SISTEMI OPERATIVI, CORSI A e B Prova del 5/2/2010 CORSO |A| |B|

OGNOME E NOME ..... MATRICOLA ..... Fila Posto

ESERCIZIO A-1 (4 punti)

In un sistema Unix sono presenti i Pthread `A, B1 e B2`. Il Pthread `A` effettua un ciclo nel quale produce una sequenza infinita di numeri interi compresi tra 1 e 100 e li memorizza in un buffer condiviso S, realizzato come una pila della capacità di 10 elementi. I Pthread B1 e B2 consumano i dati prodotti da A, in particolare il Pthread B1 consuma solo i dati minori o uguali di 50, mentre il pthread B2 consuma solo i dati maggiori di 50.
La pila S è gestita tramite il puntatore p_top tramite il quale vengono effettuate le operazioni di inserimento ed estrazione.
Oltre al buffer S, i thread condividono anche la variabile `elementi_nel_buffer,` inizializzata al valore 0.
Inoltre sono usati:

- La variabile `mutex` per gestire la mutua esclusione.
- La variabile di condizione `condA` per la sospensione del Pthread A
- La variabile di condizione `condB1` per la sospensione del Pthread B1
- La variabile di condizione `condB2` per la sospensione del Pthread B2

Si chiede di completare con le opportune operazioni di sincronizzazione della libreria p_thread lo pseudo codice dei PThread A e B1 sotto riportato. Il thread B2 svolge funzioni simmetriche rispetto a B1 e pertanto il suo pseudo codice non è richiesto.

SOLUZIONE

Pthread A:

```
while (true) {
      <produce un dato d compreso tra 1 e 100>
      Pthread_mutex_lock(&mutex);
      while (elementi_nel_buffer == 10) pthread_cond_wait(&condA,&mutex);
      p_top ++; S[p_top] = d; //inserisce l'elemento d in testa al buffer
      elementi_nel_buffer ++;
      if d<=50) pthread_cond_signal(&condB1)
      else pthread_cond_signal(&condB2);
      Pthread_mutex_unlock(&mutex);
}
```

Pthread B1:

```
while (true) {
      Pthread_mutex_lock(&mutex);
      while (elementi_nel_buffer == 0) pthread_cond_wait(&condB1,&mutex);
      while (S[p_top] <= 50) {
            x=S[p_top]; //preleva l'elemento x dalla coda del buffer
            p_top --
            elementi_nel_buffer --;
            pthread_cond_signal(&condA);
            }
      while (S[p_top] > 50)   {
            pthread_cond_signal(&condB2);
            pthread_cond_wait(&condB1,&mutex);
            }
      Pthread_mutex_unlock(&mutex);
      }
```

Pthread B2:

```
while (true) {
      Pthread_mutex_lock(&mutex);
      while (elementi_nel_buffer == 0) pthread_cond_wait(&condB1,&mutex);
      while (S[p_top] > 50) {
            x=S[p_top]; //preleva l'elemento x dalla coda del buffer
            p_top --
            elementi_nel_buffer --;
            pthread_cond_signal(&condA);
            }
      while (S[p_top] <= 50)  {
            pthread_cond_signal(&condB1);
            pthread_cond_wait(&condB1,&mutex);
            }
      Pthread_mutex_unlock(&mutex);
      }
}
```

ESERCIZIO A-2 (4 punti)

Si consideri un processore che dispone dei registri speciali PC (program counter) e PS (program status), di un banco di registri riservato allo stato utente, che comprende i registri generali R1, R2, R3, R4 e lo stack pointer SP, e di un ulteriore banco di registri riservato allo stato supervisore, che comprende i registri generali R'1, R'2, R'3, R'4 e lo stack pointer SP'. Quando riconosce un'interruzione, il processore salva i registri PC e PS nello stack del nucleo.

