Versione N° 1.0

LaurTec

TCP/IP in ambiente Linux
Calcolo dei numeri primi

Autore : *Mauro Laurenti*

email: info.laurtec@gmail.com

ID: PJ11001-IT

INFORMATIVA

Come prescritto dall'art. 1, comma 1, della legge 21 maggio 2004 n.128, l'autore avvisa di aver assolto, per la seguente opera dell'ingegno, a tutti gli obblighi della legge 22 Aprile del 1941 n. 633, sulla tutela del diritto d'autore.

AVVERTENZE

I progetti presentati non hanno la certificazione CE, quindi non possono essere utilizzati per scopi commerciali nella Comunità Economica Europea.

Chiunque decida di far uso delle nozioni riportate nella seguente opera o decida di realizzare i circuiti proposti, è tenuto pertanto a prestare la massima attenzione in osservanza alle normative in vigore sulla sicurezza.

L'autore declina ogni responsabilità per eventuali danni causati a persone, animali o cose derivante dall'utilizzo diretto o indiretto del materiale, dei dispositivi o del software presentati nella seguente opera.

Si fa inoltre presente che quanto riportato viene fornito cosi com'è, a solo scopo didattico e formativo, senza garanzia alcuna della sua correttezza.

L'autore ringrazia anticipatamente per la segnalazione di ogni errore.

Introduzione

La programmazione di sistema rappresenta una simbiosi tra il sistema operativo e il programma stesso. In questo esempio di programmazione in C in ambiente Linux viene illustrato il protocollo TCP/IP per poter calcolare i numeri primi per mezzo di più processi client identificati da un indirizzo IP.

Specifiche di progetto

Viene richiesto di realizzare un sistema software, distribuito, per il calcolo dei numeri primi interi, compresi in un intervallo inserito dall'utente. L'applicazione deve essere ottenuta realizzando un server al quale un numero indefinito di client possa connettersi tramite BSD socket. Il lavoro di ricerca di numeri primi è svolto dai client in sottointervalli dell'intervallo principale. Tali pacchetti vengono inviati dal server e le dimensioni sono scelte dall'utente. Il client deve rendere al server i numeri primi trovati nel pacchetto e al termine dell'analisi chiedere eventualmente un altro pacchetto da analizzare. Analizzato l'intervallo principale o alla chiusura del server, i numeri primi trovati devono essere salvati in maniera ordinata all'interno di un file. La sintassi per eseguire il server e il client è :

server <porta_TCP> <primo_intero> <ultimo_intero> <dim_pacchetti>

client <indirizzo_IP_server> <porta_TCP_server>

Scelte di progetto

Il server e il client sono trattati come due entità distinte, ovvero programmi differenti. Alcune grandezze, al fine di permettere in maniera corretta la comunicazione tra server e client hanno dimensioni comune (es. la dimensione dell'array che viene utilizzato per leggere e scrivere messaggi nel canale di trasmissione). Il domino sul quale viene utilizzata la socket è quello di Unix, dal momento che i processi server e client si trovano nella stessa macchina . Il client quando riesce a connettersi con il server inizia la trasmissione dei dati con un figlio del server in modo da permettere al server di continuare ad accettare le connessioni sulla porta della socket aperta.

Il client trasmette al figlio del server i numeri primi mano a mano che li trova, il figlio salva i dati all'interno di un file. Ogni figlio quando viene creato eredita dal padre la variabile che contiene il numero dei figli creati, tale numero al pari del PID identifica in maniera univoca il figlio. Questo invio di numeri e il salvataggio in un file è stato deciso in alternativa all'invio dei numeri primi a fine ricerca poiché nel primo caso, nell'eventualità di "crollo" del sistema client o server i calcoli svolti fino a quel momento non vengono persi. Il client a fine ricerca può richiedere un nuovo pacchetto da analizzare al server, per semplificare l'algoritmo di salvataggio finale, che prevede il salvataggio, in maniera ordinata, dei numeri primi trovati, si è fatto in modo che il client a fine calcolo chiuda la comunicazione con il figlio del server (inviando numero 0) per poi chiudere la propria socket. In questo modo per richiedere un nuovo pacchetto viene aperta una nuova socket e il server delega un nuovo figlio, con indice crescente, per la comunicazione con il client.

Al fine di rendere nota sia al server che al client lo stato della comunicazione si è fatto uso dei segnali inviati dal sistema operativo, in particolare si sono armati e gestiti i segnali di chiusura del processo e di chiusura pipe. Per il server si è fatto uso anche dei segnali che il sistema Unix riserva per l'utente, ovvero SIGUSR1 e SIGUSR2 in modo da controllare in maniera più sicura i "rapporti" padre figlio.

Per la scelta del tipo numerico al quale devono appartenere i numeri dell'intervallo si è fatto uso del tipo long, non si è posta attenzione a rendere l'intervallo grande (ad esempio usando il tipo long long) dal momento che tale tipologia di programmi dovrebbero trattare comunque insiemi numerici estremamente grandi, generalmente non definiti come tipi elementari. Se si fosse quindi voluto raggiungere anche questo scopo si sarebbe dovuto creare un nuovo tipo e creare particolari funzioni in

modo da trattarlo come un tipo elementare. Per tali ragioni il programma è stato concepito per soddisfare solo le specifiche algoritmiche concettuali.

