SISTEMI OPERATIVI, CORSI A e B Prova del 25/6/2009 CORSO |A| |B| Fila…..…. Posto ……

NOME……MATRICOLA …… CORSO |A| |B|

ESERCIZIO A-1 (4 punti)

Si consideri un processore che dispone dei registri speciali PC (program counter) e PS (program status), dello stack pointer SP e dei registri generali R1 e R2. Al riconoscimento di un'interruzione, l'hardware instaura lo stato supervisore, disabilita le interruzioni, salva tutti i registri nello stack del nucleo, copia nel registro SP il valore attuale dello stack pointer del nucleo e salta all'indirizzo specificato dal vettore di interruzione, cioè al punto di ingresso della funzione di servizio. Ogni funzione di servizio termina con l'istruzione IRET, che ripristina tutti i registri dallo stack del nucleo e contemporaneamente torna allo stato utente e riabilita le interruzioni.

Il sistema operativo realizza i thread a livello kernel. I thread cositituiscono l'unica unità di schedulazione e la politica di gestione del processore è basata unicamente sulla priorità dei thread.

Al tempo t sono presenti, tra gli altri, il thread T1i del processo P1, in stato di esecuzione, e il thread T2j del processo P2, che è il primo dei thread sospesi sulla *condition COND1*. Al tempo *t* il thread T1i esegue la funzione *p-thread_condition-signal(COND1)*, che esegue l'istruzione SVC. Al momento dell'esecuzione di questa istruzione i registri del processore, i descrittori di T1i e T2j e lo stack del nucleo hanno i contenuti mostrati in tabella. Lo stack pointer del nucleo ha il valore 1016.

Il vettore di interruzione associato all'interruzione generata da SVC è 0425 e la parola di stato del nucleo è 275E.

Si chiede:

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

b) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;

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

DESCRITTORE DI T1i	
Stato	Esecuzione
Priorita	2
PC	2E31
PS	26F2
SP	2873
R1	2234
R2	26CC

DESCRITTORE DI T2j	
Stato	Bloccato
Priorita	5
PC	A12C
PS	A6F2
SP	A275
R1	A5CC
R2	A000

STACK DEL NUCLEO	
……	……
1016	23BB
1015	
1014	
1013	
1012	
1011	

REGISTRI SP, R1, R2	
SP	2997
R1	2649
R2	22CE

PROCESSORE: Registri speciali e stato							
PC	2F00		PS	16F2		stato	Utente

SOLUZIONE

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

DESCRITTORE DI T1i	
Stato	
Priorita	
PC	
PS	
SP	
R1	
R2	

DESCRITTORE DI T2j	
Stato	
Priorita	
PC	
PS	
SP	
R1	
R2	

STACK DEL NUCLEO	
……	……
1016	23BB
1015	
1014	
1013	
1012	
1011	

REGISTRI SP, R1, R2	
SP	
R1	
R2	

PROCESSORE: Registri speciali e stato						
PC		PS			stato	

b) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;

DESCRITTORE DI T1i	
Stato	
Priorita	
PC	
PS	
SP	
R1	
R2	

DESCRITTORE DI T2j	
Stato	
Priorita	
PC	
PS	
SP	
R1	
R2	

STACK DEL NUCLEO	
……	……
1016	23BB
1015	
1014	
1013	
1012	
1011	

REGISTRI SP, R1, R2	
SP	
R1	
R2	

PROCESSORE: Registri speciali e stato						
PC		PS			stato	

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

DESCRITTORE DI T1i	
Stato	
Priorita	
PC	
PS	
SP	
R1	
R2	

DESCRITTORE DI T2j	
Stato	
Priorita	
PC	
PS	
SP	
R1	
R2	

STACK DEL NUCLEO	
……	……
1016	23BB
1015	
1014	
1013	
1012	
1011	

REGISTRI SP, R1, R2	
SP	
R1	
R2	

PROCESSORE: Registri speciali e stato						
PC		PS			stato	

SOLUZIONE

d) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio:

DESCRITTORE DI T1i			DESCRITTORE DI T2j			STACK DEL NUCLEO			REGISTRI SP, R1, R2	
Stato	Esecuzione		Stato	Bloccato		……	……		SP	1011
Priorita	2		Priorita	5		1016	23BB		R1	Invariato
PC	Invariato		PC	Invariato		1015	2F00		R2	Invariato
PS	Invariato		PS	Invariato		1014	16F2			
SP	Invariato		SP	Invariato		1013	2997			
R1	Invariato		R1	Invariato		1012	2649			
R2	Invariato		R2	Invariato		1011	22CE			
PROCESSORE: Registri speciali e stato										
PC	0425		PS	27CE		stato	Supervisore			

e) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;