Al tempo t il processo Pi è in stato di esecuzione ed invoca una chiamata per la generazione di un nuovo processo. La chiamata di sistema opera sul modello della primitiva *fork()* di Unix e genera un nuovo processo clonando il processo invocante. La chiamata di sistema ha quindi come effetto quello di generare un nuovo processo Pj figlio di Pi. La primitiva inoltre prevede il prerilascio a favore del processo appena generato.

Immediatamente dopo il tempo *t*, quando la chiamata di sistema viene riconosciuta, i registri del processore, il descrittore di Pi e lo stack di Pi e del nucleo hanno i contenuti mostrati in tabella.

L'interruzione determina l'intervento del nucleo, che esegue la funzione di servizio. Supponendo che il vettore di interruzione associato alla chiamata di sistema sia 0425 e che la parola di stato del nucleo sia 275E, si chiede:

a) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione della prima istruzione della funzione di servizio;
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 funzione di servizio;
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.

DESCRITTORI			
Processo	Pi	Processo	∅
Stato	Esec		
PC	4000		
PS	16F2		
SP	F000		
R1	1112		
R2	2221		
R3	3331		
R4	4441		

PROCESSORE: Registri speciali			
PC	4100	PS	16F2

STACK DEL NUCLEO	
……	……
1016	AAAA
1015	
1014	
1013	
1012	
1011	
1010	

REG. STATO UTENTE	
SP	EF00
R1	1111
R2	2222
R3	3333
R4	4444
REG. STATO SUPERV	
SP'	1016
R'1	0000
R'2	0001
R'3	0002
R'4	0003

SISTEMI OPERATIVI, CORSI A e B Prova del 5/2/2010 CORSO |A| |B|

SOLUZIONE

a) Contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione della prima istruzione della funzione di servizio:

DESCRITTORI				
Processo	Pi		Processo	∅
Stato	Esec			
PC	Invariato			
PS	Invariato			
SP	Invariato			
R1	Invariato			
R2	Invariato			
R3	Invariato			
R4	Invariato			

PROCESSORE: Registri speciali				
PC	0425		PS	275E

STACK DEL NUCLEO	
……	……
1016	AAAA
1015	4100
1014	16F2
1013	
1012	
1011	
1010	

REG. STATO UTENTE	
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato
REG. STATO SUPERV	
SP'	1014
R'1	Invariato
R'2	Invariato
R'3	Invariato
R'4	Invariato

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 funzione di servizio;

DESCRITTORI				
Processo	Pi		Processo	Pj
Stato	Pronto		Stato	Esec
PC	4100		PC	4100
PS	16F2		PS	16F2
SP	EF00		SP	EF00
R1	1111		R1	1111
R2	2222		R2	2222
R3	3333		R3	3333
R4	4444		R4	4444

PROCESSORE: Registri speciali				
PC	0425		PS	275E

STACK DEL NUCLEO	
……	……
1016	AAAA
1015	4100
1014	16F2
1013	
1012	
1011	
1010	

REG. STATO UTENTE	
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato
REG. STATO SUPERV	
SP'	1014
R'1	Invariato
R'2	Invariato
R'3	Invariato
R'4	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

DESCRITTORI				
Processo	Pi		Processo	Pj
Stato	Pronto		Stato	Esec
PC	Invariato		PC	Invariato
PS	Invariato		PS	Invariato
SP	Invariato		SP	Invariato
R1	Invariato		R1	Invariato
R2	Invariato		R2	Invariato
R3	Invariato		R3	Invariato
R4	Invariato		R4	Invariato

PROCESSORE: Registri speciali				
PC	4100		PS	16F2

STACK DEL NUCLEO	
……	……
1016	AAAA
1015	
1014	
1013	
1012	
1011	
1010	

REG. STATO UTENTE	
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato
REG. STATO SUPERV	
SP'	1014
R'1	Invariato
R'2	Invariato
R'3	Invariato
R'4	Invariato

SISTEMI OPERATIVI, CORSI A e B Prova del 5/2/2010 CORSO |A| |B|

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 [8, 6, 3, 5] adotta nei confronti dello stallo la politica di prevenzione dinamica tramite l'algoritmo del banchiere. Ad un dato istante di tempo *t* lo stato del sistema è sicuro ed è mostrato nelle tabelle seguenti.

