Università degli Studi di Bologna
Scuola di Ingegneria

Corso di

Reti di Calcolatori T

Esercitazione 4 (svolta)

Server Multiservizio: Socket C con select

Antonio Corradi, Luca Foschini
Michele Solimando, Giuseppe Martuscelli,
Marco Torello

Anno accademico 2022/2023

SERVER MULTISERVIZIO: CLIENT

Sviluppare **un'applicazione C/S** in cui **uno stesso server fornisce due servizi, richiesti da due tipi diversi di client**: **il conteggio del numero di file contenuti in un direttorio** e **il trasferimento di file dal server al client (`get`)**:

- Il **primo** tipo di **Client** chiede all'utente il nome del direttorio di cui vuole conoscere il numero di file, lo invia al server in un pacchetto di richiesta, e attende il pacchetto con la risposta

- Il **secondo** tipo di **Client** chiede all'utente il nome del **file testo** da trasferire, e usa una connessione per inviare il **nome del file** selezionato al servitore e ricevere il contenuto del file richiesto

Si noti che entrambi i clienti sono filtri e accettano ciclicamente richieste dall'utente, fino alla fine del file di input da console: per ogni ciclo fanno una richiesta di operazione relativa.

SERVER MULTISERVIZIO: IL SERVER

Il **Server** discrimina i **due tipi di richiesta utilizzando la primitiva *select***

Le richieste di **conteggio dei file** di un direttorio vengono gestite in maniera sequenziale **usando una socket datagram per la comunicazione**: il server riceve il pacchetto con il nome del direttorio dal client, esegue il conteggio dei file, e invia al client un pacchetto di risposta con il numero ottenuto

Le richieste di **get di un file** vengono gestite **usando una socket connessa per la comunicazione e con un server concorrente multiprocesso**. Il server usa la connessione con il client per ricevere il nome file e per inviare il file richiesto, dopodiché chiude la connessione (*assumiamo file testo*)

Si noti che sono possibili diversi schemi architetturali e di protocollo

SCHEMA DI SOLUZIONE: IL CLIENT DATAGRAM 1/2

Inizializzazione indirizzo del server dall'argomento di invocazione:

```
memset((char *)&servaddr, 0, sizeof(struct
sockaddr_in));
servaddr.sin_family = AF_INET;
host = gethostbyname (argv[1]);
servaddr.sin_addr.s_addr =
   ((struct in_addr *)(host->h_addr))->s_addr;
servaddr.sin_port = htons(atoi(argv[2]));
```

Creazione e bind socket datagram:

```
sd=socket(AF_INET, SOCK_DGRAM, 0);
bind(sd,...);
```

SCHEMA DI SOLUZIONE: IL CLIENT DATAGRAM

Ciclo di interazione con l'utente e di invio di richieste al client:

- Lettura dei dati da console (nome del direttorio) …
- Invio richiesta operazione al server:

```
sendto(sd, nome_dir, strlen(nome_dir)+1, 0,
   (struct sockaddr *)&servaddr, len)
```

- Ricezione risposta contenente il risultato dal server:

```
recvfrom(sd, &num_file, sizeof(num_file), 0,
   (struct sockaddr *)&servaddr, &len)
```

Fine ciclo, chiusura socket:

```
close(sd);
```

SCHEMA DI SOLUZIONE: IL CLIENT STREAM 1/2

Inizializzazione indirizzo del server dall'argomento di invocazione:

```
memset((char *)&servaddr, 0,
  sizeof(struct sockaddr_in));
servaddr.sin_family = AF_INET;
host = gethostbyname (argv[1]);
servaddr.sin_addr.s_addr =
  ((struct in_addr *)(host->h_addr))->s_addr;
servaddr.sin_port = htons(atoi(argv[2]));
```

Creazione e connessione socket stream:

```
sd=socket(AF_INET, SOCK_STREAM, 0);
connect(sd,...);
```

PROTOCOLLO DEL SERVIZIO STREAM

Sono possibili due schemi di soluzione (costi?)

