Caselle riservate

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Sistemi Operativi

Compito d'esame

21 Settembre 2016

Matricola _____ Cognome _____ Nome _____

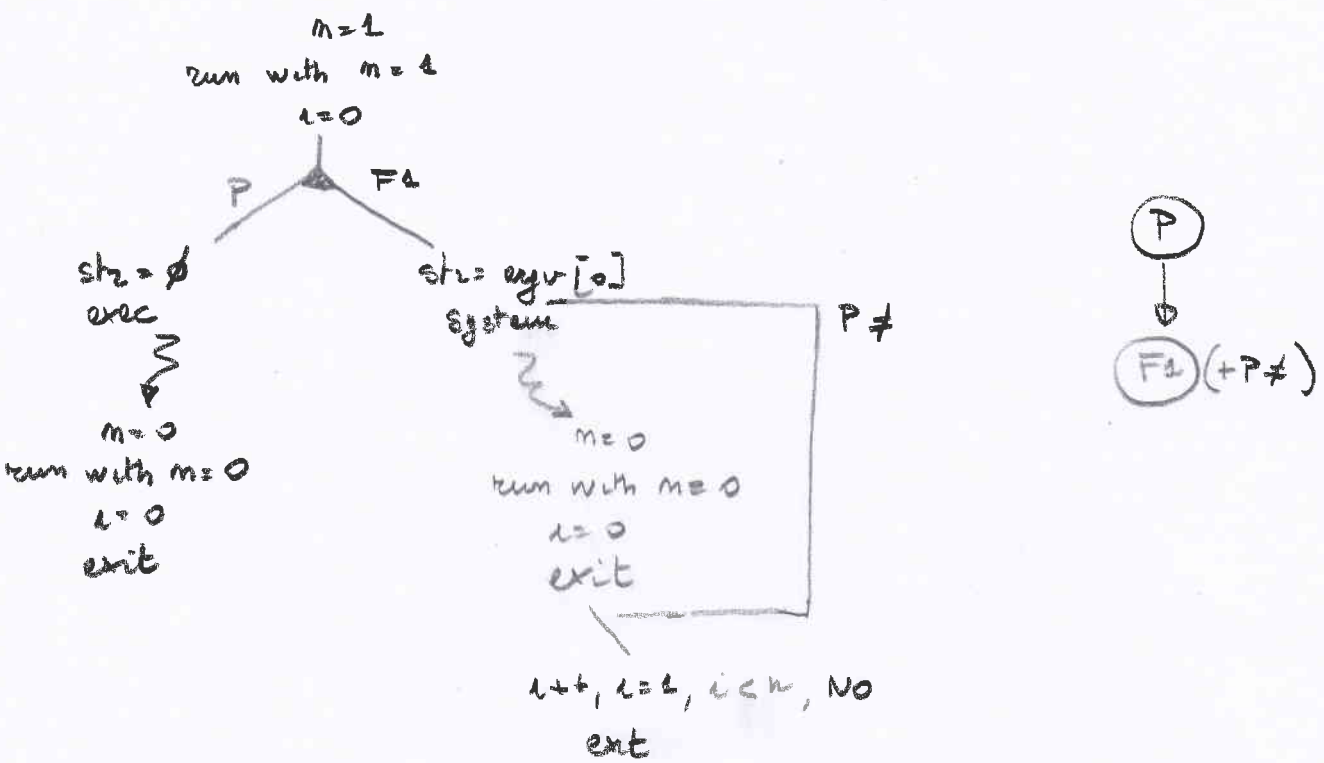
Docente: ◯ Quer ◯ Sterpone

Non si possono consultare testi, appunti o calcolatrici a parte i formulari distribuiti dal docente. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 100 **minuti.**

