [ACK] Seq=1 Ack=42 Win=29312 Len=0 TSval=542704 TSecr=558512
11	14.53361700(	172.16.2.131	172.16.2.163	TCP	69	46363→1234 [PSH, ACK] Seq=1 Ack=42 Win=29312 Len=3 TSval=581963 TSecr=558424
12	14.53374200(	172.16.2.163	172.16.2.131	TCP	66	1234→46363 [ACK] Seq=42 Ack=4 Win=29056 Len=0 TSval=562057 TSecr=581963
13	14.53387100(	172.16.2.163	172.16.2.131	TCP	71	1234→46363 [PSH, ACK] Seq=42 Ack=4 Win=29056 Len=5 TSval=562057 TSecr=581963
14	14.53456200(	172.16.2.131	172.16.2.163	TCP	66	46363→1234 [ACK] Seq=4 Ack=47 Win=29312 Len=0 TSval=581963 TSecr=562057
15	23.38888300(	172.16.2.162	172.16.2.163	TCP	69	59260→1234 [PSH, ACK] Seq=1 Ack=42 Win=29312 Len=3 TSval=548462 TSecr=558512
16	23.38898900(	172.16.2.163	172.16.2.162	TCP	66	1234→59260 [ACK] Seq=42 Ack=4 Win=29056 Len=0 TSval=564271 TSecr=548462
17	23.38915300(	172.16.2.163	172.16.2.162	TCP	71	1234→59260 [PSH, ACK] Seq=42 Ack=4 Win=29056 Len=5 TSval=564271 TSecr=548462
18	23.38951300(	172.16.2.162	172.16.2.163	TCP	66	59260→1234 [ACK] Seq=4 Ack=47 Win=29312 Len=0 TSval=548462 TSecr=564271
19	40.59012000(	172.16.2.131	172.16.2.163	TCP	66	46363→1234 [FIN, ACK] Seq=4 Ack=47 Win=29312 Len=0 TSval=588477 TSecr=562057
20	40.59023800(	172.16.2.163	172.16.2.131	TCP	80	1234→46363 [PSH, ACK] Seq=47 Ack=5 Win=29056 Len=14 TSval=568571 TSecr=588477
21	40.59096500(	172.16.2.131	172.16.2.163	TCP	66	46363→1234 [ACK] Seq=5 Ack=61 Win=29312 Len=0 TSval=588478 TSecr=568571
22	43.24046800(	172.16.2.162	172.16.2.163	TCP	66	59260→1234 [FIN, ACK] Seq=4 Ack=47 Win=29312 Len=0 TSval=553425 TSecr=564271
23	43.24058800(	172.16.2.163	172.16.2.162	TCP	80	1234→59260 [PSH, ACK] Seq=47 Ack=5 Win=29056 Len=14 TSval=569234 TSecr=553425
24	43.24082100(	172.16.2.162	172.16.2.163	TCP	66	59260→1234 [ACK] Seq=5 Ack=61 Win=29312 Len=0 TSval=553425 TSecr=569234

Voici ce qui est échangé :

→ Les 3 premières trames sont la « poignée de main » du premier client pour initialiser la connexion(connexion du client, acceptation du serveur, accusé de réception du client).

→ Les trames 4 et 5 sont l'envoi du message de bienvenue du serveur au nouveau client suivit de l'accusé de réception du client.

→ Les 3 trames suivantes sont la poignée de main du second client.

→ Les trames 9 et 10 sont l'envoi du message de bienvenue du serveur au nouveau client suivit de l'accusé de réception du client.

→ Les trames 11 et 12 sont l'envoi d'un message du premier client au serveur suivit de l'accusé de réception du serveur.

→ Les trames 13 et 14 sont l'envoi du même message du serveur au premier client suivis de l'accusé de réception du client.

→ De la trame 15 à la trame 18, il s'agit de la même chose mais pour le deuxième client.

→ Les trames 19, 20 et 21 correspondent à l'envoi du client d'un message pour prévenir le serveur qu'il se déconnecte. Le serveur lui répond qu'il a bien reçu l'information et le client envoi un accusé de réception.

→ Les trames 22, 23 et 24 font la même chose mais pour le second client.

