ISTITUTO NAZIONALE di FISICA NUCLEARE
Laboratori Nazionali di Frascati
Servizio Informazione e Documentazione Scientifica
Via Enrico Fermi, 40 – 00044 Frascati RM

Stages estivi 2014

Relazione di fine corso

Gruppo di lavoro "**L**" - Informatica

Alessandro PONTIS	Ist. Tecn. "Enzo Ferrari" - Roma
Andrea SILVI	Ist. Tecn. "Sandro Pertini" - Genzano di Roma
Gheorghe SIRBU	Ist. Tecn. "Galileo Galilei" - Roma
Massimo STANZIONE	Ist. Tecn. "Enrico Fermi" - Frascati
Marco TITI	Ist. Tecn. "Giancarlo Vallauri" - Velletri

Tutore: Giuseppe Fabio FORTUGNO

Indice generale

Obiettivo

Progettazione e gestione sotto il sistema operativo UNIX di un sistema dati per un esperimento che produce almeno 3000 Terabyte.

Ambienti di sviluppo

Macchina *guest102* *(locale)*	
Sistema Operativo	Solaris
Compilatori	gcc
Editor	vim

Macchina *ke08* *(remota)*	
Sistema Operativo	AIX 5.3
Compilatori	xlc
Editor	vi

Materiale utilizzato

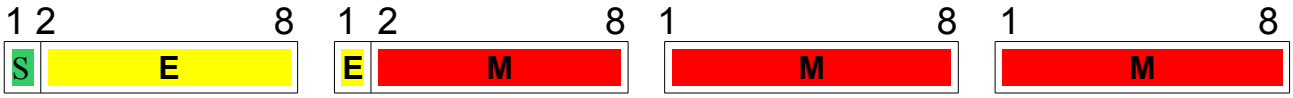
- E. Quigley, “UNIX shells by example”, ed. Prentice Hall;
- H. e P. Deitel, “Corso completo di programmazione C”, ed. Apogeo;
- M. J. Bach, “UNIX Architettura di sistema”, ed. Jackson;
- M. J. Rochkind, “UNIX Programmazione avanzata”, ed Jackson;
- M. Kernighan – D. Ritchie, “The C programming language”;
- M. J. Donahoo – K. L. Calvert, “TCP/IP sockets in C”;
- F. Ferroni, “Programmazione dei socket di rete in GNU/LINUX”
- Dispensa UNIX Comandi di base;
- Dispensa UNIX Editing e compilazione;
- Appunti.

Argomenti svolti

- **Ambiente di lavoro**
  - Struttura dell'acceleratore di particelle DAΦNE;
  - Struttura dell'esperimento KLOE;
  - Struttura degli edifici di calcolo;
  - Analisi di dati grezzi prodotti dall'esperimento KLOE.
- **Sistema operativo AIX**
  - Introduzione;
  - Kernel e bootstrap;
  - Gestione processi;
  - File-system;
  - Analisi delle shell;
  - Flussi di dati;
  - Gestione di volumi fisici e logici;
  - Gestione dei segnali;
  - Editor testuale vi;
  - Gestione multi-user;
  - Funzionamento del modulo hmc.
- **Programmazione**
  - Progettazione e realizzazione di applicazioni in linguaggio C (*v. Appendice*);
  - Funzioni di debug;
  - Analisi e struttura dei compilatori gcc e xlc;
  - Duplicazione di processi mediante *fork().*
- **Reti**
  - Introduzione alle reti;
  - Protocolli di rete UDP e TCP;
  - DNS e indirizzamento;
  - Hidden networks e VLAN;
  - Ethernet, IGRP e routing.
- **Gestione dei socket in AIX e nei Sistemi UNIX-like**
  - Gestione client/server nei protocolli UDP e TCP.
- **Nozioni di programmazione vettoriale e parallela**

Ambiente di lavoro

L'acceleratore di particelle DAΦNE è ospitato presso i Laboratori Nazionali di Frascati dell'Istituto Nazionale di Fisica Nucleare.
Su di esso è montato il detector dell'esperimento KLOE, che rileva dati relativi alle particelle in circolo.
I files grezzi (RAW) prodotti dal detector hanno dimensione di 2GB, e contengono dati in formato floating-point di dimensione fissa pari a 4 bytes.