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

Esigenza Massima (valore iniziale)				
	R1	R2	R3	R4
A	3	0	1	1
B	3	5	2	3
C	3	3	3	3
D	1	4	1	2
E	1	3	0	3

Esigenza residua (considerata l'assegnazione attuale)				
	R1	R2	R3	R4
A	2	0	1	0
B	1	4	1	2
C	1	3	3	2
D	0	2	0	2
E	1	1	0	2

Molteplicità			
R1	R2	R3	R4
8	6	3	5

Disponibilità			
2	1	1	1

Sucessivamente avvengono in sequenza le seguenti richieste:
1) il processo B richiede una risorsa di tipo R4
2) il processo C richiede una risorsa di tipo R2

Per ognuna di queste richieste si chiede:
- se può essere accettata dal gestore che applica l'algoritno del banchiere;
- lo stato raggiunto dal processo richiedente in seguito all'applicazione dell'algoritmo del banchiere

SOLUZIONE

1) Stato raggiunto dopo l'ipotetica assegnazione di 1 istanza di R4 al processo B:

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

Esigenza Massima)				
	R1	R2	R3	R4
A	3	0	1	1
B	3	5	2	3
C	3	3	3	3
D	1	4	1	2
E	1	3	0	3

Esigenza residua				
	R1	R2	R3	R4
A	2	0	1	0
B	1	4	1	1
C	1	3	3	2
D	0	2	0	2
E	1	1	0	2

Molteplicità			
R1	R2	R3	R4
8	6	3	5

Disponibilità			
2	1	1	0

Il processo A può terminare. La disponibilità di {R1, R2, R3, R4} diviene {3,1,1,1}
Ma a questo punto nessun altro processo può terminare, quindi lo stato non è sicuro.
La richiesta non è accettata, B va in stato di bloccato.

2) Stato raggiunto dopo l'ipotetica assegnazione di 1 istanza di R2 al processo C:

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

Esigenza Massima				
	R1	R2	R3	R4
A	3	0	1	1
B	3	5	2	3
C	3	3	3	3
D	1	4	1	2
E	1	3	0	3

Esigenza residua				
	R1	R2	R3	R4
A	2	0	1	0
B	1	4	1	2
C	1	2	3	2
D	0	2	0	2
E	1	1	0	2

Molteplicità			
R1	R2	R3	R4
8	6	3	5

Disponibilità			
2	0	1	1

Il processo A può terminare. La disponibilità di {R1, R2, R3, R4} diviene {3,0,1,2}
Ma a questo punto nessun altro processo può terminare, quindi lo stato non è sicuro.
La richiesta non è accettata, C va in stato di bloccato.

SISTEMI OPERATIVI, CORSI A e B Prova del 5/2/2010 CORSO |A| |B|

ESERCIZIO A-4 (2 punti)

Si consideri un sistema che realizza i thread a livello utente. I thread di uno stesso processo si sincronizzano con le operazioni *thread_lock(k), thread_ unlock(k)* e *thread_yield.* L'operazione *thread_yield* può essere eseguita indipendentemente, ma viene eseguita anche all'interno dell'operazione *thread_lock(k)* quando risulta *k= 0.* L'operazione *thread_ unlokh(k)* si limita ad assegnare *k= 1.*

Inoltre i thread possono eseguire le normali primitive di comunicazione, gestite dal kernel, che possono determinare transizioni di stato dei processi. L'uso delle primitive di comunicazione non è rilevante in questo esercizio.

