# Parte A

**ESERCIZIO A-1 (4 punti)**

Si considerino quattro processi (A,B,C,D) che cooperano secondo il paradigma produttore-consumatore. In particolare i processi A e B producono dati che depositano su un buffer ad *n* posizioni, e i processi C e D sono consumatori. L'accesso al buffer è regolato dai semafori di sincronizzazione `bufferpieno` (inizializzato ad `n`), `buffervuoto` (inizializzato a `0`) e dal semaforo di mutua esclusione `mutex` (inizializzato ad `1`).
Il codice eseguito dai processi è mostrato dalla seguente tabella:

| | Processi A e B: | | Processi C e D: |
|---|---|---|---|
| 1 | `while (true) {` | 1 | `while (true) {` |
| 2 | `<produce un dato>;` | 2 | `wait(buffervuoto);` |
| 3 | `wait(bufferpieno);` | 3 | `wait(mutex);` |
| 4 | `wait(mutex);` | 4 | `<preleva il dato nel buffer>;` |
| 5 | `<deposita il dato nel buffer>;` | 5 | `signal(mutex);` |
| 6 | `signal(mutex);` | 6 | `signal(bufferpieno);` |
| 7 | `signal(buffervuoto);` | 7 | `<consuma il dato>;` |
| 8 | `}` | 8 | `}` |

Supponendo che (a partire dal tempo iniziale 0) si sussegua la seguente sequenza di eventi:
Tempo 1: C esegue (interamente) le righe 1,2
Tempo 2: A esegue (interamente) le righe 1,2,3
Tempo 3: D esegue (interamente) le righe 1,2
Tempo 4: B esegue (interamente) le righe 1,2,3,4,5
Tempo 5: A esegue (interamente) la riga 4
Tempo 6: B esegue (interamente) le righe 6,7
qual è lo stato raggiunto negli istanti di tempo 1,2,3,4,5 e 6?

**SOLUZIONE**

| | Valore di `bufferpieno` | Valore di `buffervuoto` | Valore di `mutex` | Processi sospesi su `bufferpieno` | Processi sospesi su `buffervuoto` | Processi sospesi su `mutex` |
|---|---|---|---|---|---|---|
| 1 | `n` | `0` | `1` | - | C | - |
| 2 | `n-1` | `0` | `1` | - | C | - |
| 3 | `n-1` | `0` | `1` | - | C,D | - |
| 4 | `n-2` | `0` | `0` | - | C,D | - |
| 5 | `n-2` | `0` | `0` | - | C,D | A |
| 6 | `n-2` | `0` | `0` | - | D | - |

**ESERCIZIO A-2 (4 punti)**

Si consideri una politica di gestione del processore basata sulla priorità, dove la priorità di ogni processo è assegnata dinamicamente nel modo seguente:

- alla generazione del processo è inizializzata ad 8;
- quando un processo bloccato viene riattivato, la sua priorità viene incrementata di 2 (fino ad un massimo di 16)
- al termine di ogni quanto di tempo, la priorità del processo in esecuzione (che ha sfruttato tutto il quanto di tempo) viene decrementata di 1 (fino ad un minimo di 1) e il processo in esecuzione viene inserito nella coda pronti corrispondente alla sua nuova priorità

La politica prevede il prerilascio e il processo in esecuzione è sempre quello con il maggior valore di priorità. Si utilizza un'unica coda dei processi pronti, ordinata per valori decrescenti di priorità; a pari priorità prevale l'ordine di inserimento I processi di pari priorità sono schedulati secondo l'algoritmo round-robin con quanto di tempo pari a 10.
Al tempo considerato (indicato convenzionalmente come tempo 0), sono presenti i seguenti processi:

- A in esecuzione con priorità 12
- B bloccato con priorità 11
- C pronto con priorità 11
- D pronto con priorità 8

Si considerino le due sequenze di eventi (**da considerare in alternativa tra loro**):

**Sequenza 1**:

- al tempo 5 viene creato il processo E;
- al tempo 11 si blocca il processo in esecuzione;
- al tempo 14 si riattiva il processo B;

**Sequenza 2:**

- al tempo 5 si blocca il processo in esecuzione;
- al tempo 7 si riattiva il processo B;
- al tempo 20 si blocca il processo in esecuzione;

Per ognuna delle due sequenze descrivere il comportamento dello scheduler al verificarsi degli eventi considerati e all'esaurimento di ogni quanto di tempo, indicando per ogni intervento il processo in esecuzione, la composizione della coda pronti e la priorità di tutti i processi pronti o in esecuzione.

