Esercitazioni di base sulle Socket in C

Primo esercizio: Socket stream con ridirezione

Si progetti un'applicazione distribuita Client/Server per una rete di workstation UNIX (BSD oppure System V). In particolare, si vuole realizzare un ordinamento remoto di un file. Il Client presenta l'interfaccia:

remote_sort nodoserver file fileordinato

dove ***nodoserver*** specifica l'indirizzo logico del nodo contenente il processo Server, ***file*** è il nome assoluto del file che deve essere ordinato e ***fileordinato*** è il nome assoluto del file ordinato.

Il Client deve inviare il file di nome ***file*** al Server, che provvede a eseguirne l'ordinamento e a rispedirlo al Client stesso che lo salva nel nuovo file ***fileordinato***.

Il Server, che si lega alla porta 12345, deve essere in grado di gestire più richieste concorrentemente e deve svolgere due operazioni:

- controllare che il Client sia autorizzato a richiedere il servizio, verificando la presenza dell'indirizzo (espresso come nome logico) del Client nel file "autorizzati.txt";
- eseguire l'operazione di ordinamento in ordine alfabetico delle linee del file ricevuto dal Client;

Proposte di modifica:

- il client specifica al server la modalità di ordinamento, dalla a alla z o viceversa (opzione –r del comando sort);

- il client anziché inviare un file invia stringhe lette da console, che il server inserisce in un file da lui creato fino al ricevimento della stringa "stop", dopodiché ordina il file creato e lo spedisce al client;

- …

```
/* Client_stream.c*/

#include<stdio.h>
#include<fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define DIM_BUFF 256


main(int argc, char *argv[])
{
 int sd, fd, fd_ord, nread;
 char buff[DIM_BUFF];
 int fromlen, conn_ok;
 struct hostent *host;    /*ptr a info per host remoto*/
 struct sockaddr_in rem_indirizzo; /*per indirizzo socket remota*/


 if(argc!=4) {
      printf("Error:%s server file_da_ord file_dest_ord\n", argv[0]);
      exit(1);
 }

 memset ((char *)&rem_indirizzo, 0, sizeof(struct sockaddr_in));
      /* Preparazione indirizzo remoto a cui connettersi */
 rem_indirizzo.sin_family = AF_INET;
 host = gethostbyname(argv[1]);
 if (host == NULL) {
        printf("%s not found in /etc/hosts\n", argv[1]);
        exit(1);
 }
 rem_indirizzo.sin_addr.s_addr=
                              ((struct in_addr*)(host->h_addr))->s_addr;
 rem_indirizzo.sin_port = 12345; /*possibile uso htons()*/

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

 conn_ok=connect(sd,(struct sockaddr *) &rem_indirizzo,
                                                sizeof(struct sockaddr));

if(conn_ok<0) { perror("Errore in connect"); exit(1);}
else printf("Client: connect ok\n");

 if ((nread=read(sd, buff, DIM_BUFF))<0) {perror("read"); exit(1);}

 if(buff[0]=='S') {
      if((fd=open(argv[2], O_RDONLY))<0) {perror("open"); exit(1); }
      if((fd_ord=open(argv[3], O_WRONLY|O_CREAT, 0644))<0) {
            perror("open"); exit(1);
      }
      printf("Client: stampo e invio file da ordinare\n");
      while((nread=read(fd, buff, DIM_BUFF))>0){
            write(1,buff,nread);
            write(sd,buff,nread);
      }
      printf("Client: file inviato\n");
      shutdown(sd,1);
      printf("Client: ricevo e stampo file ordinato\n");
```

```
      while((nread=read(sd,buff,DIM_BUFF))>0){
            write(fd_ord,buff,nread);
            write(1,buff,nread);
      }

 } else printf("Client: non sono autorizzato \n");
 close(sd); exit(0);
}
```

```
/* Server_stream.c*/

#include<stdio.h>
#include<string.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 256

/* gestore per raccogliere lo stato di terminazione del figlio ed evitare che
diventi zombie */
void gestore(int signo)
{
 int stato;
 printf("esecuzione gestore di SIGCHLD\n");
 wait(&stato);
}

main(int argc, char *argv[])
{
 int sd, ns, fd, flag=0;
 char buff[DIM_BUFF];
 int fromlen, status, nread, nwrite;

 struct sockaddr_in mio_indirizzo, rem_indirizzo;
 struct hostent *host;
 memset((char *)&mio_indirizzo, 0, sizeof(struct sockaddr_in));
 fromlen=sizeof(struct sockaddr_in);

 sigset(SIGCHLD, gestore);
 sd=socket(AF_INET, SOCK_STREAM, 0); /*socket d'ascolto*/
 if(sd<0) {perror("apertura socket"); exit(1);}
 else printf("Server: creata la socket sd=%d\n", sd);


 mio_indirizzo.sin_family=AF_INET;
 mio_indirizzo.sin_port=12345;

 if(bind(sd,(struct sockaddr *) &mio_indirizzo, sizeof(struct sockaddr_in))<0) {
      perror("bind"); exit(1);
 }
 else printf("Server: bind ok, mi metto in ascolto\n");

 listen(sd,5);
```

```
 for(;;) {

      if((ns=accept(sd,(struct sockaddr *)&rem_indirizzo,&fromlen))<0) {
      if (errno==EINTR) {
            perror("Forzo la continuazione della accept");
            continue;
      } else exit(1);
   }


   if (fork()==0) { /* figlio */
      close(sd);
      host=gethostbyaddr( (char *) &rem_indirizzo.sin_addr,
                                    sizeof(rem_indirizzo.sin_addr), AF_INET);
        if (host == NULL) {
            printf("host information not found\n"); exit (1);
        }
         else printf("Server (figlio): host client e' %s \n", host->h_name);

      /* controllo di autorizzazione */

      if ((fd=open("autorizzati.txt", O_RDONLY))<0){
            perror("apertura del file autorizzati");}
      else
            while((nread=read(fd, buff, DIM_BUFF))>0) {

               if (strstr( buff, host->h_name)!=0)
                  flag=1;
            }

      if(flag==0) {
            if ((nwrite=write(ns, "N", 1))<0)
                  {perror("write autorizzazione negata"); exit(1);}
            printf("Server (figlio): autorizzazione negata\n");
            exit(0);
      }
      else {
            if ((nwrite=write(ns, "S", 1))<0)
                  {perror("write autorizzazione ok"); exit(1);}
            printf("Server (figlio): autorizzazione OK\n");

            printf("Server (figlio): eseguo l'ordinamento\n");
            close(1); close(0);
            dup(ns); dup(ns);
            execl("/bin/sort", "sort", (char *)0);
      }
   }
 close(ns);
}
}
```