Schema 1: una connessione per ogni richiesta

Una connessione è usata per un solo trasferimento di file e viene aperta ad ogni ciclo di richiesta di uno stesso cliente e chiusa alla fine del trasferimento del file

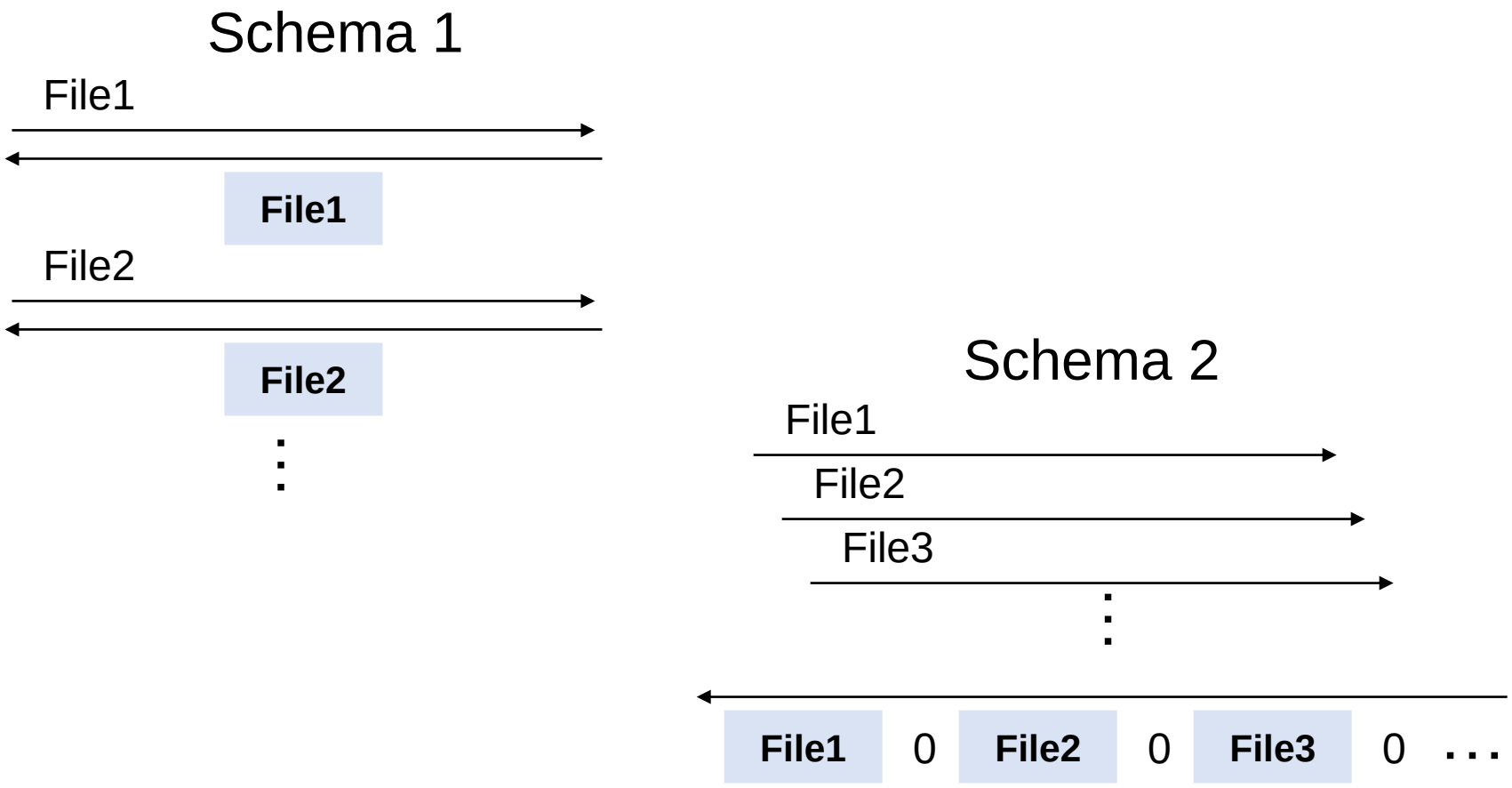
- **il protocollo applicativo è semplificato** ☺
- **ma il costo è superiore** ☹

