

POLITECNICO DI TORINO

III Facoltà di Ingegneria  
Corso di Laurea Specialistica in Ingegneria Informatica

Tesina del corso di Progetto di sistemi operativi

## Log file system di tipo ext2/ext3



LUCA ARDITO 151376  
PIETRO NICCOLI 136245  
STEFANO MANCINI 152769

Giugno 2008

## Indice

1	Descrizione del lavoro	2
2	Funzione main	3
3	Funzione controlla input	5
4	Funzione naviga cartella	6
5	Funzione lista cartella	8
6	Funzione permessi	9
7	Funzione indirezione	11
8	Funzione statistiche	14

# 1 Descrizione del lavoro

Il lavoro consiste in un programma scritto in codice `c` che permette di leggere in modalità `raw` un disco formattato in `ext2/ext3` per reperire delle informazioni relative ai file contenuti all'interno di un path specificato da linea di comando.

L'algoritmo parte dal percorso passato come parametro al programma e inizia a scandire tutti i file le cartelle e le sotto cartelle; per ogni file restituisce delle informazioni che variano a seconda della sua tipologia.

I tipi di file riconosciuti sono i seguenti:

- Directory: stampa il suo contenuto
- File regolare: stampa la sua dimensione, il numero dei frammenti, i permessi associati, il proprietario e il gruppo a cui appartiene
- Device di blocco o device di carattere: major number, minor number, permessi associati, il proprietario e il gruppo a cui appartiene
- Link simbolico: stampa la destinazione del link, i permessi associati, il proprietario e il gruppo a cui appartiene

Non sono state utilizzate chiamate di sistema che automaticamente restituiscono queste informazioni ma sono state effettuati accessi al disco in modalità `raw` per mezzo della chiamata di sistema `lseek()` che permette di posizionarsi in un punto preciso del disco. Questa posizione è ottenuta mediante dei calcoli eseguiti in base ai valori ottenuti leggendo il superblocco. Questo permette al programma di funzionare correttamente indipendentemente dalla dimensione dei blocchi del file system analizzato.

Una volta letto il superblocco si passa al group descriptor e successivamente alla tabella degli inode. Una volta raggiunta la tabella degli inode si può accedere direttamente ai blocchi contenenti i dati veri e propri.

Successivamente verranno spiegate dettagliatamente tutte le funzioni che compongono il programma: verrà fornita prima una descrizione e successivamente sarà disponibile il codice `c`.

L'algoritmo scrive inoltre all'interno di un file tutte le informazioni raccolte in modo da permettere all'utente di analizzare i dati.

## 2 Funzione main

All'avvio il programma controlla la correttezza dell'input mediante la funzione controlla input(). Successivamente esplora l'albero delle directory a partire dalla directory passata come parametro e lo memorizza in una lista contenente tutte le cartelle presenti. Per fare questo viene chiamata la funzione opendir() sulla cartella corrente e ne viene scorso il suo contenuto mediante la funzione naviga cartella() e ne viene stampato il contenuto con lista cartella(); ogni volta che la funzione naviga cartella() viene eseguita, questa riempie la lista principale che viene scorsa fino ad esaurire l'albero.

```
int main(int argc, char *argv[]){
    DIR *directory,*dir2;
    output=fopen("output.txt","w+");

    controlla_input(argc,argv[1]);

    //setto come prima cartella da visitare quella
    //specificata dall'utente da shell
    testa=(mydir*)calloc(1,sizeof(mydir));
    strcpy(testa->path,start);
    testa->next=NULL;
    coda=testa;

    while(testa!=NULL){
        directory=opendir(testa->path);
        dir2 = opendir(testa->path);
        if(directory!=NULL){
            //aperta la prima cartella della
            //lista, la visito
            fprintf(output,"\nInformazioni relative
alla directory %s\n",testa->path);
            lista_cartella(dir2);
            naviga_cartella(directory);

        }
        else
            fprintf(stderr,"Impossibile aprire %s\n"
,testa->path);
        //faccio avanzare il puntatore alla testa della
        //lista e libero la memoria usata

        appoggio=testa;
        testa=testa->next;
    }
}
```

```
        free(appoggio);
        closedir(directory);
    }
    fclose(output);
    return 0;
}
```

### 3 Funzione controlla input

La funzione permette di verificare se è stato inserito un path come parametro e di differenziare un path assoluto da un path relativo alla cartella in cui il programma è eseguito.

