Gestione avanzata delle Socket in C: la primitiva *select*

TCP e UDP Client/Server usando select

Si progetti un'applicazione distribuita Client/Server per una rete di workstation UNIX (BSD oppure System V). In particolare, si realizzi un servizio di echo in cui il Server sia in grado di rispondere a richieste provenienti sia da socket TCP sia da socket UDP.

Il Client TCP deve presentare l'interfaccia

echoTCP nodoserver msg

mentre il Client UDP deve presentare l'interfaccia

echoUDP nodoserver msg

dove ***nodoserver*** specifica l'indirizzo logico del nodo contenente il processo Server, e ***msg*** è la stringa che si vuole inviare, come stream nel primo caso e come datagram nel secondo.

Il Server, che si lega alla porta 12345 (sia per le socket TCP che per quelle UDP), provvede a rispedire ***msg*** al Client con la stessa modalità con cui l'ha ricevuta, deve quindi essere in grado di multiplexare l'output sui due tipi di socket usando la primitiva ***select,*** e deve inoltre gestire le richieste in maniera concorrente.

Proposte di modifica:

- Rendere il Server multiservizio. Il Server deve offrire due servizi che utilizzino il protocollo UDP (echo, e un altro a scelta) e due servizi che utilizzino il protocollo TCP (per esempio remote copy e remote sort).

  I Client devono presentare l'interfaccia:

  nomeClient nodoServer nomeServizio

  Il server discrimina il servizio da erogare in base alla socket da cui proviene la richiesta (usando la select) e in base al terzo argomento (per esempio con un confronto tra stringhe).

```
/* TCPEchoClient.c */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define DIM_BUFF 100


main(int argc, char **argv)
{
struct hostent *host;
struct sockaddr_in servaddr;
int  sd, conn_ok, nread;
char buff[DIM_BUFF];


if (argc!=3) {
      printf("Error! Usage: echoTCP nodoserver msg\n");
      exit(1);
}

memset((char *)&servaddr, 0, sizeof(servaddr));

sd=socket(AF_INET, SOCK_STREAM, 0);
if(sd<0) {perror("apertura socket"); exit(1);}
else printf("TCPClient: creata la socket con fd=%d\n", sd);

servaddr.sin_family = AF_INET;

host = gethostbyname (argv[1]);
if (host == NULL)
{printf("%s not found in /etc/hosts\n", argv[1]); exit(1);}

printf("TCPClient: indirizzo server : %s \n", host->h_name);

servaddr.sin_addr.s_addr=((struct in_addr *)(host->h_addr))->s_addr;
servaddr.sin_port = 12345;

conn_ok=connect(sd,(struct sockaddr *) &servaddr, sizeof(struct sockaddr));
if(conn_ok<0) { perror("Errore in connect"); exit(1);}
printf("TCPClient: connect ok\n");

if (write(sd, argv[2], strlen(argv[2])+1)<0)
{perror("write"); exit(1);}
printf("TCPClient: inviato msg %s \n", argv[2]);

if (nread=read(sd, buff, DIM_BUFF)<0)
{perror("read"); exit(1);}
printf("TCPClient: ricevo echo -> %s\n", buff);



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

```
/* UDPEchoClient.c */


#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define DIM_BUFF 100



main(int argc, char **argv)
{
struct hostent *host;
struct sockaddr_in servaddr;
int  sd, i, addrlen;
char buff[DIM_BUFF];


if (argc!=3) {
      printf("Error! Usage: echoUDP nodoserver msg\n");
      exit(1);
}


memset((char *)&servaddr, 0, sizeof(servaddr));

sd=socket(AF_INET, SOCK_DGRAM, 0);
if(sd<0) {perror("apertura socket"); exit(1);}
else printf("UDPClient: creata la socket con fd=%d\n", sd);

servaddr.sin_family = AF_INET;

host = gethostbyname (argv[1]);
if (host == NULL)
{printf("%s not found in /etc/hosts\n", argv[1]); exit(1);}

printf("UDPClient: indirizzo server: %s \n", host->h_name);

servaddr.sin_addr.s_addr=((struct in_addr *)(host->h_addr))->s_addr;
servaddr.sin_port = 12345;

if (sendto(sd, argv[2], (strlen(argv[2]) + 1), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr))<0)
      {perror("sendto"); exit(1);}
printf("UDPClient: inviato msg %s \n", argv[2]);


if (recvfrom(sd, buff, DIM_BUFF, 0, (struct sockaddr *)&servaddr, &addrlen)<0)
      {perror("recvfrom"); exit(1);}
printf("UDPClient: ricevo echo -> %s\n", buff);