Codici Sorgenti e loro descrizione

Codice Sorgente del Server[1]

```
1 #include <sys/types.h> // includo le librerie di interesse
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4 #include <netdb.h>
5 #include <stdio.h>
6 #include <signal.h>
7 #include <sys/stat.h>
8 #include "mio.h"
9
10 //prototipi di funzioni
11 void gestoreSig1(int); // funzione per gestire la terminazione di un figlio
12 void gestoreSig2 (int);       // funzione per chiudere i figli
13 void gestoreUscita (int); // gestisco l'uscita Control +C
14 void salvataggio ();    // salvo in un unico file i numeri trovati
15
16 int figli_necessari=0; // numero che dipende dall'intervallo e dal quanto
17 int num_figli =0; // è uguale al numero di client che si è connesso
18
19 int main (int argc, char *argv[])
20 {
21     int ds_sock, ds_sock_acc;
22     struct sockaddr_in server;              // struttura del server
23     struct sockaddr client;          // struttura del client
24     int lunghezza; // misura in byte della struttura client effettivamente usati
25     char ip_server [20]="128.0.0.0";        //IP del server
26     char buffer[BUF_DIM];                   //per trasmettere messaggi
27     long numero;        // per trasmettere i numeri dell'intervallo
28     int porta_TCP;             // porta di comunicazione
29     long primo_numero, ultimo_numero,inf,sup; // primo e ultimo numero  dell'intervallo
da analizzare
30     long quanto;               // dimensione dei pacchetti
31     int pid;                   // pid del figlio
32     int controlla = 0;         // misura dell'intervallo controllato
33     FILE *fp;
34     char nome_file [12];       // nome dei file creati dai figli
35
36
37 if (argc<5) // controllo il numero di argomenti inseriti
38 {
39     printf ("Inserire %s <porta_TCP> <primo_intero><secondo_intero> <dim_pacchetti>\n",
argv[0]);
40     exit(-1);
41 }
42
43     porta_TCP = atoi (argv[1]); // inizializzo le variabili con gli argomenti inseriti
44     primo_numero= atol(argv[2]);
45     ultimo_numero = atol (argv[3]);
46     quanto = atol (argv[4]);
47
48 if (primo_numero<2) // controllo alcune caratteristiche dei numeri introdotti
49 {
50     printf ("<primo_numero> deve essere maggiore di 1\n");
51     exit(-1);
52 }
53
```

[1] Per una migliore identazione del programma si rimanda ai sorgenti in formato pdf o html presenti nel file Software.zip scaricabile dal sito www.LaurTec.com.

```
54 if (primo_numero >= ultimo_numero) // controllo l'ordine dell'intervallo
55 {
56     printf ("<ultimo_numero> deve essere maggiore di <primo_numero>\n");
57     exit(-1);
58 }
59
60
61 figli_necessari = (ultimo_numero-primo_numero)/quanto; // rapporto tra interi è intero
62
63 if (((ultimo_numero-primo_numero)%quanto) !=0) figli_necessari++; // ho un figlio che
lavorerà con meno di un quanto
64
65     printf("\n");
66     printf("Premere Control + C per uscire\n");
67
68 if ((ds_sock = socket(AF_UNIX,SOCK_STREAM,0))==-1) // creo il soket
69 {
70     printf ("Errore di socket\n");
71     printf ("Impossibile avviare il Server\n");
72     exit(-1);
73 }
74
75 server.sin_family = AF_UNIX; // imposto la struttura dati del socket
76 server.sin_port = porta_TCP;
77 server.sin_addr.s_addr = inet_addr(ip_server); //converto l'indirizzo da stringa in
unsigned long
78
79 printf ("ds server %d\n",ds_sock); // visualizzo impostazioni socket
80 printf ("Server IP : %s \n",inet_ntoa (server.sin_addr.s_addr));
81 printf ("Porta di connessione : %d\n",porta_TCP);
82 printf ("Numero figli necessari : %d\n",figli_necessari);
83
84
85 if((bind (ds_sock, &server, sizeof(server)))==-1) //effettuo bind del socket creato
86 {
87     printf ("Bind fallito \n");
88     printf ("Impossibile avviare il Server\n");
89     exit(-1);
90 }
91
92 if ((listen (ds_sock,CODA_DIM)) ==-1) //imposto la coda dei client
93 printf("Impossibile accodare client\n");
94
95 while(2) //ciclo infinito
96 {   printf("*****************************************\n");
97     printf("Sono in attesa di connessioni \n");
98     while ((ds_sock_acc = accept (ds_sock, &client,&lunghezza))==-1); // attendo
connessione client
99     printf("Un client si è connesso\n");
100    printf ("ds client %d\n",ds_sock_acc);
101
102    if (controlla >= (ultimo_numero-primo_numero) ) // se l'intervallo è controllato
blocco i client
103    write(ds_sock_acc,"fine",BUF_DIM); // avviso dello stato con il messaggio fine
104
105    else // asssegno ad un altro client il pacchetto e risorse necessarie
106    {
107    num_figli++; // aumento il numero dei figli attivati = client accettati
108    controlla = quanto*num_figli; // imposto l'intervallo controllato
109    inf = primo_numero + quanto*(num_figli-1);
110    sup = inf+quanto-1;
111    if (sup> ultimo_numero) sup= ultimo_numero; // non controllo numeri oltre
l'intervallo;
112
113    signal (SIGUSR1,gestoreSig1); // armo il segnale
114
115    pid = fork(); // se l'intervallo è completo non creo altri figli
116
117    if (pid == -1)
```

```
118    {
119    printf ("Impossibile creare un nuovo figlio \n");
120    exit (-1); // non attendo il liberarsi delle risorse di sistema
121    }
122
123    if (pid!=0)
124    {
125    signal (SIGUSR2,SIG_IGN); // ignoro il segnale 2 di utente (chiude i figli)
126    signal (SIGINT,gestoreUscita); // armo il segnale di uscita per i salvataggi
necessari
127    }
128
129    else //sono il figlio
130    { close (ds_sock);
131    signal (SIGUSR2,gestoreSig2); //armo il segnale
132    read(ds_sock_acc,buffer,BUF_DIM);
133    printf("messaggio ricevuto %s \n",buffer);
134    write(ds_sock_acc,"OK",BUF_DIM);              // invia il messaggio OK al client
135    write(ds_sock_acc,&inf,sizeof(inf));          // invia l'intervallo al client
136    write(ds_sock_acc,&sup,sizeof(sup));
137
138    sprintf (nome_file,"%d",num_figli); // creo nome file con indice del figlio (univoco)
139
140    if ((fp=fopen(nome_file, "w")) == NULL)
141    {
142    printf ("Impossibile aprire il file \n");
143    exit(-1);
144    }
145
146    do // scrivo nel file i numeri inviati dal client
147    {
148    read(ds_sock_acc,&numero,sizeof(numero));
149    if (numero!= 0) fprintf (fp,"%d ",numero); // con 0 il client segnala la fine
dell'analisi pacchetto
150    }
151    while (numero!=0);
152
153    // chiudo il file aperto dal figlio
154
155    if ((fclose (fp))== -1 ) printf ("Impossibile chiudere file %s \n", nome_file);
156
157    if ((close(ds_sock_acc)) == -1) printf ("Impossibile chiudere soket\n");
158
159    while (kill(getppid(),SIGUSR1)==-1); // invio al padre il segnale usr1
160    // segnalando fine del figlio
161    exit (0); // termine del figlio
162                }
163          }
164    } // ritorno all'inizio del ciclo infinito
165 }
166
167 // gestisco le azioni da compiere prima di uscire
168 void gestoreUscita (int sig)
169 {
170    kill (0,SIGUSR2); // invio il segnale a tutti i figli,il padre lo ignora
171    salvataggio();
172    exit(1);
173 }
174
175
176 //gestisco i figli che terminano
177 void gestoreSig1 (int sig)
178 { static figli_terminati = 0; // inizializzo la variabile solo la prima volta
179
180    figli_terminati++;
181    printf ("Figlio è terminato %d°°°°°°°°°\n",figli_terminati);
182    if (figli_terminati == figli_necessari)
183    salvataggio();
184 }
```

```
185
186 // gestisco la chiusura del figlio sotto il volere del padre
187 void gestoreSig2 (int sig)
188 { printf("Figlio PID : %d chiuso\n",getpid());
189    exit(-1); // chiudo il figlio
190
191 }
192
193 // effettuo il salvataggio dei file nell'unico file FINAL.DAT
194 void salvataggio ()
195 { int i,j=0; // i contatore file letti j contatore numeri scritti per andare poi a capo
196    int numero; // per la lettura dei numeri nei file
197    char nome_file [12];
198    FILE *fp,*fp_finale;
199
200    printf("_____\n");
201    printf("I CLIENT HANNO TERMINATO L'ELABORAZIONE\n");
202    printf("\n");
203    printf("exit file : FINAL.DAT\n");
204    printf("_____\n");
205
206    if ((fp_finale=fopen("FINAL.DAT", "w")) == NULL)
207    {
208    printf ("Impossibile salvare i risultati\n");
209    exit(-1);
210    }
211
212    for (i=1; i <=num_figli; i++)
213    {
214
215    sprintf (nome_file,"%d",i); // creo il nome del file dall'indice
216
217
218    if ((fp=fopen(nome_file, "r")) == NULL)
219    {
220 printf ("Impossibile aprire file per salvataggio finale %s\n",nome_file);
221 continue;
222 }
223
224 while (fscanf(fp,"%d",&numero)!=EOF)
225 {
226    if ((++j%10)==0) fprintf(fp_finale,"\n"); // salto una riga ogni 10 numeri scritti
227    fprintf(fp_finale,"%d ",numero); // trasferisco il contenuto dei file in FINAL.DAT
228
229 }
230
231 close(fp);
232
233 //cancella il file letto
234 if ((remove(nome_file)) == -1 ) printf ("Impossibile rimuovere il file %s \n",
nome_file);
235
236    }
237    close(fp_finale);
238    exit(0);
239 }
```

