ESERCITAZIONE 3

Socket in C (funzionalitá di base)

OBIETTIVI

Sviluppare un protocollo di trasferimento file basato sul socket senza connessione(UDP) e con connessione(TCP).

Davide di Molfetta Mirko Legnini Daniele Nanni Cirulli Natanaele Stagni Lorenzo Venerandi

UDP

Sviluppare un servizio che fornisce informazioni sulla parola più lunga in un file di testo presente su un file server remoto.

SPECIFICHE CLIENT UDP:

- Il Client chiede all'utente il nome di un file, invia al server una richiesta con il nome inserito; infine attende una risposta che indica il numero di caratteri della parola più lunga del file .
- Se il file è presente sul server, riceve un intero, altrimenti una notifica di errore.

SPECIFICHE SERVER UDP:

- Riceve il nome del file.
- Se esiste, identifica la parola formata dal maggior numero di lettere.
- Invia al client un intero che indica il numero di lettere della parola più grande.

CLIENT UDP

Controllo argomenti

Creazione ed apertura socket

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

Assegnazione IP e porta

```
if (bind(sd, (struct sockaddr *)&clientaddr, sizeof(clientaddr)) < 0)
{
    perror("bind socket ");
    exit(1);
}
printf("Client: bind socket ok, alla porta %i\n", clientaddr.sin_port);
```

```
while (gets(req) != NULL)
{
    /* richiesta operazione */
    len = sizeof(servaddr);
    if (sendto(sd, &req, sizeof(char) * FILENAME_MAX, 0, (struct sockaddr *)&servaddr, len) < 0)
    {
        perror("sendto");
        continue;
    }

    /* ricezione del risultato */
    printf("Attesa del risultato...\n");
    if (recvfrom(sd, &ris, sizeof(ris), 0, (struct sockaddr *)&servaddr, &len) < 0)
    {
        perror("recvfrom");
        continue;
    }

    if ((int)ntohl(ris) < 0)
        printf("Il file %s non esiste!\n\n", req);
    else
        printf("Esito dell'operazione: %i\n", (int)ntohl(ris));
    printf("Inserire nome file, ^D per terminare\n");

} // while gets
```

- Invio richiesta contenente il nome file
- Attesa risposta (o num. caratteri parola o errore)

SERVER UDP

Controllo argomenti

Creazione e connessione socket

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

if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0)
{
    perror("set opzioni socket ");
    exit(1);
}
printf("Server: set opzioni socket ok\n");

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

int lenght = 0, f;
char c;
```

Ciclo di accettazione richieste

```
for (;;)
{
    ris = -1;
    lenght = 0;

    len = sizeof(struct sockaddr_in);
    if (recvfrom(sd, req, sizeof(char) * FILENAME_MAX, 0, (struct sockaddr *)&cliaddr, &len) < 0)
    {
        perror("recvfrom ");
        continue;
    }

    clienthost = gethostbyaddr((char *)&cliaddr.sin_addr, sizeof(cliaddr.sin_addr), AF_INET);
    if (clienthost == NULL)
        printf("client host information not found\n");
    else
        printf("Operazione richiesta da: %s %i\n", clienthost->h_name, (unsigned)ntohs(cliaddr.sin_port));
```

Conto lettere

```
if (f = open(req, O_RDONLY))
{
    while (read(f, &c, sizeof(char)) > 0)
    {
        lenght++;

        if (c == ' ' || c == '\n')
        {
            if (lenght > ris)
                ris = lenght - 1;
            //printf("%d\n", lenght-1);

            lenght = 0;
        }
    }
}
```

Invio risposta al Client

```
ris = htonl(ris);
if (sendto(sd, &ris, sizeof(ris), 0, (struct sockaddr *)&cliaddr, len) < 0)
{
    perror("sendto ");
    continue;
}
```

TCP

Sviluppare un servizio che elimina una linea all'interno di un file di testo presente sul file system del cliente.

SPECIFICHE CLIENT TCP:

- Il Client invia il nome del file e il numero di linea (forniti dall'utente) ad un Server.
- Stampa a video la risposta del server

SPECIFICHE SERVER TCP:

- Ad ogni richiesta, il padre genera un processo figlio.
- Il processo figlio riceve il file, effettua l'eliminazione richiesta e spedisce il risultato al client.

CLIENT TCP

Controllo argomenti

Creazione, apertura socket e bind

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

/* Operazione di BIND implicita nella connect */
if (connect(sd, (struct sockaddr *)&servaddr, sizeof(struct sockaddr)) < 0)
{
    perror("connect");
    exit(1);
}
printf("Client: connect ok\n");
```

Invio del n. riga, invio del file e chiusura socket spedizione

```
printf("Invio numero riga");
write(sd, &riga, sizeof(int));

/*INVIO File*/
printf("Client: stampo e invio file da ordinare\n");
while ((nread = read(fd, buff, DIM_BUFF)) > 0)
{
    write(1, buff, nread);  //stampa
    write(sd, buff, nread); //invio
}
printf("Client: file inviato\n");
/* Chiusura socket in spedizione -> invio dell'EOF */
shutdown(sd, 1);
close(fd);
```

Ricevo il file ordinato e chiudo la socket in ricezione

```
printf("Client: ricevo e stampo file ordinato\n");
while ((nread = read(sd, buff, DIM_BUFF)) > 0)
{
    write(fd, buff, nread);
    write(1, buff, nread);
}
printf("Traspefimento terminato\n");
/* Chiusura socket in ricezione */
shutdown(sd, 0);
/* Chiusura file */

close(fd);
close(sd);
```

SERVER TCP

Controllo argomenti

Creazione e connessione socket

```
listen_sd = socket(AF_INET, SOCK_STREAM, 0);
if (listen_sd < 0)
{
    perror("creazione socket ");
    exit(1);
}
printf("Server: creata la socket d'ascolto per le richieste di eliminazione, fd=%d\n", listen_sd);

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

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

if (listen(listen_sd, 5) < 0) //creazione coda d'ascolto
{
    perror("listen");
    exit(1);
}
printf("Server: listen ok\n");

signal(SIGCHLD, gestore);
```

Ciclo di accettazione richieste

```
for (;;)
{
    len = sizeof(cliaddr);
    if ((conn_sd = accept(listen_sd, (struct sockaddr *)&cliaddr, &len)) < 0)
    {

        if (errno == EINTR)
        {
            perror("Forzo la continuazione della accept");
            continue;
        }
        else
            exit(1);
    }
```

Creazione figlio

```
if (fork() == 0)
{ // figlio

    close(listen_sd);
    host = gethostbyaddr((char *)&cliaddr.sin_addr, sizeof(cliaddr.sin_addr), AF_INET);
    if (host == NULL)
    {
        printf("client host information not found\n");
        continue;
    }
    else
        printf("Server (figlio): host client e' %s \n", host->h_name);
```

Rimozione linea e invio risposta al Client

```
printf("Ricevo riga da eliminare\n");
read(conn_sd,&riga,sizeof(int));
printf("Riga: %d\n",riga);

printf("Server (figlio): eseguo la rimozione\n");
while (read(conn_sd, &c, sizeof(char)) > 0)
{
    line[ic] = c;
    ic++; // Indice carattere linea

    if (c == '\n')
    {
        if (ir != riga){
            write(conn_sd, line, ic);
            write(1, line, ic);
        }
        ir++; // Indice riga
        ic = 0;
    }
}
```

CONCLUSIONI

- In entrambi i casi il Server agisce come filtro a carattere.
- Il Server TCP è sequenziale, quindi riesce a sostenere molteplici connessioni senza dilatare i tempi di esecuzione.