Al tempo *T0*, il processo P, che è in esecuzione, ha definito i thread thr1, thr2 e thr3. Thr 1 si trova in stato di esecuzione, mentre thr 2 e thr 3 sono pronti e sono inseriti in quest'ordine nella Coda Pronti. La gestione del processore per i thread avviene con politica *Round Robin.* A partire dal tempo *T0* si verificano, in successione, i seguenti eventi:

T1 il thread in esecuzione esegue *thread_lock(k)*
T2 il thread in esecuzione esegue *thread_yield.*
T3 il thread in esecuzione esegue *thread_ unlock(k)*
T4 il thread in esecuzione esegue *thread_yield.*
T5 il thread in esecuzione esegue *thread_ unlock(k)*
T6 il thread in esecuzione esegue *thread_yield.*
T7 il thread in esecuzione esegue *thread_lock(k).*

Si chiede di compilare la seguente tabella, riportando, per ogni tempo lo stato dei thread e il valore di *k* subito prima dell'evento successivo.

SOLUZIONE

Tempo	Evento	Dopo l'evento		
		Thread in esecuzione	Coda Pronti dei thread	Valore di *k*
T0		thr1	thr2 → thr3	0
T1	il thread in esecuzione esegue *thread_lock(k) (nota 1)*	thr2	thr3 → thr1	0
T2	il thread in esecuzione esegue *thread_yield.*	thr3	thr1 → thr2	0
T3	il thread in esecuzione esegue *thread_ unlock(k)*	thr3	thr1 → thr2	1
T4	il thread in esecuzione esegue *thread_yield.*	thr1	thr2 → thr3	1
T5	il thread in esecuzione esegue *thread_ unlock(k) (nota 2)*	thr1	thr2 → thr3	1
T6	il thread in esecuzione esegue *thread_yield.*	thr2	thr3 → thr1	1
T7	il thread in esecuzione esegue *thread_lock(k).*	thr2	thr3 → thr1	0

(nota 1): eseguendo la lock rilascia il processore (nota 2): ripete l'esecuzione della lock e la supera, ma poi esegue la unlock

ESERCIZIO A-5 (2 punti)

In un sistema UNIX il processore è gestito con una politica a code multiple, che combina la politica a priorità (con precedenza al processo con il minimo valore di priorità) e quella a quanti di tempo. Al tempo *t* sono presenti i processi P1 (con priorità 1), P2 (con priorità 2) e P3 (con priorità 3). Il processo P1 è in esecuzione e i processi P2 e P3 sono pronti.

A partire dal tempo T si verificano, in successione, i seguenti eventi:

T1: P1 esegue *fork,* generando il processo F1, che eredita la priorità dal padre. Prima di questa chiamata, P1 non ha eseguito altre fork
T2: Il processo in esecuzione esegue la chiamata *pipe,* che restituisce i descrittori *fd[0], fd[1])*
T3: Il processo in esecuzione esegue la chiamata *wait*
T4: Il processo in esecuzione esegue la chiamata *exit*
T5: Il processo in esecuzione esegue la chiamata *read(fd[0]&buff, 5)*
T6: Il processo in esecuzione esegue la chiamata *signal(s1, &funct)*

Si chiede di compilare la seguente tabella, che definisce l'evoluzione nel tempo dello stato di tutti i processi,supponendo che nell'intervallo di tempo compreso tra T0 e T6 non vengano eseguite altre chiamate di sistema, oltre quelle specificate.

SOLUZIONE

Tempo	Evento	In esecuzione	Coda Pronti Priorità 1	Coda Pronti Priorità 2	Coda Pronti Priorità 3
T0		P1	∅	P2	P3
T1	P1 esegue *fork;* genera F1	P1	F1	P2	P3
T2	Il processo in esecuzione esegue la chiamata *pipe (restituisce fd[0], fd[1])*	P1	F1	P2	P3
T3	Il processo in esecuzione esegue la chiamata *wait (1)*	F1	∅	P2	P3
T4	Il processo in esecuzione esegue la chiamata *exit (2)*	P1	∅	P2	P3
T5	Il processo in esecuzione esegue la chiamata *read(fd[0]&buff, 5) (3)*	P2	∅	∅	P3
T6	Il processo in esecuzione esegue la chiamata *signal(s1, &funct)*	P2	∅	∅	P3

(1) la wait sospende P1 (2) la exit riattiva P1 (3) il file è vuoto e la read sospende P1

SISTEMI OPERATIVI, CORSI A e B Prova del 5/2/2010 CORSO |A| |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=8 e minfree=1*, 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 18. I primi 2 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione.

