

```

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
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
socket
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-primο_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      // assegno 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 socket\n");
158
159         while (kill(getppid(), SIGUSR1) == -1);    // invio
al padre il segnale usr1
160
161         // segnalando fine del figlio
162         exit (0); // termine del figlio
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 {
189     printf("Figlio PID : %d chiuso\n",getpid());
190     exit(-1);    // chiudo il figlio
191 }
192
193 // effettuo il salvataggio dei file nell'unico file FINAL.DAT
194 void salvataggio ()
195 {
196     int i,j=0; // i contatore file letti      j contatore numeri
197     // scritti per andare poi a capo
198     int numero; // per la lettura dei numeri nei file
199     char nome_file [12];
200     FILE *fp,*fp_finale;
201
202     printf("_____ \n");
203     printf("I CLIENT HANNO TERMINATO L'ELABORAZIONE\n");
204     printf("\n");
205     printf("exit file : FINAL.DAT\n");
206     printf("_____ \n");
207
208     if ((fp_finale=fopen("FINAL.DAT", "w")) == NULL)
209     {
210         printf ("Impossibile salvare i risultati\n");
211         exit(-1);
212     }
213
214     for (i=1; i <=num_figli; i++)
215     {
216         sprintf (nome_file,"%d",i);          // creo il nome del
217         // file dall'indice
218
219         if ((fp=fopen(nome_file, "r")) == NULL)
220         {
221             printf ("Impossibile aprire file per salvataggio
222             finale %s\n",nome_file);
223             continue;
224         }
225
226         while (fscanf(fp,"%d",&numero)!=EOF)
227         {
228             if ((++j%10)==0) fprintf(fp_finale,"\n"); // salto
229             // una riga ogni 10 numeri scritti
230             fprintf(fp_finale,"%d ",numero); // trasferisco il
231             // contenuto dei file in FINAL.DAT
232         }
233
234         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 }
240
```