Schema 2: Una sola connessione per tutta la sessione cliente

Una stessa connessione è il veicolo per tutti i trasferimenti di file successivi per lo stesso cliente

- **il costo in risorse è limitato** ☺
- **il protocollo applicativo è più complesso** ☹

ARCHITETTURA PER PARTE CON CONNESSIONE



SCHEMA 1 DI SOLUZIONE: IL CLIENT STREAM

Schema 1: una connessione per trasferimento

ossia per ogni ciclo si attua una richiesta su una connessione che si può chiudere al termine del file

Ciclo di interazione con l'utente e di invio di richieste dal client al server

→ connect **all'interno del ciclo**

- Lettura da console dei dati della richiesta (nome file da ricevere)
- Ricezione del file ordinato dal server:

lettura a blocchi fino alla chiusura della connessione da parte del server che viene identificata con la fine del file

```
close(sd)
```

Fine ciclo

SCHEMA 2 DI SOLUZIONE: IL CLIENT STREAM

Schema 2: una sola connessione per sessione

→ connect **prima di entrare nel ciclo**

Ciclo di interazione con l'utente e di invio di richieste dal client al server, una per ogni ciclo:

- Lettura da console dei dati della richiesta (nome file da ricevere)
- Ricezione del file ordinato dal server:

Schema 2: lettura a caratteri fino ad un separatore chiaro, ad esempio uno zero binario (questa soluzione **può dare problemi** in caso di **trasferimenti di file binari**, che potrebbero contenere diversi zeri binari – come risolvere?).

Fine ciclo

```
close(sd)
```

SCHEMA DI SOLUZIONE: IL SERVER 1/3

Inizializzazione indirizzo:

```
memset ((char *)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(port);
```

Creazione delle due socket su cui ricevere richieste:

```
udpfd=socket(AF_INET, SOCK_DGRAM, 0);
listenfd=socket(AF_INET, SOCK_STREAM, 0);
```

Settaggio delle opzioni e bind:

```
setsockopt(… di entrambe le socket…)
bind(… di entrambe le socket…)
listen(…)
```

SCHEMA DI SOLUZIONE: IL SERVER 2/3

Pulizia e settaggio della maschera dei file descriptor:

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

Ciclo di ricezione eventi dalla select:

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

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

  <gestione richieste>
```

SCHEMA DI SOLUZIONE: IL SERVER 3/3

Gestione richieste da socket stream:

```
if (FD_ISSET(listenfd, &rset))
{  accept(…)

   <generazione processo figlio>
   <invio del file richiesto>
```

Due possibili schemi di soluzione corrispondenti nel server

- una connessione unica per tutte le richieste provenienti dallo stesso client, invio di carattere terminatore

- una connessione diversa per ogni file inviato

```
}
```

Gestione richieste da socket datagram:

```
if (FD_ISSET(udpfd, &rset))
{ <conteggio dei file> }
```

CLIENT DATAGRAM 1/3

```
main(int argc, char **argv)
{
struct hostent *host;
struct sockaddr_in servaddr, clientaddr;
int  sd, len, num_file;
char nome_dir[20];

if(argc!=3) // Controllo argomenti
   { printf("Error:%s server\n", argv[0]); exit(1); }

clientaddr.sin_family = AF_INET; // Prepara indirizzo client e server
clientaddr.sin_addr.s_addr = INADDR_ANY;
clientaddr.sin_port = 0;
memset((char *)&servaddr, 0, sizeof(struct sockaddr_in));
servaddr.sin_family = AF_INET;
host = gethostbyname (argv[1]);

if (host == NULL)
{ printf("%s not found in /etc/hosts\n", argv[1]); exit(2);}
```

CLIENT DATAGRAM 2/3

```
else { servaddr.sin_addr.s_addr =
       ((struct in_addr *)(host->h_addr))->s_addr;
        servaddr.sin_port = htons(atoi(argv[2]));
}
```