Dalla riga 1 alla riga 8 vengono incluse le librerie d'interesse, in particolare alla riga 8 viene inclusa la libreria "mio.h" che deve essere memorizzata all'interno della directory in cui sono presenti i file sorgenti da compilare. Al suo interno sono definite le seguenti costanti :

```
#define BUF_DIM 8       // dimensione del buffer messaggi tra server e client

#define CODA_DIM 16     // dimensione della coda per i client
```

La seconda costante è utilizzata solo nel programma server quindi può essere definita direttamente nel file sorgente del server. Per la prima costante, pur rimanendo valida la possibilità di dichiararla all'interno dei singoli file sorgenti risulta, in caso di variazione di quest'ultima, più pratica tale soluzione, poiché, dal momento che i due sorgenti (server e client) devono averla definita di uguale valore, in questo modo variandola in un solo punto e ricompilando i sorgenti si cambia la costante ad entrambi.

Dalla riga 11 alla riga 14 vengono dichiarate i prototipi delle funzioni che verranno usate, in particolare le prime tre funzioni servono per gestire i segnali SIGUSR1, SIGUSR2 e SIGINT, mentre l'ultima funzione permette il salvataggio di tutti i numeri primi trovati in un unico file FINAL.DAT. Nelle righe 16 e 17 sono dichiarate due variabili globali in modo da poter essere viste anche dalle funzioni che gestiscono i segnali. La variabile "figli_necessari" memorizza il numero di client ovvero figli, che saranno necessari per analizzare l'intervallo di interi impostato. La variabile "num_figli" tiene il conteggio del numero dei figli che sono stati creati fino a quel momento. Dalla riga 21 alla riga 34 sono dichiarate le variabili la cui visibilità è ristretta alla funzione main( ).

Alla riga 37 viene effettuato il controllo del numero degli argomenti introdotti, se risulta essere inferiore a 5 viene visualizzata la sintassi d'invocazione del programma server, per poi terminare l'esecuzione. Gli argomenti introdotti vengono poi caricati con le opportune trasformazioni nelle variabili interne, in particolare per gli intervalli si usa la funzione di conversione int atol (char *string) che converte una stringa in un intero di tipo long.

Nella riga 48 viene effettuato il controllo sul primo numero dell'intervallo in maniera che risulti non inferiore[2] a 2. Nella riga 54 viene effettuato il controllo sull'intervallo inserito, ovvero che primo_numero sia inferiore di ultimo_numero. Nel caso in cui uno dei test dovesse verificare una stato inconsistente l'esecuzione del programma verrebbe interrotta con relativa messaggistica d'errore.

Tra la riga 61 e 64 viene impostato il numero di figli necessari, sfruttando il fatto che, il rapporto tra interi è un intero, e l'operatore modulo. Si mette in evidenza che nella riga 66 viene esplicitamente segnalata la modalità di uscita che riconosce il sistema ovvero Control + C.