S=Segno
E=Esponente
M=Mantissa

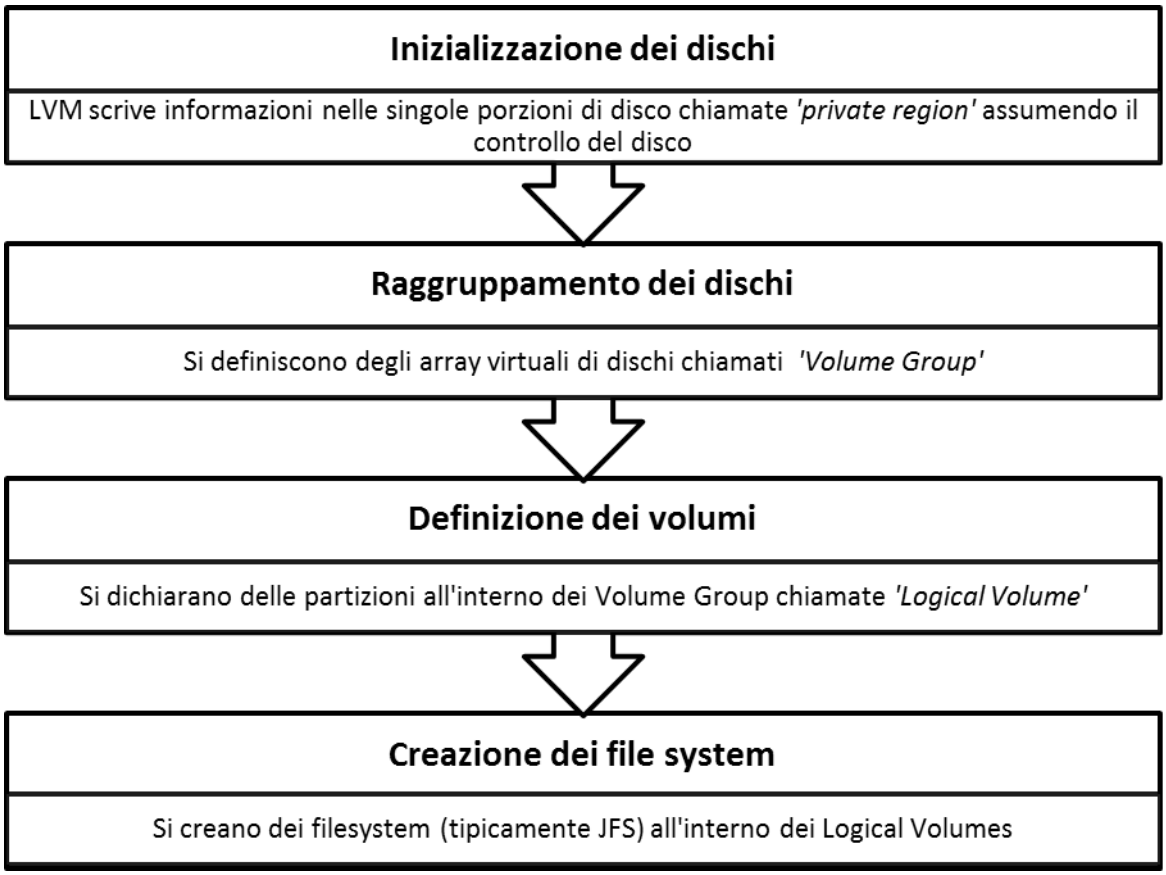
L'obiettivo da raggiungere è quello di analizzare e comprimere, se possibile, i dati in modo da ridurre lo spazio occupato e contenere i costi.
Per raggiungere tale scopo è stato predisposto il codice di cui in appendice B.

Il Sistema Operativo AIX

AIX è un sistema operativo sviluppato da IBM introdotto per la prima volta (*AIX Version 1*) nel 1986 nella IBM 6150 RT Workstation.
Il Sistema operativo AIX è basato sul file-system JFS, oltre ad utilizzare CDFS per il montaggio di dispositivi ottici e NFS per gestire pacchetti e ambienti di lavoro.

