#include <stdlib.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdio.h>

#include <errno.h>

# Exercice 1 :

for(;;)

SELECT (ensemble de socket suivis en lecture)

if un descripteur de socket dispo en lecture est le socket d’écoute

then faire ACCEPT

⇒ sauvegarde le nouveau socket de dialog, ajouter ce socket à ceux suivis par select

endif

while il existe des sockets du dialogue dispo en lecture

traiter les données (recv/send)

si le client a demandé la fermeture fermer le socket, le retirer de suivis

finsi

fin

Structure de données utilisées par SELECT :

fd\_set\_readset; ---> Inititalise à -1 puis rempli avec la socket

pour le manipuler : FD\_ZERO → remet tout à -1

FD\_SET ----> mettre tout à -1 d’abord

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| fd1 | fd2 | -1 | -1 |  |

FD\_CLEAR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -1 | fd2 (maxfdp) | -1 | -1 |  |

FD\_ISSET ---> Renvoi Vrai si le socket est dans le select

***Lien du tp :***

<http://thibault.cholez.free.fr/teaching/RSI/socket/TP-socket-C.pdf>

// Exo 1 Question 1).

#include <stdio.h>

#include <stdlib.h>

#include <sys/select.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <string.h>

#include <unistd.h>

//lit ce qui est recu sur sockfd et réécris la meme chose sur sockfd

int str\_echo(int sockfd){

int nrcv, nsnd;

char msg[BUFSIZ];

memset((char \*)msg,0,sizeof(msg));

if((nrcv=read(sockfd,msg,sizeof(msg)))){

perror("read error on socket");

exit(1);

}

msg[nrcv]='\0';

printf("msg reçu = %s", msg);

if((nsnd=write(sockfd,msg,nrcv))<0){

perror("write error on socket");

exit(1);

}

return(nsnd);

}

int main(int argc, char const \*argv[])

{

// Déclarations

int sockfd , newsockfd;

struct sockaddr\_in serv\_addr , cli\_addr ;

socklen\_t clilen = sizeof(cli\_addr);

int tab\_clients[FD\_SETSIZE];

fd\_set rset,pset;

int maxfdp1,nbfd,i,sockcli;

// Initialisation

maxfdp1 = sockfd+1;

for (i=0;i<FD\_SETSIZE;i++) {

tab\_clients[i]=-1;

}

FD\_ZERO(&rset);

FD\_SET(sockfd, &rset);

for(;;){

pset = rset;

nbfd = select(maxfdp1, &rset, NULL, NULL, NULL);

if(FD\_ISSET(sockfd,&rset)){

newsockfd = accept(sockfd,(struct sockaddr\*)&cli\_addr, &clilen);

i=0;

while((i<FD\_SETSIZE)&&(tab\_clients[i]>=0)) {

i++;

}

if (i==FD\_SETSIZE){

exit(1);

}

tab\_clients[i] = newsockfd;

FD\_SET(newsockfd, &rset);

if (newsockfd >= maxfdp1){

maxfdp1 = newsockfd+1;

}

nbfd --;

}

//parcourir le tableau des clients

i=0;

while((nbfd>0) && (i<FD\_SETSIZE)){

if(((sockcli=tab\_clients[i])>=0) && FD\_ISSET(sockcli, &rset)){

if(str\_echo(sockcli)==0){ // the server, for each client, reads data from the client and echo it back to the client

//le client a fermé sa connexion

close(sockcli);

tab\_clients[i]=-1;

FD\_CLR(sockcli,&rset);

}

nbfd--;

}

i++;

}

}

return 0;

}

## Exercice 2 :

Question 1 : Préciser, par un schéma, les primitives réseau/système que le serveur inetd devra appeler avant de se mettre en attente de requêtes clients.

inctd ⇒ démon écoute sur les ports de n services qu’une requête est reçue, démarre le service correspondant.

socket()

bind()

listen()

select()

pour chaque service configuré dans /etc/inetd.conf

Question 2 : Comment sera réalisée cette attente ?

comme inetd peut écouter sur plusieurs ports, il attend sur le SELECT().

Question 3 : Un client UDP ou TCP arrive. Illustrer par un schéma pour mettre en évidence l’algorithme effectué par

le serveur.

socket()

bind()

listen()

select()

pour chaque service configuré dans /etc/inetd.conf

newfd = accept (si TCP)

fork

close newfd

close tous les descripteurs

sauf newfd (TCP)

sauf fd (UDP)

setgid()

setuid()

exec() service

si TCP

parent

fils

Q1

Q2

Q3

Question 4 : Le flag nowait spécifie que le serveur inetd n’a pas besoin d’attendre que son fils se termine avant d’accepter une nouvelle connexion pour ce service. Pourquoi est-ce le cas avec des services utilisant TCP et non pas avec un service comme tftp ?

**En TCP :**

* accept retourne un nouveau descripteur qui est utilisé par le processus parent pour continuer l’écoute

**En UDP :**

* Le processus parent doit faire un wait pid et attendre la fin d'exécution du fils avant d’écouter à nouveau. Sinon il risque de forker pour rien

Question 5 : Que faut-il ajouter dans le processus tftp pour permettre à plusieurs clients tftp de se connecter simultanément ?

tftp doit bind un nouveau socket avec l’@IP du client et utiliser ce nouveau port pour envoyer la réponse