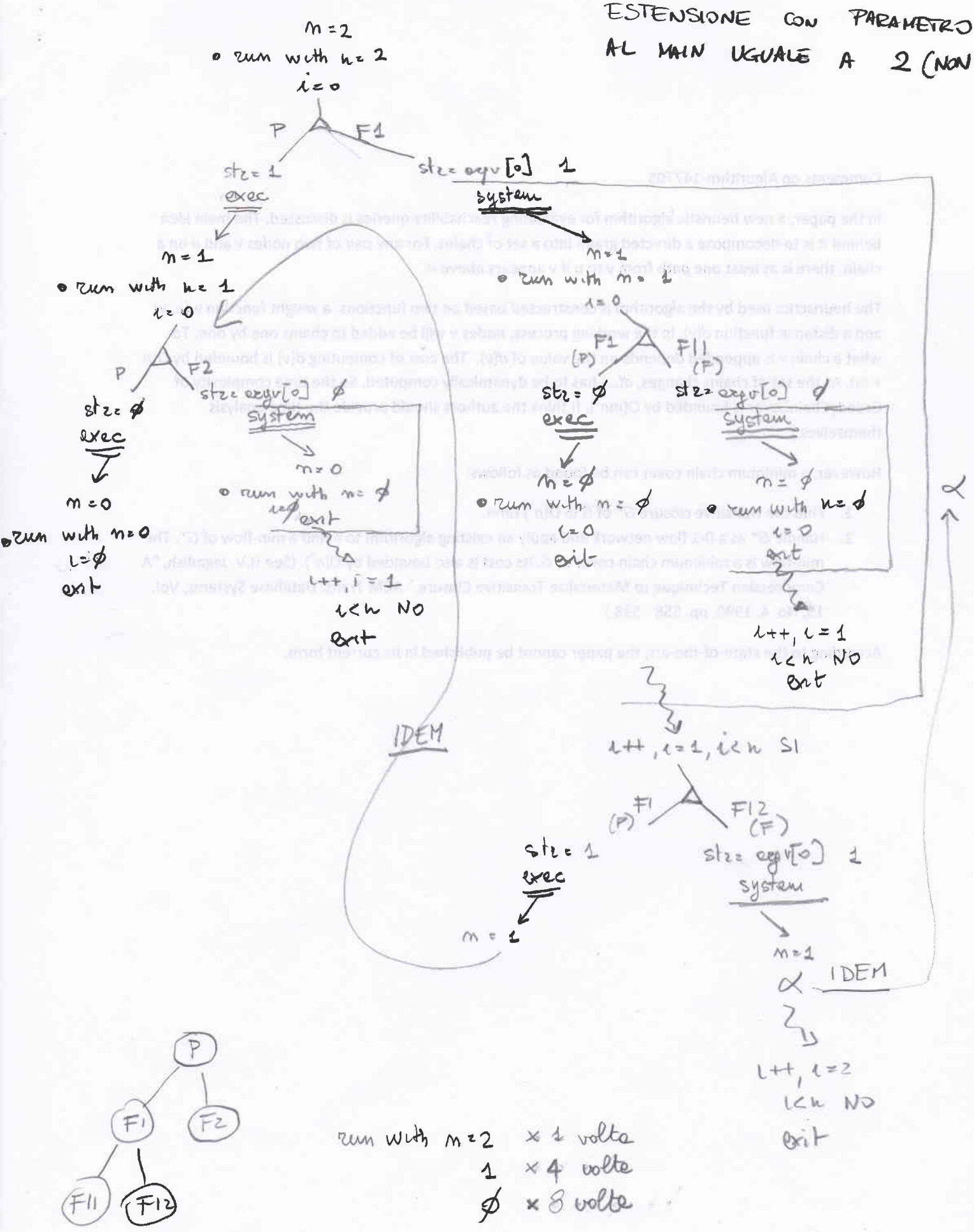
1. Si riporti il control flow graph e l'albero di generazione dei processi ottenuto dall'esecuzione del seguente tratto di codice C. Si supponga che il programma venga eseguito con un unico parametro, il valore intero 1, sulla riga di comando. Si indichi inoltre che cosa esso produce su video e per quale motivo.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/wait.h>
6
7  int main (int argc, char *argv[]) {
8    int i, n;
9    char str[50];
10
11   n = atoi (argv[1]);
12
13   printf ("run with n=%d\n", n); fflush (stdout);
14
15   for (i=0; i<n; i++) {
16     if (fork () > 0) {
17       sprintf (str, "%d", n-1);
18       execlp (argv[0], argv[0], str, NULL);
19     } else {
20       sprintf (str, "%s %d", argv[0], n-1);
21       system (str);
22     }
23   }
24
25   exit (0);
26 }
```

OUTPUT: run with n=1
run with n=0
run with n=0



ESTENSIONE CON PARAMETRO AL MAIN UGUALE A 2 (NON 1)



run with n = 2 × 1 volta
1 × 4 volte
∅ × 8 volte

2. Si descriva l'utilizzo dei segnali nel sistema operativo UNIX/Linux indicandone i principali vantaggi e svantaggi.
   Si scriva inoltre un programma in linguaggio C in grado di accettare segnali `SIGUSR1` e `SIGUSR2` e di:
   - Visualizzare su standard output un messaggio di successo ogni volta che un segnale `SIGUSR1` è seguito da un segnale `SIGUSR2` o viceversa.
   - Visualizzare su standard output un messaggio di errore ogni volta che riceve due segnali `SIGUSR1` oppure `SIGUSR2` di seguito.