Il sistema utilizza un gestore logico dei volumi (LVM) che, essendo più flessibile rispetto al normale partizionamento dei dischi, permette tramite l'estensione JFS di ridimensionare i file system senza dover riavviare la macchina e quindi senza dover interrompere i servizi offerti da un server di dati o un database.

LVM segue questo schema per definire i file system:



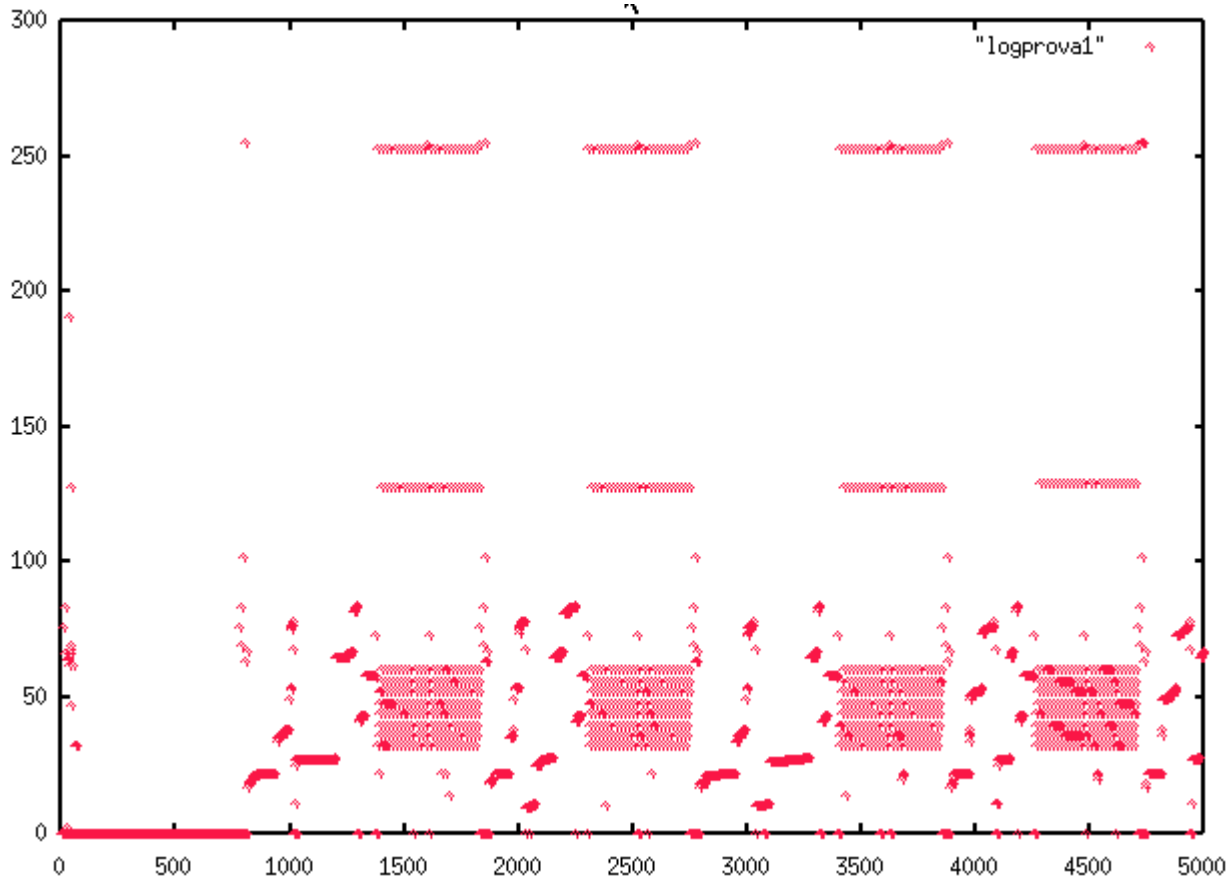
Solitamente questo sistema è utilizzato per gestire varie tipologie di RAID tra dischi.