Nel primo caso il path che sarà preso in considerazione come root path sarà semplicemente il parametro mentre, nel secondo caso, verrà concatenato il percorso corrente restituito dalla funzione `getcwd()` al parametro inserito dall'utente.

A seguito è presente il codice della funzione.

```
void controlla_input(int argc, char *arg){
    if(argc!=2){
        fprintf(stderr, "Specificare un path valido\n");
        _exit(1);
    }
    if(arg[strlen(arg)-1]=='/')
        arg[strlen(arg)-1]='\0';
    strcpy(start, arg);
    if(start[0]!='/'){
        //specificato path relativo alla dir del programma
        getcwd(start, MAXPATHLEN);
        strcat(start, "/");
        strcat(start, arg);
    }
}
```

## 4 Funzione naviga cartella

La funzione naviga cartella scandisce l'albero delle directory a partire dal root path definito nella funzione controlla input e per ogni cartella ne stampa il contenuto.

Ogni volta che viene incontrata una cartella viene scandito il proprio contenuto e, ogni volta che si incontra un'altra cartella, quest'ultima viene inserita in una coda che verrà gestita in seguito; altrimenti viene eseguita la funzione stat() che permette di ottenere il major e il minor number del device che contiene il file preso in esame. A partire da questi parametri viene creato un device di blocco temporaneo che permette alla funzione che restituisce le statistiche di aprire il device associato senza conoscerlo direttamente. Il codice seguente relativo alla funzione naviga cartella esegue ciò che è stato precedentemente descritto.

```
void naviga_cartella(DIR *directory){

    struct dirent *dent;

    char curr[MAXPATHLEN];
    char name[128];
    int status;

    strcpy(name, "");
    //prendo il primo oggetto della directory corrente
    dent=readdir(directory);

    while(dent!=NULL){
        //ciclo nella cartella aperta, quindi setto come parametro il path
        //della cartella aperta
        strcpy(current, "\0");
        strcpy(current, testa->path);
        if(dent->d_type==4){
            //trovata un'altra cartella, controllo che non siano . . . o
            //DS_STORE
            if(strcmp(".", dent->d_name)!=0 && strcmp("..", dent->d_name)!=0
                && strcmp(".DS_Store", dent->d_name)!=0){
                //quindi posso aggiungerla alla lista delle cartelle
                //da aprire
                strcpy(curr, "\0");
                strcpy(curr, current);
                appoggio=(mydir*)calloc(1, sizeof(mydir));
                strcpy(appoggio->path, strcat(strcat(curr, "/"),
                    dent->d_name));
                appoggio->next=NULL;
            }
        }
    }
}
```

```

        coda->next=appoggio;
        coda=appoggio;
        //fprintf(output,"trovata directory %s,\naggiunta in
        //coda alla lista delle cartelle\n",coda->path);
    }
}
else{
    //trovato un file generico
    fprintf(output,"Informazioni per il file %s\n",dent->d_name);
    strcat(name,testa->path);
    strcat(name,"/");
    strcat(name,dent->d_name);
    fprintf(output,"%s\n",name);
    stat(name,&buf);
    status = mknod(node,S_IFBLK|S_IRWXU|S_IRWXG|S_IRWXO,buf.st_dev);
    //qui chiamo la funzione che fa le statistiche
    statistiche(node,dent->d_ino);
    unlink(node);
    strcpy(name,"");
}
//prendo il successivo oggetto della directory
dent=readdir(directory);
}
}

```

## 5 Funzione lista cartella

La funzione lista cartella permette di stampare l'elenco dei file presenti all'interno di una cartella. Il principio di funzionamento è simile alla funzione naviga cartella con la sola differenza che si limita a stampare ciò che è contenuto all'interno della cartella senza fare alcuna distinzione tra i tipo di file in essa contenuta

```
void lista_cartella(DIR *directory){
    struct dirent *dent;
    dent=readdir(directory);
    fprintf(output,"Elenco file contenuti nella directory:\n");
    while(dent!=NULL){
        if(strcmp(".",dent->d_name)!=0 && strcmp("..",dent->d_name)!=0
        && strcmp(".DS_Store",dent->d_name)!=0)
            fprintf(output,"%s\n",dent->d_name);
        dent=readdir(directory);
    }
    fprintf(output,"-----\n");
}
```

## 6 Funzione permessi

Questa funzione permette di tradurre l'intero corrispondente al tipo di permessi appartenenti ad un file in una stringa. Questo valore intero viene reperito dal disco dalla funzione chiamante e per una visualizzazione più semplice da parte dell'utente viene convertito nella classica notazione usata nei sistemi Unix.