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

Core Map al tempo *t*

Il *PageDaemon* interviene al tempo *t*+5 e successivamente ogni 10 msec. Ad ogni intervento, *PageDaemon* avanza per 1 msec occupando in modo esclusivo il processore e scarica fino a 7 pagine o, in alternativa, esegue lo *swapout* di un numero di processi sufficiente a portare il numero di blocchi liberi a un valore non minore di *lotsfree*. La selezione dei processi candidati allo *swapout* avviene in ordine di occupazione di memoria (per primi i processi che occupano più spazio) e, in caso di parità, in ordine alfabetico. Quando un processo subisce uno swap out tutte le sue pagine presenti in memoria vengono salvate sul disco, e saranno tutte ricaricate con un successivo swapin (non considerato in questo esercizio).
In caso di errori di pagina, i blocchi liberi vengono assegnati in ordine crescente di indice.
Considerare i seguenti eventi che si susseguono tra

1. dal tempo t al tempo t+5 avanzano i processi B e D che riferiscono nell'ordine le pagine: D2, D13, D3, D1, B7, B3, B4;
2. dal tempo t+5 al tempo t+6 avanza il processo *PageDaemon*;
3. dal tempo t+6 al tempo t+15 avanzano i processi A e B che riferiscono le pagine A1, A2, A4, A5, A6, A8, B1, B2, B6, B10
4. dal tempo t+15 al tempo t+16 avanza il processo *PageDaemon*;
5. dal tempo t+17 al tempo t+20 avanzano i processi A e B che riferiscono nell'ordine le pagine: A9, A5, A4, B3, B6, B10

Mostrare la configurazione della *CoreMap* ai tempi 5, 6, 15, 16 e 20

SOLUZIONE

																			↓			
Proc			D	C	C	A	C	**D**	D	C	D	**D**	B	A	A	B	A	C	D	C	B	B
Pag			1	0	1	4	6	**2**	7	3	10	**3**	4	9	14	3	16	11	13	12	7	8
Rif			1	1	1	0	1	**1**	1	1	1	**1**	1	0	1	**1**	0	0	1	1	**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

Core Map al tempo *t+5*

																			↓			
Proc						A							B	A	A	B	A				B	B
Pag						4							4	9	14	3	16				7	8
Rif						0							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

Core Map al tempo *t+6*

Eseguito swap out dei processi C e D

																			↓			
Proc			**A**	**A**	**A**	A	**A**	**A**	**B**	**B**	**B**	**B**	B	A	A	B	A				B	B
Pag			**1**	**2**	**5**	4	**6**	**8**	**1**	**2**	**6**	**10**	4	9	14	3	16				7	8
Rif			**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**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

Core Map al tempo *t+15*

				↓																		
Proc				A	A	A	A	A	B	B	B	B	B		A	B						
Pag				2	5	4	6	8	1	2	6	10	4		14	3						
Rif				**0**	**0**	**0**	**0**	**0**	**0**	**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

Core Map al tempo *t+16*

Rimosse le pagine: A9, A16, B7, B8, A1

				↓																		
Proc			**A**	A	A	A	A	A	B	B	B	B	B		A	B						
Pag			**9**	2	5	4	6	8	1	2	6	10	4		14	3						
Rif			**1**	0	**1**	**1**	0	0	0	0	**1**	**1**	0		0	**1**						
Blocco	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Core Map al tempo *t+20*

SISTEMI OPERATIVI, CORSI A e B Prova del 5/2/2010 CORSO |A| |B|

ESERCIZIO B-2 (4 punti)

