AA 17/18. Prima provetta di Sistemi Operativi .

1. Scrivere una riga di comando linux per listare e contare I file della directory corrente che hanno il numero di link maggiore di zero, ordinandoli per numero di link crescente (2)

```
ls -l|tr -s " "|sort -nk2|cut -f9,9 -d" "; ls -l|tr -s " "|sort -nk2|cut -f9,9 -d" "|wc -l
```

Nota: Il numero di link è sempre maggiore di zero (il minimo è uno)
Per semplicità, la lista e il numero dei file è fatto con due comandi in linea separati da ;

2. Scrivere una riga di comando linux per listare i 10 processi appartenenti a root che eseguono da più tempo nel sistema. (2)

```
ps -ef|sort -nk7r|while read line; do case $(echo $line|cut -f1,1 -d" ") in root) echo $line|cut -f12,12 -d"
"; esac; done|head -10
```

Nota: Per vedere quali processi appartengono a root non basta fare grep root perchè la parola root può trovarsi anche in altri punti, mentre devo vedere se si trova nel 1o campo

3. Scrivere uno script in bash per listare il nome dei proprietari di file in ordine di numero di file posseduti, scrivendo anche il numero di file posseduti. (3)

```
#!/bin/bash
#questo script si basa su due tabelle accoppiate, diciamo table1 e table2
#la prima contiene I nomi dei proprietari distinti, la seconda contiene le occorrenze
#
declare -i nutenti
declare -i i
declare -i table1
nutenti=0

ls -l |tr -s " "|cut -f3,3 -d" "> file #in questo file c'è l'elenco dei proprietari distinti

while read owner
do
   i=0
   while (( $i <= $nutenti))  # ${#utenti}))
   do
     if [[ $owner = ${table2[$i]} ]]
     then
        table1[$i]=${table1[$i]}+1
        break
     else
        i=$i+1
     fi
   done
   if (( $i>$nutenti ))
   then
          table2[$i]=$owner
          table1[$i]=1
          nutenti=$nutenti+1
   fi
done<file
i=0
while (( $i<= $nutenti ))
do
   echo "${table2[$i]} ${table1[$i]}"
   i=$i+1
done
```

4. Scrivere uno script in bash che crea una sottodirectory `"/onlycode"` nella directory corrente dove, per ogni file con estensione `".c"` della directory corrente, vengono salvati file con gli stessi nomi ma con l'aggiunta dell'intestazione `"mio_"` al nome del file. Il contenuto sarà lo stesso del file originale ma privo da ogni tipo di commento di tipo `//....` , dove naturalmente i puntini rappresentano commenti. Per esempio, il file `draw.c` diventerà `mio_draw.c` e conterrà lo stesso codice di `draw.c` ma senza commenti. (4)

```
#!/bin/bash
#da notare che \/ è formata da backslash \ + slash /
#
mkdir onlycode
for nome in *.c            #per tutti I file .c
do
   while read linea        #leggi ogni riga dei file
   do
         if [[ $linea!="\/\/"* ]]
         then
                  echo $linea>>./onlycode/"mio_$nome"
         fi
   done<$nome
done
```

5. Data la seguente bozza di programma C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* sentence(char* s)
{
...
}
void main()
{
  char stringa[500]="Most of the knowledge on falls might be biased. Fall
simulation are widely used in algorithm development. However the results of
fall detection are not convincing. Students were instructed to fall on the
back during Experiment 1. In Experiment 2, students were instructed not to
fall, when possible, when released from a backward lean. Data acquisition
was performend using a tri-axial acceleration sensor.";

   char* start=stringa;
   iny i=1;//numero di frasi
   char* p=NULL;
   while (*(p=sentence(start)) != '\0')
   {
      printf("%s\n",p);
      start+=strlen(p)+2;
     i++;
   }
}
```

scrivere la funzione `sentence(char* s). Questa funzione, ogni volta che viene chiamata,` estrae dalla stringa di ingresso la prossima frase, cioè tutti i caratteri fino al prossimo punto. Quando si arriva alla fine della stringa, ritorna il carattere terminatore di stringa. (5)

```
char* sentence(char* s)
{
     char* p1;
     char* p2;
     char *arr;
     short i;
     p1=s;
     p2=malloc(100);
     arr=p2;
     for(i=0; *p1!='.';i++) *p2++=*p1++;
     *p2='\0';
     return arr;
}
```

6. Supponendo di avere scritto il programma del punto 5., scrivere la funzione `char** carica(char* p)` che carica le frasi estratte in un array bidimensionale di stringhe.
La funzione `char** carica(char* p)` ovviamente sostituisce l'istruzione `printf("%s\n",p);` e restituisce il puntatore all'array. (3)

```
char **carica(char* p, char **q, int i)
{
   q[i]=malloc(strlen(p));
   strcpy(q[i],p);
   return q;
}
```

Naturalmente nelle variabili locali del main ci deve essere qualcosa come:

```
  char** q=malloc(10*sizeof(char*)); //spazio per 10 frasi
```

7. Supponendo di avere scritto il programma del punto 6., scrivere la funzione `stampa(char**)` che stampa le stringhe memorizzate nell'array a partire dal puntatore all'array. (3)

```
stampa (char **q,int N)
{
    int i;
    for(i=0;i<N;i++)
            printf("stampa: %s\n",q[i]);
}
```

8. Si consideri la seguente gerarchia di directory presenti in /:

```
 /usr
 |
 -------------------------------------
 |              |              |
antonio        gianni         giulia
 |    |                       |    |
uno  due                   cinque mycat
```

dove `uno, due, cinque` sono i nomi di file dati contenuti nelle rispettive directory. L'unico file eseguibile è `mycat`. I filemode dei file sono:
usr=555, antonio=501, giulia=345, uno=434, due=545, cinque=456, mycat=555.
I proprietari dei file sono:
usr=root, antonio=Gianni, giulia=Antonio, uno=Gianni, due=Antonio, cinque=Giulia, mycat=Giulia.
I gruppi ai quali appartengono gli utenti sono:
usr=root, Antonio=studenti, Giulia=studenti, Gianni=staff

Quali di questi comandi funzionano correttamente?

Utente Antonio:	../giulia/mycat uno	../giulia/mycat due	../giulia/mycat cinque
Utente Gianni:	../giulia/mycat uno	../giulia/mycat due	../giulia/mycat cinque
Utente Giulia:	./mycat ../antonio/uno	./mycat ../antonio/due	./mycat cinque

(3)

Nota: il bit 'x' su un file directory significa che posso accedere alla directory.
In questo problema ci sono 3 utenti, Antonio, Gianni, Giulia, 3 directory, *antonio*, *gianni* e *giulia,* e alcuni file.
La prima cosa che devo vedere è se è possibile accedere alle directory. */usr* ha I bit di esecuzione alti per proprietario, gruppo e altri, quindi tutti possono accedervi.
Le altre domande che devo pormi sono:
può Antonio accedere alla directory antonio? antonio è di Gianni, diverso utente che appartiene ad un altro gruppo, quindi devo vedere I bit di 'other'. Il bit di esecuzione è alto, quindi Antonio può accedere alla directory antonio.
Può Antonio accedere alla directory giulia? Il file directory giulia è di Antonio, quindi devo vedere I bit del proprietario → Antonio può accedere a giulia.
Può Antonio eseguire mycat?
Il file eseguibile mycat è di Giulia, che appartiene allo stesso gruppo di Antonio. I bit del gruppo dicono che Antonio può leggere ed eseguire mycat.
Può Antonio leggere il file uno? Uno è di Gianni che non appartiene allo stesso gruppo di Antonio quindi devo vedere I file di other → Antonio può eseguire *mycat uno*