Tra la riga 68 e 72 viene creata la socket del server e gestito l'eventuale fallimento dell'operazione, in caso di corretta esecuzione in ds_sock è presente il descrittore della socket. Il dominio impostato è AF_UNIX il tipo di comunicazione è SOCK_STREAM e tramite lo 0 segnalo che voglio utilizzare il protocollo di comunicazione standard definito per il tipo di comunicazione selezionata. Tra la riga 75 e 77 vengono impostati i campi della struttura sock_addr della variabile server, in particolare si ripete il dominio, si imposta la porta inserita precedentemente dall'utente e l'indirizzo IP del server[3]. Tra la riga 79 e 82 vengono riportati sullo standard output il valore dei campi della struttura e il numero dei figli necessari.

Creata la socket, al fine di renderla utilizzabile è necessario assegnarle un indirizzo interno, questo è effettuato tra la riga 85 e 90 tramite la chiamata di sistema

int bind (int ds_sock, struct sock_addr *my_addr, int addr_len)

Prima di accettare connessioni viene abilitata la possibilità di avere code di attesa in modo da evitare di perdere tentativi di connessione da parte di client esterni, il numero accettato è definito dalla costante CODA_DIM presente nel file mio.h.

Il programma arrivato a questo punto avendo impostato tutti i parametri necessari è pronto ad accettare eventuali connessioni da parte di client, questo viene svolto all'interno di un ciclo infinito che può essere interrotto solo dai figli dei server (con opportuna segnalazione tramite SIGUSR1) appena viene analizzato l'intero intervallo e dall'utente premendo Control + C.

Il ciclo infinito è ottenuto da while (2), ovvero da una condizione il cui risultato è maggiore di 0 quindi sempre vera. Un ulteriore ciclo che può risultare infinito è dato dal while in cui l'operazione di controllo è svolta sul valore ritornato dalla funzione

[2] In questo modo vengono esclusi i numeri negativi lo 0 e 1 (non si è fatto uso degli unsigned long int).

[3] Viene usata la funzione unsigned int inet_addr(* char) per convertire la stringa IP nel formato del campo d'interesse.

int accept (int ds_sock, struct sock_addr *addr, int *addrlen)

questa ritorna –1 se nessun client si è connesso altrimenti ritorna un descrittore di socket diverso da quello del server ma connesso allo stesso canale di comunicazione, la struttura client viene riempita con gli attributi del client che ha richiesto la connessione.
Alla riga 102 viene effettuato un test per vedere se l'intervallo controllato[4] coincide con quello impostato dall'utente, in tal caso il client viene avvisato dello stato con il messaggio "fine", sarà compito dell'ultimo figlio far terminare il processo padre. Se l'intervallo non è stato ancora tutto controllato, si entra nel blocco istruzione della condizione else.

Viene incrementata la variabile "num_figli", questo avviene prima della fork perché deve essere disponibile sia al padre che al figlio[5]. Alla riga 108 viene aggiornata la variabile controlla e poi vengono calcolati gli indici dell'intervallo del pacchetto da inviare al client, in particolare sup nella riga 111 viene ritoccato se l'intervallo dell'utente non è multiplo del quanto impostato.
Viene poi armato il segnale con il quale il padre riconosce la terminazione di un suo figlio, successivamente viene eseguita la fork ( ). Se il pid è uguale a –1 vuol dire che non è andata a buon fine quindi il processo termina[6]. Se il pid è diverso da 0[7] (è il padre a eseguire tale codice) armo il segnale SIGUSR2 in modo da ignorarlo[8] e poi armo il segnale per gestire il segnale inviato dal sistema operativo quando viene premuto Control + C ovvero SIGINT.

Nel caso in cui il pid sia nullo è il figlio ad eseguire tali righe di codice. Alla riga 130 viene chiusa la socket ereditata dal padre che non verrà usata. Viene successivamente armato il segnale SIGUSR2 gestito dalla funzione gestoreSig2, in modo da poter gestire la chiusura del figlio quando il padre invia tale segnale. In tale funzione viene segnalato il pid del figlio e poi avviene la chiusura del processo associato[9].

Il figlio poi legge lo stato di pronto inviato dal client e invia lo stato di OK al client e il pacchetto da analizzare. Con la variabile "num_figli" viene creata una stringa riportata nella variabile "nome_file". Quest'ultima variabile viene utilizzata per creare il file in cui il figlio memorizzerà i numeri primi che il client gli invierà.

Una volta creato il file il figlio si mette a ricevere dal canale di comunicazione tutti i numeri che vengono inviati dal client e se risultano essere diversi da 0 vengono memorizzati all'interno del file. Il numero 0 viene ricevuto quando il client finisce di analizzare l'intervallo assegnatogli o quando viene terminato premendo il tasto Control + C. Ricevuto il codice 0 il file associato al figlio viene chiuso e anche la relativa socket[10]. Alla riga 159 viene inviato il segnale SIGUSR1 al processo padre[11], dal momento che possono essere presenti più figli concorrenti e il padre svolge anche altre attività, al fine di evitare di perdere il segnale, lo si è trasformato tramite un ciclo di controllo while in un segnale bloccante qualora non vada a buon fine. Questa precauzione è necessaria poiché se si fosse usato il segnale SIGCHLD, essendo non bloccante, si sarebbe potuto perdere il conteggio dei figli terminati, che coincide con il termine dell'analisi di un pacchetto da parte di un client, impedendo in questo modo la chiusura automatica del server quando viene controllato l'intero intervallo impostato dall'utente. Quando il segnale viene riconosciuto, il figlio termina la propria esecuzione. Si capisce quindi che ad ogni pacchetto è associato un figlio, dunque uno stesso client in richieste successive verrà servito da figli differenti.

