SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

ESERCIZIO A-1 (4 punti)

Si consideri un processore che dispone dei seguenti registri:

- i registri speciali PC (program counter), PS (program status) e SP(stack pointer);
- i registri generali R1 e R2, utilizzati sia nello stato utente, sia nello stato supervisore

Inoltre il sistema riserva un'area di memoria (appartenente al nucleo) per il vettore di interruzione e per lo stack del nucleo. Il vettore di interruzione contiene, per ogni interruzione, un indirizzo nell'area di memoria del nucleo e una parola di stato.

- Al riconoscimento di un'interruzione, l'hardware salva i registri generali e speciali nello stack del nucleo e salta alla funzione di servizio dell'interruzione.
- Nella fase di esecuzione dell'istruzione IRET, l'hardware ripristina **tutti** i registri dallo stack del nucleo.

Si assuma che il sistema operativo sia di tipo Unix e che in un dato istante di tempo sia presente il processo Q (in stato *pronto*) e il suo processo figlio P (in stato *esecuzione*). Nel medesimo istante il processo P esegue l'istruzione SVC per invocare la primitiva exit(). La primitiva eseguita con questo meccanismo mette in esecuzione il processo Q.

All'istante in cui viene estratta l'istruzione SVC, i registri del processore, i descrittori di P e Q e lo stack del nucleo hanno i contenuti mostrati in figura, che mostra anche il contenuto dell'elemento del vettore di interruzione associato alle interruzioni da programma.

Si chiede:

1) Lo stato del processore dopo l'esecuzione della SVC;
2) Il contenuto dei registri, dei descrittori e dello stack del nucleo dopo l'esecuzione della SVC;
3) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito prima dell'esecuzione della IRET;
4) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito dopo l'esecuzione della IRET;
5) Lo stato del processore dopo l'esecuzione della IRET

Descrittore di P	
Stato	esecuzione
PC	A000
PS	B000
SP	C000
R1	D000
R2	E000

Descrittore di Q	
Stato	pronto
PC	1000
PS	2000
SP	3000
R1	4000
R2	5000

Stack del nucleo	
1000	ABCD
1001	
1002	
1003	
1004	
1005	
1006	

Registri	
PC	AB00
PS	B000
SP	C111
R1	D222
R2	E333

INDIRIZZO	0F9A
PAROLA DI STATO	0AF6
Vettore di interruzione	

Stack pointer del nucleo	1001

SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

SOLUZIONE

1) Stato del processore: SUPERVISORE

2) Il contenuto dei registri, dei descrittori e dello stack del nucleo dopo l'esecuzione della SVC;

Descrittore di P	
Stato	esecuzione
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Descrittore di Q	
Stato	pronto
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Stack del nucleo	
1000	ABCD
1001	AB00
1002	B000
1003	C111
1004	D222
1005	E333
1006	

Registri	
PC	0F9A
PS	0AF6
SP	C111
R1	D222
R2	E333

INDIRIZZO	0F9A
PAROLA DI STATO	0AF6
Vettore di interruzione	

Stack pointer del nucleo	1006

3) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito prima dell'esecuzione della IRET;

Descrittore di P:	
Stato	Zombie
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Descrittore di Q	
Stato	esecuzione
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Stack del nucleo	
1000	ABCD
1001	1000
1002	2000
1003	3000
1004	4000
1005	5000
1006	

Registri	
PC	0F9A+ ?
PS	0AF6
SP	C111
R1	?
R2	?

INDIRIZZO	0F9A
PAROLA DI STATO	0AF6
Vettore di interruzione	

Stack pointer del nucleo	1006

4) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito dopo l'esecuzione della IRET;

Descrittore di P:	
Stato	Zombie
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Descrittore di Q	
Stato	esecuzione
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Stack del nucleo	
1000	ABCD
1001	
1002	
1003	
1004	
1005	
1006	

Registri	
PC	1000
PS	2000
SP	3000
R1	4000
R2	5000

INDIRIZZO	0F9A
PAROLA DI STATO	0AF6
Vettore di interruzione	

Stack pointer del nucleo	1001

5) Lo stato del processore dopo l'esecuzione della IRET: UTENTE

SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

ESERCIZIO A-2 (4 punti)