Può Antonio leggere il file due? Due è di Antonio, quindi devo vedere I bit dell'owner di due → Antonio può eseguire *mycat due*
Può Antonio leggere il file cinque? cinque è di Giulia, quindi devo vedere I bit del gruppo Antonio può eseguire *mycat cinque.*
Un simile ragionamento porta a dire che Gianni può eseguire mycat e può leggere tutti I file.
Ugualmente, si può concludere che Giulia non può accedere alla directory giulia e quindi non può eseguiere mycat. Quindi le risposte sono:
si si si
si si si
no no no

9.Si consideri la struttura di directory del punto 8. Quali file possono essere correttamente letti e scritti dall'utente Alex che appartiene al gruppo staff? (1)

Assumiamo che Alex possa eseguire un programma che può leggere e scrivere un file.

Alex può accedere alla directory antonio? I bit del gruppo mi dicono di no.
Alex può accedere alla directory giulia? Si
Alex può leggere e scrivere cinque? Si

Quindi le risposte sono: Alex può leggere e scrivere solo il file cinque.

10. Su una schedina basata sul processore ARM per applicazioni dedicate viene installato una certa distribuzione Linux con disco a stato solido con blocchi di disco di 1KByte.

Il file system è così configurato: boot blocco+super bloccol+lista di INODE + lista di 10^6 blocchi dati. La lista degli INODE è divisa in due parti: una lista di INODE utilizzata per rappresentare I file speciali di tipo directory e una lista di INODE utilizzata per rappresentare I file regolari. La lista di INODE utilizzata per rappresentare I file speciali di tipo directory ha 10 blocchi, quindi ci sono solo 10 directory in questo sistema.

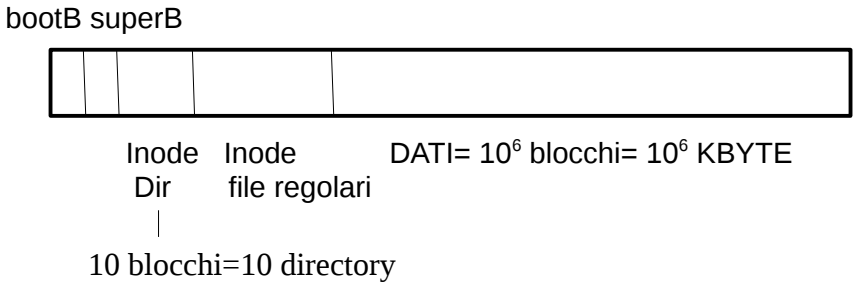
Gli INODE contengono una lista di 12 numeri di blocchi dati. I numeri sono di 4byte. Altri blocchi dati sono listati mediante singola indirettezza.

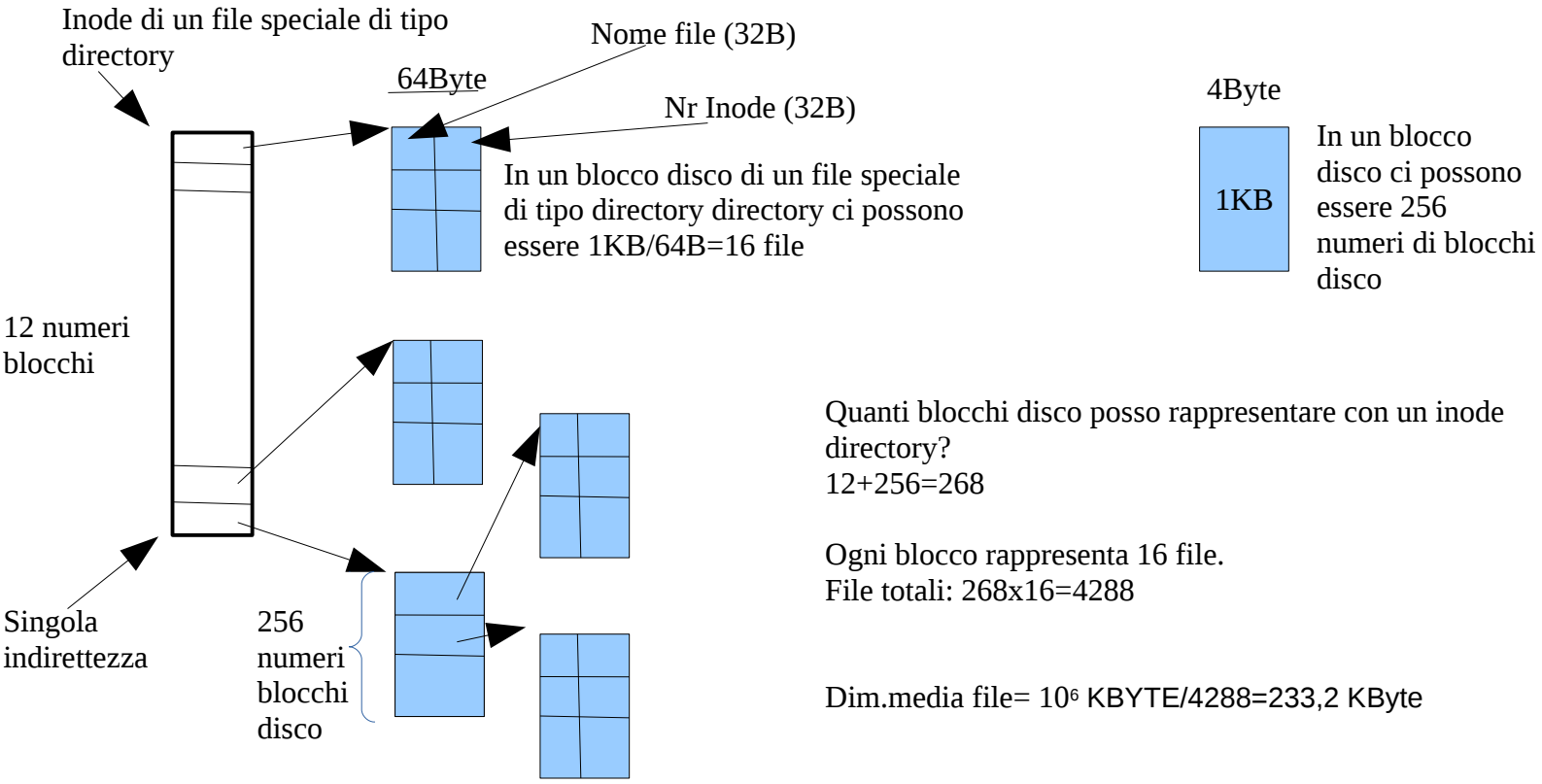
I blocchi di disco che rappresentano directory sono formattati nel seguente modo: 32 byte per indicare il nome dei file più 32 byte per il numero di INODE del file.

Calcolare la dimensione media dei file. (4)

==========

Nota: in un file speciale di tipo directory I blocche che costituiscono il file non rappresentano dati binari, ma tabelle inode-nome file (contenuto directory)





Quanti blocchi disco posso rappresentare con un inode directory?
12+256=268

Ogni blocco rappresenta 16 file.
File totali: 268x16=4288

Dim.media file= 10^6 KBYTE/4288=233,2 KByte