```
sd=socket(AF_INET, SOCK_DGRAM, 0);    // Creazione e connessione
if(sd<0) {perror("apertura socket"); exit(3);}
```

```
bind(sd,(struct sockaddr*) &clientaddr, sizeof(clientaddr));
```

// Corpo del client: ciclo di accettazione di richieste di conteggio

```
printf("Nome del direttorio: ");
while (gets(nome_dir))
{ len=sizeof(servaddr);     // Invio richiesta
  if (sendto(sd, nome_dir, (strlen(nome_dir)+1), 0,
  (struct sockaddr *)&servaddr, len)<0)
  { perror("scrittura socket");
    printf("Nome del direttorio: ");
    continue; // Se l'invio fallisce nuovo ciclo
  }
```

CLIENT DATAGRAM 3/3

```
// Ricezione del risultato
if (recvfrom(sd, &num_file, sizeof(num_file), 0,
                 (struct sockaddr *)&servaddr, &len)<0)
  { perror("recvfrom");
    printf("Nome del direttorio: ");
    continue; /* se la ricezione fallisce nuovo ciclo */
  }

  printf("Numero di file: %i\n", num_file);
  printf("Nuovo nome del direttorio: ");
}   // while

printf("\nClient: termino...\n");
close(sd);
} // main
```

CLIENT STREAM 1/5

```
main(int argc, char *argv[])
{
int sd, nread, nwrite;
char c, ok, buff[DIM_BUFF], nome_file[15];
struct hostent *host;
struct sockaddr_in servaddr;
const int on = 1;

if(argc!=3)  // Controllo argomenti
{ printf("Error:%s server\n", argv[0]); exit(1); }

// Preparazione indirizzo server
memset((char *)&servaddr, 0, sizeof(struct sockaddr_in));
servaddr.sin_family = AF_INET;
 host = gethostbyname(argv[1]);
if (host == NULL)
{ printf("%s not found in /etc/hosts\n", argv[1]); exit(2); }

servaddr.sin_addr.s_addr =
  ((struct in_addr*) (host->h_addr))->s_addr;
servaddr.sin_port = htons(atoi(argv[2]));
```

CLIENT STREAM 2/5

Primo schema:

creo e connetto socket ad ogni ciclo di interazione con l'utente perché viene utilizzata una connessione diversa per ogni file richiesto . . .

```
printf("Nome del file da richiedere: "); // Accettazione richieste
while (gets(nome_file))
{/* Creazione socket e connessione DENTRO il ciclo -- */
 sd=socket(AF_INET, SOCK_STREAM, 0);
 if (sd <0){perror("apertura socket "); exit(3);}
 if (connect(sd,(struct sockaddr *) &servaddr,
 sizeof(struct sockaddr))<0)
   {perror("Errore in connect"); exit(4);}
 if (write(sd, nome_file, (strlen(nome_file)+1))<0)
   {  perror("write"); close(sd); /*...*/ continue; }
 if (read(sd, &ok, 1)<0)
   { perror("read"); /*...*/ continue;  }
```

CLIENT STREAM 3/5

```
 if (ok=='S')
  { /* Leggo fino alla chiusura della connessione da parte del server */
    while((nread=read(sd, buff, sizeof(buff)))>0)
    {  if (nwrite=write(1, buff, nread)<0) // Scrittura a video
         { perror("write"); break;   }
    }
    if ( nread <0 ){ perror("read"); close(sd);
                     /* ... */ continue; }
   }
  else if (ok=='N') printf("File inesistente\n");
  // Chiusura dentro il while
  close(sd);
  printf("Nome del file da richiedere: ");
 }    // while
}    // main
```

CLIENT STREAM 4/5

Secondo schema:

creo e connetto socket prima di entrare nel ciclo di interazione con l'utente perché viene utilizzata sempre la stessa connessione per tutte le richieste dello stesso cliente