Ciò è possibile per mezzo di un and binario con il singolo valore che rappresenta il permesso interessato ed è possibile notarlo dal codice della funzione qui riportato.

```
char * permessi(int mode){
    int risultato = mode & 0x1FF;
    char *stringa= calloc(1,sizeof(char*));
    if (risultato & 0x100) {
        strcat(stringa,"r");
    } else {
        strcat(stringa,"-");
    }
    if (risultato & 0x80) {
        strcat(stringa,"w");
    } else {
        strcat(stringa,"-");
    }
    if (risultato & 0x40) {
        strcat(stringa,"x");
    } else {
        strcat(stringa,"-");
    }
    if (risultato & 0x20) {
        strcat(stringa,"r");
    } else {
        strcat(stringa,"-");
    }
    if (risultato & 0x10) {
        strcat(stringa,"w");
    } else {
        strcat(stringa,"-");
    }
    if (risultato & 0x08) {
        strcat(stringa,"x");
    } else {
        strcat(stringa,"-");
    }
    if (risultato & 0x04) {
```

```
    strcat(stringa,"r");
    } else {
    strcat(stringa,"-");
    }
    if (risultato & 0x02) {
    strcat(stringa,"w");
    } else {
    strcat(stringa,"-");
    }
    if (risultato & 0x01) {
    strcat(stringa,"x");
    } else {
    strcat(stringa,"-");
    }
    return stringa;
}
```

## 7 Funzione indirezione

Questa funzione ricorsiva permette di andare a leggere il valore del blocco contenente il dato. Viene chiamata quando si legge l'inode relativo ad un file regolare e permette di leggere tutti i blocchi nei tre livelli di indirezione. E' strutturata in modo da ricevere l'offset da passare alla lseek in modo da posizionarsi nel punto giusto del disco in cui sono contenute le informazioni necessarie. Il parametro livello indica l'indirezione:

- 0: legge i primi 12 elementi del blocco
- 1: legge l'array di 256 elementi contenenti i blocchi
- 2: legge l'array di 256 elementi ognuno dei quali contiene un array di 256 elementi contenenti i blocchi
- 3: legge l'array di 256 elementi ognuno dei quali contiene un array di 256 elementi ognuno dei quali contiene un array di 256 elementi contenenti i blocchi.

Ad ogni livello è presente un controllo che fa fermare ogni ciclo se l'elemento visitato ha valore 0. Ciò significa che da quel punto in avanti non sono più presenti dati relativi a quel file. Questa funzione inoltre, verifica se sono presenti delle discontinuità nella memorizzazione dei dati relativi al file preso in esame.

A seguire è riportato il codice della funzione.

```
void indirezione(int livello, int salto){
    int tmp2[256],m=0,prev,i,j;
    unsigned char test[sb.s_log_block_size];
    if(!livello){
        for(i=0;i<15;i++) {
            if(i<12){
                if(inode.i_block[i] != 0){
                    if(inode.i_block[i+1] != 0){
                        if(inode.i_block[i]+1!=inode.i_block[i+1])
                            discont++;
                    }
                }
            }
            else
                break;
        }
    }
    else{
        if(i==12)
            indirezione(1,inode.i_block[i]);
        else if(i==13)
```

```

        indirezione(2,inode.i_block[i]);
    else if(i==14)
        indirezione(3,inode.i_block[i]);
    }
}
}
else if(livello==1){
    lseek(fd,salto*sb.s_log_block_size,0);
    read(fd,&test,sizeof(test));
    prev=test[0] + (test[1] << 8) + (test[2] << 16) + (test[3] <<24);
    m=4;
    for(j=1;j<256;j++){
        tmp2[j]= test[m] + (test[m+1] << 8) + (test[m+2] << 16) + (test[m+3] <<24);
        if(tmp2[j]==0)
            break;
        else{
            if(prev+1!=tmp2[j]){
                discont++;
            }
            prev=tmp2[j];
            m+=4;
        }
    }
}
else if(livello==2){
    m=0;
    lseek(fd,salto*sb.s_log_block_size,0);
    read(fd,&test,sizeof(test));
    for(j=0;j<256;j++){
        tmp2[j]= test[m] + (test[m+1] << 8) + (test[m+2] << 16) + (test[m+3] <<24);
        if(tmp2[j]==0)
            break;
        indirezione(1,tmp2[j]);
        m+=4;
    }
}
else if(livello==3){
    m=0;
    lseek(fd,salto*sb.s_log_block_size,0);
    read(fd,&test,sizeof(test));
    for(j=0;j<256;j++){
        tmp2[j]= test[m] + (test[m+1] << 8) + (test[m+2] << 16) + (test[m+3] <<24);
        if(tmp2[j]==0)

```

```
        break;
        indirezione(2,tmp2[j]);
        m+=4;
    }
}
```

## 8 Funzione statistiche

Questa funzione rappresenta il cuore di tutto il programma. Riceve come parametro il device da aprire ed il numero dell'inode da visitare.