In un sistema operativo che realizza i threads a livello utente, i thread *Produttore* e *Consumatore* del processo P cooperano per inserire ed estrarre messaggi di lunghezza unitaria da un buffer circolare della capacità di *N* messaggi. Per il controllo del buffer, i due thread condividono le variabili *PrimoMessaggio* (intero nell'intervallo *[0, N)*) e *MessaggiNelBuffer* (intero nell'intervallo *[0, N]*) e, per la mutua esclusione, utilizzano i protocolli *lock(&K)* e *unlock(&K)*. L'accesso alla mutua esclusione è consentito quando la chiave binaria *K* ha il valore 1. L'espansione assembler dei protocolli *lock* e *unlock* è la seguente:

```
lock(&K)                                        unlock&K)
                                                    MOV K, #1  //scrive1 in K//
loop        TSL K, R1 //copia K in R1 e scrive 0 in K//
            JNZ R1, Fine //esegue il salto se R1≠0//
            CALL thread_yield
            JMP loop  //salto incondizionato//
Fine        NOP //No OPeration//
```

I due thread alternano tra lo stato di esecuzione e quello di pronto. La transizione di un thread nello stato di pronto, e la conseguente transizione dell'altro thread in stato di esecuzione, avviene quando viene eseguita l'operazione *thread_yield* oppure quando termina il quanto di tempo, che ha il valore di *25 μsec.*

Il codice eseguito dai due thread è riportato nelle tabelle seguenti. Il tempo di esecuzione di ogni riga di comando è indicato in tabella.

	Thread Produttore
P0 (1 μsec)	**while** true {
P1 (10 μsec)	Nuovo= DefinisceMessaggioCriptato;
P2 (1 μsec)	**lock** (&K);
P3 (1 μsec)	**while** MessaggiNelBuffer== N {
P4 (1 μsec)	**unlock**(&K);
P5 (1 μsec)	**thread_yield**
P6 (1 μsec)	**lock**(&k); }
P7 (1 μsec)	Buffer[PrimoMessaggio+MessaggiNelBuffer **mod** N]= Nuovo;
P8 (1 μsec)	MessaggiNelBuffer ++;
P9 (1 μsec)	**unlock**(&K) }

	Thread Consumatore
C0 (1 μsec)	**while** true {
C1 (1 μsec)	**lock** (&K);
C2 (1 μsec)	**while** MessaggiNelBuffer==0 {
C3 (1 μsec)	**unlock**(&K);
C4 (1 μsec)	**thread_yield**
C5 (1 μsec)	**lock**(&K); }
C6 (1 μsec)	MessaggioCriptato= Buffer[PrimoMessaggio];
C7 (10 μsec)	MessaggioRicevuto= Decripta(MessaggioCriptato);
C8 (1 μsec)	PrimoMessaggio =PrimoMessaggio ++ **mod** N;
C9 (1 μsec)	MessaggiNelBuffer - - ;
C10(1 μsec)	**unlock**(&K);
C11 (10 μsec)	IniziaElaborazione(MessaggioRicevuto)
C12 (1 μsec)	**thread_yield**
C13 (10 μsec)	CompletaElaborazione(MessaggioRicevuto) }

Al tempo *1* si ha *MessaggiNelBuffer== N* (e quindi il buffer è pieno) e *K= 1*, e il thread *Produttore* passa in stato di esecuzione a partire dalla linea di comando P2, disponendo di un intero quanto di tempo. Allo stesso tempo il thread *Consumatore* passa in stato di pronto, avendo eseguito la riga di comando *C12.*

Si chiede a quale tempo il thread *Produttore* completa il deposito di un messaggio nel buffer, tenendo presente che l'operazione si considera completata dopo l'esecuzione della riga di comando P9. Si chiede anche di compilare la tabella riportata nello schema di soluzione, specificando per ogni riga le linee di comando eseguite da un thread, a partire dal suo passaggio in esecuzione e fino al successivo passaggio in stato di pronto (oppure, nell'ultima riga, fino al deposito del messaggio).