---

4 Si suppone che ogni figlio abbia portato o porterà a termine il controllo sul pacchetto ricevuto.
5 Si ricorda inoltre che tale variabile è globale.
6 Questo può avvenire se il numero di processi che il S.O. è stato impostato ad eseguire è stato raggiunto, tale situazione non è comunque gestita e il processo viene terminato.
7 Il padre possiede il pid del figlio e il figlio non possiede il pid del padre.
8 Questo segnale viene inviato a tutti i processi del gruppo in modo da chiudere i figli attivi, per evitare che venga chiuso anche il padre prima del dovuto salvataggio viene ignorato.
9 Tutti i figli ereditano tale funzione e quindi la eseguiranno alla ricezione del rispettivo segnale.
10 Se la socket viene chiusa poiché il server è stato chiuso dall'utente il client rileva la chiusura della pipe.
11 Il pid del padre viene prelevato dal figlio usando la funzione int getppid ( );

Rimangono ora da analizzare le funzioni che completano il file sorgente del server.

void gestoreUscita (int sig)

Questa funzione serve per gestire, da parte del padre il segnale di interruzione del processo (Control+C). Come prima cosa vengono interrotti tutti i processi del gruppo[12] in modo da poter poi salvare tramite la funzione salvataggio ( ) i vari file dei figli nell'unico file FINAL.DAT. il processo termina riportando il valore –1, segnalando a chi volesse utilizzarlo che è terminato prematuramente per volontà dell'utente.

void gestoreSig1 (int sig)

Tale funzione serve per gestire il conteggio dei figli che terminano, per tale ragione la variabile "figli_terminati" è dichiarata statica.
Quando il numero dei figli terminati è pari al numero dei figli necessari[13] viene effettuato il salvataggio automatico richiamando la funzione salvataggio ( ).

void gestoreSig2 (int sig)

Tale funzione serve per gestire la chiusura dei figli in risposta al segnale SIGUSR2 inviato dal processo padre quando gestisce la chiusura volontaria da parte dell'utente Control+C. L'uscita avviene con il valore –1.

void salvataggio ( )

In tale funzione i vari numeri primi contenuti nei file creati dai figli vengono salvati nell'unico file FINAL.DAT.

I file vengono aperti per mezzo del nome dell'indice, che va da 1 al numero contenuto nella variabile "num_figli"[14], trasformato nel nome del file ovvero in una stringa. I numeri vengono prelevati dal file aperto e copiati all'interno del file finale per mezzo del ciclo compreso tra le righe 224 e 229, ogni dieci numeri scritti viene scritto il carattere speciale di ritorno accapo. L'uscita dal ciclo avviene quando viene prelevato il codice di fine file (EOF), dopo l'uscita il file scritto dal figlio viene chiuso e cancellato. Alla fine del loop principale che ha inizio dalla riga 212 viene chiuso anche il file FINAL.DAT e il processo ha termine regolarmente.

Codice Sorgente del Client[15]

```
1 #include <sys/types.h>  //librerie da includere
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4 #include <netdb.h>
5 #include <stdio.h>
6 #include <string.h>
7 #include <signal.h>
8 #include "mio.h"        // cerco nel percorso del file client
9
10 // definizioni di funzioni
11
12 void gestoreFineConn (int);   // gestisce segnale PIPE
13 void gestoreUscita (int);     //gestisco il segnale di uscita
14
15 int isPrimo(long);            // funzioni computazionali
16 int analizza(long,long);
```

[12] Il padre non è interrotto poiché ignora tale segnale.
[13] Si ricorda che tale variabile è globale e posso quindi accedere al suo indirizzo anche da tale funzione.
[14] Si ricorda che tale variabile è globale e mantiene il numero dei figli creati quindi il numero dei file creati. Se il figlio non ha fatto in tempo ad aprire il file il salvataggio relativo al suo pacchetto salta, segnalando l'errore, al pacchetto successivo (istruzione "continue" riga 225).
[15] Per una migliore identazione del programma si rimanda ai sorgenti in formato pdf o html presenti nel file Software.zip scaricabile dal sito www.LaurTec.com.

```
17 void invia(long);
18
19 int ds_sock; // tutte le funzioni sono a conoscenza di questo parametro
20
21
22 int main (int argc, char *argv[])
23 {
24     struct sockaddr_in client;
25     char buffer [BUF_DIM];     // buffer per trasmettere messaggi stringa al server
26     char *ip_Server;    // ip del server con cui ci si vuole connettere
27     int porta_Server;   // stessa porta del server
28     long inf, sup; // estremi dell'intervallo da analizzare
29     char risposta; // memorizza la risposta se continuare o meno l'analisi
30
31     if (argc<3)  // controllo che il numero dei parametri sia corretto
32     {
33     printf("Introdurre %s <IP_Server> <porta_TCP_Server>\n",argv[0]);
34     exit(-1);
35     }
36
37     ip_Server = argv[1]; // memorizzo nelle variabili i parametri introdotti
38     porta_Server = atoi(argv[2]);
39
40     while (3)    //il ciclo finisce con la chiusura del server
41     {
42
43     if ((ds_sock = socket(AF_UNIX,SOCK_STREAM,0))<0) // creo la socket
44           {
45           printf ("Errore di socket");
46           printf ("Impossibile connetersi al Server \n ");
47           exit (-1);
48           }
49
50     printf("**************************************\n");
51     printf("Premere Control + C per uscire\n");
52     printf("\n");
53     printf ("ds server %d\n",ds_sock);
54
55     client.sin_family = AF_UNIX; // imposta la struttura del server a cui connettermi
56     client.sin_port = porta_Server;
57     client.sin_addr.s_addr = inet_addr(ip_Server); // converto IP stringa in long uns.
58
59
60
61     if ((connect(ds_sock,&client, sizeof(client))) <0 ) //provo a connettermi con il
server
62     {
63     printf("Impossibile effettuare connessione\n");
64     exit (-1);
65     }
66
67     signal (SIGINT,gestoreUscita);   // armo il segnale di uscita
68     signal (SIGPIPE,gestoreFineConn);       // armo il segnale di chiusura PIPE
69
70     write (ds_sock,"Pronto",BUF_DIM);       // invio il mio stato di pronto al server
71     read (ds_sock,buffer,BUF_DIM);
72     printf ("Stato connessione : %s\n",buffer);// legge lo stato del server o fine o OK
73     if (strcmp(buffer,"fine")==0)    // il server ha finito i pacchetti
74     {
75     printf ("Il Server ha concluso la computazione\n");
76     if ((close(ds_sock))== -1 )
77     printf ("Impossibile chiudere la socket server\n");
78     exit(-1);
79     }
80     read (ds_sock,&inf,sizeof(inf)); // leggo l'intervallo da analizzare inviato dal
server
81     read (ds_sock,&sup,sizeof(sup));
82
83     printf ("Intervallo da analizzare [%d,%d] \n",inf,sup);
```

```
84
85      analizza(inf,sup); // funzione di analisi del pacchetto
86
87      printf("\n");
88
89      printf ("Sto per chiudere la porta\n");
90
91      if ((close(ds_sock))== -1 )
92              printf ("Impossibile chiudere la socket server\n");
93
94      printf ("Ho chiuso le porte\n");
95      } // ritorno all'inizio del ciclo
96  }
97
98
99  void gestoreUscita (int sig ) // gestisco una chiusura del client prematura
100 {
101     invia (0); // in questo modo interrompo il figlio del server
102     exit (-1);
103 }
104
105
106
107 void gestoreFineConn (int sig)      // gestisco il segnale chiusura PIPE
108 {
109     printf ("\n");
110     printf ("Il server ha chiuso la socket\n"); // avviso del segnale ricevuto ed esco
111     exit(-1);
112 }
113
114
115 int analizza (long inf, long sup) // funzione che scandisce i vari numeri
dell'intervallo
116 {  long i;
117     for (i = inf; i<=sup; i++)
118     if ((isPrimo(i))==1) invia(i); // se i è primo lo invio al figlio del server
119
120     invia(0); // termino la comunicazione con il figlio del server a fine ciclo
121 }
122
123
124 int isPrimo (long numero)    // funzione che controlla se il numero è primo
125 {  long i;
126     long resto;
127     for (i=2; i<numero; i++)  // non considero il numero 1 (non inviato dal server)
128     {
129     resto = numero%i;
130     if (resto==0) return (0); // il numero non è primo
131     }
132     return(1); // se sono arrivato fino a qui il numero è primo
133 }
134
135
136 void invia (long numero) // funzione che invia il segnale
137 {
138     write (ds_sock,&numero,sizeof(numero));
139 }
```