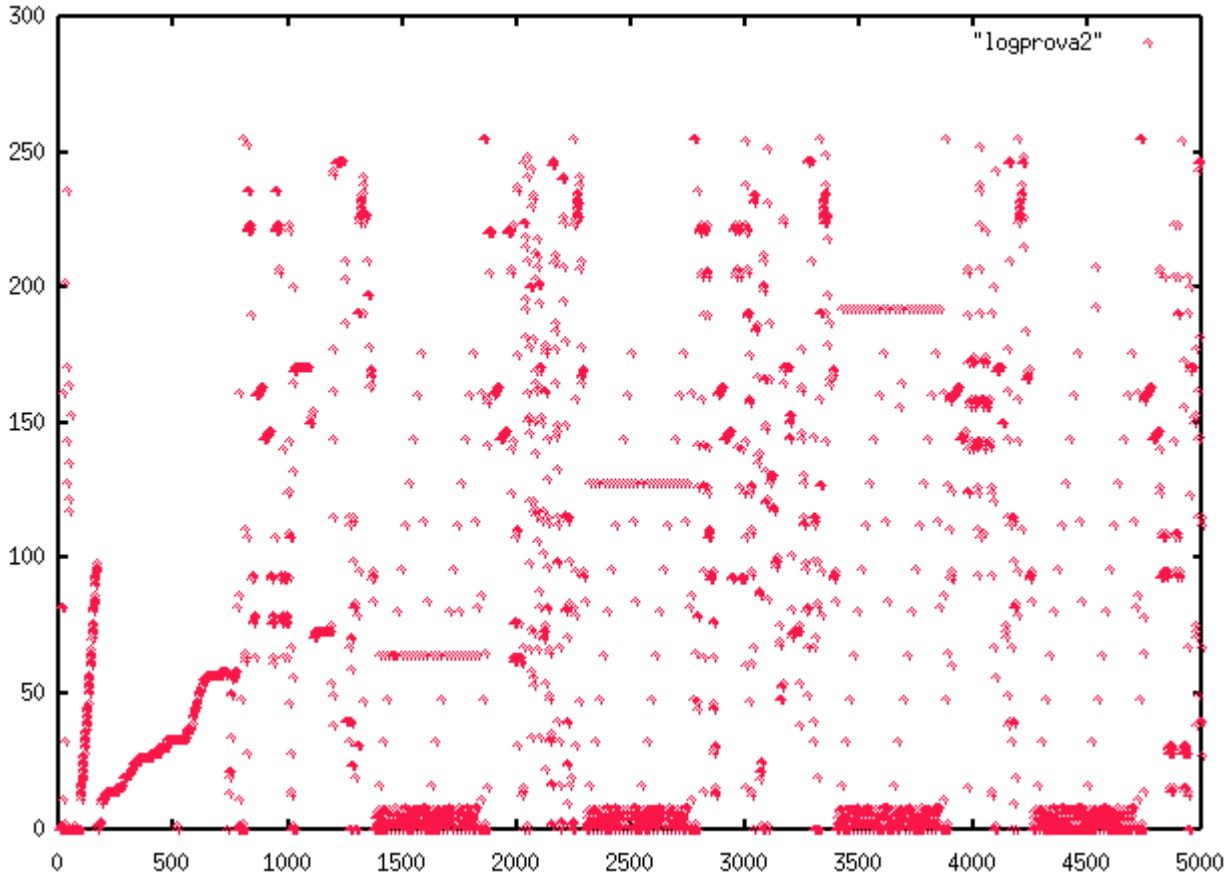
Struttura del centro di calcolo

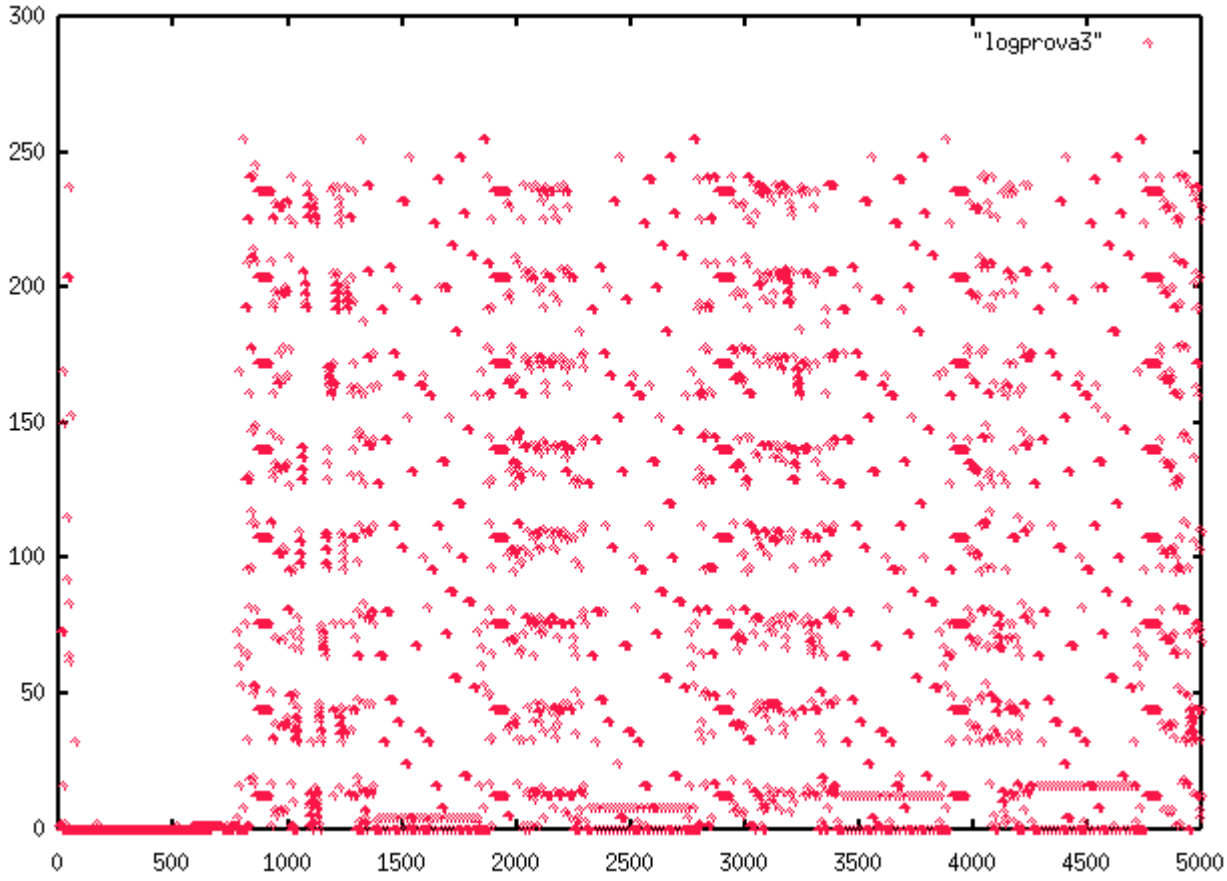
Nel centro di calcolo sono presenti dispositivi di elaborazione ed instradamento necessari a ricevere, analizzare ed immagazzinare i dati provenienti dal rilevatore KLOE.
Una parte del costo del centro di calcolo deriva dall'acquisto di supporti di memorizzazione dati, quindi lavorando sulla compressione di questi ultimi si possono abbattere notevolmente i costi (fino al 60%)

