Esercizio con select

Si progetti un'applicazione distribuita Client/Server che consenta di gestire i conti corrente di una banca. Il Server deve fornire due tipi di servizio:

- **gestione di movimenti** (versamenti e prelievi)
- **visualizzazione dell'estratto conto** completo, inteso come insieme di tutti i movimenti

L'applicazione prevede l'implementazione di un Server e di due Client.

Server

Fase di inizializzazione

➔ creazione di un certo numero di conti su cui lavorare (nome file estratto conto = numero cc).

Fase di accettazione delle richieste di operazione

➔ discriminazione tra i due tipi di richieste ricevute attraverso socket diverse, usando la primitiva ***select***

Richieste di versamento/prelievo

➔ da gestire in maniera sequenziale

Richieste di visualizzazione dell'estratto conto

➔ da gestire in modo parallelo,
vista la probabile maggiore durata

Clienti

Il **primo Client** effettua solo richieste di versamento/prelievo su un conto specificato. Ciclicamente legge da tastiera:

numero di conto, tipo di operazione, importo dell'operazione

e richiede l'operazione attendendone il risultato o trattando le opportune condizioni anomale, tipo il prelievo di un importo superiore a quello disponibile.

Esempio di interazione:

- Numero conto corrente? 123
- Operazione (V=versamento, P=prelievo)? V
- Importo? 100
- Operazione effettuata correttamente

ecc.

Il **secondo Client** effettua solo richieste di visualizzazione dell'estratto conto. Ciclicamente legge da tastiera:

numero di conto

e richiede l'operazione attendendo l'invio del file di testo corrispondente, che contiene una riga per ogni movimento effettuato sul conto, o trattando le opportune condizioni anomale, tipo la richiesta dell'estratto conto di un conto inesistente.

Il file ricevuto viene visualizzato sulla console dell'utente.

Esempio di interazione:

- Numero conto corrente? 121
- Estratto conto del cc n. 121:

300 V
200 P
680 V
…
150 P

- Numero conto corrente? 999
- Errore: numero di conto inesistente

Entrambi i Client sono implementati come processi ciclici che continuano a fare richieste sincrone fino ad esaurire tutte le richieste utente (fino alla fine del file di ingresso dell'utente).

Soluzione

- Gestione delle **richieste di movimenti**

socket datagram

limitato numero delle informazioni da inviare e ricevere

➔ superfluo mantenere una connessione dedicata

- Gestione delle **richieste di estratti conto**

socket stream

trasferimento di file di dimensioni eventualmente elevate

➔ giustificato l'impiego di una connessione.

Server

```
....

/* 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(1);}
 }



/* GESTIONE RICHIESTE DI ESTRATTO CONTO -------------- */

if (FD_ISSET(listenfd, &rset))
{
 printf("Server: ricevuta richiesta di estratto conto\n");
 len = sizeof(cliaddr);

 if((connfd = accept(listenfd,(struct sockaddr *)
                                    &cliaddr,&len))<0)
 {
   if (errno==EINTR) continue;
   else {perror("accept"); exit(1);}
 }

 // gestione richieste di EC

}
```



```
/* GESTIONE RICHIESTE DI MOVIMENTO ------------------- */

if (FD_ISSET(udpfd, &rset))
{

 printf("Server: ricevuta richiesta di movimento\n");
 len=sizeof(struct sockaddr_in);

 strcpy(ris, "OK");

 if (recvfrom(udpfd, req, sizeof(Request), 0,
                (struct sockaddr *)&cliaddr, &len)<0)
  {perror("recvfrom"); exit(1);}

 /* trattiamo le conversioni possibili */
 num=ntohl(req->numcc);
 imp=ntohl(req->impOp);


 printf("Movimento richiesto: %c di %i  su cc n. %i\n",
                                  req->tipoOp, imp, num);



/* APERTURA O CREAZIONE DEL FILE ESTRATTO CONTO ------ */

if((fd_mov=open(conti[num].ec,
                O_WRONLY|O_CREAT|O_APPEND, PERMS))<0)
    { perror("open"); exit(1);}



/* RICHIESTA DI VERSAMENTO --------------------------- */

if(req->tipoOp=='V')
 {
  printf("versamento...\n");
  conti[num].deposito=+imp;
  strcpy(movimento,"versamento di ");
  write(fd_mov, movimento, strlen(movimento));
  sprintf(simp, "%i\n", imp);
  write(fd_mov, simp, strlen(simp));
 }
```

```
/* RICHIESTA DI PRELIEVO ----------------------------- */

 if(req->tipoOp==’P’)
 {
  printf("prelievo...\n");
  if ( (conti[num].deposito-imp) <0)
    {strcpy(ris, "errore: prelievo superiore al
                                        deposito\n");}
  conti[num].deposito=-imp;
  strcpy(movimento,"prelievo di ");
  write(fd_mov, movimento, strlen(movimento));
  sprintf(simp, "%i\n", imp);
  write(fd_mov, simp, strlen(simp));
 }

 i=sendto(udpfd, ris, strlen(ris)+1, 0,
                      (struct sockaddr *)&cliaddr, len);
 if (i<0)
    {perror("sendto"); exit(1);}

 } /* fine gestione richieste di movimento

} /* ciclo for della select */
```

