



# Presentazione dell'esercitazione 3

## Conteggio Parole / Elimina Linea Socket C senza e con connessione

Alice Turrini - Serena Bertaccini – Anna Vandi – Caterina Leonelli

Anno accademico 2021/2022

# Obiettivi

---

## ***Senza connessione:***

realizzare il **server parallelo**, dato che l'operazione da compiere su ogni file è read-only.

## ***Con connessione:***

Il server deve eliminare la linea del file senza salvarlo in locale, ma agendo solo sull'input della socket.


## ***Con connessione:***

Il server deve poter mandare comunque l'esito al cliente anche in caso di lettura non completata.

Uso **ottimizzato** e ridotto delle risorse

# ClientDatagram:

```
/* SET UP DEL CLIENT: inizializzazione indirizzo client e server, verifica
intero, verifica port e host, creazione socket, bind socket a una porta
scelta dal sistema
/* CORPO DEL CLIENT: ciclo di accettazione di richieste da utente */
printf("Inserire nome file remoto, EOF per terminare: ");
gets(temp);
while(temp != NULL){
    strcpy(req.nomeFile, temp);
    /* richiesta operazione */
    len=sizeof(servaddr);
    if(sendto(sd, &req, sizeof(Request), 0,
        (struct sockaddr *)&servaddr, len)<0){
        /*errore*/ continue;}
    /* ricezione del risultato */
    if ((recvfrom(sd, &ris, sizeof(ris), 0,
        (struct sockaddr *)&servaddr, &len))<0){
        /*errore*/ continue;}
    printf("Esito operazione: lunghezza parola più grande e' %d\n",
        (int)ntohl(ris), ris );
    printf("Inserisci nomefile remoto, EOF per terminare: ");
    gets(temp);
} //while utente
close(sd); exit(0); //CLEAN OUT
}
```



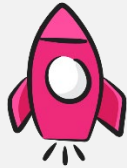
# ServerDatagram

```
/* CONTROLLO ARGOMENTI, INIZIALIZZAZIONE INDIRIZZO SERVER, CREAZIONE, SETAGGIO  
OPZIONI E CONNESSIONE SOCKET */
```

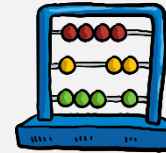
```
for(;;){ /* CICLO DI RICEZIONE RICHIESTE */
```



```
    len=sizeof(struct sockaddr_in);  
    if (recvfrom(sd, req, sizeof(Request), 0,  
                (struct sockaddr *)&cliaddr, &len)<0){ /*errore*/}  
    if ((fd=open(req->nomeFile, O_RDONLY)) < 0){ /*errore*/}  
    else{
```



```
        if((pid=fork())<0){ /*errore*/}  
        else if(pid==0){ //processo figlio  
            while((nread=read(fd, &c, sizeof(char)))) {  
                if(nread<0){ /*invio al client messaggio di errore*/  
                    ris=-ris;  
                    if (sendto(sd, &ris, sizeof(ris), 0,  
                                (struct sockaddr *)&cliaddr, len)<0){ /*errore*/}  
                    if(c != ' ' && c!='\n') count++;  
                    else{ if(count>ris)ris=count;  
                        count=1;  
                    }  
                } //fine while  
            }  
            ris=htonl(ris);  
            if (sendto(sd, &ris, sizeof(ris), 0, (struct sockaddr *)&cliaddr,  
                      len)<0){ /*errore*/ }  
            close(fd); //fine figlio!  
        } else{ /*codice padre*/  
    } //for, demone
```



# Esempio di esecuzione datagram

```
student@student:~/eclipse-workspace/Es3_11$ ./client 127.0.0.1 4555
Client: creata la socket sd=3
Client: bind socket ok, alla porta 2345
Inserire nome file remoto, EOF per terminare: prova.txt
Ho letto il nomeFile: prova.txtAttesa del risultato...
Esito dell'operazione: lunghezza della parola più grande e' 0
Inserisci nomefile remoto, EOF per terminare: ^C
student@student:~/eclipse-workspace/Es3_11$
```

```
student@student:~/eclipse-workspace/Es3_12$ ./server 4555
Server: creata la socket, sd=3
Server: set opzioni socket ok
Server: bind socket ok
Sono il padre, ho creato il figlio (pid: 1070)
Figlio (pid 1070): Lunghezza max del file prova.txt e' 0
```

# ClientStream

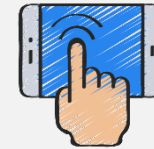
```
/* CLIENT SETUP: CONTROLLO ARGOMENTI, INIZIALIZZAZIONE INDIRIZZO SERVER,  
VERIFICA INTERO, VERIFICA PORT e HOST
```

```
/* CORPO DEL CLIENT:
```

