SISTEMI OPERATIVI, CORSI A e B - SECONDO APPELLO 2008 - 1/2/2008

ESERCIZIO A.1 (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 riservato allo stato utente, che comprende i registri generali R1, R2, R3, R4 e lo stack pointer SP,
- 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.

Il sistema gestisce il processore con la politica delle code multiple. Al tempo t, quando sono presenti nel sistema il processo Pi, in stato di esecuzione, e il processo Pj, che è sospeso in attesa dei input da tastiera, il controller della tastiera lancia un'interruzione che segnala l'avvenuta pressione di un tasto. Immediatamente dopo il tempo *t*, quando l'interruzione viene riconosciuta, i registri del processore, i descrittori di Pi e Pj e lo stack del nucleo hanno i contenuti mostrati in figura.

L'interruzione determina l'intervento del nucleo, che esegue una funzione di servizio che riattiva il processo Pj e incrementa di 4 la sua priorità. Supponendo che il vettore di interruzione associato all'interruzione generata dalla tastiera 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.

DESCRITTORE DI P_I	
Stato	Esec
priorità	8
PC	8AA1
PS	16F2
SP	8CC1
R1	8181
R2	9898
R3	A000
R4	8000

DESCRITTORE DI P_J	
Stato	Bloccato
priorità	5
PC	51AB
PS	16F2
SP	5FF6
R1	1453
R2	5E54
R3	0056
R4	0038

PROCESSORE: Registri speciali			
PC	8B00	PS	16F2

STACK DEL NUCLEO	
……	……
1016	23BB
1015	070A
1014	
1013	
1012	
1011	
1010	

REG. STATO UTENTE	
SP	8CD1
R1	8209
R2	89AA
R3	D899
R4	ABAB
REG. STATO SUPERV.	
SP'	1015
R'1	F1F1
R'2	FF55
R'3	FF01
R'4	F0F0

SISTEMI OPERATIVI, CORSI A e B - SECONDO APPELLO 2008 - 1/2/2008

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:

DESCRITTORE DI P_I	
Stato	Esec
priorità	8
PC	Invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato

DESCRITTORE DI P_J	
Stato	Bloccato
priorità	5
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	23BB
1015	070A
1014	8B00
1013	16F2
1012	
1011	
1010	

REG. STATO UTENTE	
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato
REG. STATO SUPERV.	
SP'	1013
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;

DESCRITTORE DI P_I	
Stato	Pronto
priorità	8
PC	8B00
PS	16F2
SP	8CD1
R1	8209
R2	89AA
R3	D899
R4	ABAB

DESCRITTORE DI P_J	
Stato	Esec
priorità	9
PC	Invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato

PROCESSORE: Registri speciali			
PC	??	PS	275E

STACK DEL NUCLEO	
……	……
1016	23BB
1015	070A
1014	51AB
1013	16F2
1012	
1011	
1010	

REG. STATO UTENTE	
SP	5FF6
R1	1453
R2	5E54
R3	0056
R4	0038
REG. STATO SUPERV.	
SP'	1013
R'1	??
R'2	??
R'3	??
R'4	??

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 P_I	
Stato	Pronto
priorità	8
PC	Invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato

DESCRITTORE DI P_J	
Stato	Esec
priorità	9
PC	Invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato

PROCESSORE: Registri speciali			
PC	51AB	PS	16F2

STACK DEL NUCLEO	
……	……
1016	23BB
1015	070A
1014	
1013	
1012	
1011	
1010	

REG. STATO UTENTE	
SP	Invariato
R1	Invariato
R2	Invariato
R3	Invariato
R4	Invariato
REG. STATO SUPERV.	
SP'	1015
R'1	??
R'2	??
R'3	??
R'4	??