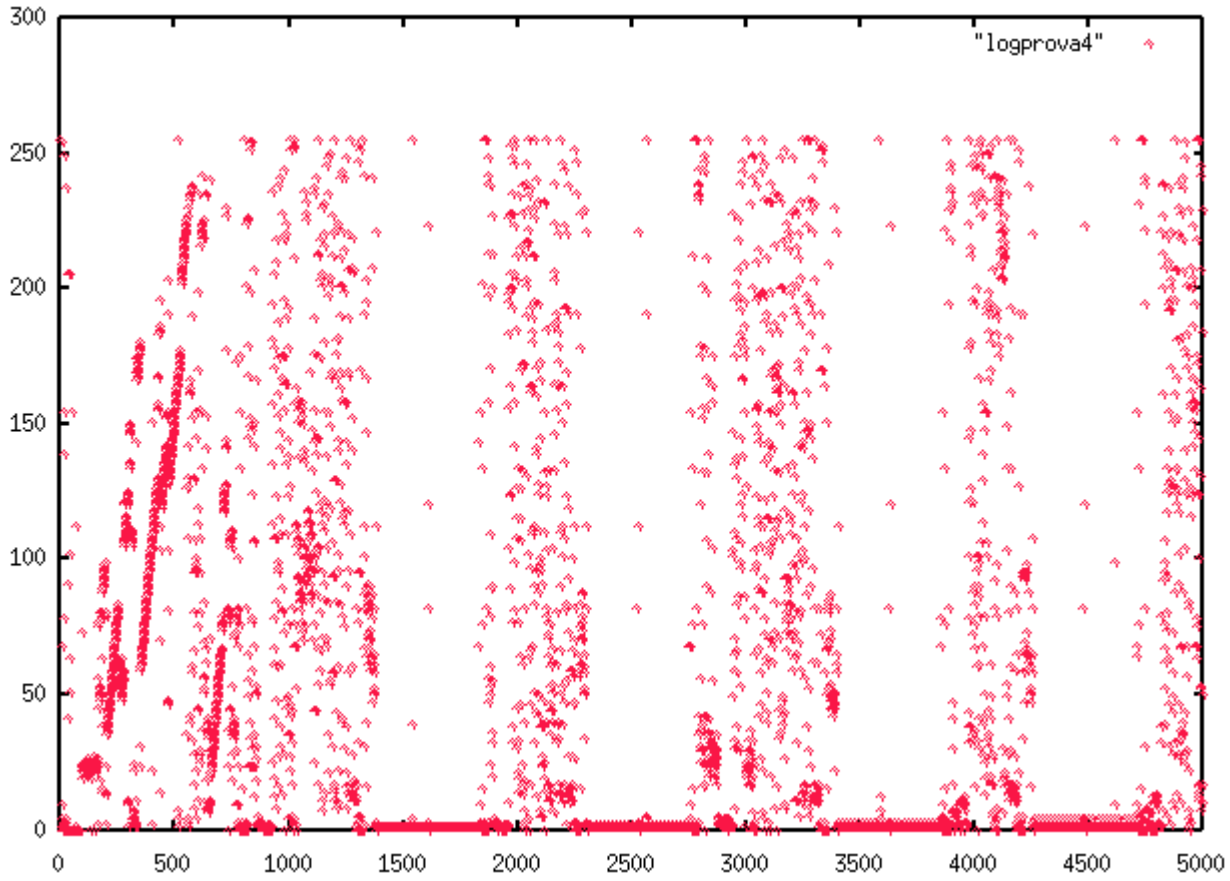
Analisi dei dati

Analizzando i dati RAW utilizzando anche applicazioni grafiche abbiamo scoperto che suddividendoli in quattro parti e comprimendoli separatamente, si ottiene una percentuale di compressione maggiore rispetto alla compressione del file originale. Questo perché come si può evincere dal grafico c'è una costante ripetizione dei dati.




"logprova2"
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000


"logprova3"
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000



Sul file raw sono memorizzati svariati dati, ognuno composto da quattro byte. Suddividendo la sequenzialità dei byte ciascuno con il corrispettivo file e poi comprimendo ogni file tramite il comando “compress” siamo riusciti a comprimere questi quattro file per un totale del 60% (rispetto al 37% originale) che è un risparmio notevole per un centro di calcolo.
Per velocizzare l'operazione abbiamo eseguito la compressione dei quattro file contemporaneamente tramite dei fork.

Comunicazione tra apparati di rete

La gestione del laboratorio di calcolo comprende anche una rete, gestita da macchine CISCO per l'instradamento dei dati. Alla base di tutto questo c'è il protocollo TCP/IP.
Il modello di architettura trattato e quello client/server.

Gestione dei socket

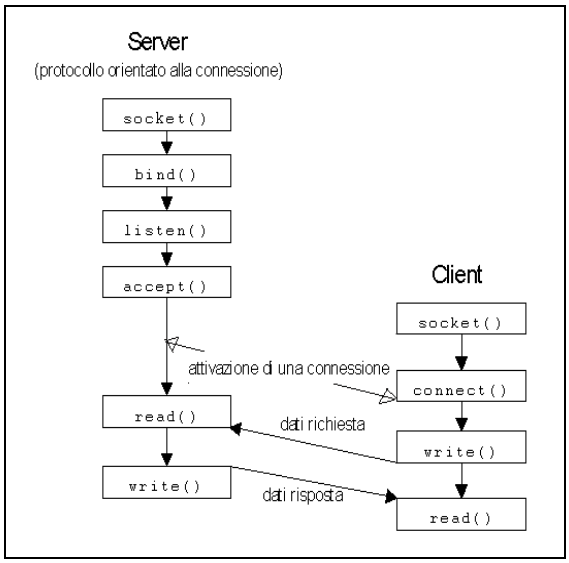
I processi che intendono stabilire una connessione, devono utilizzare sockets della stessa famiglia di protocolli e lo stesso stile di comunicazione (con/senza connessione). Ad esempio per la suite di protocolli di Internet sono resi disponibili i seguenti stili di comunicazione:

- Stream socket(SOCK_STREAM, TCP);
- Datagram socket(SOCK_DGRAM, UDP);
- Raw socket(SOCK_RAW, ICMP).

Il primo consente di stabilire un collegamento affidabile orientato alla connessione con controllo di flusso. Il tipo di comunicazione puo' essere usato quando l'integrità dell'informazione non e' fondamentale, come nei messaggi a diffusione (broadcast). Il terzo tipo di socket invece, permette collegamenti inaffidabili a basso livello.
Uno dei parametri che si deve specificare quando si crea un socket è l'indirizzo, poiché protocolli diversi utilizzano formati di indirizzo diversi.
Una volta creato il socket si deve fornirgli l' indirizzo-locale , in modo che sia individuabile univocamente dagli altri processi.
Nel funzionamento dei sockets di tipo SOCK_STREAM, essi sono in grado di accodare le richieste di connessione.
Una volta creato, il socket del server e' posto in uno stato di attesa fino a che non arrivi una richiesta, ciò è ottenuto tramite la chiamata di sistema accept().
A questo punto viene creato un socket per il cliente che manda una richiesta di connessione al socket del server tramite la chiamata di sistema connect().

