Esercizio c.1

> Il consiglio e' quello di dimenticarsi dei colori, e fare una funzione comune ad entrambi

```
int nr = 0
int nb = 0
condition c
float other
float result

def entry meanblack(float f):
   nb++
   rv = meanf(f)
   nb--
   return rv

def entry meanred(float f):
   nr++
   rv = meanf(f)
   nr--
   return rv

# Questa e' una funzione interna al monitor, non una procedure entry
def float meanf(float f):
   if (nr == 0 or nb == 0):
      c.wait()
      result = (f + other) / 2
   else:
      other = f
      c.signal()
   return result
```

Esercizio c.2

> `db` e' una struttura dati che ha `add(msg, sender)` e `get(sender)` come metodi di accesso

```
def nbl_send(m, dest):
   asend(<getpid(), m>, dest)

def nbl_receive(sender):
   asend(<getpid(), TAG>, getpid())
   while (True):
      <s, m> = arecv(ANY)
      if (s == getpid() and m == TAG):
```

```
            break
        else:
            db.add(m, s)
    return db.get(sender)
```

Esercizio g.1

File structure

```
/
|- f
|- g
|- d
   |- d1
   |- d2
   |- d3
```

Indice	FAT	Contenuto
0	Res	
1	Res	
2	E	`/`
3	E	`/f`
4	4	
5	EOC	`/g`, secondo blocco
6	5	`/g`, primo blocco
7	EOC	`FREE`, quarto blocco e `/d/d3`
8	12	`FREE`, primo blocco
9	EOC	`/d/`
10	EOC	Sia `/d/d2` che secondo blocco `/d/d1`
11	10	`/d/d2`, primo blocco
12	13	`FREE`, secondo blocco
13	7	`FREE`, terzo blocco

Abbiamo incongruenza nel blocco 4, 7 e 10 (che si ripercuote anche al blocco 11)

Per risolvere quella a blocco 4 aggiungiamo 4 alla lista dei blocchi `FREE`

Per risolvere quella a blocco 7 abbiamo togliamo il blocco 7 dalla lista dei blocchi liberi

Per risolvere quella a blocco 10 e 11 abbiamo piu' strade: - O copiamo i dati del blocco 10 in un altro blocco, facendo poi puntare `/d/d1` al blocco appena duplicato - O aggiungiamo i due blocchi alla lista dei blocchi liberi, aggiornando la cartella (togliendo quindi le due entry per `d1` e `d2`) - O mettiamo a indice 11 EOC, cosi' da avere il file `d2` troncato ma il file system coerente

Esercizio g.2

Domanda a

Vengono messi i parametri in registri specifici della CPU

La chiamata della system call causa una trap, che viene catturata dal SO

Alla fine dell'esecuzione della system call viene in serito in un altro registro il kernel pone il valore di ritorno

Domanda b

Raid 5 usa 3 dischi per i dati e 1 per la parita', quindi ha 3GiB utilizzabili

Raid 10 usa 2 dischi in striping, e poi fa mirroring, quindi ha 2GiB utilizzabili

Quindi lo spazio disponibile in Raid 5 e' maggiore di Raid 10

Domanda c

Dalla definizione di algoritmo a stack sappiamo che aumentando il frame buffer abbiamo almeno le pagine con dimensione del frame buffer minore

Quindi non e' possibile che `m` frame causino piu' page fault di `n` frame (con `n < m`)

Domanda d

Il problema e' che i processi non danno il massimo delle risorse che utilizzano all'inizio delle esecuzione

Si puo' stimare, ma non sara' mai esatto come prevede l'algoritmo del banchiere

Nota: Non va bene dire che e' inefficiente, in realta' come dice il relativo teorema e' efficiente perche' viene aggiornata solo l'ultima riga della tabella