ESERCIZIO A.2 (4 punti)

In un sistema operativo che realizza i threads a livello utente, i thread T_1, T_2 e T_3 del processo P competono in un gioco di carte. I tre threads avanzano senza sincronizzarsi ed eseguono tutti lo stesso codice:

```
1. While (true) {
2.     lock (&Chiave);
3.           EstraiCarta(&Carta);
4.     unlock (&Chiave);
5.     ThreadYield();
6.     UtilizzaCarta(Carta);
7.     lock (&Chiave);
8.           InserisciCarta(&Mazzo,Carta);
9.           ThreadYield();
10.          MescolaCarte(&Mazzo);
11.    unlock (&Chiave);
12.    ThreadYield();
       }
```

Dove lo scheduling dei thread avviene senza prerilascio e l'espansione assembler dei comandi **lock** e **unlock** è la seguente:

```
thread_lock(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//

thread_unlock(K)
     MOV K, #1  //scrive1 in K//
```

Si fanno inoltre le seguenti ipotesi:

- le operazioni `MescolaCarte`, `EstraiCarta`, `UtilizzaCarta` e `InserisciCarta` richiedono 10 msec ciascuna. Tutte le altre operazioni sono eseguite in un tempo trascurabile.
- Al tempo t, i thread T_2 e T_3 sono in stato di pronto e devono entrambi eseguire l'istruzione 6. L'ordinamento dei thread nella coda pronti è T_2 -> T_3 ;
- Al tempo t, il thread T_1 è in stato di esecuzione e deve eseguire l'istruzione 1.

Supponendo che il processo P rimanga in stato di esecuzione, mostrare come si modifica lo stato del sistema ad ogni riassegnazione del processore ai thread, fino all'istante T+80.

SOLUZIONE

(riempire una riga della tabella ad ogni nuova assegnazione del processore ai thread)

Tempo	Thread in esecuzione	Prossima istruzione di T1	Prossima istruzione di T2	Prossima istruzione di T3	Coda dei thread pronti
T	T1	1	6	6	T2 -> T3
T+10	T2	6	6	6	T3 -> T1
T+30	T3	6	10	6	T1 -> T2
T+40	T1	6	10	7	T2 -> T3
T+50	T2	7	10	7	T3 -> T1
T+60	T3	7	1	7	T1 -> T2
T+70	T1	7	1	10	T2 -> T3
T+70	T2	7	2	10	T3 -> T1
T+70	T3	7	2	10	T1 -> T2
T+80	T1	7	2	1	T2 -> T3

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. Il sistema non è attualmente in stallo.

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

Esigenza Iniziale (da modificare in base all'assegnazione attuale)				
	R1	R2	R3	R4
A	4	6	4	6
B	4	2		2
C	3	4	3	2
D	5	6	4	3
E	2	1		3

Molteplicità			
R1	R2	R3	R4
6	7	4	6

Disponibilità			
0	1	1	3

Successivamente, i processi E e C eseguono in sequenza le seguenti richieste:

1. E richiede 2 istanze di R4
2. C richiede 1 istanza di R3

Verificare se l'accettazione di queste richieste porta il sistema in uno stato sicuro.

SOLUZIONE

Stato raggiunto dopo l'assegnazione di 2 istanze di R4 al processo E:

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

Esigenza Attuale				
	R1	R2	R3	R4
A	4	6	4	6
B	4	2		2
C	3	4	3	2
D	5	6	4	3
E	2	1		1

Molteplicità			
R1	R2	R3	R4
6	7	4	6

Disponibilità			
0	1	1	1

Il processo E può terminare, il vettore disponibilità diventa: 2,2,1,3
Il processo B può terminare, il vettore disponibilità diventa: 4,2,1,4
Il processo C può terminare, il vettore disponibilità diventa: 4,5,3,4
Il processo A può terminare, il vettore disponibilità diventa: 5,6,4,6
Il processo D può terminare, il vettore disponibilità diventa: 6,7,4,6

- Quindi lo stato e sicuro e la richiesta può essere accettata dal sistema operativo.

Stato raggiunto dopo l'assegnazione di 1 istanza di R3 al processo C:

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

Esigenza Attuale				
	R1	R2	R3	R4
A	4	6	4	6
B	4	2		2
C	3	3	3	2
D	5	6	4	3
E	2	1		1

Molteplicità			
R1	R2	R3	R4
6	7	4	6

Disponibilità			
0	1	0	1

Il processo E può terminare, il vettore disponibilità diventa: 2,2,0,3
Il processo B può terminare, il vettore disponibilità diventa: 4,2,0,4
Il processo C può terminare, il vettore disponibilità diventa: 4,5,3,4
Il processo A può terminare, il vettore disponibilità diventa: 5,6,4,6
Il processo D può terminare, il vettore disponibilità diventa: 6,7,4,6

Quindi la richiesta lascia il sistema in uno stato sicuro e può essere accettata dal sistema operativo.