```
// Creazione socket e connessione PRIMA del ciclo
sd=socket(AF_INET, SOCK_STREAM, 0);
if (sd <0) {perror("apertura socket "); exit(3);}
printf("Creata la socket sd=%d\n", sd);
if (connect(sd,(struct sockaddr *) &servaddr,
            sizeof(struct sockaddr))<0)
   {perror("Errore in connect"); exit(4);}

// Corpo del client: accettazione richieste
printf("Nome del file da richiedere: ");
while (gets(nome_file))
{ if (write(sd, nome_file, (strlen(nome_file)+1))<0)
      { perror("write"); /*...*/ break; }
  if (read(sd, &ok, 1)<0)
     { perror("read"); /*...*/ break; }
```

CLIENT STREAM 5/5

```
if (ok=='S')
  {
   while((nread=read(sd,&c,1))>0)
     if (c!='\0'){ write(1,&c,1);} // Stampo a video fino a EOF
     else break;
   if( nread<0 ) { perror("read"); /*...*/ break; }
  }
 else if (ok=='N') printf("File inesistente\n");
       // Controllare sempre che il protocollo sia rispettato
       else printf("Errore di protocollo!!!\n");
 printf("Nome del file da richiedere: ");
}//while

// Chiusura FUORI dal while
close(sd);
printf("\nClient: termino...\n");
}
```

SERVER CON SELECT 1/8

```
int conta_file (char *name)  // Funzione di conteggio dei file nel direttorio
{ DIR *dir; struct dirent * dd;
  int count = 0;
  dir = opendir (name);
  while ((dd = readdir(dir)) != NULL)
  { printf("Trovato il file %s\n", dd-> d_name);count++; }
   /* conta anche il direttorio stesso e il padre e altri direttori! */
  printf("Numero totale di file %d\n", count);
  closedir (dir);
  return count;
}
void gestore(int signo)
   // gestore del segnale per eliminare i processi figli
{int stato;
 printf("esecuzione gestore di SIGCHLD\n");
 wait(&stato);
}
```

SERVER CON SELECT 2/8

```
main(int argc, char **argv)
{ int  listenfd, connfd, udpfd, fd_file, nready, maxfdp1;
  char zero=0, buff[DIM_BUFF], nome_file[20], nome_dir[20];
  const int on = 1;
  fd_set rset;
  int len, nread, nwrite, num, ris, port;
  struct sockaddr_in cliaddr, servaddr;
```

`if( argc!=2 ){ ... } else{ ... }` **// Controllo argomenti**

// Inizializzazione indirizzo server

```
memset ((char *)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY; // no htonl
servaddr.sin_port = htons(port);
```

// NOTA: si possono usare lo stesso indirizzo e stesso numero di porta per le due socket

// Creazione socket TCP di ascolto

```
listenfd = socket(AF_INET, SOCK_STREAM, 0);
if (listenfd <0)
{perror("apertura socket TCP "); exit(1);}
```

SERVER CON SELECT 3/8

```
if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on))<0)
  {perror("set opzioni socket TCP"); exit(2);}

if (bind(listenfd,(struct sockaddr *) &servaddr,
                              sizeof(servaddr))<0)
   {perror("bind socket TCP"); exit(3);}

if (listen(listenfd, 5)<0)
  {perror("listen"); exit(4);}

udpfd = socket(AF_INET, SOCK_DGRAM, 0); // Creazione socket UDP
if(udpfd <0) {perror("apertura socket UDP"); exit(5);}

if (setsockopt(udpfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on))<0)
   {perror("set opzioni socket UDP"); exit(6);}

if(bind(udpfd,(struct sockaddr *) &servaddr,
                              sizeof(servaddr))<0)
   {perror("bind socket UDP"); exit(7);}
```

SERVER CON SELECT 4/8

