SISTEMI OPERATIVI, CORSI A e B Prova del 10/9/2008 CORSO |A| |B|

ESERCIZIO A-1 (4 punti)

Si considerino tre thread linux A,B e C che cooperano tramite un buffer circolare condiviso n posizioni, numerate da 0 a n-1. Il thread A deposita messaggi sul buffer circolare (ogni messaggio occupa una posizione del buffer), mentre i thread B e C prelevano i messaggi con il vincolo che ogni messaggio deve essere letto da entrambi i thread B e C una e una sola volta.
A tal fine i thread condividono una variabile di tipo mutex, denominata *mux* e inizializzata a 1, e tre variabili di tipo condition, denominate *NonPieno*, *attesaB* e *attesaC*. Inoltre il buffer è indirizzato dalle variabili *testa* (valore iniziale 0) che identifica la posizione nella quale il thread A inserisce i messaggi, *codaB* e *codaC* (valore iniziale 0 per entrambe) che sono i puntatori agli elementi dai quali vengono estratti i messaggi dei thread B e C, rispettivamente, e le variabili *n_elemB* e *n_elemC* (valore iniziale 0 per entrambe) che indicano il numero di messaggi da leggere per i thread B e C, rispettivamente.
L'interazione tra i thread A e B si svolge nel modo seguente:

Thread A:
```
    while (true) {
    <produce  v>
    <verifica la condizione (max ( n_elemB , n_elemC ) == n) ed eventualmente si sospende>
    buf[testa] =v;
    testa = (testa + 1) mod n;
    n_elemB  ++;
    n_elemC ++;
    riattiva (eventualmente) il thread B
    riattiva (eventualmente) il thread C
        }
```

Thread B:
```
    while (true) {
    <verifica la condizione (n_elemB == 0) ed eventualmente si
    sospende>
    v=buf[codaB];
    codaB = (codaB + 1) mod n;
    nelemB --;
   <verifica la condizione max ( n_elemB , n_elemC ) < n  ed
   eventualmente riattiva  il thread A>
    <consuma v>
        }
```

Thread C:
```
     while (true) {
   <verifica la condizione (n_elemC == 0) ed eventualmente si
   sospende>
   v=buf[codaC];
   codaC = (codaC + 1) mod n;
   nelemC --;
   <verifica la condizione max ( n_elemB , n_elemC ) < n  ed
   eventualmente riattiva  il thread A>
   <consuma v>
        }
```

Inserire nel precedente schema le funzioni **pthread_mutex_lock**, **pthread_mutex_unlock**, **pthread_cond_wait**, **pthread_cond_signal** con i rispettivi argomenti.

SOLUZIONE

Thread A:
```
a.1)   while (true) {
a.2)   <produce  v>
a.3)   pthread_mutex_lock(&mux);
a.4)   while (max ( n_elemB , n_elemC ) == n) pthread_cond_wait(&NonPieno,&mux);
a.5)   buf[testa] =v;
a.6)   testa = (testa + 1) mod n;
a.7)   n_elemB  ++;
a.8)   n_elemC ++;
a.9)   pthread_cond_signal(&attesaB);
a.10)  pthread_cond_signal(&attesaC);
a.11)  pthread_mutex_unlock(&mux);
       }
```

Thread B:
```
b.1)   while (true) {
b.2)   pthread_mutex_lock(&mux);
b.3)   if (n_elemB == 0) pthread_cond_wait(&attesaB , &mux);
b.4)   v=buf[codaB];
b.5)   codaB = (codaB + 1) mod n;
b.6)   nelemB --;
b.7)   if (max ( n_elemB , n_elemC ) < n) pthread_cond_signal(&NonPieno);
b.8)   pthread_mutex_unlock(&mux);
b.9)   <consuma v>
       }
```

Thread C:

```
c.1)  while (true) {
c.2)  pthread_mutex_lock(&mux);
c.3)  if (n_elemC == 0) pthread_cond_wait(&attesaC , &mux);
c.4)  v=buf[codaC];
c.5)  codaC = (codaC + 1) mod n;
c.6)  nelemC --;
c.7)  if (max ( n_elemB , n_elemC ) < n)  pthread_cond_signal(&NonPieno);
c.8)  pthread_mutex_unlock(&mux);
c.9)  <consuma v>
      }
```

ESERCIZIO A-2 (4 punti)