**SOLUZIONE**

**Sequenza 1:**

| | | | |
|---|---|---|---|
| t=0 | in esecuzione A (12) | Coda pronti C(11)->D(8) | bloccati: B(11) |
| t=5 | in esecuzione A (12) | Coda pronti C(11)->D(8)->E(8) | bloccati: B(11) |
| t=10 | in esecuzione C (11) | Coda pronti A(11)->D(8)->E(8) | bloccati: B(11) |
| t=11 | in esecuzione A (11) | Coda pronti D(8)->E(8) | bloccati: B(11); C(11) |
| t=14 | in esecuzione B (13) | Coda pronti A(11)->D(8)->E(8) | bloccati: C(11) |

**Sequenza 2:**

| | | | |
|---|---|---|---|
| t=0 | in esecuzione A (12) | Coda pronti C(11)->D(8) | bloccati: B(11) |
| t=5 | in esecuzione C(11) | Coda pronti D(8) | bloccati: B(11); A (12) |
| t=7 | in esecuzione B (13) | Coda pronti C(11)->D(8) | bloccati: A (12) |
| t=17 | in esecuzione B (12) | Coda pronti C(11)->D(8) | bloccati: A (12) |
| t=20 | in esecuzione C (11) | Coda pronti D(8) | bloccati: A (12); B(12) |

**ESERCIZIO A-3 (3 punti)**

Un sistema con processi A, B, C, D, E e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [4,5,6,7], ha raggiunto lo stato mostrato nelle tabelle seguenti.
Lo stato è sicuro?

| Assegnazione attuale | | | | |
|---|---|---|---|---|
| | R1 | R2 | R3 | R4 |
| A | | | 1 | 1 |
| B | 1 | | | 2 |
| C | | 1 | 1 | 1 |
| D | | 2 | 2 | 1 |
| E | 2 | | 1 | 1 |

| Esigenza residua (esclusa l'assegnazione attuale) | | | | |
|---|---|---|---|---|
| | R1 | R2 | R3 | R4 |
| A | 2 | 2 | 3 | 2 |
| B | 2 | 2 | 4 | 3 |
| C | 1 | 2 | 2 | 1 |
| D | 4 | 2 | 1 | 2 |
| E | 1 | 2 | 1 | 1 |

| Molteplicità | | | |
|---|---|---|---|
| R1 | R2 | R3 | R4 |
| 4 | 5 | 6 | 7 |

| Disponibilità | | | |
|---|---|---|---|
| 1 | 2 | 1 | 1 |

**SOLUZIONE**

Per la verifica dello stato sicuro:
1) Il processo E può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 3 , 2 , 2 , 2 }
2) Il processo C può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 3 , 3 , 3 , 3 }
3) Il processo A può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 3 , 3 , 4 , 4 }
4) Il processo B può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 4 , 3 , 4 , 6 }
5) Il processo D può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 4 , 5 , 6 , 7 }

Di conseguenza: lo stato è sicuro.

**ESERCIZIO A-4 (2 punti)**

In un sistema con thread realizzati a livello utente, sono presenti i processi P1 con thread T11, T12, il processo P2 con i thread T21 e T22 e il processo P3 con i thread T31 e T32. Ogni processo gestisce i suoi thread senza prerilascio. La politica di scheduling dei processi è la round robin.
Al tempo *T* è in esecuzione il processo P1, il processo P2 è sospeso sul semaforo *sem1* e il processo P3 è pronto. Nel processo P1 è in esecuzione il thread T11 e i rimanenti thread dei tre processi sono pronti, con il seguente ordinamento nelle rispettive code:
Processo P1: T12 Processo P2: T21->T22 Processo P3: T31->T32.
Quale thread è in esecuzione dopo che si sono verificate (in alternativa) le due seguenti sequenze di eventi:

a) 1. Il thread in esecuzione esegue una operazione di *thread_yield*;
   2. il processo in esecuzione esaurisce il quanto di tempo;
   3. il thread in esecuzione esegue una *signal* sul semaforo *sem1*;
   4. il processo in esecuzione esaurisce il quanto di tempo;
b) 1. Il thread in esecuzione esegue una *wait* sul semaforo *sem1*;
   2. il thread in esecuzione esegue una *signal* su *sem1*;
   3. il processo in esecuzione esaurisce il quanto di tempo;
   4. Il thread in esecuzione esegue una operazione di *thread_yield.*