DESCRITTORE DI T1i			DESCRITTORE DI T2j			STACK DEL NUCLEO			REGISTRI SP, R1, R2	
Stato	Pronto		Stato	Esecuzione		……	……		SP	1011
Priorita	2		Priorita	5		1016	23BB		R1	??
PC	2F00		PC	Invariato		1015	A12C		R2	??
PS	16F2		PS	Invariato		1014	A6F2			
SP	2997		SP	Invariato		1013	A275			
R1	2649		R1	Invariato		1012	A5CC			
R2	22CE		R2	Invariato		1011	A000			
PROCESSORE: Registri speciali e stato										
PC	0425+ ??		PS	27CE		stato	Supervisore			

f) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

DESCRITTORE DI T1i			DESCRITTORE DI T2j			STACK DEL NUCLEO			REGISTRI SP, R1, R2	
Stato	Pronto		Stato	Esecuzione		……	……		SP	A275
Priorita	2		Priorita	5		1016	23BB		R1	A5CC
PC	Invariato		PC	Invariato		1015			R2	A000
PS	Invariato		PS	Invariato		1014				
SP	Invariato		SP	Invariato		1013				
R1	Invariato		R1	Invariato		1012				
R2	Invariato		R2	Invariato		1011				
PROCESSORE: Registri speciali e stato										
PC	A12C		PS	A6F2		stato	Utente			

ESERCIZIO A-2 (4 punti)

Un ristorante Fast-Food che somministra un unico tipo di hamburger è gestito da un inserviente e da un cuoco, che interagiscono attraverso uno scaffale, dove possono essere accumulati fino a 10 hamburger pronti per essere serviti. II cuoco utilizza un forno e, in sequenza, prepara gli hamburger e li depone sullo scaffale, eventualmente attendendo che ci sia posto. Il generico cliente fa il suo ordine all'inserviente e quindi attende la consegna dell'hamburger. L'inserviente riceve in sequenza gli ordini, preleva gli hamburger dallo scaffale (eventualmente attendendo la disponibilità) e li consegna ai clienti.
Il ristorante è un processo, i cui thread sono i clienti, l'inserviente, il cuoco e il forno. I thread sono realizzati a livello kernel e costituiscono l'unità di schedulazione.
Lo scaffale è un buffer circolare di 10 celle, ciascuna capace di contenere un hamburger, con associati i puntatori *primo* e *ultimo.* Inizialmente il buffer è vuoto e si ha *primo= ultimo= 0.*
Per l'interazione tra i thread si utilizzano i semafori:
ordine (valore iniziale 0); *consegna* (valore iniziale 0);
HamburgerNelloScaffale (valore iniziale 0); *PostoLiberoNelloScaffale* (valore iniziale 10);
InizioCottura (valore iniziale 0); *FineCottura* (valore iniziale 0).
La funzione eseguita dal generico cliente e, a meno delle interazioni, quelle eseguite dagli altri thread sono riportate nello schema di soluzione.
Si chiede di completare queste funzioni inserendo le operazioni *wait* e *signal* sui semafori necessarie per le interazioni tra i thread.
Si chiede inoltre se è necessario eseguire in mutua esclusione le operazioni sul buffer condiviso *Scaffale,* evidenziate in grassetto (motivare la risposta).

SOLUZIONE

thread GenericoCliente
…….
<entra nel negozio>;
<ordina un hamburger e paga>;
signal (ordine);
wait (consegna);
……..

thread Inserviente
do while true {

…….. ; //attende un *ordine*//

…….. ; //attende la disponibilità di almeno un *hamburger nello scaffale*//

<preleva un hamburger da Scaffale[primo]>

primo= (primo + 1) mod 10;

…….. ; //notifica la disponibiltà di un nuovo *posto libero nello scaffale*//

…….. ; //*consegna* l'hamburger al cliente//

}

thread Cuoco

…….. ; //avvia il forno per l'*inizio cottura* //

do while true {

…….. ; //attende la *fine cottura*//

…….. ; //attende la disponibiltà di almeno un *posto libero nello scaffale* //

<deposita un hamburger in Scaffale[ultimo]>
ultimo= (ultimo + 1) mod 10;

…….. ; //segnala la disponibilità di almeno un *hamburger nello scaffale*//

…….. ; //avvia il forno per l'*inizio cottura* //

}

thread Forno
do while true {

…….. ; //attende il segnale di *inizio cottura*//

<esegue la cottura>

…….. ; //segnala *la fine cottura*//

}

Per le operazioni sul buffer condiviso *Scaffale* **E' / NON E'** necessaria la mutua esclusione

Motivo: ……..