Si consideri un File System FAT ospitato da un disco con capacità di 2^{16} *blocchi* di *1 kByte,* individuati da *indici di blocco fisico* compresi nell'intervallo $[0, 2^{16}-1]$. Il disco contiene la copia permanente della FAT, che occupa 2^{7} blocchi, con *indici di blocco fisico* compresi nell'intervallo $[0, 2^{7}-1]$. I rimanenti blocchi del disco, con *indici di blocco fisico* compresi nell'intervallo $[2^{7}, 2^{16}-1]$, sono riservati ai blocchi dati del File System. Si ignorano il blocco di *boot* e gli ulteriori blocchi riservati a dati di controllo, come la *bitmap*.
Gli elementi della FAT sono in corrispondenza uno a uno con i blocchi del disco, ma solo quelli corrispondenti a blocchi dati sono significativi. Ogni elemento significativo della FAT è individuato dall'*indice di blocco fisico* del corrispondente blocco dati, e contiene *l'indice di blocco fisico* di un altro blocco dati o il valore *nil.* Ogni elemento della FAT occupa un numero intero di byte, sufficienti a indirizzare tutti i blocchi del disco.
Su questo File System sono definiti, tra gli altri, il file *miofile,* che ha una lunghezza di 500 caratteri , e il file *tuofile,* la cui lunghezza corrente è di 20.400 caratteri. Il file *miofile* è identificato nella sua directory dalla coppia *(miofile, 128),* e il file *tuofile* è identificato nella sua directory dalla coppia *(tuofile, 25.000)*. Entrambi i file sono memorizzati sul disco come *run* di blocchi consecutivi di opportuna lunghezza.
Si chiede:

1. la lunghezza in byte di ciascun elemento della FAT;
2. il numero di elementi della FAT contenuti in ciascuno dei blocchi ad essa riservati;
3. la lunghezza in blocchi del file *miofile* e gli *indici di blocco fisico* dei blocchi dati assegnati a questo file;
4. la lunghezza in blocchi del file *tuofile* e gli *indici di blocco fisico* dei blocchi dati assegnati a questo file;
5. A quali elementi della FAT si deve accedere per leggere dal file *miofile* 100 caratteri a partire da quello di indice 200;
6. A quali elementi della FAT si deve accedere per leggere dal file *tuofile* 1.000 caratteri a partire da quello di indice 20.000
7. Su quali blocchi del disco sono distribuiti gli elementi della FAT di cui al punto precedente;

SOLUZIONE

1. Lunghezza di ciascun elemento della FAT: 16 bit → 2 byte;
2. Numero di elementi della FAT contenuti in ciascuno dei blocchi ad essa riservati: 1024/2= 512;
3. Lunghezza in blocchi del file *miofile* : 1 blocco
   indice di blocco fisico dell'unico blocco dati assegnato a questo file: 128;
4. lunghezza in blocchi del file *tuofile*: $\lceil 20400/\ 1024 \rceil = 20$
   indici di blocco fisico dei blocchi dati assegnati a questo file: 25.000 …. 25.019;
5. elementi della FAT ai quali si deve accedere per leggere dal file *miofile* 100 caratteri a partire dal carattere 200:
   - tutti i caratteri sono contenuti nel blocco dati di indice 128, indirizzato dalla directory: pertanto non si deve leggere nessun elemento della FAT.
6. Elementi della FAT ai quali si deve accedere per leggere dal file *tuofile* 1.000 caratteri a partire dal carattere 20.000:
   - il primo blocco dati del file *tuofile* ha indice 25.000;
   - il primo carattere da leggere appartiene al blocco logico 20000 **div** 1024= 19 ed è contenuto nel blocco fisico 25.019;
   - l'ultimo carattere da leggere appartiene al blocco logico (20000+ 1000- 1) **div** 1024= 20 ed è contenuto nel blocco dati 25.020;
   - per accedere al blocco dati 25.019 si devono leggere in sequenza gli elementi della FAT di indice 25.000, 25.001, … , 25.018 (il primo elemento di questa sequenza è individuato dalla directory, l'ultimo contiene l'indice del blocco 25.019); quindi per accedere al blocco dati 25.020 si deve leggere anche l'elemento della FAT di indice 25.019;
7. gli elementi della FAT di indice 25.000 e 25.019, come tutti quelli intermedi, risiedono nel blocco del disco di indice 25.000 **div** 512= 25.019 **div** 512= 48.