   - Terminare non appena riceve tre segnali `SIGUSR1` oppure `SIGUSR2` di seguito.

   Si indichi infine come è possibile inviare tali segnali al processo utilizzando comandi di shell.

Interrupt s/w

Gestiscono eventi asincroni

Possono essere usati per comunicare

Occorre generare, consegnare e gestire il segnale: signal, kill, pause, alarm

La memoria dei segnali "pending" è limitata

I segnali possono essere bloccati

Richiedono funzioni rientranti e producono race condition

10

kill - SIGUSR1 pid

kill - SIGUSR2 pid

12

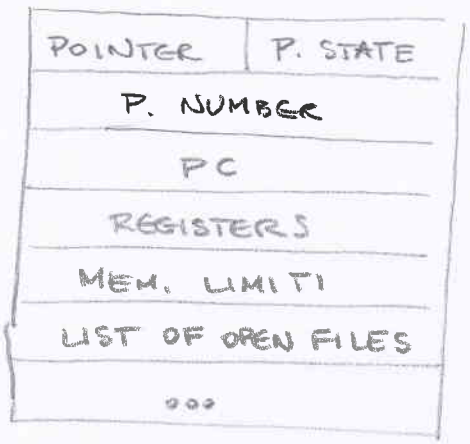
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

static void sigUsr (int);

int s1, s2, s3;
int n = 0;

static void
sigUsr (
  int signo
) {
  n++;
  s1 = s2;
  s2 = s3;

  if (signo == SIGUSR1)
    s3 = 1;

  if (signo == SIGUSR2)
    s3 = -1;

  // Only for First Call
  if (n==1)
    return;

  if (s1==s2 && s2==s3) {
    fprintf (stdout, "Stop ...\n");
    exit (1);
  }

  if (s2==s3)
    fprintf (stdout, "Error ...\n");

  if (s2!=s3)
    fprintf (stdout, "Success ...\n");

  return;
}

int
main (void) {
  if (signal(SIGUSR1, sigUsr) == SIG_ERR) {
    fprintf (stderr, "Signal Handler Error.\n");
    return (1);
  }
  if (signal(SIGUSR2, sigUsr) == SIG_ERR) {
    fprintf (stderr, "Signal Handler Error.\n");
    return (1);
  }

  while (1) {
    //fprintf (stdout, "Before pause.\n");
    pause ();
    //fprintf (stdout, "After pause.\n");
  }

  return (0);
}
```