SOLUZIONE

```
thread GenericoCliente
…….
<entra nel negozio>;
<ordina un hamburger e paga>;
signal (ordine);
wait (consegna);
……..

thread Inserviente
do while true        {
        wait (ordine); //attende un ordine//
        wait (HamburgerNelloScaffale)        //attende la disponibilità di almeno un hamburger nello scaffale//
        <preleva un hamburger da Scaffale[primo]>
        primo= (primo + 1) mod 10;
        signal (PostoLiberoNelloScaffale) //notifica la disponibiltà di un nuovo posto libero nello scaffale//
        signal (consegna)  //consegna l'hamburger al cliente//
}

thread Cuoco
signal (InizioCottura); //avvia il forno per l'inizio cottura //
do while true        {
        wait (FineCottura); //attende la fine cottura//
        wait (PostoLiberoNelloScaffale); //attende la disponibiltà di almeno un posto libero nello scaffale //
        <deposita un hamburger in Scaffale[ultimo]>
        ultimo= (ultimo + 1) mod 10;
        signal (HamburgerNelloScaffale); //segnala la disponibilità di almeno un hamburger nello scaffale//
        signal (InizioCottura); //segnala l'inizio cottura//
        }

thread Forno
        do while true        {
        wait (InizioCottura); //attende il segnale di inizio cottura//
        <esegue la cottura>
        signal (FineCottura) //segnala la fine cottura//
        }
```

Per le operazioni sul buffer condiviso *Scaffale* NON E' necessaria la mutua esclusione
Motivo: l'unico inserviente e l'unico cuoco non accedono mai al medesimo puntatore, né alla medesima cella

ESERCIZIO A-3 (3 punti)

Un sistema con processi A, B, C, D, E e risorse dei tipi R1, R2, R3, R4 ha raggiunto lo stato mostrato nelle tabelle seguenti:

Assegnazione attuale				
	R1	R2	R3	R4
A	2	1		1
B	1	2	1	
C		2		2
D			1	1
E	1		3	2

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

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			
0	0	0	0

Successivamente il processo A richiede un'istanza di R1, il processo B richiede un'istanza di R2, il processo C richiede un'istanza di R2, il processo D richiede un'istanza di R3 e il processo E richiede un'istanza di R4: di conseguenza tutti i processi si sospendono e il sistema è in stallo.
Per eliminare lo stallo si sopprimono i pocessi D ed E.
Si chiede:

1. lo stato raggiunto dopo l'eliminazione di D ed E,
2. lo stato raggiunto dopo aver soddisfatto (compatibilmente con la disponibilità) le richieste pendenti dei processi rimanenti;
3. a questo punto lo stallo è definitivamente eliminato? (ovvero, è sempre garantita la terminazione dei processi A, B e C ?)

SOLUZIONE

1. Stato raggiunto dopo la soppressione dei processi D ed E:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza residua (considerata la molteplicità e l'assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			

2. Stato raggiunto dopo aver soddisfatto (compatibilmente con la disponibilità) le richieste pendenti dei processi rimanenti:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza residua (considerata la molteplicità e l'assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			

3. A partire da questo stato :

Il processo ……. **PUO' / NON PUO'** terminare

La disponibilità di {R1, R2, R3, R4} diviene …….

- Il processo ……. **PUO' / NON PUO'** terminare

La disponibilità di {R1, R2, R3, R4} diviene …….

- Il processo ……. **PUO' / NON PUO'** terminare

La disponibilità di {R1, R2, R3, R4} diviene …….

Quindi lo stallo **E' / NON E'** definitivamente eliminato.

SOLUZIONE

1. Stato raggiunto dopo la soppressione dei processi D ed E:

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

Esigenza residua (considerata la molteplicità e l'assegnazione attuale				
	R1	R2	R3	R4
A	1	0	1	2
B	2	1	1	3
C	2	1	2	1
D	-	-	-	-
E	-	-	-	-

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			
1	0	4	3

2. Stato raggiunto dopo l'assegnazione di un'istanza di R1 al processo A (le richieste pendenti di B e C non possono essere soddisfatte con l'attuale disponibilità):

Assegnazione attuale				
	R1	R2	R3	R4
A	3	1		1
B	1	2	1	
C		2		2
D	-	-	-	-
E	-	-	-	-

Esigenza residua (considerata la molteplicità e l'assegnazione attuale				
	R1	R2	R3	R4
A	0	0	1	2
B	2	1	1	3
C	2	1	2	1
D	-	-	-	-
E	-	-	-	-

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			
0	0	4	3

3. A partire da questo stato la terminazione dei processi è sempre garantita. Infatti:

Il processo A può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 3,1,4,4 }
- Il processo B può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 4,3,5,4}
- Il processo C può terminare

In altri termini, lo stato è sicuro e lo stallo è definitivamente eliminato.