Gestione richieste di estratto conto

➔ Due schemi di soluzione

Primo schema di soluzione

Tutte le **richieste di uno stesso client**
vengono gestite dallo **stesso figlio**
mantenendo aperta sempre la **stessa connessione**.

Il server (figlio) manda al client uno zero binario con cui segnala la terminazione dell’invio del file, e resta in attesa di nuove richieste; finisce la sua esecuzione solo quando il client che sta servendo chiude la connessione.

Il client continua ad inviare richieste sempre sulla stessa socket, e chiude la connessione soltanto al termine della sua esecuzione

Client Estratto Conto Schema 1:

```
…
while (gets(ok))
{
  printf("Numero di contocorrente: ");
  while (scanf("%i", &num) != 1)
  {
    do
    {c=getchar(); printf("%c ", c);}
    while (c!= '\n');
    printf("Numero di contocorrente: ");
  }
  numcc=htonl(num);
  gets(ok);

  if (write(sd,&numcc,4)<0)
    {perror("write"); exit(1);}
  printf("Richiesta di estratto del cc n.%i inviata...
                                        \n", ntohl(numcc));

  if (read(sd, ok, 1)<0)
    {perror("read"); exit(1);}

 if (ok[0]=='S')
  {
   printf("ESTRATTO CONTO DEL CC N. %i:\n", num);
   while((nread=read(sd,&c,1))>0)
      if (c!='\0'){
    write(1,&c,1);
    } else break;
  }
 else if (ok[0]=='N')printf("Numero di conto
                                          inesistente\n");
     else if (ok[0]=='B')printf("Nessuna movimentazione
                                 sul conto richiesto\n");
 printf("Qualsiasi tasto per procedere,
                                  EOF per terminare: ");
}//while

printf("Client estratto conto: termino...");
close(sd);
exit(0);

}
```


Server Schema 1 per Estratto Conto:

```
…

if (fork()==0)
 { /* processo figlio che serve
                            la richiesta di operazione */
   close(listenfd);
   printf("Dentro il figlio...\n");

  /* visualizzazione ciclica
                     degli estratti conto richiesti */
  for (;;){

    i=read(connfd, &n, sizeof(int));

    if (i<0) { perror("read"); exit(2);}
    if (i==0) break;

    numcc=ntohl(n);
    printf("Richiesto conto N. %i\n", numcc);

    if ((numcc<0) || (numcc>5))
        {printf("Numero conto inesistente\n");
         write(connfd, "N", 1);}
    else {
        fd_ec=open(conti[numcc].ec, O_RDONLY);
        if(fd_ec<0)
            {printf("File inesistente\n");
             write(connfd, "B", 1);}
        else {
            write(connfd, "S", 1);

    /* lettura dal file (a blocchi, non a linee)
                          e scrittura sulla socket */
            while((nread=read(fd_ec, buffTCP,
                                       DIM_BUFF))>0){
                if (nwrite=write(connfd, buffTCP,
                                            nread)<0)
                          {perror("write"); exit(4);}
            }
            printf("Terminato invio file EC\n");
```

```
/* invio al client segnale di terminazione: zero binario */

            write(connfd, &zero, 1);
            close(fd_ec);

        }//else
    }//else

    }//for

    close(connfd);
    exit(0);

 }
/* padre chiude la socket dell'operazione */
 close(connfd);

} /* fine gestione richieste di EC */
```


Secondo schema di soluzione

Generato un **nuovo figlio**
e quindi accettata **una nuova connessione**
per ogni richiesta
anche se proveniente dallo stesso client.