**SOLUZIONE**

| | Sequenze di eventi | Thread in esecuzione dopo l'evento |
|---|---|---|
| a-1 | Il thread in esecuzione esegue una operazione di *thread_yield* | T12 |
| a-2 | il processo in esecuzione esaurisce il quanto di tempo | T31 |
| a-3 | il thread in esecuzione esegue una *signal* sul semaforo *sem1* | T31 |
| a-4 | il processo in esecuzione esaurisce il quanto di tempo | T12 |
| | | |
| b-1 | il thread in esecuzione esegue una *wait* sul semaforo *sem1* | T31 |
| b-2 | il thread in esecuzione esegue una *signal* su *sem1* | T31 |
| b-3 | il processo in esecuzione esaurisce il quanto di tempo | T21 |
| b-4 | Il thread in esecuzione esegue una operazione di *thread_yield* | T22 |

**ESERCIZIO A- 5 (2 punti)**

Dire quali delle seguenti affermazioni sono vere o false:

| | |
|---|---|
| Se il sistema è in uno stato sicuro non si può verificare stallo, qualunque sia l'ordine di richiesta e assegnazione delle risorse | FALSO |
| Se il sistema è in uno stato non sicuro, è inevitabile lo stallo | FALSO |
| Alcuni processi in stallo possono essere in stato di *pronto* | FALSO |
| Se due processi sono in stallo per aver richiesto le risorse A e B, allora A e B sono non prerilasciabili | VERO |
| Lo spooling è una tecnica per la prevenzione dinamica dello stallo | FALSO |
| L'algoritmo del banchiere deve essere applicato allo scadere di ogni quanto di tempo | FALSO |

# Parte B

**ESERCIZIO B-1 (4 punti)**

Un sistema operativo simile a UNIX, che gestisce la memoria con paginazione a domanda, utilizza il processo *PageDaemon,* con parametri *lotsfree= 5 e minfree=2,* e l'algoritmo di sostituzione *Second Chance.* Gli elementi della *CoreMap* hanno i campi *Proc* (processo a cui è assegnato il blocco; il campo è vuoto se il blocco è libero); *Pag* (pagina del processo caricata nel blocco), *Rif* (bit di pagina riferita utilizzato da *Second Chance*). Al tempo *t* sono presenti i processi A, B, C, D e la *Core Map* ha la configurazione mostrata in figura, con il puntatore dell'algoritmo di sostituzione posizionato sul blocco 11. I primi 6 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione.

| Proc | | | | | | | C | A | B | A | | B | C | D | D | | D | B | A | B | | C | | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pag | | | | | | | 0 | 1 | 0 | 2 | | 6 | 3 | 1 | 2 | | 6 | 2 | 7 | 3 | | 7 | | 2 |
| Rif | | | | | | | 0 | 1 | 0 | 0 | | 1 | 1 | 0 | 1 | | 0 | 0 | 1 | 0 | | 1 | | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Core Map al tempo *t*

Il *PageDaemon* interviene al tempo *t* e successivamente ogni 10 msec. Ad ogni intervento, *PageDaemon* avanza per 1 msec occupando in modo esclusivo il processore e scarica fino a 3 pagine o, in alternativa, esegue lo *swapout* di un processo.
La selezione dei processi candidati allo *swapout* avviene in ordine alfabetico. In caso di errori di pagina, i blocchi liberi vengono assegnati in ordine crescente di indice.
Considerare la seguente evoluzione del sistema:

1. Dal tempo *t* al tempo *t+1* avanza il processo *PageDaemon*;
2. Dal tempo *t+1* al tempo *t+10* avanzano i processi B e C, che riferiscono nell'ordine le pagine B0, B1, B6, C2, C0, C4, B2, B5
3. Dal tempo *t+10* al tempo *t+11* avanza il processo *PageDaemon*;
4. Dal tempo *t+11* al tempo *t+20* avanzano i processi D e B, che riferiscono nell'ordine le pagine D3, D6, D2, D1, B0, B2, B3, B0;
5. Dal tempo t+20 al tempo t+21 avanza il processo *PageDaemon.*