ESERCIZIO A-4 (2 punti)

In un sistema UNIX, un processo che genera un figlio esegue il seguente frammento di codice.

```
…
printf("uno");
a = fork();
execl("/bin/pippo",NULL);
if (a>0) {
       printf("tre");
       }
else
   if (a==0)printf("quattro");
   else printf("cinque");
…
```

Che cosa stampa il processo eseguendo questo frammento di codice se il file /bin/pippo è eseguibile?

SOLUZIONE

Inizialmente il processo padre stampa "`uno`". Successivamente stampa "cinque" solo se la fork fallisce.
Se la fork ha successo, sia il padre che il figlio eseguono la `execl` e non eseguono le stampe specificate nel programma originario, ma quelle eventualmente specificate nel programma `/bin/pippo`

ESERCIZIO A-5 (2 punti)

In un sistema operativo sono presenti i processi P1, P2 e P3, e sono definiti i semafori *sem1* e *sem2*.
Al tempo *t* i semafori hanno la seguente configurazione:
Sem1: valore 0, coda P2
Sem2: valore 2, coda Ø
Allo stesso tempo, la CodaPronti contiene P3 e il processo P1 è in esecuzione.

SISTEMI OPERATIVI, CORSI A e B Prova del 25/6/2009 CORSO |A| |B|

Lo scheduler dei processi non prevede il prerilascio del processore.
Dire come si modificano i semafori e la CodaPronti e quale processo è in esecuzione se si verificano (in alternativa) le due seguenti sequenze di eventi:

a) P1 esegue *signal (Sem1)* e successivamente il processo in esecuzione esegue *wait(Sem1);*
b) P1 esegue *wait (Sem2)* e successivamente il processo in esecuzione esegue *wait(Sem2);*

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti	Sem1	Sem2
a-1	P1 esegue *signal(Sem1)*				
a-2	Il processo in esecuzione esegue *wait(Sem1)*				
b-1	P1 esegue *wait (Sem2)*				
b-2	Il processo in esecuzione esegue *wait (Sem2)*				

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti	Sem1	Sem2
a-1	P1 esegue *signal(Sem1)*	P1	P3, P2	0, Ø	2, Ø
a-2	Il processo in esecuzione esegue *wait(Sem1)*	P3	P2	0, P1	2, Ø
b-1	P1 esegue *wait (Sem2)*	P1	P3	0, P2	1, Ø
b-2	Il processo in esecuzione esegue *wait (Sem2)*	P1	P3	0, P2	0, Ø

SISTEMI OPERATIVI, CORSI A e B Prova del 25/6/2009 CORSO |A| |B|

ESERCIZIO B-1 (4 punti)

Un sistema operativo simile a UNIX, che gestisce la memoria con paginazione a domanda, utilizza l'algoritmo di sostituzione *Second Chance* e il processo *PageDaemon,* con parametri *Lotsfree = 4* e *MinFree =2*. 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* (indice della pagina del processo caricata nel blocco) e *Rif* (bit di pagina riferita utilizzato dall'algoritmo *Second Chance*). Al tempo *t*=0, quando il *PageDaemon* ha appena terminato la sua esecuzione, sono presenti i processi A, B, C, D e la *Core Map* ha la configurazione mostrata in figura, con il puntatore dell'algoritmo di *SecondChance* posizionato sul blocco 6. Sono ignorati i blocchi di memoria occupati dal sistema operativo.

							▼																		
Proc	C		D			A	C	B	A	A	D		C		C	B	D	A	C		D	B	B	
Pag	3		12			8	5	2	5	6	10		7		8	7	9	2	10		4	8	9	
Rif	1		1			0	0	0	1	0	1		1		1	1	1	1	0		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=0*

Il *PageDaemon* interviene periodicamente (con periodo 10, quindi ai tempi *t=10*, *t=20*, etc.), e inoltre quando si verifica un *PageFault* e, in seguito al caricamento della pagina, il numero di blocchi liberi diventa minore di *MinFree*. Ad ogni intervento, *PageDaemon* si comporta nel modo seguente:

- se *NumeroDiBlocchiLiberi* ≥ *LotsFree* termina senza operare nessuna modifica;
- se *NumeroDiBlocchiLiberi* ≥ *MinFree* e *NumeroDiBlocchiLiberi* < *LotsFree*, scarica iterativamente le pagine selezionate dall'algoritmo *Second Chance*, finchè *NumeroDiBlocchiLiberi* = *Lotsfree* + 3;
- se *NumeroDiBlocchiLiberi* < *MinFree* esegue lo swapout di uno o più processi, finchè diviene *NumeroDiBlocchiLiberi* ≥ *Lotsfree* + 3. Per lo swapout dei processi si seleziona di volta in volta quello che ha il maggior numero di pagine caricate in memoria principale (in caso di parità si segue l'ordine alfabetico).

Quando si verifica un errore di pagina, la pagina riferita viene caricata nel blocco libero di indice minore.
Considerare in la seguente sequenza di eventi:

a.1) Dal tempo *0* al tempo *5* avanza il processo A, che riferisce nell'ordine le pagine 13, 12, 8, 6, 7, 4, 1
a.2) Dal tempo *6* al tempo *8* avanza il processo B, che riferisce le pagine 2, 7, 8, 10;
a.3) Dal tempo *9* al tempo *10* avanza il processo D, che riferisce le pagine 13, 11, 10, 4;
a.4) Dal tempo *10* al tempo *15* avanza il processo C, che riferisce le pagine 4, 6, 10, 9;
a.5) Dal tempo *16* al tempo *20* avanza il processo B, che riferisce le pagine 10, 2, 1, 5

Si suppone per semplicità che la durata di ogni intervento del PageDaemon sia trascurabile
Si richiede di mostrare la configurazione della *CoreMap* all'inizio e alla fine di ogni intervento del processo *PageDaemon*.

SOLUZIONE (usare solo le tabelle e le righe necessarie)

Configurazione della Core Map **prima** del **primo** intervento del *PageDaemon*, che avviene al tempo t= …….

Proc																									
Pag																									
Rif																									
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

Configurazione della Core Map **al termine** del **primo** intervento del *PageDaemon* (tempo t= ……)

Proc																									
Pag																									
Rif																									
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

Eseguito swap out dei processi: …… Rimosse le pagine: ……

Configurazione della Core Map **prima** del **secondo** intervento del *PageDaemon*, che avviene al tempo t= ……

Proc																									
Pag																									
Rif																									
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

Configurazione della Core Map **al termine** del **secondo** intervento del *PageDaemon* (tempo t= ……)

Proc																									
Pag																									
Rif																									
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

Eseguito swap out dei processi: …… Rimosse le pagine: ……

Configurazione della Core Map **prima** del **terzo** intervento del *PageDaemon*, che avviene al tempo t= ……

Proc																									
Pag																									
Rif																									
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

Configurazione della Core Map **al termine** del **terzo** intervento del *PageDaemon* (tempo t= ……)

Proc																									
Pag																									
Rif																									
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

Eseguito swap out dei processi: …… Rimosse le pagine: ……

SOLUZIONE

Configurazione della Core Map **prima** del **primo** intervento del *PageDaemon*, che avviene al tempo t= 8

							▼																	
Proc	C	A	D	A	A	A	C	B	A	A	D	A	C	A	C	B	D	A	C	B	D	B	B	
Pag	3	13	12	12	7	8	5	2	5	6	10	4	7	1	8	7	9	2	10	10	4	8	9	
Rif	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	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

Configurazione della Core Map **al termine** del **primo** intervento del *PageDaemon* (tempo t= 8;)

							▼																	
Proc	C		D				C	B			D		C		C	B	D		C	B	D	B	B	
Pag	3		12				5	2			10		7		8	7	9		10	10	4	8	9	
Rif	1		1				0	0			1		1		1	1	1		0	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

Eseguito lo swapout del processo A

Configurazione della Core Map **prima** del **secondo** intervento del *PageDaemon*, che avviene al tempo 10

							▼																	
Proc	C	D	D	D			C	B			D		C		C	B	D		C	B	D	B	B	
Pag	3	13	12	11			5	2			10		7		8	7	9		10	10	4	8	9	
Rif	1	1	1	1			0	0			1		1		1	1	1		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	22	23

Configurazione della Core Map **al termine** del **secondo** intervento del *PageDaemon* (tempo t= 10)

							▼																	
Proc	C	D	D	D			C	B			D		C		C	B	D		C	B	D	B	B	
Pag	3	13	12	11			5	2			10		7		8	7	9		10	10	4	8	9	
Rif	1	1	1	1			0	0			1		1		1	1	1		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	22	23