Dopo che si e' stabilita la connessione può iniziare lo scambio di informazioni che si realizza normalmente tramite l'utilizzo delle funzioni read() e write(), una volta terminato la connessione viene interrotta tramite la chiamata close() nella comunicazione tra processi si possono accettare nuove connessioni mentre si sta elaborando una precedente, tramite la chiamata di sistema fork().I

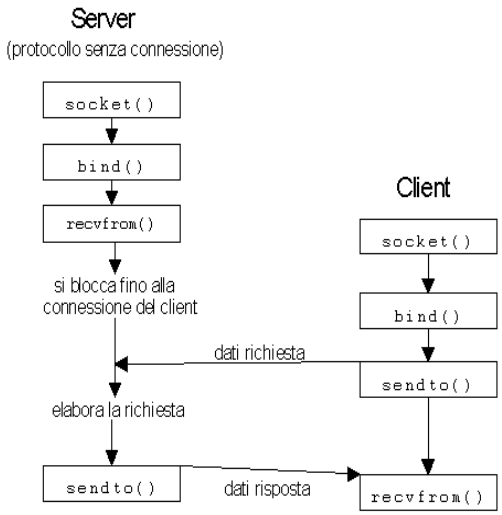
nomi possono essere associati ai socket dalla chiamata bind().
Il server attende l'arrivo di una richiesta di connessione attraverso la chiamata accept(). Se la richiesta e' già arrivata, sarà presa dalla code del socket, altrimenti si bloccherà. Quando una richiesta e' disponibile, viene creato un nuovo socket.
Per effettuare una connessione con un socket remoto, un processo utente puo' invocare la chiamata connect(), specificando il nome socket locale e l'indirizzo di quello remoto. Per inviare i messaggi vengono impiegate le chiamate send() e recv() nel caso di una connessione orientata e sendto() recvfrom() nel caso di una connessione non orientata.
La comunicazione orientata alla connessione e' per definizione asimmetrica; il client agisce per richiedere una connessione, mentre il server crea un socket e rimane in attesa di aventuali chiamate.
L'ordine delle chiamate di sistema di socket per un protocollo orientato alla connessione e mostrato del seguente schema:



Per un client/server che impiega un protocollo senza connessione, le chiamate di sistema sono differenti. Il client non stabilisce una connessione con il server, ma si limita ad inviare al server un datagramma mediante la chiamata sendto(), che richiede l'indirizzo di destinazione. Similmente, il server non ha l'obbligo di accettare una connesione da un client, ma si limita ad inviare una chiamata di

sistema recvfrom(), la quale rimane bloccata fino all'arrivo da dati da qualche client.

Le chiamate di sistema di socket per un protocollo senza connessione è mostrato del seguente schema:



Appendice A: Frequenza caratteri

Programma che stampa in output la frequenza dei caratteri presenti in un file.

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * ncaratteri.c: programma che restituisce la frequenza dei
 * caratteri all'interno di un file
 */
#include <stdio.h>
int main()
{
     printf("\n\n\tISTITUTO NAZIONALE di FISICA
NUCLEARE\n\tLaboratori Nazionali di
Frascati\n\t**********************************\n\tStages estivi
e residenziali 2014\n\t16 giugno - 4 luglio 2014\n\n\tGruppo di
lavoro 'L' - Informatica\n\tPONTIS Alessandro\n\tSILVI
Andrea\n\tSIRBU Gheorghe\n\tSTANZIONE Massimo\n\tTITI
Marco\n\t**********************************\n\tProgramma che,
dato un file in ingresso,\n\tne ricava il numero di complessivo
di\n\tcaratteri in esso contenuti.\n\n\t18 giugno 2014\n   \n
     \n");
     int cont=0;
     char app;
     FILE *fo;
     fo=fopen("testo","r");
     printf("\nContenuto del file:\n");
     while((app=getc(fo))!=EOF)
     {
          if(app!='\n')
          cont++;
          printf("%c", app);
     }
     fclose(fo);
     printf("\n\nNel file sono stati rilevati %d
caratteri.\n\n",cont);
     return 0;
}
```

Appendice B: Analisi e compressione

Analisi e compressione dei dati grezzi prodotti dall'esperimento KLOE

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * analisi_comp.c: analisi e compressione dei dati grezzi
 * prodotti dall'esperimento KLOE
 */
#include <stdio.h>
int main()
{
     int pid0, pid1, pid2, pid3, pid4;
     pid0=fork();
     if(pid0==0)
     {
          FILE *fread;
          FILE *f1, *f2, *f3, *f4;
          fread=fopen("provaanalisi", "r");
          f1=fopen("DEF_log1", "w");
          f2=fopen("DEF_log2", "w");
          f3=fopen("DEF_log3", "w");
          f4=fopen("DEF_log4", "w");
          /*limite per campione 50mb*/
          long int lim=200000000;
          char var;
          int i=0;
          while(!feof(fread))
          {
               var=getc(fread);
               putc(var, f1);
               var=getc(fread);
               putc(var, f2);
               var=getc(fread);
               putc(var, f3);
               var=getc(fread);
               putc(var, f4);
               var=getc(fread);
               i+=4;
               if(i%10==0)
               {
                    printf("Sono stati processati %d bytes.\n", i);
```

```
                    system("clear");
               }
               printf("Sono stati processati %d bytes.\n", i);
          }
          fclose(fread);
          fclose(f1);
          fclose(f2);
          fclose(f3);
          fclose(f4);
          system("rm DEF_log1.Z DEF_log2.Z DEF_log3.Z DEF_log4.Z");
          pid1=fork();
          if(pid1==0)
               system("compress -v DEF_log1");
          else
          {
               pid2=fork();
               if(pid2==0)
                    system("compress -v DEF_log2");

               else
               {
                    pid3=fork();
                    if(pid3==0)
                         system("compress -v DEF_log3");
                    else
                    {
                         pid4=fork();
                         if(pid4==0)
                              system("compress -v DEF_log4");
                         else
                         {
                              waitpid(pid1,NULL,0);
                              waitpid(pid2,NULL,0);
                              waitpid(pid3,NULL,0);
                              waitpid(pid4,NULL,0);
                         }
                    }
               }

          }

     }
     else
          waitpid(pid0, NULL, 0);
     return 0;
}
```