4. Proposer un serveur mono-processus en utilisant select :

Maintenant, je vais utiliser la fonction select pour gérer les connexions multiple de clients au serveur :

```c
/* echo / serveur simpliste
   Master Informatique 2012 -- Université Aix-Marseille
   Emmanuel Godard
*/
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netdb.h>

/* taille maximale des lignes */
#define MAXLIGNE 80
#define CIAO "Au revoir ...\n"

char echo(int f, char* hote, char* port, char politesse);

int main(int argc, char *argv[])
{
  int s,n; /* descripteurs de socket */
  int len,on; /* utilitaires divers */
  struct addrinfo * resol; /* résolution */
  struct addrinfo indic = {AI_PASSIVE, /* Toute interface */
                           PF_INET,SOCK_STREAM,0, /* IP mode connecté */
                           0,NULL,NULL,NULL};
  char * port; /* Port pour le service */
  int err; /* code d'erreur */

  /* Traitement des arguments */
  if (argc!=2) { /* erreur de syntaxe */
    printf("Usage: %s port\n",argv[0]);
    exit(1);
  }

  port=argv[1]; fprintf(stderr,"Ecoute sur le port %s\n",port);
  err = getaddrinfo(NULL,port,&indic,&resol);
```

```c
  if (err<0){
    fprintf(stderr,"Résolution: %s\n",gai_strerror(err));
    exit(2);
  }

  /* Création de la socket, de type TCP / IP */
  if ((s=socket(resol->ai_family,resol->ai_socktype,resol->ai_protocol))<0) {
    perror("allocation de socket");
    exit(3);
  }
  fprintf(stderr,"le n° de la socket est : %i\n",s);

  /* On rend le port réutilisable rapidement /!\ */
  on = 1;
  if (setsockopt(s,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on))<0) {
    perror("option socket");
    exit(4);
  }
  fprintf(stderr,"Option(s) OK!\n");

  /* Association de la socket s à l'adresse obtenue par résolution */
  if (bind(s,resol->ai_addr,sizeof(struct sockaddr_in))<0) {
    perror("bind");
    exit(5);
  }
  freeaddrinfo(resol); /* /!\ Libération mémoire */
  fprintf(stderr,"bind!\n");

  /* la socket est prête à recevoir */
  if (listen(s,SOMAXCONN)<0) {
    perror("listen");
    exit(6);
  }
  fprintf(stderr,"listen!\n");
  fd_set rdfs;
  int actual = 0;
  int max = s;
  int clients[42];
```

```c
  while(1)
  {
     int i = 0;
     int j = 0;
     FD_ZERO(&rdfs);
     FD_SET(s, &rdfs);
     for( i = 0; i < actual; i++)
     {
          FD_SET(clients[i], &rdfs);
     }
     if(select(max + 1, &rdfs, NULL, NULL, NULL) == -1)
     {
          printf("error\n");
          exit(2);
     }
     struct sockaddr_in csin;
     len=sizeof(struct sockaddr_in);
     char hotec[NI_MAXHOST];
     char portc[NI_MAXSERV];
```

```
97      if(FD_ISSET(s, &rdfs))
98      {
99           printf("Un nouveau client\n");
100          len = sizeof csin;
101          if( (n = accept(s, (struct sockaddr *) &csin, (socklen_t*)&len)) < 0 )
102          {
103                    perror("accept");
104                    exit(7);
105          }
106          err = getnameinfo((struct sockaddr*)&csin,len,hotec,NI_MAXHOST,portc,NI_MAXSERV,0);
107          if (err < 0 )
108          {
109                    fprintf(stderr,"résolution client (%i): %s\n",n,gai_strerror(err));
110          }
111          else
112          {
113                    fprintf(stderr,"accept! (%i) ip=%s port=%s\n",n,hotec,portc);
114          }
115          echo(n,hotec,portc, 1);
116          max = n > max ? n : max;
117          FD_SET(n, &rdfs);
118          clients[actual++] = n;
119     }
```

```
120     else
121     {
122          for(j = 0; j < actual; j++)
123          {
124                    /* a client is talking */
125                    if(FD_ISSET(clients[j], &rdfs))
126                    {
127                    err = getnameinfo((struct sockaddr*)&clients[j],len,hotec,NI_MAXHOST,portc,NI_MAXSERV,0);
128                    if ( echo(clients[j],hotec,portc, 0) == 0)
129                    {
130                         close(clients[j]);
131                         //remove client socket
132                         memmove(clients + j, clients + j + 1, (actual - j - 1) * sizeof(int));
133                         actual--;
134                    }
135               }
136          }
137     }
138  }
139  return EXIT_SUCCESS;
140 }
```

```
142 /* echo des messages reçus (le tout via le descripteur f) */
143 char echo(int f, char* hote, char* port, char politesse)
144 {
145   ssize_t lu; /* nb d'octets reçus */
146   char msg[MAXLIGNE+1]; /* tampons pour les communications */
147   char tampon[MAXLIGNE+1];
148   int pid = getpid(); /* pid du processus */
149
150   /* message d'accueil */
151   if(politesse)
152   {
153      snprintf(msg,MAXLIGNE,"Bonjour %s! (vous utilisez le port %s)\n",hote,port);
154      /* envoi du message d'accueil */
155      send(f,msg,strlen(msg),0);
156   }
157   else
158   {
159      /* Faire echo et logguer */
160      lu = recv(f,tampon,MAXLIGNE,0);
161
162      if (lu <= 0 )
163      {
164           return 0;
165      }
166      tampon[lu] = '\0';
167      fprintf(stderr,"[%s:%s](%i) :%s\n",hote,port,pid,tampon);
168      snprintf(msg,MAXLIGNE,"> %s",tampon);
169         /* echo vers le client */
170      send(f, msg, strlen(msg),0);
171   }
172   return 1;
173 }
```