3. Si indichino le principali differenze tra *processi* e *thread*, riportandone le caratteristiche principali e i relativi vantaggi e svantaggi. Quale strategia potrebbe essere utilizzata per ordinare un vettore di dimensioni enormi in maniera concorrente? Si indichi inoltre che cosa si intende per *Process Control Block*, *context switching* e *implementazione ibrida* dei thread.

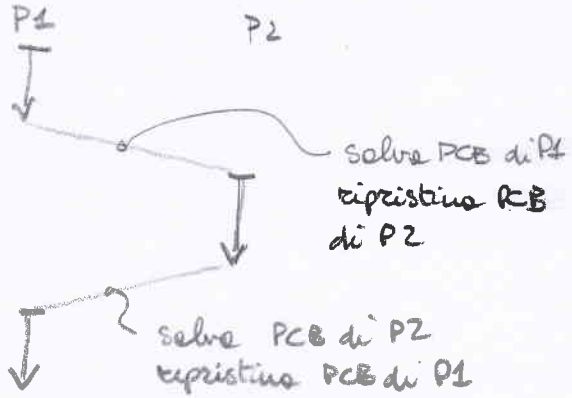
P spazio di indirizzamento proprio
singola traccia di esecuzione
costosi (tempo e memoria)

T spazio di indirizzamento condiviso (codice & dati)
Tracce esecuzione diverse nello stesso codice (PC, stack, registri propri)
meno costosi (tempo e memoria)

PCB

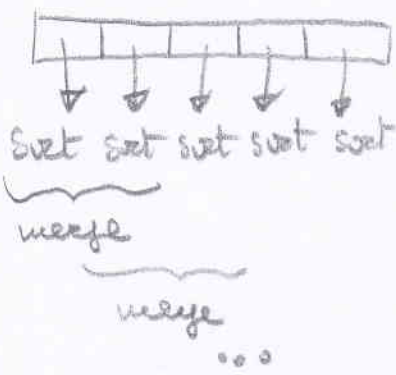


CONTEXT SWITCHING — CPU assegnata a P≠ (overhead per la CPU)



Implementazione Ibrida o Mista
Utilizza tanto user level thread quanto kernel level thread
n thread utente sono accoppiati a m thread kernel
(normalmente $n \gg m$)

Ordinamento di un vettore:
divide-and-conquer ⇒ dividiamo il vettore in n sezioni
ordiniamo ciascuna sezione con un thread diverso
fondiamo (merge) le sezioni ordinate



n ordering thread
1 merge thread (o più)

4. Si scriva uno script BASH che, una volta ricevuto su linea di comando il nome di un file di tipo "elenco" contenente su righe successive un elenco di cartelle, esegua le seguenti operazioni:
   - Controlli il corretto passaggio del parametro e l'esistenza del file specificato, stampando opportuni messaggi d'errore in caso rilevi dei problemi.
   - Esegua il programma di nome `tester` su ciascun file con estensione `.test` presente nelle cartelle riportate nel file di elenco. Il programma `tester` produce su standard output una riga del tipo
     `result  time`
     dove `result` è uguale a `PASS` oppure `FAIL` e `time` è un valore intero pari al tempo di esecuzione (e.g., `PASS 108`).
   - Memorizzi tutti i nomi dei file di estensione `.test` che hanno ricevuto risultato `PASS` nel file di nome `pass.txt`, indicando per ognuno il nome e il tempo di processamento.
   - Memorizzi tutti i nomi dei file di estensione `.test` che hanno ricevuto risultato `FAIL` nel file di nome `fail.txt`, indicando per ognuno il nome e il tempo di processamento.

   Al termine del procedimento lo script visualizzi su standard output il numero totale di file processati con risultato `PASS` e il relativo tempo totale di processamento e il numero totale di file processati con risultato `FAIL` e il relativo tempo totale di processamento.

   Lo script deve occuparsi della rimozione di eventuali file temporanei generati. Si supponga che le cartelle riportate nel file di elenco fornito quale parametro non contengano sotto-cartelle.

```
#!/bin/bash

# Check arguments
if [ $# -ne 1 ]; then
    echo "Usage: es4.sh <dirfile>"
    exit 1
fi

if [ ! -f $1 ]; then
    echo "File $1 does not exist"
fi

# Script initialization
echo -n "" > "pass.txt"
echo -n "" > "fail.txt"
pass_tot=0
fail_tot=0
pass_time=0
fail_time=0

# Do the job
for dir in $(cat $1); do
    for file in $(find $dir -maxdepth 1 -name "*.test"); do
        out=$(./tester $dir"/"$file)
        res=$(echo $out | cut -d " " -f 1)
        time=$(echo $out | cut -d " " -f 2)
        if [ $res == "PASS" ]; then
            echo "$file $time" >> "pass.txt"
            let "pass_tot++"
            let "pass_time+=time"
        elif [ $res == "FAIL" ]; then
            echo "$file $time" >> "fail.txt"
            let "fail_tot++"
            let "fail_time+=time"
        fi
    done
done

# Output results
echo "Passed tests: $pass_tot ($pass_time tot time)"
echo "Failed tests: $fail_tot ($fail_time tot time)"
```