Appendice C: Conversione ASCII – binario

Programma che converte un file ASCII in binario, poi riconvertito in ASCII tramite una fork().

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * asciibin.c: programma che converte un file ASCII in binario,
 * poi riconvertito in ASCII tramite una fork()
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
     if(argc!=2)
     {
          printf("\nSpecificare argomento\nUsage: provafork
<nomefilein>\n");
          return 0;
     }
     int pid;
     FILE *fread;
     fread=fopen(argv[1], "r");
     pid=fork();

     if(pid==0)
     {
          printf("Processo figlio 1 creato\n");
          int var;
          printf("Creazione file binario\n");
          FILE *fwrite=fopen("out.bin", "wb");
          printf("File binario creato\n");
          while(!feof(fread))
          {

               putc(getc(fread),fwrite);

          }
          fclose(fwrite);
          fclose(fread);
```

```
          int pid2=fork();
          if(pid2==0)
          {
               printf("Processo figlio 2 creato\n");
               /*Apertura del file binario creato*/
               FILE *fread2=fopen("out.bin", "rb");
               /*Scrittura del file ASCII*/
               FILE *fwrite2=fopen("out.ascii", "w");
               while(!feof(fread2))
               {
                    putc(getc(fread2), fwrite);
               }
          fclose(fwrite2);
          fclose(fread2);
          }
          else
          {
               waitpid(pid,NULL,0);
               printf("Processo figlio 2 terminato\n");
          }
     }
     else
     {
          waitpid(pid, NULL, 0);
          printf("Processo figlio 1 terminato\n");
     }

     return 0;
}
```

Appendice D: Gestore di segnali

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * signal.c Gestore di segnali
 */
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
int clock=2;
void esegue(int sign)
{
     printf("Segnale SIGALARM ricevuto\n");
     alarm(clock);
}

void uscita(int sign)
{
     printf("Segnale SIGINT ricevuto. Chiusura in corso...\n");
     exit();
}

int main()
{
     printf("\n\n\t\tPer interrompere il programma inviare SIGINT
tramite il comando\n\t\t\t********** kill -2 %d **********\n\n",
getpid());
     for(;;)
     {
          alarm(clock);
          signal(SIGALRM, esegue);
          signal(SIGINT, uscita);
          pause();
     }
}
```

Appendice E: Server UDP

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * serverUDP.c: server UDP
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

void error(char *msg)
     {
     perror(msg);
     exit(0);
     }

int main(int argc, char *argv[])
     {
     int sock, length, clientlen, n, ex=1;
     struct sockaddr_in server;
     struct sockaddr_in client;
     char buf[1024];
     char obuf[1024];

     if(argc<2)
          {
          fprintf(stderr, "ERRORE - Nessuna porta specificata\n");
          exit(0);
          }
     sock=socket(AF_INET, SOCK_DGRAM, 0);
     if(sock<0)
          error("Apertura socket");
     length=sizeof(server);
     bzero(&server, length);
     server.sin_family=AF_INET;
     server.sin_addr.s_addr=INADDR_ANY;
```

```
     server.sin_port=htons(stoi(argv[1]));
     if(bind(sock,(struct sockaddr *)&server,length)<0)
          error("Binding");
     clientlen=sizeof(struct sockaddr_in);
     while(ex!=0)
          {
          bzero(buf, sizeof(buf));
          n=recvfrom(sock,buf,1024,0,(struct sockaddr *)&client,
(unsigned int *)&clientlen);
          if(n<0)
               error("recvfrom");
          write(1,"Datagram ricevuto: ", 21);
          write(1, buf, n);
          sprintf(obuf, "Ricevuto messaggio: %s", buf);
          n=sendto(sock, obuf, strlen(obuf), 0, (struct sockaddr
*)&vclient, clientlen);
          if(n<0)
               error("sendto");
          ex=strcmp(buf, "EXIT", 4);
          }
     close(sock);
     return 0;
}
```

Appendice F: Client UDP

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * clientUDP.c: client UDP
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

void error(char *msg)
     {
     perror(msg);
     exit(0);
     }

int main(int argc, char *argv[])
     {
     int sock, length, n, ex=1;
     struct sockaddr_in server, client;
     struct hostent *hp;
     char buffer[256];
          if(argc!=3)
          {
          fprintf("Usage: server port");
          exit(1);
          }
     sock=socket(AF_INET, SOCK_DGRAM, 0);
     if(sock<0)
          error("Socket");
     server.sin_family=AF_INET;
     hp=gethostname(argv[1]);
     if(hp==0)
          error("Host sconosiuto");
     bcopy((char *)hp->h_addr,(char *)&server.sin_addr, hp-
```

```
>h_length);
     server.sin_port=htons(atoi(argv[2]));
     length=sizeof(struct sockaddr_in);
     while(ex==0)
          {

          printf("Inserire il messaggio: ");
          bzero(buffer, 256);
          fgets(buffer, 256, stdin);
          n=sendto(sock, buffer, strlen(buffer), 0, (struct
sockaddr *)&server, length);
          if(n<0)
               error("sendto");
          n=recvform(sock, buffer, 256, 0, (struct sockaddr
*)&client, (unsigned int *)&length);
          if(n<0)
               error("recvfrom");
          write(1, "Acknowledge ricevuto:\n"m 13);
          write(1, buffer, n);
          write(1, "\n",2);
          if(!strncmp(buffer,"EXIT", 4);
               exit(0);
          }
     return 0;
}
```