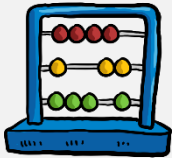
```
//INTERAZIONE UTENTE:
```

```
printf("[ClientStream] Inserire nome file remoto, EOF per terminare: \n");
```

```
do{//client cicla fino a che il cliente non immette EOF  
gets(req.nomeFile);
```



```
    if((fd = open(req.nomeFile, O_RDONLY)) < 0){/*error*/  
//conto il numero di righe che il file contiene
```



```
        tot_righe=0;  
        while( read(fd, &ch, sizeof(char))>0 )  
            if( ch=='\n') tot_righe++;  
        tot_righe++;
```



```
typedef struct{  
    char  
    nomeFile[MAX_LENGTH];  
    long num_riga;  
}Request;
```



```
printf("[ClientStream] Inserire numero riga da eliminare e premere invio:\n");
```



```
while((ok=scanf("%lu", &req.num_riga)!=1)){ /*errore e svuoto stdin*/ }  
if(req.num_riga<=0 && req.num_riga>tot_righe){ /*errore*/ }
```

# ClientStream

```
/* CREAZIONE SOCKET e Operazione di BIND implicita nella connect */
/*INVIO File e riga da eliminare*/
 write(sd, &req.num_riga, sizeof(long));
printf("Client: stampo e invio file da ordinare\n");
lseek(fd, SEEK_SET, 0);
while((nread=read(fd, buff, DIM_BUFF))>0) write(sd,buff,nread);
 close(fd);
shutdown(sd,1); /* Chiusura socket in spedizione -> invio dell'EOF */

/*RICEZIONE File*/
if((fd = open(req.nomeFile, O_TRUNC|O_WRONLY)) < 0){/*error*/}
 while((nread=read(sd,buff,DIM_BUFF))>0){
    write(1,buff,nread);
    write(fd,buff,nread);
}
 close(fd);
shutdown(sd, 0); /* Chiusura socket in ricezione */
close(sd);
printf("\n[ClientStream] Inserisci nomefile remoto, EOF per terminare: \n");

}while(gets(req.nomeFile)!=NULL );

/* terminazione classe Client stream*/
```

# ServerStream:

```
/* controllo argomenti, inizializzazione indirizzo server, settaggi socket d'ascolto*/
```

```
for(;;){
```

```
    //settaggio con la accept
```

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

```
        char c; int nread, riga_corrente=1;
```

```
        if((nread=read(conn_sd, &req->num_riga,sizeof(int)))<0){
```



```
            while((nread=read(conn_sd, &c, sizeof(char)))){
```

```
                if(req->num_riga !=riga_corrente){
```



```
                    if (write(conn_sd, &c,sizeof(char))<0){/*errore*/}
```

```
                        if(c=='\n') riga_corrente++;
```

```
                    } //fine while lettura carattere
```

```
                    shutdown(conn_sd, 1);//chiudo l'output
```

```
                    exit(EXIT_SUCCESS);
```

```
                } //fine figlio!
```

```
            } // fine ciclo for infinito
```

```
            close(conn_sd);// padre chiude socket di connessione non di ascolto
```

```
}
```



# Esempio di esecuzione stream

```
student@student:~/eclipse-workspace/Es3_21$ ./client 127.0.0.1 4555
[ClientStream] Inserire nome file remoto, EOF per terminare:
prova.txt
Il file ha 2 righe
[ClientStream] Inserire numero riga da eliminare e premere invio:
1
Client: crea la socket sd=4
Client: connect ok
[ClientStream] Spedita riga da eliminare
Client: stampo e invio file da ordinare
riga3

[ClientStream] Spedito contenuto file
Client: ricevo e stampo file modificato
```

```
student@student:~/eclipse-workspace/Es3_22$ ./server 4555
Server: crea la socket d'ascolto per le richieste di ordinamento, fd=3
Server: set opzioni socket d'ascolto ok
Server: bind socket d'ascolto ok
Server: listen ok
(PID 1793) ho letto il numero di linea 1
riga3
esecuzione gestore di SIGCHLD
```

# Conclusioni

---

**Server parallelo**, importanza gestire le chiusure degli opportuni socket descriptor tra padre e figlio;

Importanza della gestione di eventuali **fallimenti** delle **primitive di connessione**

**Nel Server Stream** non è stato possibile fare una lettura bufferizzata da file perché abbiamo dovuto **discriminare le righe**, al fine di trovare quella da eliminare.

Necessario l'uso **ottimizzato** e ridotto delle risorse, soprattutto quando si lavora nel distribuito