Mostrare la configurazione della *CoreMap* ai tempi 1, 10, 11, 20 e 21

**SOLUZIONE**

(Modificare incrementalmente la configurazione iniziale della *CoreMap*, aggiornando anche la posizione del puntatore)

| Proc | | | | | | | C | A | B | A | | B | C | | D | | D | B | A | B | | C | | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pag | | | | | | | 0 | 1 | 0 | 2 | | 6 | 3 | | 2 | | 6 | 2 | 7 | 3 | | 7 | | 2 |
| Rif | | | | | | | 0 | 1 | 0 | 0 | | 0 | 0 | | 1 | | 0 | 0 | 1 | 0 | | 1 | | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Core Map al tempo *t+1*

| Proc | | | | | | | C | A | B | A | B | B | C | C | D | B | D | B | A | B | | C | | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pag | | | | | | | 0 | 1 | 0 | 2 | 1 | 6 | 3 | 4 | 2 | 5 | 6 | 2 | 7 | 3 | | 7 | | 2 |
| Rif | | | | | | | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | 1 | | 1 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Core Map al tempo *t+10*

| Proc | | | | | | | C | A | B | | B | B | C | C | D | B | | B | A | | | C | | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pag | | | | | | | 0 | 1 | 0 | | 1 | 6 | 3 | 4 | 2 | 5 | | 2 | 7 | | | 7 | | 2 |
| Rif | | | | | | | 0 | 0 | 0 | | 1 | 1 | 0 | 1 | 0 | 0 | | 0 | 0 | | | 0 | | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Core Map al tempo *t+11*

| Proc | | | | | | | C | A | B | D | B | B | C | C | D | B | D | B | A | D | B | C | | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pag | | | | | | | 0 | 1 | 0 | 3 | 1 | 6 | 3 | 4 | 2 | 5 | 6 | 2 | 7 | 1 | 3 | 7 | | 2 |
| Rif | | | | | | | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Core Map al tempo *t+20*

| Proc | | | | | | | C | | B | D | B | B | C | C | D | B | D | B | | D | B | C | | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pag | | | | | | | 0 | | 0 | 3 | 1 | 6 | 3 | 4 | 2 | 5 | 6 | 2 | | 1 | 3 | 7 | | 2 |
| Rif | | | | | | | 0 | | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | | 1 | 1 | 0 | | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

**ESERCIZIO B-2 (4 punti)**

In un file system di tipo Unix gli i-node contengono 5 puntatori diretti, un puntatore indiretto singolo e un puntatore indiretto doppio. Tutti i puntatori sono a 16 bit e i blocchi del disco hanno dimensione 128 bytes.

Si consideri il file A allocato sui blocchi: 33, 68, 69, da 709 a 712, da 800 a 820, da 1000 a 1049, da 400 a 560.
Specificare il contenuto dello i-node di A e dei blocchi che contengono i puntatori indiretti, supponendo che:

- il blocco 100 contenga i puntatori indiretti singoli;
- il blocco 101 contenga i puntatori indiretti doppi;
- Il contenuto del blocco 101 sia: 103, 104, 105.

**SOLUZIONE**

**i-node A:**

| **Puntatore** | **0** | **1** | **2** | **3** | **4** | **Ind. singolo** | **Ind. doppio** |
|---|---|---|---|---|---|---|---|
| **valore** | 33 | 68 | 69 | 709 | 710 | **100** | **101** |

Ogni blocco indiretto contiene 64 puntatori.
Contenuto dei blocchi indiretti:

- **Blocco 100:** 711, 712, 800 – 820, 1000 - 1040
- **Blocco 101:** 103, 104, 105
- **Blocco 103:** 1041 – 1049, 400 – 454
- **Blocco 104:** 455 - 518
- **Blocco 105:** 519 - 560

# SISTEMI OPERATIVI, CORSI A e B - QUARTO APPELLO - 29/6/2006

**ESERCIZIO B-3 (3 punti)**

In un sistema operativo che gestisce la memoria con partizioni variabili, la memoria fisica ha un'ampiezza di 35 Mbyte. La partizione 1, della lunghezza di 7 Mbyte, è riservata al sistema operativo, mentre il resto della memoria è disponibile per il caricamento dei processi.
Al tempo *t* sono caricati in memoria i seguenti processi:

- Processo A, nella partizione con origine 7 Mbyte e lunghezza 8 Mbyte;
- Processo B, nella partizione con origine 16 Mbyte e lunghezza 5 Mbyte;
- Processo C, nella partizione con origine 21 Mbyte e lunghezza 3 Mbyte;
- Processo D, nella partizione con origine 24 Mbyte e lunghezza 4 Mbyte;
- Processo E, nella partizione con origine 30 Mbyte e lunghezza 2 Mbyt

Successivamente si verificano i seguenti eventi:

1. al tempo *t+2* termina il processo C;
2. al tempo *t+ 5* viene generato il processo F il cui spazio virtuale occupa 2 Mbyte;
3. al tempo *t+10* termina il processo B;
4. al tempo *t+ 15* viene generato il processo G il cui spazio virtuale occupa 4 Mbyte.

Per l'assegnazione delle partizioni si utilizza la politica *best fit.*
Le partizioni libere sono concatenate con puntatori interni alle partizioni medesime, con associata la lunghezza. La prima partizione libera è individuata dal puntatore *PrimaPartizioneLibera*.
Mostrare la configurazione della memoria e il contenuto delle partizioni libere ai tempi t, t+2, t+5, t+10 e t+15.

**SOLUZIONE**

1) tempo t: *PrimaPartizioneLibera*: 15

| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | A | A | A | 28, 1 | B | B | B | B | B | C | C | C | D | D | D | D | 32, 2 | | E | E | Ø, 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

2) tempo t+2: *PrimaPartizioneLibera*: 15

| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | A | A | A | 21, 1 | B | B | B | B | B | 28, 3 | | | D | D | D | D | 32, 2 | | E | E | Ø, 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

3) tempo t+5: *PrimaPartizioneLibera*: 15

| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | A | A | A | 21, 1 | B | B | B | B | B | 32, 3 | | | D | D | D | D | F | F | E | E | Ø, 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

4) tempo t+10: *PrimaPartizioneLibera*: 15

| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | A | A | A | 32, 9 | | | | | | | | | D | D | D | D | F | F | E | E | Ø, 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

5) tempo t+15: *PrimaPartizioneLibera*: 19

| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | A | A | A | G | G | G | G | 32, 5 | | | | | D | D | D | D | F | F | E | E | Ø, 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

**ESERCIZIO B-4 (2 punti)**

Si consideri un disco con 200 cilindri, 4 facce e 100 settori per traccia.

a) Quanti blocchi contiene il disco?
b) Tradurre i seguenti indici di blocco nelle rispettive terne <cilindro, faccia, settore>: 76456, 65222, 3456, 25399, 46340, 9876.

**SOLUZIONE**

a) Il disco contiene 80.000 blocchi
b) Traduzione degli indici di blocco in terne:
cilindro = IndiceDiBlocco **div** (4*100)
faccia = (IndiceDiBlocco **mod** (4*100)) **div** 100
settore = (IndiceDiBlocco **mod** (4*100)) **mod** 100

| Indice di blocco | cilindro | faccia | settore |
|---|---|---|---|
| 76456 | 191 | 0 | 56 |
| 65222 | 163 | 0 | 22 |
| 3456 | 8 | 2 | 56 |
| 25399 | 63 | 1 | 99 |
| 46340 | 115 | 3 | 40 |
| 9876 | 24 | 2 | 76 |

**ESERCIZIO B-5 (2 punti)**

Calcolare la dimensione della FAT nei seguenti casi:

1. puntatori a 16 bit, blocchi di 16 KB, disco di 800 MB
2. puntatori di 24 bit, blocchi di 8 KB, disco di 1GB
3. puntatori di 32 bit, blocchi di 16 KB, disco di 100 GB
4. puntatori di 24 bit, blocchi di 2 KB, disco di 400 MB

**SOLUZIONE**

1. numero di blocchi del disco: $50 * 2^{10}$; Dimensione della FAT 100 Kbyte
2. numero di blocchi del disco: $128 * 2^{10}$ Dimensione della FAT 384 Kbyte
3. numero di blocchi del disco: $6{,}25 * 2^{20}$ Dimensione della FAT 25 Mbyte
4. numero di blocchi del disco: $200 * 2^{10}$ Dimensione della FAT 600 Kbyte