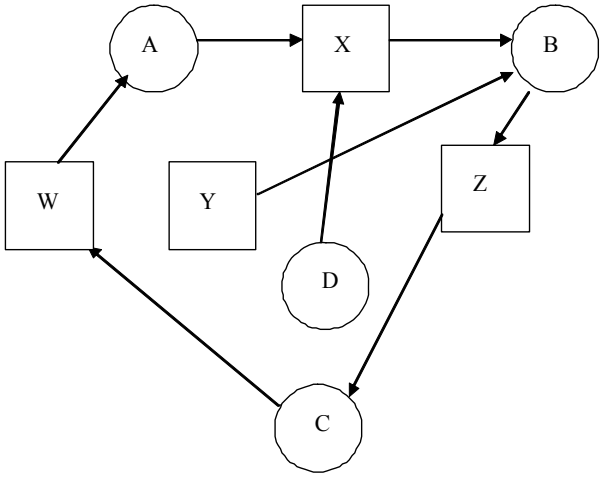
SOLUZIONE

Al tempo:	Passa in esecuzione il thread:	Evento che determina la transizione in stato di esecuzione:	Sequenza di linee di comando eseguite fino alla successiva transizione in stato di pronto oppure fino al completamento del deposito del messaggio	… che avviene al tempo
1	Produttore	Eseguito C12	P2, P3, P4, P5	5
5	Consumatore	Eseguito P5	C13, C0, C1, C2, C6, C7, C8	30
30	Produttore	Esaurito Quanto di Tempo	P6	31
31	Consumatore	Eseguito P6	C9, C10, C11, C12	44
44	Produttore	Eseguito C12	P6, P7, P8, P9	48

Il thread *Produttore* termina il deposito del messaggio al tempo 48.

SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

ESERCIZIO A- 3 (3 punti)
Un sistema con 3 processi (A,B,C,D) e quattro risorse singole (W, X, Y e Z) ha raggiunto lo stato descritto dal grafo seguente:



Domande:

a) Quali processi sono in stallo?
b) E' possibile che lo stato sia stato raggiunto con la seguente sequenza di richieste di assegnazione, a partire dallo stato iniziale in cui tutte le risorse sono disponibili?

1) A richiede W
2) B richiede X
3) A richiede X
4) B richiede Y
5) D richiede X
6) C richiede W
7) B richiede Z
8) C richiede Z

c) Cosa succede se viene forzata la terminazione del processo D?

SOLUZIONE

a) Sono tutti in stallo. Infatti i processi A,B e C sono in attesa circolare, il processo D è in attesa che B liberi X
b) L sequenza di richieste produce i seguenti risultati:

1) A richiede W Risultato: W assegnato ad A
2) B richiede X Risultato: X assegnato a B
3) A richiede X Risultato: A attende X
4) B richiede Y Risultato: Y assegnato a B
5) D richiede X Risultato: D attende X
6) C richiede W Risultato: C attende W
7) B richiede Z Risultato: Z assegnato a B
8) C richiede Z Risultato: Richiesta impossibile: C è sospeso.

La sequenza non porta il sistema nello stato rappresentato nel grafo, perchè 7) determina un'assegnazione invece che la sospensione del processo, e 8) è impossibile.
c) Se la terminazione del processo D è alternativa alla sequenza di richieste di cui al punto b), il sistema rimane in stallo perché alla terminazione di D non viene resa disponibile nessuna risorsa.

SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

ESERCIZIO A-4 (2 punti)

In un sistema operativo che realizza i threads a livello kernel, i thread T_{11} e T_{12} del processo P_1 e il thread T_{21} del processo P_2 cooperano scambiando messaggi attraverso un buffer di una sola cella. Per l'interazione si utilizzano i semafori BufferVuoto (inizializzato ad 1), BufferPieno (inizializzato a 0) e Mutex (inizializzato ad 1). I thread T11 e T12 si comportano da produttori eseguendo la funzione *deposito,* e il thread T21 si comporta da consumatore eseguendo la funzione *prelievo.* Le funzioni sono definite nel modo seguente:

```
void function deposito(&messaggio);
      wait(&BufferVuoto);
      wait(&Mutex);
      insert(messaggio);
      signal(&Mutex);
      signal(&BufferPieno);
      }
```

```
void function prelievo(&mess);
{     wait(&BufferPieno);
      wait(&Mutex);
      remove(&mess);
      signal(&Mutex);
      signal(&BufferVuoto);
      }
```

La soluzione è corretta? Motivare la risposta.

SOLUZIONE

La soluzione è errata.
Giustificazione della risposta: : Il thread P21 non può condividere il buffer con i thread T11 e T12, perchè appartiene a un processo diverso.

ESERCIZIO A-5 (2 punti)
Si consideri un sistema operativo dove i thread realizzati a livello kernel. e costituiscono l'unità di schedulazione . Si chiede quali dei seguenti dati sono contenuti nel descrittore di processo e quali nel descrittore di thread.