Secondo esercizio: Socket datagram

Si progetti un'applicazione distribuita Client/Server utilizzando le chiamate di sistema UNIX. Il Client deve offrire la seguente interfaccia:

send_msg nodo1 nodo2 … nodoN stringa

dove ***nodo1***, .. ***nodoN*** sono nomi logici di nodi sulla rete Internet e ***stringa*** è una stringa di caratteri qualsiasi.

Il Client deve inviare la stringa a tutti i processi Server in esecuzione sui nodi passati come parametro.

Ogni Server deve visualizzare la stringa ricevuta sulla "console". Il Server si deve legare alla porta 22375.

Proposte di modifica:

- il server rispedisce il msg ricevuto al client (echo), il quale lo stampa a video specificando da quale server proviene;
- il client indica al server a quale indirizzo inoltrare il msg, che sarà poi stampato a video da chi lo riceve;
- …

```
/* Client_datagram.c */

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

main(int argc, char **argv)
{
struct hostent *host;   /*ptr a info per host remoto*/
struct sockaddr_in indirizzo_server; /*per indirizzo socket remota*/
int  sock, i, lun;

memset((char *)&indirizzo_server, 0, sizeof(struct sockaddr_in));

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

indirizzo_server.sin_family = AF_INET;

for (i=1; i<=argc-2; i++) {
      host = gethostbyname (argv[i]);
      if (host == NULL) {
            printf("%s not found in /etc/hosts\n", argv[i]);
            exit(2);
      }
      else {
            printf("Client: indirizzo server n. %d: %s \n", i, host->h_name);
            indirizzo_server.sin_addr.s_addr=
                                 ((struct in_addr *)(host->h_addr))->s_addr;
            indirizzo_server.sin_port = 22375;
            sendto(sock, argv[argc-1], (strlen(argv[argc-1]) + 1), 0,
                              (struct sockaddr *)&indirizzo_server,
                                                sizeof(struct sockaddr_in));
            printf("Client: inviato msg %s \n", argv[argc-1]);
      }
} /*end for*/
close(sock);
exit(0);
}
```

```
/* Server_datagram.c */

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define MAX_DIM 20

main(int argc, char **argv)
{
int  sock, retval, addrlen;
char buff[MAX_DIM];
struct sockaddr_in mio_indirizzo;  /* local socket address */
struct sockaddr_in indirizzo_client;  /* peer socket address */

memset ((char *)&mio_indirizzo, 0, sizeof(struct sockaddr_in));
memset ((char *)&indirizzo_client, 0, sizeof(struct sockaddr_in));

/* inizializzazione mio indirizzo */
mio_indirizzo.sin_family=AF_INET;
mio_indirizzo.sin_port=22375;

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

if(bind(sock, (struct sockaddr *)&mio_indirizzo, sizeof(struct sockaddr_in))<0)
      {perror("bind"); exit(1); }
else printf("Server: bind ok\n");

for (;;) {
  recvfrom(sock, buff, MAX_DIM, 0,
                  (struct sockaddr *)&indirizzo_client, &addrlen);
  write(1, buff, strlen(buff));
  printf("\n");
}
close(sock);
exit(0);
}
```