Si consideri un processore che dispone dei seguenti registri:
- i registri speciali PC (program counter) e PS (program status)
- un banco di registri utilizzato in stato utente che comprende i registri generali R1, R2 e lo stack pointer SP
- un banco di registri utilizzato in stato supervisore che comprende i registri generali R1', R2' e lo stack pointer SP'

L'hardware provvede al salvataggio dei registri PC e PS nello stack del nucleo quando riconosce un'interruzione e al loro ripristino dallo stack del nucleo quando esegue un'istruzione IRET.
Tutti i processi hanno uguale priorità e il sistema gestisce il processore con politica Round Robin. Al tempo t, quando sono presenti nel sistema (tra gli altri) il processo A, in stato di esecuzione, e il processo B che è bloccato in attesa della lettura di un blocco di dati dal disco, arriva un'interruzione dal disco che segnala il completamento dell'operazione di lettura.
L'interruzione è riconosciuta dal processore che individua il vettore di interruzione corrispondente, contenente i valori PC=0F01 e PS=A3AF.
Immediatamente dopo il tempo *t*, quando l'interruzione generata dal disco viene riconosciuta, i registri del processore, i descrittori di A e B e lo stack del nucleo hanno i contenuti mostrati in figura.
L'interruzione determina l'intervento del nucleo, che gestisce l'interruzione. Supponendo che il processo A non esaurisca il proprio quanto di tempo, si chiede:

a) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell' istruzione eseguita subito dopo il riconoscimento dell'interruzione;
b) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione IRET con la quale termina la gestione dell'interruzione;
c) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. GENERALI	
Stato	Esec		Stato	Bloccato		……	……		SP	3EAA
PC	23FA		PC	409A		3F0A			R1	07AA
PS	16F2		PS	16F2		3F09			R2	08AA
SP	3EFA		SP	500B		3F08				
R1	AAA1		R1	BBB6		3F07				
R2	AAA2		R2	BBB5		3F06			REG. NUCLEO	
						3F05			SP'	3F0A
						3F04			R1'	0000
PROCESSORE: Registri speciali									R2'	0001
PC	240A		PS	16F2						

SISTEMI OPERATIVI, CORSI A e B Prova del 10/9/2008 CORSO |A| |B|

SOLUZIONE

a) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell' istruzione eseguita subito dopo il riconoscimento dell'interruzione:

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. GENERALI	
Stato	Esec		Stato	Bloccato		……	……		SP	invariato
PC	invariato		PC	invariato		3F0A	240A		R1	invariato
PS	invariato		PS	invariato		3F09	16F2		R2	invariato
SP	invariato		SP	invariato		3F08				
R1	invariato		R1	invariato		3F07				
R2	invariato		R2	invariato		3F06			REG. NUCLEO	
						3F05			SP'	3F08
						3F04			R1'	invariato
PROCESSORE: Registri speciali									R2'	invariato
PC	0F01		PS	A3AF						

b) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione IRET con la quale termina la gestione dell'interruzione:

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. GENERALI	
Stato	Esec		Stato	Pronto		……	……		SP	invariato
PC	invariato		PC	invariato		3F0A	invariato		R1	invariato
PS	invariato		PS	invariato		3F09	invariato		R2	invariato
SP	invariato		SP	invariato		3F08				
R1	invariato		R1	invariato		3F07				
R2	invariato		R2	invariato		3F06			REG. NUCLEO	
						3F05			SP'	invariato
						3F04			R1'	??
PROCESSORE: Registri speciali									R2'	??
PC	0F01+ ??		PS	invariato						

c) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET:

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. GENERALI	
Stato	Esec		Stato	Pronto		……	……		SP	invariato
PC	invariato		PC	invariato		3F0A			R1	invariato
PS	invariato		PS	invariato		3F09			R2	invariato
SP	invariato		SP	invariato		3F08				
R1	invariato		R1	invariato		3F07				
R2	invariato		R2	invariato		3F06			REG. NUCLEO	
						3F05			SP'	3F0A
						3F04			R1'	??
PROCESSORE: Registri speciali									R2'	??
PC	240A		PS	16F2						

ESERCIZIO A-3 (3 punti)

Un sistema con 5 processi A, B, C, D, E e risorse dei tipi R1, R2, R3 rispettivamente di molteplicità [5, 4, 7] adotta nei confronti dello stallo la politica di riconoscimento ed eliminazione. Al tempo *t* il sistema raggiunge lo stato mostrato nella seguente tabella.