1	Immagine del contatore di programma
2	Puntatore alle aree dati in memoria
3	Immagine dei registri del processore
4	Stato (pronto/esecuzione/bloccato)
5	Puntatore alla tabella dei file aperti
6	Puntatore allo stack

SOLUZIONE

	Dato	Descr. Processo	Descr Thread
1	Immagine del contatore di programma	NO	SI
2	Puntatore alle aree dati in memoria	SI	NO
3	Immagine dei registri del processore	NO	SI
4	Stato (pronto/esecuzione/bloccato)	NO	SI
5	Puntatore alla tabella dei file aperti	SI	NO
6	Puntatore allo stack	NO	SI

SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

ESERCIZIO B-1 (4 punti)

Un sistema simile a Unix ha a disposizione 30 Mbyte di memoria, che gestisce con segmentazione e caricamento in partizioni variabili. Il sistema operativo è allocato permanentemente in una partizione con origine 0 e lunghezza 2 Mbyte. I segmenti codice condivisi tra più processi sono caricati in un'unica copia. Quando un processo si sospende, si esegue l'immediato *swap-out* dei segmenti non condivisi. Il successivo *swap-in* è deciso dal sistema operativo con un'opportuna politica. La politica adottata per l'assegnazione delle partizioni è la *Best Fit.*

Al tempo T0 sono caricati in memoria i processi A e B che occupano le seguenti partizioni:

- A1, con origine 11 e lunghezza 2, che contiene il codice del processo A;
- A2, con origine 7 e lunghezza 3, che contiene i dati del processo A;
- A3, con origine 22 e lunghezza 1, che contiene lo stack del processo A;
- B1, con origine 13 e lunghezza 3, che contiene il codice del processo B;
- B2, con origine 17 e lunghezza 2, , che contiene i dati del processo B;
- B3, con origine 26 e lunghezza 1, che contiene lo stack del processo B.

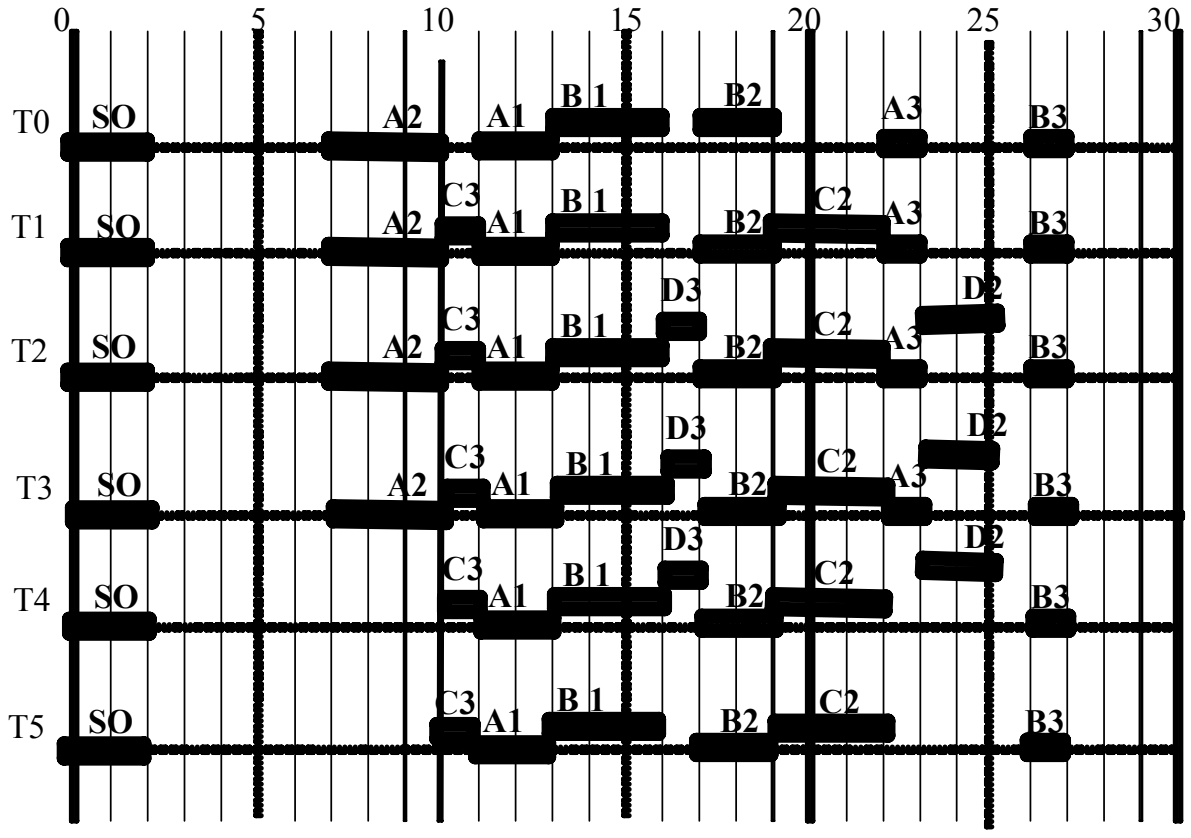
Dopo il tempo T0 avvengono i seguenti eventi rilevanti per la gestione della memoria:

- tempo T1: il processo A esegue una fork() che crea il processo figlio C;
- tempo T2: il processo B esegue una fork() che crea il processo figlio D;
- al tempo T3 il processo D esegue una exit();
- al tempo T4: il processo A esegue una waitpid();
- al tempo T5: il processo B esegue una waitpid();

Utilizzando il grafico sotto riportato, mostrare come evolve l'occupazione della memoria ai tempi T1, T2, T3, T4 e T5.

SOLUZIONE

- T1: Si assegnano al processo C le partizioni C2 e C3, copiando le partizioni A2 e A3;
- T2: Si assegnano al processo D le partizioni D2 e D3, copiando le partizioni B2 e B3;
- T3: il processo D passa nello stato di zombie e rimane caricato in memoria;
- T4: Il processo A si sospende e si esegue lo swapout dei segmenti A2 e A3. Il segmento A1 rimane caricato perché è condiviso;
- T5: Il processo D termina e si rilasciano le partizioni occupate dai segmenti D2 e D3.



SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

ESERCIZIO B-2 (4 punti)

Si consideri un sistema che gestisce la memoria utilizzando un algoritmo di sostituzione LRU locale e che attribuisce ad ogni processo un numero `MaxBlocchi` di blocchi a disposizione del working set. Appartengono al working set di un processo le pagine il cui *tempo virtuale di ultimo riferimento* è maggiore o uguale di *t - 100* dove *t* è il valore attuale del *tempo virtuale* del processo. Il numero `MaxBlocchi` è definito dinamicamente e viene incrementato o decrementato dal processo di sistema `WorkingSetManager`, che interviene periodicamente e applica la seguente politica:

- Scarica tutte le pagine in memoria che non appartengono al working set; sia *pag_rimosse* il numero di pagine rimosse;
- Se *pag_rimosse* == 0 incrementa `MaxBlocchi` di 1
- Se *pag_rimosse* > 1 si decrementa di *pag_rimosse*–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=1500, 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	200	1390
1		
2		
3	218	1318
4		
5	220	1444
6		
7	207	1401
8		
9	201	1299
10		
11	208	1210
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 1501 riferisce la pagina 3
- al tempo 1502 riferisce la pagina 6
- al tempo 1503 riferisce la pagina 5

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 1501,1502 e 1503; il valore di `MaxBlocchi` assegnato al processo A al tempo 1503; la configurazione della tabella delle pagine del processo A al tempo 1503

La lista `BlocchiDisponibili` contiene i blocchi: 300, 303, 335, …. Si assuma che i blocchi associati alle pagine rimosse vengano aggiunti in coda alla lista `Blocchi Disponibili.`

SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

SOLUZIONE

a-1) Pagine scaricate dal `WorkingSetManager` : 0,3,9,11

a-2) Valore di `MaxBlocchi` assegnato al processo A quando termina l'esecuzione di `WorkingSetManager:` 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		
1		
2		
3		
4		
5	220	1444
6		
7	207	1401
8		
9		
10		
11		
Processo A		

b-1) Caricamenti e gli eventuali scaricamenti di pagine eseguiti nei tempi 1501,1502 e 1503

- tempo 1501: caricata la pagina 3 nel blocco 300
- tempo 1502: rimossa la pagina 7; caricata la pagina 6 nel blocco 303
- tempo 1503: aggiornato l'istante di ultimo riferimento della pagina 5

b-2) Valore di `MaxBlocchi` assegnato al processo A al tempo 1503: `MaxBlocchi=` 3

b-3) Configurazione della tabella delle pagine del processo A al tempo 1503 :

Pagina	Blocco	*Tempo virtuale di ultimo riferimento*
0		
1		
2		
3	300	1501
4		
5	220	1503
6	303	1502
7		
8		
9		
10		
11		
Processo A		

ESERCIZIO B-3 (3 punti)