ESERCIZIO A- 4 (2 punti)

In un sistema UNIX il processo P esegue con successo una *fork*, generando il processo F, che passa immediatamente in esecuzione ed esegue con successo una *exec.* Si chiede quali dati di *F* risiedono nella sua *process structure,* quali risiedono nella sua *user structure,* e quali sono identici a quelle del processo padre subito dopo la *exec.*

SOLUZIONE

DATO	Process structure	User Structure	Identico?
PID del processo	Si		No
PID del padre del processo	Si		No
Immagine del program counter (PC)		Si	No
Informazioni sui segnali pendenti (signal bitmap)	Si		Si
Descrittori dei pipe aperti		Si	Si
Tabella dei descrittori di file		Si	Si

ESERCIZIO A-5 (2 punti)

In un sistema con thread realizzati a livello utente, sono presenti i processi P1 con thread T11, T12, T13, il processo P2 con thread T21 e T22 e il processo P3 con il solo thread T31. Per ogni processo, lo scheduler dei thread applica la politica FIFO e interviene quando un thread esegue l'operazione *thread_yield.* La politica di scheduling dei processi non prevede il prerilascio.

Al tempo *T* è in esecuzione il processo P1, il processo P2 è pronto e il processo P3 è sospeso sul semaforo *sem1*. Nel processo P1 è in esecuzione il thread T11 e i rimanenti thread dei tre processi sono pronti, con il seguente ordinamento nelle rispettive code:

Processo P1: T12-> T13 Processo P2: T21->T22 Processo P3: T31

Si chiede qual è il contenuto della coda pronti dei processi e quale thread è in esecuzione dopo che si sono verificate (in alternativa) le due seguenti sequenze di eventi:

a. Il thread in esecuzione esegue *thread_yield,* quindi il thread in esecuzione esegue una *signal* sul semaforo *sem1*; quindi il thread in esecuzione esegue una *wait* sul semaforo *sem2,* che ha valore 0;
b. Il thread in esecuzione esegue una *wait* sul semaforo *sem1*; il thread in esecuzione esegue una *thread_yield*; quindi il thread in esecuzione esegue una *signal* sul semaforo *sem1.*

SOLUZIONE

	Sequenze di eventi	Thread in esecuzione	Coda pronti
a-1	Il thread in esecuzione esegue *thread_yield*	T12	P2
a-2	il thread in esecuzione esegue *signal*(*sem1)*	T12	P2 → P3
a-3	il thread in esecuzione esegue *wait(sem2)* con valore 0	T21	P3
b-1	il thread in esecuzione esegue *wait*(*sem1)*	T21	-
b-2	il thread in esecuzione esegue *thread_yield*	T22	-
b-3	il thread in esecuzione esegue *signal*(*sem1)*	T22	P3

SISTEMI OPERATIVI, CORSI A e B - SECONDO APPELLO 2008 - 1/2/2008

ESERCIZIO B.1 (4 punti)

In un sistema che gestisce la memoria con swapping e partizioni variabili, la memoria viene allocata in multipli interi di unità di allocazione. La memoria ha in totale 40 unità di allocazione, e il sistema operativo occupa permanentemente una partizione con origine 0 e lunghezza 5. Al tempo 0 sono stati generati e caricati in memoria i seguenti processi:

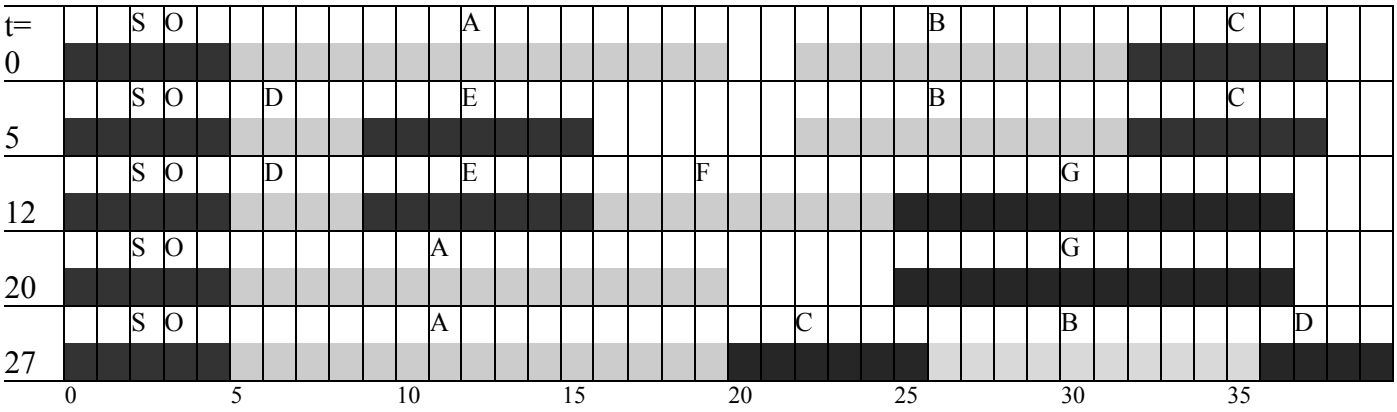
- il processo A, che risiede in memoria da 10 sec ed è caricato nella partizione con origine 5 e lunghezza 15;
- il processo B, che risiede in memoria da 6 sec ed è caricato nella partizione con origine 22 e lunghezza 10;
- il processo C, che risiede in memoria da 8 sec ed è caricato nella partizione con origine 32 e lunghezza 6;

Inoltre sono stati generati i seguenti processi, che al tempo 0 sono in attesa di caricamento:

- il processo D, che richiede una partizione di lunghezza 4 ed è in attesa di caricamento da 10 secondi;
- il processo E, che richiede una partizione di lunghezza 7 ed è in attesa di caricamento da 5 secondi;
- il processo F, che richiede una partizione di lunghezza 9 ed è in attesa di caricamento da 3 secondi;
- il processo G, che richiede una partizione di lunghezza 12 ed è in attesa di caricamento da 2 secondi.

Il codice dei processi è rilocabile. I processi in attesa di caricamento sono inseriti in una coda *FIFO*.
Quando si libera spazio sufficiente in memoria per effetto della terminazione o dello *swap-out* di un processo, il sistema carica uno o più processi in attesa di caricamento, utilizzando la politica *FIFO* per la scelta dei processi da caricare e la politica *first-fit* per la scelta delle partizioni da assegnare. Lo swapout avviene quando il tempo di attesa di caricamento raggiunge 15 secondi per almeno un processo. Al verificarsi di questa circostanza viene eseguito lo *swap-out* di uno o più processi in memoria, selezionandoli in ordine decrescente di tempo di residenza (a parità di tempo si considera l'ordine con il quale sono stati caricati), fino a creare lo spazio per il caricamento di almeno il primo processo tra quelli in attesa di caricamento.
Utilizzando il grafico sotto riportato, mostrare come evolvono l'occupazione della memoria e la coda dei processi in attesa di caricamento fino al tempo 27 (incluso), supponendo ce nessun processo termini in questo intervallo di tempo.

SOLUZIONE



	PROCESSI							
	A (15)	B(10)	C(6)	D(4)	E(7)	F(9)	G(12)	CODA
t= 0	SI, 10	SI, 6	SI, 8	NO, 10	NO, 5	NO, 3	NO, 2	D → E → F → G
t= 5	NO, 0	SI, 11	SI, 13	SI, 0	SI, 0	NO, 8	NO, 7	F → G → A
t= 12	NO, 7	NO, 0	NO, 0	SI, 7	SI, 7	SI, 0	SI, 0	A → C → B
t= 20	SI, 0	NO, 8	NO, 8	NO, 0	NO, 0	NO, 0	SI, 8	C → B → D → E → F
t= 27	SI, 7	SI, 0	SI, 0	SI, 0	NO, 7	NO, 7	NO, 0	E → F → G

ESERCIZIO B.2 (4 PUNTI)

Un sistema operativo simile a UNIX gestisce la memoria con paginazione a domanda e utilizza il processo *PageDaemon,* con parametri *lotsfree= 6 e minfree=3,* e l'algoritmo di sostituzione *Second Chance.* Gli elementi della *CoreMap* hanno i campi *Proc* (processo a cui è assegnato il blocco), *Pag* (pagina del processo caricata nel blocco), *Rif* (bit di pagina riferita, utilizzato da *Second Chance*). Tutti questi campi sono vuoti se il blocco è libero. Al tempo *t* sono presenti i processi A, B, C e la *Core Map* ha la configurazione mostrata in figura, con il puntatore dell'algoritmo di sostituzione posizionato sul blocco 11. I primi 6 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione.