Assegnazione attuale			
	R1	R2	R3
A	2	1	
B			2
C		1	3
D	1		1
E	2	1	

Esigenza Massima (assegnazione attuale + esigenza residua)			
	R1	R2	R3
A	2	3	6
B	1	2	3
C		4	4
D	3	2	4
E	2	1	1

Esigenza residua (tiene conto dell' assgnazione attuale			
	R1	R2	R3
A	0	2	6
B	1	2	1
C	0	3	1
D	2	2	3
E	0	0	1

Molteplicità		
R1	R2	R3
5	4	7

Disponibilità		
0	1	1

Dire se il sistema è in stallo. Inoltre:

- In caso affermativo dire quali processi sono in stallo e quale processo è possibile far terminare per rimuovere lo stallo. Motivare la risposta.
- In caso negativo dire se l'assegnazione di una risorsa di tipo R3 al processo E porta il sistema in stallo

SOLUZIONE

Verifica dello stallo:
1) Il processo E può terminare
   La disponibilità di {R1, R2, R3} diviene { 2,2,1 }
2) Il processo B può terminare
   La disponibilità di {R1, R2, R3} diviene { 2,2,3 }
3) Il processo D può terminare
   La disponibilità di {R1, R2, R3} diviene { 3,2,4 }

I processi A e C non possono terminare, quindi il sistema è in stallo.

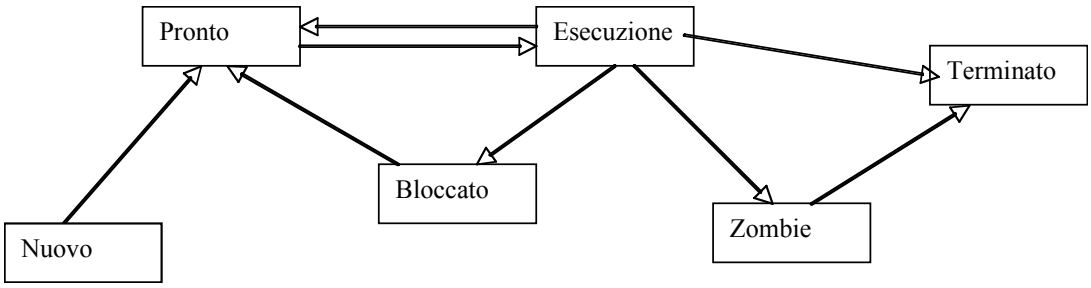
I processi in stallo sono A e C.

Per rimuovere lo stallo è possibile sopprimere uno qualsiasi dei processi A e C. Infatti se si sopprime A i processi E, B e D possono terminare (come si è verificato in precedenza) e a questo punto C ha a disposizione tutte le risorse del sistema per terminare. In modo simile, se si sopprime C, oltre ai processi E, B e D anche A può terminare.

ESERCIZIO A.4 (2 punti)

Ricordando che nei sistemi UNIX i processi possono assumere gli stati *Nuovo, Pronto, Esecuzione, Bloccato, Terminato* e *Zombie,* costruire il diagramma dello possibili transizioni di stato, collegando i nodi corrispondenti agli stati con opportuni archi orientati.

SOLUZIONE



ESERCIZIO A.5 (2 punti)

Dire quali delle seguenti primitive Unix possono causare la sospensione o la terminazione del processo invocante.

SOLUZIONE

	Può far terminare il processo in esecuzione?	Può provocare la sospensione del processo in esecuzione?
exit	SI	NO
wait	NO	SI
signal	NO	NO
exec	NO	NO
kill	NO *	NO
pipe	NO	NO

***** Se si esclude il caso in cui il destinatario del segnale è lo stesso processo in esecuzione

ESERCIZIO B-1 (4 punti)

Un sistema simile a Unix ha a disposizione 30 Mbyte di memoria che gestisce con segmentazione, caricamento in partizioni variabili e swapping. Il sistema operativo è allocato permanentemente in una partizione con origine 0 e lunghezza 5 Mbyte. Il codice condiviso tra due o più processi è caricato in memoria in una sola copia.
Al tempo t sono presenti in memoria due processi A e B che occupano le seguenti partizioni:

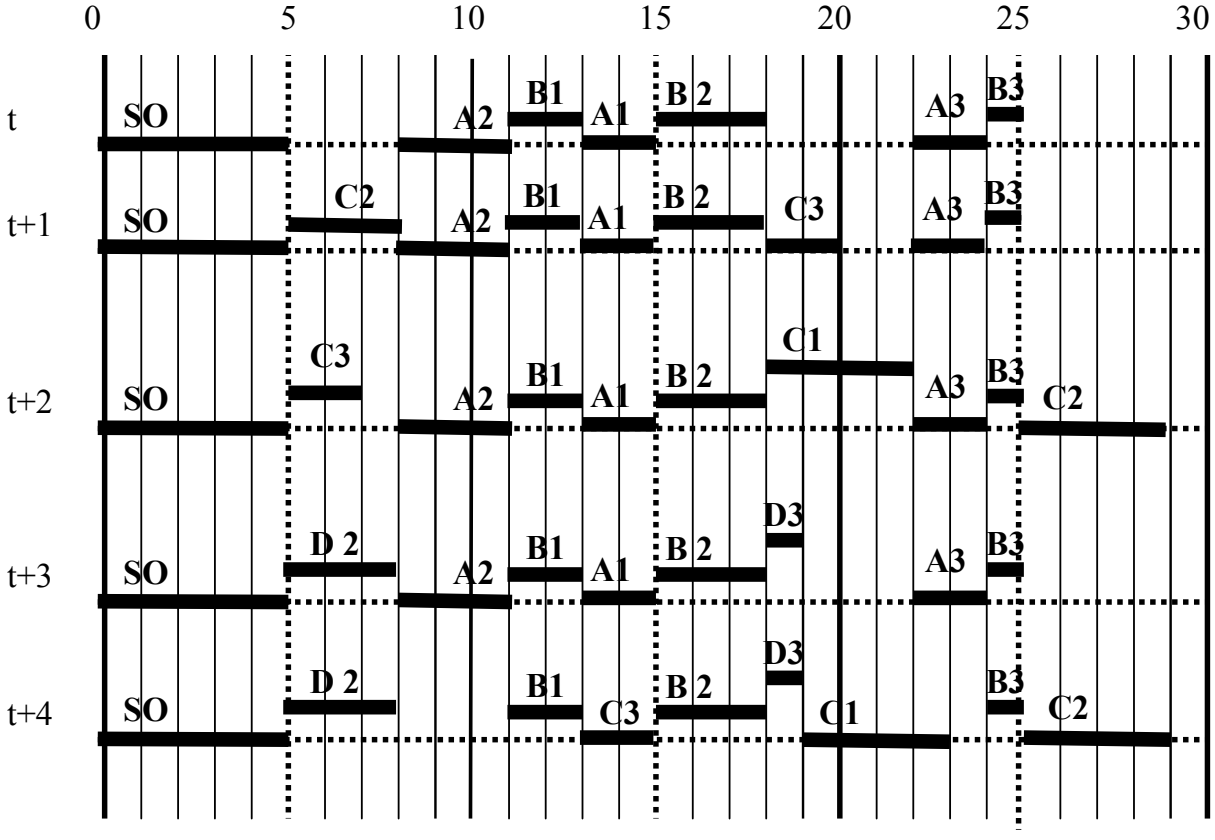
- il processo A, occupa tre partizioni:
  - una partizione A1 con origine 13 e lunghezza 2 per il codice
  - una partizione A2 con origine 8 e lunghezza 3 per i dati
  - una partizione A3 con origine 22 e lunghezza 2 per lo stack
- il processo B, che occupa tre partizioni:
  - una partizione B1 con origine 11 e lunghezza 2 per il codice
  - una partizione B2 con origine 15 e lunghezza 3 per i dati
  - una partizione B3 con origine 24 e lunghezza 1 per lo stack

Il gestore della memoria adotta politica *best fit* per l'assegnazione delle partizioni. Inoltre, se il sistema non ha spazio sufficiente per allocare un nuovo processo, il gestore della memoria effettua lo swap-out in memoria secondaria di uno o più processi, in ordine decrescente di occupazione complessiva per dati e stack, finché non libera spazio sufficiente per allocare il nuovo processo. I processi scaricati in memoria secondaria vengono caricati in memoria principale (swap-in) quando la terminazione di uno o più processi libera uno spazio sufficiente.
Negli istanti successivi al tempo t avvengono (solo ed esclusivamente) i seguenti eventi rilevanti per il gestore della memoria:

1. tempo t+1: A esegue una fork() che crea il processo figlio C
2. tempo t+2: C esegue una exec() che rilascia le partizioni allocate e successivamente alloca tre partizioni: una per il codice di ampiezza 4, una per i dati di ampiezza 4 e una per lo stack di ampiezza 2
3. tempo t+3: B esegue una fork() che crea il processo D
4. tempo t+4: A esegue una exit() senza passare dallo stato zombie

Utilizzando il grafico sotto riportato, mostrare come evolve l'occupazione della memoria ai tempi t+1, t+2, t+3 e t+4.