PageDaemon termina senza operare modifiche dato che il numero di blocchi liberi è maggiore di *LotsFree*

Configurazione della Core Map **prima** del **terzo** intervento del *PageDaemon*, che avviene al tempo t= 20

								▼																
Proc	C	D	D	D	C	C	C	B	C	B	D	B	C		C	B	D		C	B	D	B	B	
Pag	3	13	12	11	4	6	5	2	9	1	10	5	7		8	7	9		10	10	4	8	9	
Rif	1	1	1	1	1	1	0	1	1	1	1	1	1		1	1	1		1	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	22	23

Configurazione della Core Map **al termine** del **terzo** intervento del *PageDaemon* (tempo t= 20)

											▼													
Proc	C	D	D	D	C	C					D	B	C		C	B	D		C	B	D	B	B	
Pag	3	13	12	11	4	6					10	5	7		8	7	9		10	10	4	8	9	
Rif	0	0	0	0	0	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	22	23

Applicato l'algoritmo second chance per liberare altre 4 pagine, rimosse le pagine: C5, B2, C9, B1

ESERCIZIO B-2 (4 punti)

Un sistema operativo gestisce la memoria con paginazione dinamica con pagine di 2 Kbyte e utilizza un file system di tipo FAT-32 (con indirizzi a 32 bit) con blocchi di 2 Kbyte per gestire un disco di 10 Gbyte. La FAT è allocata sul disco a partire dal blocco 3. Nel file system il file `foto` contenuto nella directory `personale` occupa 9 blocchi logici allocati come segue:

Blocco logico:	0	1	2	3	4	5	6	7	8
Blocco fisico:	398.203	1.716.882	106.987	2.448.123	12.186	65.637	4.876.999	3.161.002	908.543

Ad un dato istante di tempo, quando nessun blocco della FAT è caricato in memoria principale e la directory `personale` è caricata in memoria, il sistema operativo riceve una richiesta di lettura dei caratteri del file `foto` compresi tra 8.000 e 14.000 (estremi inclusi).

Si chiede:

1. Quali blocchi logici del file `foto` vengono letti dal sistema operativo
2. Quali blocchi fisici del file `foto` vengono letti dal sistema operativo
3. Quali blocchi fisici contenenti la FAT vengono letti dal sistema operativo
4. Quanti page fault causa l'operazione di lettura e che i buffer destinati ad accogliere i dati letti dal file siano già allocati e presenti in memoria.

SOLUZIONE

1. Blocco logico del file `foto` che contiene il byte 8.000: ……

   Blocco logico del file `foto` che contiene il byte 14.000: ……

   Blocchi logici del file `foto` letti dal sistema operativo: ……
2. Blocchi fisici del file `foto` letti dal sistema operativo: ……
3. Blocchi della FAT letti:

   Ogni blocco del disco contiene …… puntatori della FAT, di conseguenza:

   Il puntatore …… è contenuto nel blocco ……della FAT, che corrisponde al blocco…… del disco;

   Il puntatore …… è contenuto nel blocco ……della FAT, che corrisponde al blocco …… del disco;

   Il puntatore …… è contenuto nel blocco ……della FAT, che corrisponde al blocco …… del disco;

   Il puntatore …… è contenuto nel blocco ……della FAT, che corrisponde al blocco …… del disco;

   Quindi i blocchi fisici da leggere contenenti la FAT sono: ……
4. Page fault causati dall'operazione di lettura: …… per la lettura della FAT

SOLUZIONE

1. Blocco logico del file `foto` che contiene il byte 8.000: 8.000 **div** 2048 = 3;
   Blocco logico del file `foto` che contiene il byte 14.000: 14.000 **div** 2048 = 6;
   Blocchi logici del file `foto` letti dal sistema operativo: 3, 4, 5, 6
2. Blocchi fisici del file `foto` letti dal sistema operativo: 2.448.123; 12.186; 65.637; 4.876.999
3. Blocchi della FAT letti:
   Ogni blocco del disco contiene 2048/4 = 512 puntatori della FAT, di conseguenza:
   Il puntatore 2.448.123 è contenuto nel blocco 4781 della FAT, che corrisponde al blocco 3+4781=4784 del disco
   Il puntatore 12.186 è contenuto nel blocco 23 della FAT, che corrisponde al blocco 3+23=26 del disco
   Il puntatore 65.637 è contenuto nel blocco 131 della FAT, che corrisponde al blocco 3+128=131 del disco
   Il puntatore 4.876.999 è contenuto nel blocco 9525 della FAT, che corrisponde al blocco 3+9525=9528 del disco