5. Il Web Server di un'azienda mantiene un log degli accessi, con il seguente formato:

```
178.1.192.33 goodguy [10/07/2015:13:55:36] GET index.html 200
34.52.1.33 badguy [10/07/2015:14:04:10] POST /services/ask 200
34.52.1.37 badguy [10/07/2015:14:04:10] POST /intranet/login 401
34.52.177.48 okguy [10/07/2015:14:32:00] POST /intranet/login 401
178.1.192.41 nastyguy [10/07/2015:18:29:01] POST /intranet/login 401
178.1.192.42 nastyguy [10/07/2015:18:56:01] POST /intranet/login 200
178.1.192.32 goodguy [11/07/2015:15:21:43] POST /intranet/login 200
123.154.48.1 worstguy [11/07/2015:00:21:32] GET /services/list 200
```

in cui per ogni riga sono indicati: l'indirizzo IP dell'host da cui proviene la richiesta di accesso, il nome dell'utente che effettua la richiesta, la data e l'ora della richiesta, il tipo della richiesta (`GET` o `POST`), il nome della risorsa oggetto della richiesta e la risposta del server alla richiesta d'accesso (`200` nel caso di richiesta accettata, `401` nel caso di richiesta respinta).

Si scriva uno script AWK che, dato un file contenente un elenco di nomi utente, visualizzi su standard output l'elenco degli indirizzi IP da cui almeno uno degli utenti specificati ha effettuato una richiesta di tipo `POST` in una determinata data e a cui il server ha dato risposta `200`. L'output non deve contenere indirizzi duplicati. Lo script deve ricevere su linea di comando il nome del file di log, quello contenente l'elenco degli utenti (specificati uno per riga) e la data per la quale effettuare il controllo.

Nell'esempio precedente, se il file degli utenti è il seguente:

```
badguy
nastyguy
worstguy
```

e la data è `10/07/2015`, lo script deve produrre su standard output quanto segue:

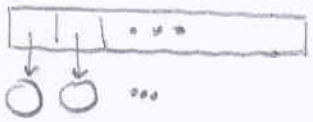
```
34.52.1.33
178.1.192.42
```

```
#!/bin/usr/awk -f

BEGIN {

    # Load users we are interested in
    while(getline < ARGV[2]){
        users[$1] = $1
    }

    # Manage parameters that should not be parsed
    date=ARGV[3]
    ARGC-=2
}

{
    # Select IP address satisfying the required
    # conditions (storing them in a vector
    # in order to remove duplicates)
    if($2 in users){
        if(match($3, "\\["date".+\\]") && $4 == "POST" && $6 == "200"){
            ips[$1] = $1
        }
    }
}

END {
    # Output selected IP addresses
    for(ip in ips){
        print(ips[ip])
    }
}
```

6. Si descriva l'organizzazione di un file system a un livello, due livelli, ad albero, a grafo aciclico e a grafo ciclico, indicandone caratteristiche, vantaggi e svantaggi. In particolare, si illustri in tale ambito il concetto di "link". Si illustrino inoltre, mediante l'utilizzo di esempi, i concetti di hard-link, soft-link e inode nel sistema operativo UNIX/Linux.

Direttori a :

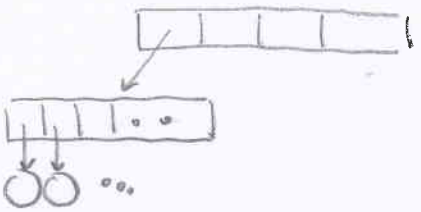
- 1 livello



dir
file

Semplice
molti file ⇒ nomi tutti diversi

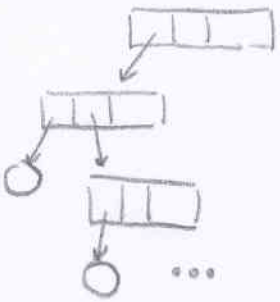
- 2 livelli



dir user
dir
file

$\exists$ 1 dir $\neq$ $\forall$ user

- Albero



Generalizza precedente
Non permette condivisione

- Grafo aciclico ...

Permette condivisione (link)
Gestione più complessa (link = riferimento indiretto ; lo stesso oggetto ha più path ; cancellazione ... ?)

- Grafo ciclo ...

No link ai direttori

N O N O N O

Link :

- Hard link = ↑ della directory entry all' i-node del file link
- Soft link = file con path della entry richiesta
  ln -s file link

I-node : $\forall$ file, l' i-node identifica diverse sue caratteristiche/riferimenti ; nella directory entry è contenuto il nome della entry e il puntatore al suo i-node ; nell' i-node sono contenuti i puntatori ai blocchi dati del file.