SISTEMI OPERATIVI, CORSI A e B Prova del 5/2/2010 CORSO |A| |B|

ESERCIZIO B-3 (3 punti)

Un disco RAID di livello 4 è composto da 4 dischi fisici indipendenti, individuati dagli indici 0, 1, 2 e 3. Ogni disco fisico ha 120 cilindri, 4 facce e 30 settori per traccia, con tempo di seek proporzionale al numero di cilindri attraversati e uguale a 0,5ms per ogni cilindro. Il periodo di rotazione è di 3 msec: conseguentemente ogni settore viene percorso in 0,1 msec. Ogni disco esegue, indipendentemente dagli altri, i comandi pendenti, che gestisce con politica FIFO. Per semplicità si assume che dopo ogni operazione di seek, il ritardo rotazionale necessario per raggiungere il primo setore da leggere o scrivere sia uguale a un intero periodo di rotazione (3 msec).
I blocchi del disco virtuale hanno ampiezza uguale a quella dei blocchi fisici, e le strip coincidono con i blocchi. Il blocco *b* del disco virtuale V è mappato nel blocco *BloccoFis= b* ***div*** *3* del disco fisico di indice *DiscoFis= b* ***mod*** *3*. Il disco 3 è ridondante e ogni suo settore contiene la parità dei settori omologhi dei dischi non ridondanti di indice 0, 1 e 2 .
Al tempo *t* si ha la seguente situazione:

- il disco di indice *0* è ha in corso di esecuzione un'operazione sul cilindro 10 che terminerà al tempo t+2, e non ha altri comandi pendenti;
- il disco di indice *1* è guasto e il suo controllore, se riceve un comando di lettura o scrittura, notifica il *crash fault* con tempo di risposta trascurabile;
- il disco di indice *2* ha appena terminato una lettura sul cilindro 20 non ha in corso di esecuzione o pendenti altri comandi;
- il disco di indice *3* in corso di esecuzione un'operazione sul cilindro 50 che terminerà al tempo t+1, e ha un altro comando pendente per il medesimo cilindro, la cui esecuzione richiederà 5,1 msec .
- Al tempo *t* il disco virtuale V riceve un comando di lettura che interessa il blocco di indice virtuale 3031;

Si chiede

1. L'indice del disco fisico e, al suo interno, l'indice del blocco fisico sul quale è mappato il blocco virtuale 3031;
2. La traduzione in terna *(cilindro, faccia, settore)* dell'indice di questo blocco fisico
3. L'operazione dà luogo a un crash fault? | NO | | SI |
4. Se la risposta è NO:
   il tempo di terminazione dell'operazione (espresso in msec a partire dal tempo *t)*.
5. Se la risposta è SI:
   - quali operazioni si devono eseguire per ricostruire il blocco da leggere, e quali sono i dischi fisici interessati;
   - il tempo di terminazione dell'operazione eseguita da ciascun disco fisico;
   - il tempo di terminazione dell'operazione di lettura del blocco virtuale 3031 (espresso in msec a partire dal tempo *t*; considerare solo il tempo impiegato per le operazioni su disco).

