

5° Esercitazione (svolta):

Socket C con select

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 da trasferire, e usa una connessione per inviare il nome del file selezionato e ricevere il file richiesto.

Entrambi i client accettano ciclicamente richieste dall'utente, fino alla fine del file di input da console.

- 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: 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 in maniera 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.

Schema di soluzione: il client datagram

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,...);
```

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

- Lettura da console dei dati della richiesta (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

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,...);
```

Schema 1: una sola connessione

=> connect prima di entrare nel ciclo

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

Schema 2: una connessione per ogni richiesta

=> connect all'interno del ciclo

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

Schema 1: lettura a caratteri fino a zero binario (si noti che questa soluzione, **può dare problemi** in caso di **trasferimenti di file binari**, che potrebbero contenere diversi zeri binari).

Schema 2: lettura a blocchi fino alla chiusura della connessione da parte del server.
close(sd)

Fine ciclo.

Schema 1: close(sd)

Schema di soluzione: il server

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(...)
```

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)<0)  
  
    <gestione richieste>
```

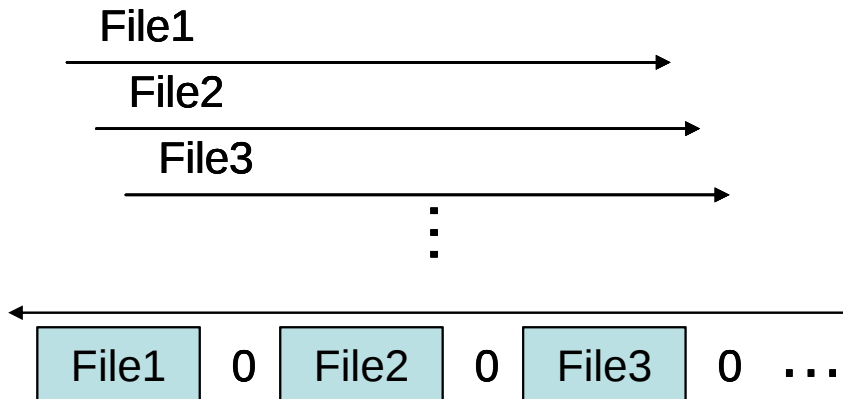
Gestione richieste da socket stream:

```
if (FD_ISSET(listenfd, &rset))  
{  
    accept(...)  
  
    <generazione processo figlio>  
  
    <invio del file richiesto>  
    Due possibili schemi di soluzione:  
    - una connessione unica per tutte le richieste  
      provenienti dallo stesso client, invio di  
      carattere terminatore  
    - una connessione diversa per ogni file  
      inviato  
}
```

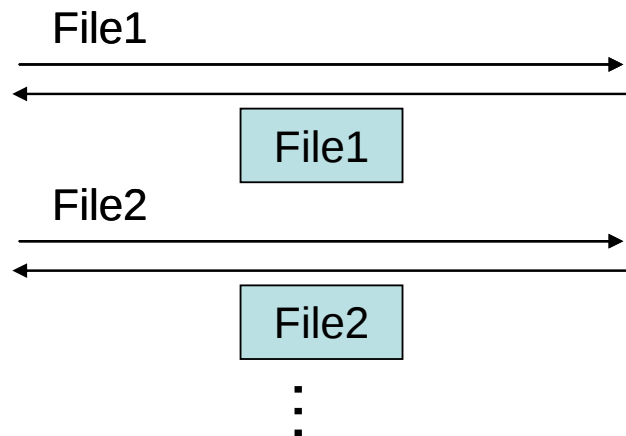
Gestione richieste da socket datagram:

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

Schema 1



Schema 2



Client datagram

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

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

    /* PREPARAZIONE INDIRIZZO CLIENT e SERVER ----- */
    clientaddr.sin_family = AF_INET;
    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);
    } else
    {
        servaddr.sin_addr.s_addr =
            ((struct in_addr *) (host->h_addr))->s_addr;
        servaddr.sin_port = htons(atoi(argv[2]));
    }

    /* CREAZIONE E CONNESSIONE SOCKET ---- */
    sd=socket(AF_INET, SOCK_DGRAM, 0);
    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))
{
    /* invio richiesta */
    len=sizeof(servaddr);
    if (sendto(sd, nome_dir, (strlen(nome_dir)+1), 0,
               (struct sockaddr *)&servaddr, len)<0)
    {
        perror("scrittura socket");
        printf("Nome del direttorio: ");
        /* se l'invio fallisce nuovo ciclo */
        continue;
    }