SOLUZIONE



ESERCIZIO B-2 (4 punti)

Si consideri un sistema che gestisce la memoria utilizzando un algoritmo di sostituzione LRU locale e che attribuisce a ogni processo un numero `MaxBlocchi` di blocchi a disposizione del working set. Questo numero è definito dinamicamente e viene incrementato o decrementato dal processo di sistema `WorkingSetManager`, che interviene periodicamente e applica la seguente politica:

- appartengono al working set di un processo le pagine il cui *tempo virtuale di ultimo riferimento* è maggiore o uguale di *t - 10* dove *t* è il valore attuale del *tempo virtuale* del processo
- se il numero *PagineInEccesso* delle pagine non appartenenti al working set che sono caricate in memoria è maggiore di 1, si scaricano *PagineInEccesso*- 1 pagine applicando l'algoritmo LRU e si decrementa di 1 il valore di `MaxBlocchi`;
- se tutte le pagine caricate in memoria appartengono al working set,si incrementa di 1 il valore di `MaxBlocchi`;

Al verificarsi di un errore di pagina si ha il seguente comportamento:

- se il numero di pagine attualmente caricate è minore di `MaxBlocchi`, la pagina riferita viene caricata assegnando il primo dei blocchi dalla lista `BlocchiDisponibili`.
- se il numero di pagine attualmente caricate è uguale a `MaxBlocchi`, viene scaricata la pagina prescelta dall'algoritmo di sostituzione e la pagina riferita viene caricata al suo posto.

A un certo istante, quando il tempo virtuale del processo A è t=100, il valore di `MaxBlocchi` assegnato al processo A è `MaxBlocchi= 6` la tabella delle pagine del processo A è la seguente:

Pagina	Blocco	*Tempo virtuale di ultimo riferimento*
0	12	93
1	13	88
2	-	
3	14	81
4	15	78
5	-	
6	16	63
7	-	
8	17	99
9	-	
Processo A		

A partire da questo istante interviene il processo `WorkingSetManager`, che applica la politica sopra definita . Durante l'esecuzione di `WorkingSetManager` il tempo virtuale dei processi rimane invariato.
Successivamente il processo A avanza ed esegue i seguenti riferimenti alla memoria:

- al tempo 101 riferisce la pagina 3
- al tempo 102 riferisce la pagina 0
- al tempo 103 riferisce la pagina 1

Si chiede:

a) Le pagine eventualmente scaricate da `WorkingSetManager;` il valore di `MaxBlocchi` assegnato al processo A quando termina l'esecuzione di `WorkingSetManager`; la configurazione della tabella delle pagine del processo A quando termina l'esecuzione di `WorkingSetManager.`
b) i caricamenti e gli eventuali scaricamenti di pagine eseguiti nei tempi 101,102 e 103; il valore di `MaxBlocchi` assegnato al processo A al tempo 103; la configurazione della tabella delle pagine del processo A al tempo 103

Si noti che il contenuto della lista `BlocchiDisponibili` non è rilevante per la risoluzione di questo esercizio.

SOLUZIONE

a-1) Pagine scaricate dal `WorkingSetManager` : 6, 4, 3

a-2) Valore di `MaxBlocchi` assegnato al processo A quando termina l'esecuzione di `WorkingSetManager:` 3

b-1) Caricamenti e gli eventuali scaricamenti di pagine eseguiti nei tempi 101,102 e 103

- tempo 101: rimossa la pagina 1 e caricata la pagina 3 nel blocco 88
- tempo 102: aggiornato l'istante di ultimo riferimento della pagina 0
- tempo 103: rimossa la pagina 8 e caricata la pagina 1 nel blocco 17

b-2) Valore di `MaxBlocchi` assegnato al processo A al tempo 103: `MaxBlocchi=` 3

a-3) Configurazione della tabella delle pagine del processo A quando termina l'esecuzione di `WorkingSetManager:`

Pagina	Blocco	*Tempo virtuale di ultimo riferimento*
0	12	93
1	13	88
2	-	
3	-	
4	-	
5	-	
6	-	
7	-	
8	17	99
9	-	
Processo A		

b-3) Configurazione della tabella delle pagine del processo A al tempo 103 :