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

Core Map al tempo *t*

Il *PageDaemon* interviene al tempo *t* e successivamente ogni 10 msec. Ad ogni intervento, *PageDaemon* avanza utilizzando in modo esclusivo il processore e scarica pagine finchè il numero di pagine libere raggiunge il valore *lotsfree* o, in alternativa, esegue lo *swapout* di un processo. Ai fini dello *swapout,* la selezione del processo avviene in ordine decrescente di pagine caricate in memoria. In caso di errori di pagina, i blocchi liberi vengono assegnati in ordine crescente di indice.
Considerare la seguente evoluzione del sistema:

1. Dal tempo *t* al tempo *t+1* avanza il processo *PageDaemon*;
2. Dal tempo *t+1* al tempo *t+10* avanzano i processi B e C, che riferiscono nell'ordine le pagine B2, B5, B0, B1, B6, C1, C0, C4.
3. Dal tempo *t+10* al tempo *t+11* avanza il processo *PageDaemon*;
4. Dal tempo *t+11* al tempo *t+20* avanzano i processi A e B, che riferiscono nell'ordine le pagine A3, A6, A2, A1, B0, B2, B3, B0;

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

SOLUZIONE

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

Core Map al tempo *t+1 (scaricate 2 pagine)*

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

Core Map al tempo *t+10 (2 blocchi liberi)*

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

Core Map al tempo *t+11 (swapout del processo C)*

Proc							A	A	B	A	B	B	A	B				B	A	B				
Pag							3	1	0	2	5	6	6	1				2	7	3				
Rif							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

Core Map al tempo *t+20 (6 blocchi liberi)*

SISTEMI OPERATIVI, CORSI A e B - SECONDO APPELLO 2008 - 1/2/2008

ESERCIZIO B-3 (3 punti)

In un file system UNIX i blocchi del disco hanno ampiezza di 1Kbyte e gli i-node,che occupano 1 blocco, contengono 10 indirizzi diretti (numerati da 0 a 9) e gli indirizzi indiretti semplice, doppio e triplo, numerati rispettivamente 10, 11 e 12.. Si consideri il file individuato dal pathname *SistemiOperativi*/*SO/appunti,* relativo alla directory corrente, e si facciano le seguenti ipotesi:

- la directory corrente è *Francese* ed è caricata in memoria
- la directory *SO,* che occupa 1 blocco, e il file *appunti* risiedono su disco.
- gli i-node della directory *SO* e quello del file *appunti* risiedono su disco, rispettivamente nei blocchi 15 e 88.
- lo *i-node* della directory *SO* contiene un solo indirizzo diretto, con valore *27.*
- gli indirizzi contenuti nello i-node del file *appunti* hanno i valori:

Indirizzo	0	1	2	3	4	5	6	7	8	9	10	11	12
Valore	51	52	53	70	72	73	75	100	101	76	96	∅	∅

- gli indirizzi contenuti nel blocco indiretto semplice hanno i valori:

Indirizzo	0	1	2	3	4	5	6	7	8	9	……..
Valore	42	64	65	66	78	90	91	93	75	∅	

- il valore corrente del puntatore di lettura è 8716 e la lunghezza corrente del file è di 18632 byte.

In questa situazione, l'utente *Francesco*, che ha i necessari diritti, esegue una chiamata di sistema per leggere 9000 caratteri dal file *appunti.* Si chiede:

1. quanti e quali blocchi del disco vengono letti per completare l'esecuzione;
2. quanti caratteri vengono estratti da ciascun blocco dati del file *appunti* che viene letto per eseguire l'operazione.

SOLUZIONE

Il puntatore di lettura è posizionato sul carattere 8700 *mod* 1024= 508 del blocco logico 8700 *div* 1024= 8, indirizzato dall'indirizzo diretto 8.