```
signal(SIGCHLD, gestore); // Aggancio gestore
FD_ZERO(&rset); // Pulizia e settaggio maschera file descriptor
maxfdp1 = max(listenfd, udpfd)+1; // Prepara il primo parametro

for(;;)  // Ciclo di ricezione eventi della select e preparazione maschera ad ogni giro
{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(8);} }


 if (FD_ISSET(udpfd, &rset)) // Gestione richieste conteggio
 { len=sizeof(struct sockaddr_in);
   if (recvfrom(udpfd, &nome_dir, sizeof(nome_dir), 0,
        (struct sockaddr *)&cliaddr, &len)<0)
   {perror("recvfrom"); continue;}
   num = conta_file(nome_dir); ris=num;
   if (sendto(udpfd, &ris, sizeof(ris), 0,
          (struct sockaddr *)&cliaddr, len)<0)
  {perror("sendto"); continue;}
 }
```

SERVER CON SELECT 5/8

Primo schema connessione:

nuova connessione e processo figlio per ogni file da inviare

```
if (FD_ISSET(listenfd, &rset))
{ len = sizeof(struct sockaddr_in);
  if((connfd = accept(listenfd,
             (struct sockaddr *)&cliaddr,&len))<0)
  { if (errno==EINTR) continue;
    else {perror("accept"); exit(9);}
  }
  if (fork()==0)  // Processo figlio: servizio richiesta
  { close(listenfd);
    printf("Dentro il figlio, pid=%i\n", getpid());
    // Non c'è ciclo di richieste, viene creato un nuovo figlio per ogni richiesta di file
    if (read(connfd, &nome_file, sizeof(nome_file))<=0)
      {perror("read"); break;}
    fd_file=open(nome_file, O_RDONLY);
    if (fd_file<0) {write(connfd, "N", 1);}   // File inesistente
```

SERVER CON SELECT 6/8

```
    else {     // Il file esiste
      write(connfd, "S", 1);
      // Invio file (a blocchi)
       while((nread=read(fd_file,buff,sizeof(buff)))>0)
       { if (nwrite=write(connfd, buff, nread)<0)
            {perror("write"); break;}
       }
       close(fd_file);  // Libero la risorsa sessione sul file
      // NOTA: non è necessario separare i file tra loro con terminatore
    }
    // Chiusura connessione. NOTA: ogni connessione assegnata ad un nuovo figlio
     close(connfd);
    exit(0);
  }   // figlio

 // Padre chiude la socket (NON di accettazione)
 close(connfd);
  }   // if
 }   // for
}   // main
```

SERVER CON SELECT 7/8

Secondo schema di connessione:

una sola connessione e un unico figlio per gestire invii di diversi file richiesti dallo stesso utente

```
if (FD_ISSET(listenfd, &rset))
{printf("Ricevuta richiesta di get di un file\n");
 len = sizeof(struct sockaddr_in);
 if((connfd = accept(listenfd,
  (struct sockaddr *)&cliaddr,&len))<0)
 { if (errno==EINTR) continue;
   else {perror("accept"); exit(9);}
 }
 if (fork()==0)
 { /* FIGLIO */
   close(listenfd);
   printf("Dentro il figlio, pid=%i\n", getpid());
```

SERVER CON SELECT 8/8

```
for (;;) // Ciclo di gestione richieste con un'unica socket da parte del figlio
 { if ((nread=read(connfd, &nome_file,sizeof(nome_file)))<0)
                {perror("read"); break;}
   else if (nread == 0) // Quando il figlio riceve EOF esce dal ciclo
          {printf("Ricevuto EOF\n"); break;}
  printf("Richiesto file %s\n", nome_file);
   fd_file=open(nome_file, O_RDONLY);
   if (fd_file<0){write(connfd, "N", 1);}
  else
  { write(connfd, "S", 1);
     // lettura/scrittura file (a blocchi)
      while((nread=read(fd_file, buff, sizeof(buff)))>0)
      { if (nwrite=write(connfd, buff, nread)<0)
        {perror("write"); break;}
      }
      write(connfd, &zero, 1); // Invio messaggio terminazione file: zero binario
       close(fd_file);     // Libero la risorsa sessione file
  }//else
}//for
 close(connfd); exit(0); // Chiusura della connessione all'uscita dal ciclo
} // figlio
close(connfd); // Padre: chiusura socket di comunicazione e suo ciclo
} /* if */     } /* for */      } //main
```