   Quindi i blocchi fisici da leggere contenenti la FAT sono: 4784, 26, 131 e 9528
4. Page fault causati dall'operazione di lettura: 4 per la lettura della FAT

ESERCIZIO B-3 (3 punti)

Un disco RAID di livello 4 è composto da 7 dischi fisici, numerati da 0 a 6. Le strip corrispondono a blocchi.
Il disco 6 è ridondante e il suo blocco di indice *i* contiene la parità dei blocchi di indice *i* dei dischi non ridondanti 0, 1, 2, 3, 4, 5.
Al tempo T, i blocchi 5 e 8 dei dischi fisici hanno il seguente contenuto:

Disco 0, blocco 5	0	1	1	0	1	0	0	1
Disco 1, blocco 5	0	1	0	0	1	1	0	1
Disco 2, blocco 5	1	1	1	0	1	1	0	0
Disco 3, blocco 5	0	1	1	1	1	0	0	1
Disco 4, blocco 5	1	0	1	1	0	0	0	1
Disco 5, blocco 5	0	0	0	1	1	0	1	0
Disco 6, blocco 5	0	0	0	1	1	0	1	0

Disco 0, blocco 8	0	0	1	1	1	0	0	1
Disco 1, blocco 8	0	1	0	0	1	1	0	1
Disco 2, blocco 8	1	1	1	1	1	1	0	0
Disco 3, blocco 8	0	1	1	1	1	0	0	1
Disco 4, blocco 8	1	1	1	1	0	0	0	1
Disco 5, blocco 8	0	0	1	0	1	1	0	1
Disco 6, blocco 8	0	0	1	0	1	1	0	1

e sono pendenti due operazioni di scrittura che interessano i blocchi del disco virtuale mappati rispettivamente nel blocco 8 del disco fisico 0 e nel blocco 5 del disco fisico 2. Le configurazioni da scrivere sono le seguenti:

Disco 0, blocco 8	1	0	0	0	1	0	1	0

Disco 2, blocco 5	0	0	0	1	1	0	1	1

Si chiede:

1) la configurazione finale del blocco 5 e del blocco 8 di tuti i dischi fisici;
2) quali sono le operazioni di lettura e di scrittura da eseguire sui dischi fisici 0, 2 e 6
3) tra le operazioni del punto 2), quali sono indipendenti e pertanto possono essere eseguite contemporaneamente (la risposta può non essere univoca)
4) il numero minimo di accessi ai dischi fisici, tenendo conto delle operazioni indipendenti.

SOLUZIONE

1) Configurazione finale dei blocchi 5 e 8 di tutti i dischi fisici dopo le due scritture:

Disco 0, blocco 5								
Disco 1, blocco 5								
Disco 2, blocco 5								
Disco 3, blocco 5								
Disco 4, blocco 5								
Disco 5, blocco 5								
Disco 6, blocco 5								

Disco 0, blocco 8								
Disco 1, blocco 8								
Disco 2, blocco 8								
Disco 3, blocco 8								
Disco 4, blocco 8								
Disco 5, blocco 8								
Disco 6, blocco 8								

2) operazioni di lettura e di scrittura da eseguire sui dischi fisici 0, 2 e 6:

Per la scrittura sul blocco 8 del disco 0:

a.1) lettura/scrittura del blocco ……. del disco ……..

a.2) lettura/scrittura del blocco ……. del disco ……..

a.3) lettura/scrittura del blocco ……. del disco ……..

a.4) ……..

a.5) ……..

Per la scrittura sul blocco 5 del disco 2:

b.1) lettura/scrittura del blocco ……. del disco ……..

b.2) lettura/scrittura del blocco ……. del disco ……..

b.3) lettura/scrittura del blocco ……. del disco ……..

b.4) ……..

b.5) ……..

3) Insiemi di operazioni indipendenti (selezionare dagli elenchi del punto 2)

insieme 1: operazione ….., operazione ……; ……

insieme 2: operazione ….., operazione ……; ……

insieme 3: operazione ….., operazione ……; ……

……..

……..

