Compte rendu : TP4 Socket (partie 2)

Auteur : PELLETIER Sébastien M1 Info TP1b

Programmation

Les solutions à ma connaissances pour rendre un serveur multi-client sont :

- **Select** (On vérifie à chaque itération si le socket à été modifier en lecture ou écriture pour agir sur celui-çi)
- **Multi-processus / threading** (Le principe est de traiter chaque client dans une autre exécution parallèle du programme que ce soit via un **fork** ou un **thread**)

Voici un version modifier de echoserveur.c en multi-client, j'utilise ici un fork() pour executer la fonction « echo » qui traite toutes les demandes du client. Le traitement des demandes s'exécutent seulement dans le processus fils (fork() == 0) quand le client coupe la connection, le fils sort de la boucle (break) et se termine.

La seule modification consiste à l'ajout du contenu en sélection dans la boucle while principale.

```
  while(1) {
    /* attendre et gérer indéfiniment les connexions entrantes */
    len=sizeof(struct sockaddr_in);
    if( (n=accept(s,(struct sockaddr *)&client,(socklen_t*)&len)) < 0 ) {
      perror("accept");
      exit(7);
    }
    /* Nom réseau du client */
    char hotec[NI_MAXHOST];  char portc[NI_MAXSERV];
    err = getnameinfo((struct sockaddr*)&client,len,hotec,NI_MAXHOST,portc,NI_MAXSERV,0);
    if (err < 0 ){
      fprintf(stderr,"résolution client (%i): %s\n",n,gai_strerror(err));
    }else{
      fprintf(stderr,"accept! (%i) ip=%s port=%s\n",n,hotec,portc);
    }
    /* traitement */
    if(fork() == 0)
    {
        echo(n,hotec,portc);
        break;
    }
  }
  return EXIT_SUCCESS;
}
```

Voici une capture de wireshark après que deux clients se soit connectés au serveur et et fait une requête sur le serveur multi-client fork avec quelques annotations :

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 TSec
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=
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 TSec
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=
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

Connexion du premier client & envoi du msg de bienvenu

Connexion du 2em client & envoi du msg de bienvenu

Envoi & ré-envoi d'une requete du client 1

Envoi & ré-envoi d'une requete du client 2

Fermeture de la connexion du client 1

Fermeture de la connexion du client 2

Pour la version multi-client et mono-processus j'ai choisi d'utiliser la fonction select en C. Pour faire cela j'ai modifié la fonction echo initialement codé pour traiter une seule « requête » du client. j'ai aussi créé une structure Client contenant le socket d'un client ainsi que le port et l'ip de celui-ci en tant que char * pour plus de simplicité. Chaque client est ajouté à une liste de Client quand il se connecte.
Il ne reste plus qu'à vérifier dans l'ordre, le socket serveur pour vérifier si un nouveau client se connecte puis chaque client pour vérifier s'il envoient des données au serveur.

```c
#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"

#define NBCLIENT_MAX 100

typedef struct
{
  int    sock;
  char * host;
  char * port;
} Client;


int  echoSelect  (Client client);
void echoWelcome (Client client);
void DeleteClient(Client* clients, int i, int* len_clients);

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

```

```c
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

  Client clients[NBCLIENT_MAX]; //Liste des clients.
  fd_set rdfs;                  //Ensemble des descripteur du select.

  int len_clients = 0; //taille de la liste des client (nb de client connecté).
  int max_socket;      //Socket plus grand (pour le select).
  int i = 0;

  /* 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);
  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");

  //Initialisation du plus grand socket.
  max_socket = s;
```

```
104   while(1) {
105     //mise en place du select
106     FD_ZERO(&rdfs);
107
108     //Ajout du soket serveur
109     FD_SET(s, &rdfs);
110
111     //Ajout de chaque client
112     for( i = 0; i < len_clients; i++)
113       FD_SET(clients[i].sock, &rdfs);
114
115     if( select(max_socket + 1, &rdfs, NULL, NULL, NULL) == -1 )
116       {
117         perror("select()");
118         exit(7);
119       }
120
121     // Si il y a un nouveau client
122     if( FD_ISSET(s, &rdfs) )
123       {
124         struct sockaddr_in client; /* adresse de socket du client */
125         len = sizeof(struct sockaddr_in);
126         if( (n = accept(s, (struct sockaddr *) &client, (socklen_t*) &len) ) < 0 )
127           {
128             perror("Erreur à la réception d'un nouveaux client.");
129             exit(7);
130           }
131
132         //mise à jour du plus grand socket.
133         if( n > max_socket )
134           max_socket = n;
135
136         //Ajout du nouveau client au select.
137         FD_SET(n, &rdfs);
138
139         // Nom réseau du client.
140         char hotec[NI_MAXHOST];
141         char portc[NI_MAXSERV];
142         err = getnameinfo( (struct sockaddr*) &client, len, hotec, NI_MAXHOST, portc, NI_MAXSERV, 0);
143         if (err < 0 )
144           fprintf(stderr, "résolution client (%i): %s\n", n, gai_strerror(err));
145         else
146           fprintf(stderr, "accept! (%i) ip=%s port=%s\n", n, hotec, portc);
147
148         //Ajout du nouveau client à la liste.
149         Client new_client = {n, hotec, portc};
150         clients[len_clients] = new_client;
151         len_clients++;
152
153         //Message de bienvenu.
154         echoWelcome( clients[len_clients - 1] );
155
156       }
157     else // Sinon un client nous parle.
158       {
159         for( i = 0; i < len_clients; i++) //On parcour touts les clients
160         {
161             if( FD_ISSET(clients[i].sock, &rdfs) ) //traitement du client
162               if( echoSelect(clients[i]) < 1) //Si le client s'est déconnecté.
163                 DeleteClient(clients, i, &len_clients);
164         }
165       }
166   }
167   return EXIT_SUCCESS;
168 }
```

```c
void DeleteClient(Client* clients, int i, int* len_clients)
{
  printf("Supression du client no : %d\n", i);
  if( *len_clients > 1 )
    {
      int j;
      for( j = i; j < (*len_clients - 1); j++)
        clients[j] = clients[j + 1];
    }
  (*len_clients)--;
}

void echoWelcome(Client client)
{
  char msg[MAXLIGNE + 1];
  snprintf(msg, MAXLIGNE, "Bonjour %s! (vous utilisez le port %s)\n", client.host, client.port);
  send(client.sock, msg, strlen(msg), 0);
}

int echoSelect(Client client)
{
  ssize_t lu;                   /* nb d'octets reçus */
  char    msg[MAXLIGNE + 1] ;  /* tampons pour les communications */
  char    tampon[MAXLIGNE+1];
  int     pid      = getpid(); /* pid du processus */
  int     compteur = 0;

  //Info client.
  char* hote          = client.host;
  char* port          = client.port;
  int   socket_client = client.sock;

  lu = recv(socket_client, tampon, MAXLIGNE, 0);

  if (lu > 0 )
    {
      compteur++;
      tampon[lu] = '\0';
      /* log */
      fprintf(stderr, "[%s:%s](%i): %3i :%s", hote, port, pid, compteur, tampon);
      snprintf(msg, MAXLIGNE, "> %s", tampon);
      /* echo vers le client */
      send(socket_client, msg, strlen(msg), 0);
      return 1;
    }
  else
    {
      /* le correspondant a quitté */
      send(socket_client, CIAO, strlen(CIAO), 0);
      close(socket_client);
      fprintf(stderr, "[%s:%s](%i): Terminé.\n", hote, port, pid);
      return 0;
    }
}
```