Legge il superblocco del disco preso in esame e memorizza in apposite variabili le informazioni sensibili per procedere successivamente con i calcoli.

Le informazioni che variano in base alle diverse caratteristiche del file system sono:

- dimensione del blocco
- dimensione dell'inode
- numero di inode per gruppo
- posizione dell'inode table

Una volta letto il superblocco si calcola la distribuzione degli inode per ogni group descriptor in modo da leggere quello corrispondente all'inode specificato e da lì raggiungere la tabella degli inode. I group descriptor sono organizzati in un unico array memorizzato dopo il superblocco. La posizione della inode table è espressa da un campo all'interno del group descriptor che indica il numero di blocco assoluto del disco nel quale ha inizio. Per fare questo, si effettua una divisione con resto sul numero dell'inode usando come divisore la variabile che indica il numero di inode per gruppo; il quoziente indica il gruppo, mentre il resto l'indice all'interno dell'inode table del gruppo in questione. Ottenuto il gruppo, si legge al suo interno la posizione della inode table e vi si accede con una lseek al byte corrispondente al numero di blocchi per la loro dimensione più il numero dell'inode per la dimensione dello stesso (fissa a 128 byte). Dopo aver analizzato l'inode e averne decifrato i campi che lo compongono, la loro dimensione e il loro ordine, onde evitare ripetute chiamate alla funzione read per leggere di volta in volta i byte che compongono ogni campo della struttura (riportata in seguito) e memorizzarli in tante variabili, abbiamo preferito sfruttare la struttura già esistente ext3 inode nella quale copiamo tutti i 128 byte che lo compongono. Successivamente la funzione stampa le informazioni relative al file corrispondente all'inode.

Struttura di un inode:

offset	size	description
0	2	i_mode
2	2	i_uid
4	4	i_size
8	4	i_atime
12	4	i_ctime
16	4	i_mtime
20	4	i_dtime
24	2	i_gid
26	2	i_links_count
28	4	i_blocks
32	4	i_flags
36	4	i_osd1
40	4	i_block (ripetuto 15 volte)
100	4	i_generation
104	4	i_file_acl
108	4	i_dir_acl
112	4	i_faddr
116	12	i_osd2

Struttura di un group descriptor:

offset	size	description
0	4	bg_block_bitmap
4	4	bg_inode_bitmap
<b>8</b>	<b>4</b>	<b>bg_inode_table</b>
12	2	bg_free_blocks_count
14	2	bg_free_inodes_count
16	2	bg_used_dirs_count
18	2	bg_pad
20	12	bg_reserved

Struttura del superblocco:

offset	size	description
0	4	s_inodes_count
4	4	s_blocks_count
8	4	s_r_blocks_count
12	4	s_free_blocks_count
16	4	s_free_inodes_count
20	4	s_first_data_block
<b>24</b>	<b>4</b>	<b>s_log_block_size</b>
28	4	s_log_frag_size
32	4	s_blocks_per_group
36	4	s_frags_per_group
<b>40</b>	<b>4</b>	<b>s_inodes_per_group</b>
44	4	s_mtime
48	4	s_wtime
52	2	s_mnt_count
54	2	s_max_mnt_count
56	2	s_magic
58	2	s_state
60	2	s_errors
62	2	s_minor_rev_level
64	4	s_lastcheck
68	4	s_checkinterval
72	4	s_creator_os
76	4	s_rev_level
80	2	s_def_resuid
82	2	s_def_resgid
84	4	s_first_ino
<b>88</b>	<b>2</b>	<b>s_inode_size</b>
90	2	s_block_group_nr
92	4	s_feature_compat
96	4	s_feature_incompat
100	4	s_feature_ro_compat
104	16	s_uuid
120	16	s_volume_name
136	64	s_last_mounted
200	4	s_algo_bitmap
204	1	s_prealloc_blocks
205	1	s_prealloc_dir_blocks
206	2	(byte di allineamento – non usati)
208	16	s_journal_uuid
224	4	s_journal_inum
228	4	s_journal_dev