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

```
/* EchoServer.c */


#include <stdio.h>
#include<signal.h>
#include<errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define DIM_BUFF 100
#define max(a,b) ((a) > (b) ? (a) : (b))



main(int argc, char **argv)
{
int  listenfd, connfd, udpfd, nready, maxfdp1;
char buffTCP[DIM_BUFF], buffUDP[DIM_BUFF];
fd_set rset;
int n, len, nread, nwrite;
const int on = 1;
struct sockaddr_in cliaddr, servaddr;

void gestore(int signo)
{
 int stato;
 printf("esecuzione gestore di SIGCHLD\n");
 wait(&stato);
}


/* creazione socket d'ascolto */
listenfd=socket(AF_INET, SOCK_STREAM, 0);
if(listenfd <0) {perror("apertura socket d'ascolto"); exit(1);}
printf("Server: creata la socket d'ascolto fd=%d\n", listenfd);


/* inizializzazione indirizzo server */
memset ((char *)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = 12345;

if(setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on))<0)
{perror("settaggio opzioni socket d'ascolto"); exit(1);}
printf("Server: settaggio opzioni socket d'ascolto ok\n");

if(bind(listenfd,(struct sockaddr *) &servaddr, sizeof(servaddr))<0)
{perror("bind socket d'ascolto"); exit(1);}
printf("Server: bind socket d'ascolto ok\n");

if (listen(listenfd, 5)<0)
{perror("listen"); exit(1);}
printf("Server: listen ok\n");


/* creazione socket UDP */
udpfd=socket(AF_INET, SOCK_DGRAM, 0);
```

```
if(udpfd <0) {perror("apertura socket UDP"); exit(1);}
printf("Server: creata la socket UDP fd=%d\n", udpfd);

/* inizializzazione indirizzo server */
memset ((char *)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = 12345;


if(bind(udpfd,(struct sockaddr *) &servaddr, sizeof(servaddr))<0)
{perror("bind socket udp"); exit(1);}
printf("Server: bind socket udp ok\n");


sigset(SIGCHLD, gestore);


FD_ZERO(&rset);
maxfdp1=max(listenfd, udpfd)+1;

for(;;){
      FD_SET(listenfd, &rset);
      FD_SET(udpfd, &rset);

if ((nready=select(maxfdp1, &rset, NULL, NULL, NULL))<0){
      if (errno==EINTR) continue;
      else {perror("select"); exit(1);}
}

if (FD_ISSET(listenfd, &rset)){ /* richiesta proveniente da client TCP */
      len = sizeof(cliaddr);
      if((connfd = accept(listenfd,(struct sockaddr *)&cliaddr,&len))<0) {
            if (errno==EINTR) continue;
            else {perror("accept"); exit(1);}
      }
      if (fork()==0){
            close(listenfd);
            if(nread=read(connfd, buffTCP, DIM_BUFF)<0)
                  { perror("read"); exit(1);}
            printf("Server: stringa ricevuta da client TCP -> %s\n", buffTCP);
            if(nwrite=write(connfd, buffTCP, DIM_BUFF)<0)
                  { perror("write"); exit(1);}
            exit(0);
      }
      close(connfd);
} /* if TCP */

if (FD_ISSET(udpfd, &rset)){ /* richiesta proveniente da client UDP */
      len=sizeof(cliaddr);
      if ((n=recvfrom(udpfd, buffUDP, DIM_BUFF, 0, (struct sockaddr *)&cliaddr,
                                                            &len))<0)
                  {perror("recvfrom"); exit(1);}
      printf("Server: stringa ricevuta da client UDP -> %s\n", buffUDP);
      if (sendto(udpfd, buffUDP, n, 0, (struct sockaddr *)&cliaddr, len)<0)
                  {perror("sendto"); exit(1);}
      } /* if UDP */

   } /* for */

}
```