In un file system di tipo Unix i blocchi hanno dimensione di 1 KB e i puntatori ai blocchi sono codificati con 24 bit (3 byte). Gli i-node contengono 10 puntatori diretti, un puntatore indiretto singolo, un puntatore indiretto doppio e un puntatore indiretto triplo.
Calcolare:

1. Il numero di indirizzi contenuti in un blocco di indirizzamento indiretto
2. il massimo numero di byte indirizzabili con i soli puntatori diretti
3. il massimo numero di byte indirizzabili con i soli puntatori indiretti singoli
4. il massimo numero di byte indirizzabili con i soli puntatori indiretti doppi
5. il massimo numero di byte indirizzabili con i soli puntatori indiretti tripli
6. la massima dimensione (in blocchi) di un file

SOLUZIONE

1. I blocchi indiretti contengono $\lfloor 2^{10}/3 \rfloor = 341$ indirizzi
2. il massimo numero di byte indirizzabili con i soli puntatori diretti: 10 KB
3. il massimo numero di byte indirizzabili con i soli puntatori indiretti singoli: 341 KB
4. il massimo numero di byte indirizzabili con i soli puntatori indiretti doppi: 116.281 KB (circa 113,5 MB)
5. il massimo numero di byte indirizzabili con i soli puntatori indiretti tripli: circa 37,81 GB
6. la massima dimensione (in blocchi) di un file: circa 39,77 milioni di blocchi

SISTEMI OPERATIVI, CORSI A e B - Secondo Appello - 6/2/2009 CORSO |A| |B|

ESERCIZIO B-4 (2 punti)
Si consideri un disco con 3000 cilindri, 6 faccie e 400 settori per traccia. Completare la seguente tabella che associa ad ogni numero di settore una terna (cilindro, faccia, settore).

num. sett.	cilindro	faccia	settore
5.000.000			
981.001			
234.892			
	1.310	4	62
	807	1	262
	142	4	386

SOLUZIONE

num. sett.	cilindro	faccia	settore
5.000.000	2.083	2	0
981.001	408	4	201
234.892	97	5	92
3.145.662	1.310	4	62
1.937.462	807	1	262
342.786	142	4	386

ESERCIZIO B- 5 (2 punti)
Un disco RAID di livello 4 è composto da 5 dischi fisici, numerati da 0 a 4. I blocchi del disco virtuale V sono mappati nei dischi 0, 1, 2, 3: precisamente il blocco *b* del disco V è mappato nel blocco *b* ***div*** *4* del disco fisico di indice *b* ***mod*** *4*. Il disco 4 è ridondante e il suo blocco di indice *i* contiene la parità dei blocchi di indice *i* dei dischi 0, 1, 2, 3.
Ad un dato istante di tempo *t* il contenuto del blocco 5 del RAID è il seguente:

Disco 0	0	1	0	0	1	1	0	1
Disco 1	1	0	1	1	0	0	0	1
Disco 2	0	1	1	0	1	0	0	1
Disco 3	0	1	1	1	1	0	0	1
Disco 4	1	1	1	0	1	1	0	0

A partire dall'istante *t* avviene la seguente sequenza di eventi:

1. arriva una richiesta di scrittura sul blocco 5 del disco 2 del dato

1	0	0	1	0	1	0	0

2. si rileva l'indisponibilità del disco 0
3. arriva una richiesta di lettura del blocco 5 del disco 0

Si chiede:
- Il numero di accessi al disco (operazioni a basso livello di lettura/scrittura) necessari per eseguire l'operazione 1.
- Il contenuto del blocco 5 in seguito all'operazione 1.
- Il numero di accessi al disco necessari per eseguire l'operazione 3.

SOLUZIONE

Numero di accessi al disco (operazioni a basso livello di lettura/scrittura) necessari per eseguire l'operazione 1:
2 operazioni di lettura (per leggere il blocco 5 dei dischi 2 e 4)
2 operazioni di scrittura (per aggiornare il blocco 5 dei dischi 2 e 4)

Contenuto del blocco 5 in seguito all'operazione 1 (scrivere solo i bit modificati):

Disco 0								
Disco 1								
Disco 2	1	0	0	1	0	1	0	0
Disco 3								
Disco 4	0	0	0	1	0	0	0	1

Numero di accessi al disco necessari per eseguire l'operazione 3:
dato che il disco 0 è danneggiato è necessario leggere il blocco 5 dei dischi 1,2,3 e 4