4) Numero minimo di accessi ai dischi fisici: ……

SOLUZIONE

SISTEMI OPERATIVI, CORSI A e B Prova del 25/6/2009 CORSO |A| |B|

1) Configurazione finale dei blocchi 5 e 8 dei dischi fisici dopo le due scritture:

Disco 0, blocco 5	0	1	1	0	1	0	0	1
Disco 1, blocco 5	0	1	0	0	1	1	0	1
Disco 2, blocco 5	0	0	0	1	1	0	1	1
Disco 3, blocco 5	0	1	1	1	1	0	0	1
Disco 4, blocco 5	1	0	1	1	0	0	0	1
Disco 5, blocco 5	0	0	0	1	1	0	1	0
Disco 6, blocco 5	1	1	1	0	1	1	0	1

Disco 0, blocco 8	1	0	0	0	1	0	1	0
Disco 1, blocco 8	0	1	0	0	1	1	0	1
Disco 2, blocco 8	1	1	1	1	1	1	0	0
Disco 3, blocco 8	0	1	1	1	1	0	0	1
Disco 4, blocco 8	1	1	1	1	0	0	0	1
Disco 5, blocco 8	0	0	1	0	1	1	0	1
Disco 6, blocco 8	1	0	0	1	1	1	1	0

2) operazioni di lettura e di scrittura da eseguire sui dischi fisici 0, 2 e 6

Per la scrittura sul blocco 8 del disco 0:
- lettura del blocco 8 del disco 0
- lettura del blocco 8 del disco 6
- scrittura del blocco 8 del disco 0
- scrittura del blocco 8 del disco 6

Per la scrittura sul blocco 5 del disco 2:
- lettura del blocco 5 del disco 2
- lettura del blocco 5 del disco 6
- scrittura del blocco 5 del disco 2
- scrittura del blocco 5 del disco 6

Insiemi di operazioni indipendenti:

a) lettura del blocco 8 del disco 0 , lettura del blocco 8 del disco 6 ,
b) scrittura del blocco 8 del disco 0 , scrittura del blocco 8 del disco 6 ,
c) lettura del blocco 5 del disco 2, lettura del blocco 5 del disco 6;
d) scrittura del blocco 5 del disco 2, scrittura del blocco 5 del disco 6;

4) Numero minimo di accessi ai dischi fisici: 4.

ESERCIZIO B.4 (2 punti)

Data la seguente lista di capability:

```
Utente 20:  <File 1:R,W,X>→<File2:W>→<File3:R>→<Stampante:W>→<Lettore:R>
Utente 21:   <File2:W,X> → <File3:RW>
Gruppo 812:  <File1:X> → <Stampante:W> → <File2:R,W>
Gruppo 33:   <Lettore:R> → <File3:W> → <File1:R,W,X> → <Stampante:W>
```

Convertirla nella corrispondente matrice di protezione

SOLUZIONE

	File1	File2	File3	Stampante	Lettore
Utente 20	RWX	W	R	W	R
Utente 21		WX	RW		
Gruppo 812	X	RW		W	
Gruppo 33	RWX		W	W	R

ESERCIZIO B.5 (2 punti)

In un disco con capacità di 64 Gbyte (= 2^{36} byte) e blocchi di 4 Kbyte (= 2^{12} byte), è definito un file system di tipo Unix con puntatori a 32 bit (4 byte) e I/O pointer di 32 bit. Ogni i-node occupa 128 byte e contiene, oltre agli attributi del file, 10 puntatori diretti e tre puntatori indiretti (indiretto singolo, doppio e triplo).
Il blocco 0 e 1 del disco sono occupati dal boot block e dal superblocco, rispettivamente, mentre la i-list occupa i blocchi compresi tra 3 e 8.194 (estremi inclusi).
Si chiede:

1. La dimensione della i-list in byte:
2. Il numero di i-node contenuti nella i-list;
3. Il massimo numero di file che possono essere contenuti nel file system;
4. La massima dimensione di un file in numero di blocchi;

SOLUZIONE

1. La dimensione della i-list in byte: ……
2. Il numero di i-node contenuti nella i-list: ……
3. Il massimo numero di file che possono essere contenuti nel file system : ……
4. Considerato che:

   - ogni i-node può rappresentare file di lunghezza fino a …… blocchi
   - lo I/O pointer può indirizzare fino a …… caratteri.

   La massima dimensione di un file è ……. caratteri → …….. blocchi

SOLUZIONE

1. La dimensione della i-list in byte: 8192*4 KByte = 32 MByte
2. Il numero di i-node contenuti nella i-list: 32MByte / 128 Byte = 256 K = 262.144
3. Il massimo numero di file che possono essere contenuti nel file system : 262.144
4. La massima dimensione di un file :
   ogni i-node può rappresentare file fino a 10+ 1024 + 1024^2 + 1024^3 blocchi = 10 + 1K + 1M + 1G blocchi, dato che ogni blocco occupa 4KB il file può essere più grande di 4 TB. Dato che l'I/O pointer può rappresentare fino a 4GB risulta che la massima dimensione del file è di 4GB e quindi di 4M Blocchi.