Appendice G: Server TCP

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * serverTCP.c: Server TCP
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

int sock, nsock, pid;

void uscita(int sign)
{
     printf("chiusura primo socket\n");
     close(nsock);
     printf("chiusura secondo socket\n");
     close(sock);
     printf("uscita\n");
     exit(0);
}

void error(char *msg)
{
     perror(msg);
     exit(0);
}

int main(int argc, char *argv[])
{
     int length, clientlen, n, m, ex=1;
     signal(SIGINT, uscita);
     struct sockaddr_in nome;
     socklen_t lung;
     lung=sizeof(nome);
```

```
     struct sockaddr_in server;
     struct sockaddr_in client;
     char buf[1024];
     char obuf[1024];

     if(argc<2)
     {
          fprintf(stderr, "ERRORE - Nessuna porta specificata\n");
          exit(0);
     }
     sock=socket(AF_INET, SOCK_STREAM, 0);
     if(sock<0)
          error("Apertura socket");
     length=sizeof(server);
     bzero(&server,length);
     server.sin_family=AF_INET;
     server.sin_addr.s_addr=INADDR_ANY;
     server.sin_port=htons(atoi(argv[1]));
     if(bind(sock,(struct sockaddr *)&server, length)<0)
          error("Binding");
     if(listen(sock, 5)<0)
          error("listen");
     clientlen=sizeof(struct sockaddr_in);
     while(ex!=0)
     {
          nsock=accept(sock,(struct sockaddr *)&client,(unsigned
int *)&clientlen);
          if(nsock<0)
               error("accept");
          bzero(buf, sizeof(buf));
          bzero(obuf, sizeof(obuf));
          pid=fork();
          if(pid==0)
          {
               n=0;
               while(n>=0)
               {
                    n=recv(nsock, buf, 1024, 0);
                    printf("\n n=%d ", n); fflush(stdout);
                    write(1, "Ricevuto un datagram: ", 21);
                    write(1, buf, n);
                    getpeername(nsock,(struct sockaddr *)&nome,
&lung);
                    printf("Indirizzo IP: %s\n",
inet_ntoa(nome.sin_addr));
                    if(n<=0) break;
                    sprintf(obuf, "Ricevuto messaggio: %s", buf);
                    m=send(nsock, obuf, strlen(obuf),0);
                    if(m<0)
                         error("send");
```

```
                    bzero(buf, sizeof(buf));
                    bzero(obuf, sizeof(obuf));
               }
               close(nsock);
               return 0;
          }
          else
          {
               close(nsock);
          }
     }
     close(sock);
     return 0;
}
```

Appendice H: Client TCP

```
/*
 * ISTITUTO NAZIONALE di FISICA NUCLEARE
 * Laboratori Nazionali di Frascati
 * Stages estivi e residenziali 2014
 * Gruppo di lavoro "L" - Informatica
 * PONTIS Alessandro, SILVI Andrea, SIRBU Gheorghe,
 * STANZIONE Massimo, TITI Marco
 * ------------------------------------------------------------
 * clientTCP.c: client TCP
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

void error(char *);

int main(int argc, char *argv[])
{
     int sock, length, n , m;
     struct sockaddr_in server;
     struct hostent *hp;
     char buffer[256];
     if(argc!=3)
     {
          printf("Usage: server port\n");
          exit(1);
     }
     sock=socket(AF_INET, SOCK_STREAM, 0);
     if(sock<0)
          error("socket");
     server.sin_family=AF_INET;
     hp=gethostbyname(argv[1]);
     if(hp==0)
          error("Host sconosciuto");
     bcopy((char *)hp->h_addr,(char *)&server.sin_addr,hp-
>h_length);
     server.sin_port=htons(atoi(argv[2]));
     length=sizeof(struct sockaddr_in);
     if(connect(sock, (struct sockaddr *)&server, length)<0)
          error("Connect");
```

```
     m=1;
     while(m!=0)
     {
          printf("\nInserire il messaggio: ");
          bzero(buffer, 256);
          fgets(buffer, 255, stdin);
          m=strncmp(buffer,"quit", 4);
          if(m==0) break;
          n=send(sock, buffer, strlen(buffer), 0);
          if(n<0)
               error("Sendto");
          n=recv(sock, buffer, 256, 0);
          if(n<0)
               error("recvfrom");
          write(1, "Ricevuto acknowledge: \n", 13);
          write(1, buffer, n);
          write(1, "\n", 2);
     }
     close(sock);
     return 0;
}

void error(char *msg)
{
     perror(msg);
     exit(0);
}
```