232 4 s\_last\_orphan  
236 788 (byte non usati – zero padding)

Organizzazione del disco:

offset	# of blocks	description
0	1	boot record
-- block group --		
1	1	superblock
2	1	group descriptors
3	1	block bitmap
4	1	inode bitmap
5	variabile	inode table
X	variabile	data blocks

```

int statistiche(char *dev,int ino) {
    INODE=ino;
    struct passwd *pwd;
    struct group *grp;

    if((fd=open(dev,O_RDONLY,0))<0) {
        fprintf(stderr,"Impossibile aprire %s\n",dev);
        _exit(1);
    }

    lseek(fd,1024,0); //SALTO IL BOOT RECORD

    if(read(fd,&sb,sizeof(sb))!=sizeof(sb)) {
        fprintf(stderr,"Impossibile leggere il superblocco\n");
        _exit(1);
    }
    if(sb.s_log_block_size==0)
        sb.s_log_block_size=1024;

    else if(sb.s_log_block_size==1)
        sb.s_log_block_size=2048;

    else if(sb.s_log_block_size==2)
        sb.s_log_block_size=4096;

    uint32_t group = (INODE -1) / sb.s_inodes_per_group;
    uint32_t index = (INODE -1) % sb.s_inodes_per_group;

    //VADO AL GROUP DESCRIPTOR
    if(sb.s_log_block_size==1024)
        lseek(fd,(sb.s_log_block_size+BOOTSECTOR)+32*group,0);
    else
        lseek(fd,sb.s_log_block_size+32*group,0);

    if(read(fd,&gd,sizeof(gd))!=sizeof(gd)) {
        fprintf(stderr,"Impossibile leggere il group descriptor\n");
        _exit(1);
    }
}

```

```

fprintf(output, "Inode: %d\n", INODE);

//VADO ALL'INODE DESIDERATO

lseek(fd, gd.bg_inode_table*sb.s_log_block_size+index*sb.s_inode_size, 0);

char *tipo, *utente, *gruppo;
read(fd, &inode, sizeof(inode));
if((pwd = getpwuid(inode.i_uid)) != NULL)
    utente = pwd->pw_name;

if((grp = getgrgid(inode.i_gid)) != NULL)
    gruppo = grp->gr_name;

indirezione(0, 0);

switch (inode.i_mode & S_IFMT) {
    case S_IFDIR:           tipo="Directory";
                           break;
    case S_IFLNK:          tipo="Link simbolico";
                           break;
    case S_IFCHR:          tipo="Device di carattere";
                           break;
    case S_IFIFO:          tipo="Fifo";
                           break;
    case S_IFBLK:          tipo="Device di blocco";
                           break;
    case S_IFREG:          tipo="File regolare";
                           break;
    case S_IFSOCK:        tipo="Socket";
                           break;
}

fprintf(output, "Tipo file: %s\n", tipo);

if(!strcmp(tipo, "Link simbolico")){
    lseek(fd, gd.bg_inode_table*sb.s_log_block_size+
index*sb.s_inode_size+40, 0);
    unsigned char b[inode.i_size];
    read(fd, &b, inode.i_size);
    b[inode.i_size]='\0';
    fprintf(output, "link punta a -> %s\n", b);
}

```

```

else if ((!strcmp(tipo, "Device di carattere")
|| (!strcmp(tipo, "Device di blocco"))) ){
    fprintf(output, "Major number %d\n", (((int)buf.st_rdev & 0xFF00) >> 8));
    fprintf(output, "Minor number %d\n", (((int)buf.st_rdev & 0x00FF) >> 0));
}

else if (!strcmp(tipo, "File regolare")){
    fprintf(output, "N. discontinuita': %d\n", discontin);
    fprintf(output, "Link al file: %d\n", inode.i_links_count);
    fprintf(output, "Dimensione: %d\n", inode.i_size);
}

else if (!strcmp(tipo, "Socket")){
}

else if (!strcmp(tipo, "Fifo")){
}

else if (!strcmp(tipo, "Directory")){
}

discont=0;

fprintf(output, "Proprietario: %s\n", utente);
fprintf(output, "Gruppo: %s\n", gruppo);
fprintf(output, "%s\n", permessi(inode.i_mode));
fprintf(output, "-----\n");
close(fd);
return 0;
}

```