Pagina	Blocco	*Tempo virtuale di ultimo riferimento*
0	12	102
1	17	103
2	-	
3	13	101
4	-	
5	-	
6	-	
7	-	
8	-	
9	-	
Processo A		

ESERCIZIO B-3 (3 punti)

Un disco di 200 GByte ospita un file system basato su i-node che utilizza blocchi di 4 Kbyte e puntatori di 32 bit, e riserva 10.000 blocchi per la memorizzazione dell'i-list. Ogni i-node occupa 128 bytes e include 10 puntatori diretti e tre puntatori indiretti: uno indiretto singolo, uno indiretto doppio e uno indiretto triplo
Si chiede:

1. Il massimo numero di blocchi indirizzabili tramite un puntatore indiretto singolo
2. Il massimo numero di blocchi indirizzabili tramite un puntatore indiretto doppio
3. Il massimo numero di blocchi indirizzabili tramite un puntatore indiretto triplo
4. Il massimo numero di blocchi indirizzabili da un i-node
5. la massima dimensione di un file
6. il massimo numero di files memorizzabili nel disco

SOLUZIONE

1. Massimo numero di blocchi indirizzabili tramite un puntatore indiretto singolo: 4Kbyte / 4 bytes = 1024
2. Massimo numero di blocchi indirizzabili tramite un puntatore indiretto doppio: 1024 ^ 2
3. Massimo numero di blocchi indirizzabili tramite un puntatore indiretto triplo: 1024 ^3
4. Massimo numero di blocchi indirizzabili da un i-node: 10+ 1024 + 1024^2 + 1024 ^ 3= 10+ 2^10++ 2^20+ 2^30
5. Massima dimensione di un file: 200 GB, pari alla dimensione del disco.
   infatti la capacità di indirizzamento di un i-node è superiore a 2^30 blocchi, pari 4* 2^40 byte , ed eccede quindi la capacità del disco.
6. Massimo numero di file memorizzabili nel disco: 10.000 * 4KBytes/128 bytes = 320.000

ESERCIZIO B.4 (2 punti)

Dato un file system FAT con blocchi di 4KB (4096 byte) e il seguente frammento di FAT, dire in quali blocchi fisici sono collocati i seguenti byte:

1. byte 6758 del file che inizia al blocco 48
2. byte 8192 del file che inizia al blocco 49

Frammento di FAT :

blocco fisico	contenuto dell'elemento corrispondente della FAT
…..	
45	46
46	52
47	51
48	47
49	50
50	45
51	55
52	56
…..	

SOLUZIONE

1) il byte 6758 è nel blocco logico 1 del file, quindi il blocco fisico è 47
2) il byte 8192 è nel blocco logico 2 del file, quindi il blocco fisico è 45

ESERCIZIO B.5 (2 punti)

Si consideri il seguente RAID di livello 4 composto di 4 dischi con il seguente contenuto:

	Disco 1	Disco 2	Disco 3	Disco 4
Strip 0	1000	1101	1110	1011
Strip 1	0001	1000	0001	1000
Strip 2	1111	0000	0011	1100
…				

Il disco 4 è ridondante e contiene la parità dei dischi non ridondanti.
Si considerino le seguenti operazioni:

1. scrittura della configurazione `1000` nella strip 1 del disco 1
2. ricostruzione del contenuto della Strip 0 del disco 3, in caso di guasto del disco 3

Per ciascuna di queste operazioni si chiede:

a) quali operazioni sui singoli dischi fisici sono necessarie per eseguirla
b) il risultato dell'operazione

SOLUZIONE

Operazione 1

a) Operazioni sui singoli dischi fisici necessarie per eseguirla

- leggere la strip 1 del disco 1
- leggere la strip 1 del disco 4
- scrivere la configurazione 1000 nella strip 1 del disco 1
- scrivere il nuovo valore della parità nella strip 1 del disco 4

b) risultato dell'operazione:

	Disco 1	Disco 2	Disco 3	Disco 4
Strip 1	1000	1000	0001	0001

Operazione 2

a) Operazioni sui singoli dischi fisici necessarie per eseguirla:

- leggere la strip 0 del disco 1
- leggere la strip 0 del disco 2
- leggere la strip 0 del disco 4
- scrivere la parità delle strip 0 dei dischi 1, 2 e 4 nella strip 0 del disco 3

b) risultato dell'operazione:

	Disco 1	Disco 2	Disco 3	Disco 4
Strip 0	1000	1101	1110	1011