Dalla riga 1 alla riga 17 vengono incluse le librerie d'interesse e dichiarati i prototipi delle funzioni che verranno utilizzate. Alla riga 19 viene dichiarata la variabile "ds_sock" come globale. Si vede subito che la funzione main accetta gli argomenti in ingresso, che permettono di introdurre l'IP del Server e il numero di porta da utilizzare per la comunicazione.

Dopo la dichiarazione delle variabili che saranno visibili all'interno della funzione main si fa il controllo sul numero degli argomenti introdotti e se il controllo passa vengono assegnati gli argomenti introdotti alle rispettive variabili interne (con opportune trasformazioni sul tipo). Il client secondo le specifiche deve poter richiedere nuovi pacchetti quindi deve poter inoltrare più richieste. Per scopi

pratici la richiesta avviene automaticamente, tramite un ciclo infinito[16], alla fine dell'analisi di un pacchetto, l'utente può comunque interrompere l'analisi premendo Control+C.

Alla riga 43 viene creata la socket con cui il client comunicherà con il server. Dalla riga 55 alla riga 57 vengono settati gli attributi della struttura della variabile client[17]. Impostati tali valori viene inoltrata la richiesta verso il Server per la connessione tramite la chiamata di sistema

int connect (int ds_sock, struct sock_addr *addr, int addrlen)

Dopo la connessione, se questa è accettata o comunque va a buon fine, viene armato il segnale per gestire l'uscita tramite Control+C e il segnale per gestire la chiusura della pipe (SIGPIPE). Viene successivamente inviato il messaggio di "pronto" al server e si legge il messaggio del server, se il messaggio è "fine" vuol dire che il server ha terminato i pacchetti quindi viene segnalato tale stato e il processo client ha termine, se il messaggio è "OK" vengono letti i due intervalli del pacchetto da analizzare e vengono stampati sullo standard output. Il pacchetto ricevuto viene inviato alla funzione analizza (inf,sup) che cerca i numeri primi in tale intervallo (si veda di seguito per maggiori dettagli). Analizzato il pacchetto viene chiusa la socket avvisando del nuovo stato e viene iniziato un nuovo ciclo in cui verrà aperta una nuova connessione con il server, che dedicherà un altro figlio per tale comunicazione.
Rimangono ora da analizzare le funzioni che completano il file sorgente del server.

void gestoreUscita (int sig)

In questa funzione viene inviato tramite la funzione invia ( ) il numero 0 in modo da avvisare che il client ha terminato la computazione. Tale tipo di terminazione avviene a seguito della pressione dei tasti Control+C, quindi la computazione non è in realtà terminata regolarmente. Per avvisare il server del termine anomalo e fare in modo che faccia terminare l'analisi ad un altro client è possibile inviare per esempio un numero negativo.

void gestoreFineConn (int sig)

Tale funzione viene invocata quando il sistema operativo invia il segnale di chiusura pipe, ovvero il canale con cui il client si è connesso con il server. Tale stato viene segnalato e poi avviene la terminazione del processo.

void analizza (long inf, long sup)

In tale funzione avviene il controllo del pacchetto inviato, questa potrebbe essere sostituita con qualunque funzione in cui venga richiesta l'analisi su un intervallo di interi. L'analisi avviene per mezzo di un ciclo che scansiona i numeri all'interno dell'intervallo assegnato. Il controllo se il numero è primo o meno viene delegato alla funzione isPrimo ( ). Se il numero è primo questo viene inviato al server tramite la funzione invia ( ). Al termine del ciclo viene inviato il numero 0 per segnalare al server che il client ha terminato l'analisi dell'intervallo assegnatogli.

int isPrimo (long inf, long sup)

Questa funzione controlla se è verificata la definizione di numero primo facendo il controllo sul resto della divisione tra il numero da controllare e i numeri compresi tra 2 e (numero-1). Il ciclo di controllo termina se una divisione dà resto nullo (numero non primo), si capisce quindi che il ciclo termina solo raggiunto (numero-1) qualora il numero risultasse primo, quindi piano piano che il numero cresce l'analisi diviene sempre più lunga (la funzione non è ottimizzata per il calcolo dei numeri primi). In particolare se il numero è primo è ritornato il valore 1 mentre se non lo è viene riportato il numero 0.