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

    printf("Numero di file: %i\n", ntohl(num_file));
    printf("Nome del direttorio: ");
}

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

```

Client stream

```

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

    /* CONTROLLO ARGOMENTI ----- */
    if (argc!=3)
    {
        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]));
}

```

Primo schema:

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

```
/* 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);}
printf("Connect ok\n");

/* 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"); /*...*/ continue; }
    if (read(sd, &ok, 1)<0)
    { perror("read"); /*...*/ continue; }
    if (ok=='S')
    {
        while((nread=read(sd,&c,1))>0)
            /* Leggo a caratteri per individuare il simbolo
            di terminazione invio file. Ci sono alternative
            alla lettura a singolo carattere? */
            if (c!='\0')
            {
                write(1,&c,1);
            } else 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");
}
```

Secondo schema:

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

```
...

/* CORPO DEL CLIENT: accettazione richieste -- */
printf("Nome del file da richiedere: ");

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; }
    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)
            {
                perror("write");
                break;
            }
        }
    }
    else if (ok=='N') printf("File inesistente\n");
    /* Chiusura dentro il while */
    close(sd);
    printf("Nome del file da richiedere: ");
}
}
```

Server con select

```
/* FUNZIONE PER IL CONTEGGIO DEI FILE PRESENTI IN
   UN DIRETTORIO */
int conta_file (char *name)
{ 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++;
  }
  printf("Numero totale di file %d\n", count);
  closedir (dir);
  return count;
}

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

*****/

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

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

/* 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 = htons(port);

/* N.B.: stesso indirizzo e stessa porta sia
   per la socket stream che per la socket
   datagram */
/* CREAZIONE SOCKET TCP ----- */
listenfd=socket(AF_INET, SOCK_STREAM, 0);
if (listenfd < 0)
{perror("apertura socket TCP "); exit(1);}

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);}

/* CREAZIONE SOCKET UDP ----- */
udpfd=socket(AF_INET, SOCK_DGRAM, 0);
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);}
```

```

/* AGGANCIO GESTORE PER EVITARE FIGLI ZOMBIE ----- */
signal(SIGCHLD, gestore);

/* PULIZIA E SETTAGGIO 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);

    if ((nready=select(maxfdp1, &rset, NULL, NULL, NULL)<0)
    {
        if (errno==EINTR) continue;
        else {perror("select"); exit(8);}
    }

    /* GESTIONE RICHIESTE DI CONTEGGIO ----- */
    if (FD_ISSET(udpfd, &rset))
    {
        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=htonl(num);

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

```

Primo schema:

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());

        /* Ciclo con cui un unico figlio su un'unica connessione
        gestisce tutte le richieste dello stesso utente */
        for (;;)
        {
            if ((nread=read(connfd, &nome_file,
                            sizeof(nome_file))<0)
            {perror("read"); break;}
            else if (nread == 0)
                /* Quando il client chiude la connessione
                (e quindi il server riceve un EOF) esco 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)
            {
                printf("File inesistente\n");
                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;}
    }
    /* invio al client del segnale di terminazione:
       zero binario */
    write(connfd, &zero, 1);
    /* Libero le risorse: chiudo il file */
    close(fd_file);
} //else
} //for
printf("Figlio %i: chiudo connessione e termino\n",
       getpid());
/* chiusura della connessione all'uscita dal ciclo */
close(connfd);
exit(0);
}

/* padre chiude la socket (NON quella di ascolto) */
close(connfd);
}
}
}

```

Secondo schema:

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 più il ciclo, 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)
        {
            printf("File inesistente\n");
            write(connfd, "N", 1);
        }
        else {
            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;}
            }
            /* Libero risorse */
            close(fd_file);

            /* NOTA:
               non è più necessario inviare lo 0 binario */
        }
    }
}

```

```
printf("Figlio %i: chiudo connessione e termino\n",
      getpid());

/* Chiudo la connessione dentro il ciclo for */
close(connfd);

/* chiudo qua perchè ogni connessione accettata è
   assegnata ad un nuovo figlio */
exit(0);
} //figlio

/* padre chiude la socket (NON di accettazione) */
close(connfd);
}
}
```