SOLUZIONE

1) Dalle formule *DiscoFis= b* ***mod*** *3; BloccoFis= b* ***div*** *3:*
DiscoFis= 1; Blocco Fis= 1010;
2) Dalle formule:
cilindro= BloccoFis div *(4*30), faccia=(BloccoFis* mod *(4*30))* div *30; settore= (BloccoFis'* mod *(4*30))* mod *30,*
cilindro= 8; faccia= 1; settore= 20.;
3) L'operazione dà luogo a un crash fault? | | SI |

5) - Operazioni da eseguire per ricostruire il blocco di indice virtuale 3031:
Disco 0: lettura del blocco con cilindro= 8; faccia= 1; settore= 20
Disco 2: lettura del blocco con cilindro= 8; faccia= 1; settore= 20
Disco 3: lettura del blocco con cilindro= 8; faccia= 1; settore= 20
- Tempo di terminazione di ciascuna operazione eseguita dai dischi fisici:

Disco 0

cilindro:	8	settore:	20	faccia :	1				
inizio:	2	seek:	2*0,5	rotazione:	3	percorrenza:	0,1	fine:	6,1

Disco 2

cilindro:	8	settore:	20	faccia :	1				
inizio:	0	seek:	12*0,5	rotazione:	3	percorrenza:	0,1	fine:	9,1

Disco 3

cilindro:	8	settore:	20	faccia :	1				
inizio:	6,1	seek:	42*0,5	rotazione:	3	percorrenza:	0,1	fine:	30,2

- tempo di terminazione dell'operazione di lettura del blocco virtuale 3031: *t+ 30,2. msec.*

ESERCIZIO B-4 (2 punti)

Si consideri un sistema dove gli indirizzi logici hanno la lunghezza di 32 bit e le pagine logiche e i blocchi fisici hanno ampiezza di 2 kByte.Per la gestione della memoria con paginazione a domanda si utilizzano tabelle delle pagine a 3 livelli.
Le tabelle di primo, secondo o terzo livello hanno tutte uguale lunghezza.Per ogni tabella, gli elementi occupano 3 byte e riservano 4 bit agli indicatori (bit di presenza, riferimento, modifica e protezione in scrittura). I rimanenti bit codificano un indice di blocco fisico.
Si chiede:

1. la lunghezza del campo offset, in numero di bit;
2. la lunghezza delle tabelle di secondo livello (numero di elementi);
3. lo spazio occupato in memoria da ogni tabella di secondo livello (numero di byte);
4. la massima estensione della memoria fisica (numero di blocchi e di byte, espressi come potenze di 2).

SOLUZIONE

1. lunghezza del campo offset : 11 bit
2. lunghezza, in numero di elementi, di ogni tabella di primo, secondo o terzo livello: 2^7 elementi
3. spazio occupato da ogni tabella di primo, secondo o terzo livello, in numero di byte : $3 * 2^7 = 384$ byte
4. massima estensione della memoria fisica : 2^{20} blocchi → $2^{20} * 2^{11} = 2^{31}$ byte → 2 Gbyte.

ESERCIZIO B-5 (2 punti)

Dato il seguente frammento di codice del processo P:

```
f1=open("documenti",…);
pid=fork();
if (pid>0) {
       n=write(f1,"esercizio",9);
       close(f1);
}
if (pid==0) {
       n=read(f1,&buf,3);
       printf(&buf);
}
```

Si assuma che il file documenti sia inizialmente vuoto, che tutte le chiamate di sistema abbiano successo, e che il processo P completi l'esecuzione del frammento prima che il suo processo figlio venga messo in esecuzione. Al termine dell'esecuzione del frammento di codice si chiede:

1. il valore dell'I/O pointer al file documenti per il processo P
2. il valore dell'I/O pointer al file documenti per il processo figlio di P
3. Cosa stampa il processo P
4. Cosa stampa il figlio del processo P

SOLUZIONE

1. Valore dell'I/O pointer al file documenti per il processo P: -- (P ha chiuso il file)
2. Valore dell'I/O pointer al file documenti per il processo figlio di P: 9
3. Il processo P non esegue la stampa
4. Il figlio del processo P stampa una stringa vuota (lo I/O è posizionato dopo l'ultimo carattere del file)