[16] Il ciclo termina o per volontà dell'utente o per termine pacchetti o per chiusura del server (chiusura pipe).
[17] L'indirizzo IP è quello del server a cui ci si deve connettere.

void invia (long numero)

Tale funzione viene utilizzata per inviare al server il numero trovato, pur venendo svolta una sola operazione è stata usata come funzione poiché viene utilizzata in più punti ed inoltre potrebbe essere modificata facendo uso di un protocollo di comunicazione ad hoc tra il server e il client.

Appendici

Come compilare e installare i programmi

I programmi sono stati scritti in C e li si è compilati facendo uso del compilatore distribuito da GNU ovvero gcc. Si consiglia di creare una directory nella quale inserire i sorgenti da compilare, dal momento che il programma client una volta in esecuzione creerà dei file che in caso di crollo del sistema e impossibilità di salvataggio nel file unico FINAL.DAT saranno confinati in una directory.

mkdir prove

Inserire il dischetto[18] nell'apposita unità e montarla.

mount /dev/fd0 /mnt

copiare i files all'interno della cartella precedentemente creata.

cp <origine> <destinazione>

esempio :

cp /mnt/server.c /home/prove

Tale procedura, una volta montata l'unità floppy, può essere svolta automaticamente eseguendo lo script setup dalla directory mnt.

./setup

Lo script crea una directory prove nella directory home, copiando file sorgenti e compilati.

La compilazione dei programmi tramite il compilatore gcc può essere comunque eseguita :

gcc server.c –o server

gcc client.c –o client

Per eseguire i programmi è necessario aprire almeno due shell, una per il server e una per il client (usare una shell per ogni client). In ogni shell ci si deve posizionare nella cartella prove e lanciare prima il server e successivamente i client. Per eseguire il server e il client bisogna utilizzare tale sintassi (nelle rispettive shell) :

./server <porta_TCP> <primo_intero> <ultimo_intero> <dim_pacchetti>

./client <indirizzo_IP_server> <porta_TCP_server>

[18] Si sta supponendo che i sorgenti siano stati memorizzati in un dischetto.

un esempio di esecuzione è il seguente :

./server 2000 3 10000 1000

./client 128.0.0.0 2000

Collaudo dei programmi

I programmai realizzati, dal momento che il server si mette in attesa "occupando" la shell nella quale è stata eseguita, si necessita di un'altra shell per ogni processo client che si vuole far comunicare con il server. Il server, al fine di evitare errore nel tentativo di connessione del client deve essere eseguito per prima. Tenendo conto della sintassi di invocazione dei processi

```
server <porta_TCP> <primo_intero> <ultimo_intero> <dim_pacchetti>

client <indirizzo_IP_server> <porta_TCP_server>
```

possibili prove da effettuare sono le seguenti :

```
./server 2000 1000 10000 3000
./client 128.0.0.0 2000
```

tale sequenza (un comando per shell), sempre che la porta non risulti utilizzata da altri processi, o si verifichino altri errori di carattere non predittivo andrebbe a buon fine.

Se successivamente venisse eseguita quest'altra sequenza

```
./server 2000 1000 10000 2000
./client 128.0.0.0 2000
```

il server non riuscirebbe ad indirizzare la socket poiché la porta è stata precedentemente usata da un altro processo.

Aumentando l'intervallo da controllare, rimanendo sempre nel limite permesso da un tipo long (2^{16}-1), è possibile vedere l'attività del client e del server potendo in questo modo interrompere l'uno o l'altro tramite la pressione dei tasti Control+C.

L'aumento dell'intervallo permette anche l'utilizzo di più client dal momento che ogni client inoltrerà una nuova richiesta di pacchetti alla fine di ogni analisi, quindi con intervalli piccoli l'analisi terminerebbe prima di avere tempo di avviare un altro processa client. Si consiglia comunque di predisporre le shell per i processi client in modo da avviarli entrambi in poco tempo.
Come sarà visibile dagli output in Appendice C, quando più client concorrono tra loro nell'analisi di diversi pacchetti avranno da analizzare pacchetti non contigui, ciò nonostante l'ordine dei pacchetti viene mantenuto grazie al particolare nome del file che ogni processo figlio del server adotta per immagazzinare i numeri primi inviati dal client.

Altre tipologie di errori possono essere testate alterando le informazioni che il server salva durante la sua esecuzione, inserendo parametri in numero errato o utilizzando sintassi non corretta.

Esempi di input/output dei programmi

Il server e il client sono due programmi distinti, in particolare ad uno stesso processo server si

possono connettere più processi client. Nell'esempio 1 viene riportato un errore del processo server che non riesce ad effettuare l'indirizzamento della socket (errore di bind).

```
mauro@linux:~/prove> ./server 1000 3 100000 10000

Premere Control + C per uscire
ds server 3
Server IP : 128.0.0.0
Porta di connessione : 1000
Numero figli necessari : 10
Bind fallito
Impossibile avviare il Server
mauro@linux:~/prove>
```

Esempio 1:

Nell'esempio 2 viene riportato l'errore di un processo client che cerca di connettersi ad un server che non è attivo all'indirizzo o alla porta (o ad entrambi) selezionate.

```
mauro@linux:~/prove> ./client 128.0.0.0 2
************************************
Premere Control + C per uscire

ds server 3
Impossibile effettuare connessione
mauro@linux:~/prove>
```

Esempio 2:

Nell'esempio 3 viene riportata l'uscita del processo server al quale si connette un processo client che ripetutamente richiede pacchetti da analizzare. Il processo server viene chiuso quando il client ha chiesto i cinque pacchetti con cui è stato suddiviso l'intervallo, alla sesta richiesta il server invia lo stato "fine".

```
mauro@linux:~/prove> ./server 1000 3 10000 2000

Premere Control + C per uscire
ds server 3
Server IP : 128.0.0.0
Porta di connessione : 1000
Numero figli necessari : 5
****************************************
Sono in attesa di connessioni
Un client si è connesso
ds client 4
messaggio ricevuto Pronto
****************************************
Sono in attesa di connessioni
Un mio figlio è terminato 1
Un client si è connesso
ds client 5
messaggio ricevuto Pronto
****************************************
****************************************
Sono in attesa di connessioni
Un mio figlio è terminato 2
Un client si è connesso
ds client 6
```

```
****************************************
Sono in attesa di connessioni
messaggio ricevuto Pronto
Un mio figlio è terminato 3
Un client si è connesso
ds client 7
****************************************
Sono in attesa di connessioni
messaggio ricevuto Pronto
Un mio figlio è terminato 4
Un client si è connesso
ds client 8
messaggio ricevuto Pronto
****************************************
Sono in attesa di connessioni
Un mio figlio è terminato 5

_____________________________________________
I CLIENT HANNO TERMINATO L'ELABORAZIONE

exit file : FINAL.DAT

_____________________________________________
mauro@linux:~/prove>
```

Esempio 3:

Si può osservare che i figli essendo processi concorrenti non terminano tutti nello stesso punto.

Nell'esempio 4 viene riportato l'uscita del processo client che si è connesso al server la cui uscita è riportata nell'esempio 3. Si può vedere che il client termina quando terminano i pacchetti del server.

```
mauro@linux:~/prove> ./client 128.0.0.0 1000
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [3,2002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [2003,4002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire
ds server 3
Stato connessione : OK
Intervallo da analizzare [4003,6002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire
```

```
ds server 3
Stato connessione : OK
Intervallo da analizzare [6003,8002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [8003,10000]

Sto per chiudere la porta
Ho chiuso le porte
************************************

Premere Control + C per uscire

ds server 3
Stato connessione : fine
Il Server ha concluso la computazione
mauro@linux:~/prove>
```

Esempio 4:

Nell'esempio 5 è riportato il contenuto del file di uscita FINAL.DAT.

```
3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409

[…]

9391 9397 9403 9413 9419 9421 9431 9433 9437 9439
9461 9463 9467 9473 9479 9491 9497 9511 9521 9533
9539 9547 9551 9587 9601 9613 9619 9623 9629 9631
9643 9649 9661 9677 9679 9689 9697 9719 9721 9733
9739 9743 9749 9767 9769 9781 9787 9791 9803 9811
9817 9829 9833 9839 9851 9857 9859 9871 9883 9887
9901 9907 9923 9929 9931 9941 9949 9967 9973
```

Esempio 5:

Nell'esempio 6 viene riportato un'altra analisi avviata con il server, che però viene interrotta dall'utente prima che i client abbiano finito la ricerca dei pacchetti assegnati e in particolare dell'intervallo assegnato al server. Questa volta i client, come visibile dai PID dei processi figli interrotti, sono due, infatti si sono eseguiti due processi client su due shell distinte, che chiedono ad ogni fine analisi un nuovo pacchetto fino a quando rilevano la chiusura del canale di comunicazione, si veda esempio 7 e 8.

```
mauro@linux:~/prove> ./server 2000 3 100000 10000
```

```
Premere Control + C per uscire
ds server 3
Server IP : 128.0.0.0
Porta di connessione : 2000
Numero figli necessari : 10
****************************************
Sono in attesa di connessioni
Un client si è connesso
ds client 4
messaggio ricevuto Pronto
****************************************
Sono in attesa di connessioni
Un client si è connesso
ds client 5
messaggio ricevuto Pronto
****************************************
Sono in attesa di connessioni
Un mio figlio è terminato 1
Un client si è connesso
ds client 6
****************************************
Sono in attesa di connessioni
messaggio ricevuto Pronto
Un mio figlio è terminato 2
Un client si è connesso
ds client 7
****************************************
Sono in attesa di connessioni
messaggio ricevuto Pronto
Un mio figlio è terminato 3
Un client si è connesso
ds client 8
****************************************
Sono in attesa di connessioni
messaggio ricevuto Pronto
Un mio figlio è terminato 4
Un client si è connesso
ds client 9
messaggio ricevuto Pronto
****************************************
Sono in attesa di connessioni
Un mio figlio è terminato 5
Un client si è connesso
ds client 10
****************************************
Sono in attesa di connessioni
messaggio ricevuto Pronto
Figlio PID : 1089 chiuso
Figlio PID : 1088 chiuso
_____________________________________________
I CLIENT HANNO TERMINATO L'ELABORAZIONE

exit file : FINAL.DAT

_____________________________________________
mauro@linux:~/prove>
```

Esempio 6:

Sono sotto riportate le uscite dei due client connessi al server dell'esempio 6.

```
mauro@linux:~> cd prove
mauro@linux:~/prove> ./client 128.0.0.0 2000
```

```
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [3,10002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [20003,30002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire

Il server ha chiuso la socket
mauro@linux:~/prove>
```

Esempio 7:

```
mauro@linux:~/prove> ./client 128.0.0.0 2000
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [10003,20002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [30003,40002]

Sto per chiudere la porta
Ho chiuso le porte
************************************
Premere Control + C per uscire

ds server 3
Stato connessione : OK
Intervallo da analizzare [50003,60002]

Il server ha chiuso la socket
mauro@linux:~/prove>
```

Esempio 8:

Si fa notare che gli intervalli analizzati dai client non sono tra loro contigui, questo, per il modo con cui vengono salvati i numeri primi, non altera l'ordine numerico al salvataggio nel file FINAL.DAT.

Bibliografia

www.LaurTec.com : sito di elettronica dove poter scaricare gli altri articoli menzionati, aggiornamenti e progetti.