Il server (figlio) termina la sua esecuzione immediatamente dopo aver inviato il file, o comunque risposto al client.

Il client in questo caso invia le richieste sempre su una socket nuova, e deve quindi ripetere tutte le operazioni di creazione della socket, settaggio delle opzioni, connect e infine chiusura della connessione ad ogni ciclo.

Client Estratto Conto Schema 2:

```
…
while (gets(ok))
{
  /* CREAZIONE E CONNESSIONE SOCKET ------------------ */
  /* in questo schema è necessario ripetere creazione,
    settaggio opzioni e connect ad ogni ciclo, perchè il
    client fa una nuova connect ad ogni ciclo */

  sd=socket(AF_INET, SOCK_STREAM, 0);
  printf("Client estratto conto:
                          creata la socket sd=%d\n", sd);
  set_opt=setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on,
                                        sizeof(on));
  if(set_opt<0)
    {perror("set opzioni socket TCP"); exit(1);}
  printf("Client estratto conto:
                             set opzioni socket ok\n");
  conn_ok=connect(sd,(struct sockaddr *) &servaddr,
                               sizeof(struct sockaddr));
  if(conn_ok<0)
    { perror("Errore in connect"); exit(1);}
  printf("Client estratto conto: connect ok\n");


  printf("Numero di contocorrente: ");
  while (scanf("%i", &num) != 1)
  {
    do
    {c=getchar(); printf("%c ", c);}
    while (c!= '\n');
    printf("Numero di contocorrente: ");
  }
  numcc=htonl(num);
  gets(ok);

  if (write(sd,&numcc,4)<0)
    {perror("write"); exit(1);}
  printf("Richiesta di estratto del cc n.%i inviata... \n",
                                          ntohl(numcc));
```



```
  if (read(sd, ok, 1)<0)
    {perror("read"); exit(1);}


if (ok[0]=='S')
  {
   printf("ESTRATTO CONTO DEL CC N. %i:\n", num);
   while((nread=read(sd,buff,DIM_BUFF))>0)
    write(1,buff,nread);
   }
   else if (ok[0]=='N')printf("Numero di conto
                                          inesistente\n");
    else if (ok[0]=='B')printf("Nessuna movimentazione
                                  sul conto richiesto\n");

 close(sd);
    /* chiudo qua perchè faccio una
                nuova connect ad ogni ciclo... */

printf("Qualsiasi tasto per procedere,
                                   EOF per terminare: ");

}//while


printf("Client estratto conto: termino...");
exit(0);

}
```

Server Schema 2 per Estratto Conto:

```
if (fork()==0)

 { /* processo figlio che serve la richiesta di operazione */
   close(listenfd);
   printf("Dentro il figlio...\n");

   /* non c'è più il ciclo perchè viene creato un
            nuovo figlio per ogni richiesta di EC */


    i=read(connfd, &n, sizeof(int));

    if (i<0) { perror("read"); exit(2);}
    if (i==0) break;

    numcc=ntohl(n);
    printf("Richiesto conto N. %i\n", numcc);

    if ((numcc<0) || (numcc>5))
        {printf("Numero conto inesistente\n");
         write(connfd, "N", 1);}
    else {
        fd_ec=open(conti[numcc].ec, O_RDONLY);
        if(fd_ec<0)
            {printf("File inesistente\n");
             write(connfd, "B", 1);}
        else {
            write(connfd, "S", 1);



    /* lettura dal file (a blocchi, non a linee)
                         e scrittura sulla socket */
            while((nread=read(fd_ec, buffTCP,
                                      DIM_BUFF))>0){
                if (nwrite=write(connfd, buffTCP,
                                          nread)<0)
                        {perror("write"); exit(4);}
            }
```



```
            printf("Terminato invio file EC\n");
            close(fd_ec);
        }//else
    }//else


    close(connfd);
    exit(0);

 }

/* padre chiude la socket dell'operazione */
 close(connfd);

} /* fine gestione richieste di EC */
```