L'ultimo carattere da leggere ha indice 8700+ 9000= 17700 e corrisponde al carattere 292 del blocco logico 17, individuato dall'indirizzo 7 del blocco indiretto semplice. Pertanto:

1. Per completare l'operazione vengono letti i seguenti blocchi:
   - blocco 15 , che contiene lo i-node di *SO*
   - blocco 27, che contiene la directory *SO*
   - blocco 88 , che contiene lo i-node di *appunti*
   - blocchi 101 e 76, indirizzati dagli indirizzi diretti 8 e 9
   - blocco 96 , che contiene il blocco indiretto semplice
   - blocchi 42, 64, 65, 66, 78, 90, 91, 93 indirizzati dai primi 8 indirizzi del blocco indiretto semplice.
2. Caratteri estratti dai blocchi dati del file *appunti* che sono interessati alla lettura:
   - 516 caratteri dal blocco 101
   - 1024 caratteri dal blocco 76
   - 1024 caratteri da ciascuno del blocchi 42, 64, 65, 66, 78, 90, 91
   - 293 caratteri dal blocco 93

SISTEMI OPERATIVI, CORSI A e B - SECONDO APPELLO 2008 - 1/2/2008

ESERCIZIO B-4 (2 PUNTI)

In sistema che gestisce la memoria con paginazione a domanda e utilizza l'algoritmo di sostituzione *Second Chance* globale, sono presenti i processi A, B, C, D. Lo stato della memoria fisica è descritto dalla seguente *Core Map,* dove ogni elemento ha campi *Proc* (processo a cui è assegnato il blocco); *Pag* (pagina del processo caricata nel blocco), *R* (bit di pagina riferita). Si trascurano i blocchi assegnati al sistema operativo.

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

Core Map al tempo t

Al tempo t il puntatore è posizionato sul blocco 14 e il processo C riferisce la pagina 8. Al tempo t+1 continua ad avanzare il processo C che riferisce la pagina 4. Quale è la posizione finale del puntatore e come si modifica la core map?
Nota: dopo ogni intervento, il puntatore si posiziona sul blocco successivo a quello che conteneva la vittima.

SOLUZIONE

1. Posizione del puntatore dopo l'intervento di *Second Chance* al tempo t : 17
2. Posizione finale del puntatore (dopo l'intervento di *Second Chance* al tempo t+ 1)*:* 18
3. Configurazione finale della *Core Map*:

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

Core Map al tempo t+1

ESERCIZIO B-5 (2 PUNTI)

In un file system FAT è definito il file *Appunti/Esercizi.* Il contenuto parziale dell'elemento della directory *Appunti* che lo individua è *(Esercizi, 2).*
Il disco che ospita il file system ha 30 blocchi. Il file *Esercizi* occupa 10 blocchi, individuati dalla FAT, il cui contenuto parziale (per la parte che descrive questo file) è il seguente:

		11		5	6	16					12	28				24								Ø				29	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

La FAT risiede su disco e occupa 3 blocchi, che contengono rispettivamente gli elementi (0 .. 9), (10 .. 19/ e (20 .. 29) della FAT. Il gestore del disco ha un buffer di un blocco. Per scandire la FAT, si copia nel buffer il blocco che contiene il primo elemento da scandire, si continua la scansione nel buffer finché gli elementi da percorrere si trovano al suo interno, dopo di che si copia nel buffer il blocco che contiene l'elemento successivo, e così via fino a completare la scansione.
Quanti accessi al disco sono necessari per caricare in memoria il blocco fisico corrispondente al blocco logico 8 del file *Esercizi* partendo dalla directory *Appunti*, supponendo che questa directory sia caricata in memoria?

SOLUZIONE

Per trovare l'indice del blocco fisico che contiene il blocco logico 8 si devono scandire in sequenza gli elementi 2 (contenuto nel primo blocco assegnato alla FAT), 11, 12 (contenuti nel secondo blocco assegnato alla FAT), 28, 29 (contenuti nel terzo blocco assegnato alla FAT), 4,5 (contenuto nel primo blocco assegnato alla FAT).

Quindi si devono eseguire gli accessi per leggere i seguenti blocchi:

1. blocco che contiene l'elemento **2** della FAT
2. blocco che contiene gli elementi **11, 12** della FAT
3. blocco che contiene gli elementi **28,29** della FAT
4. blocco che contiene gli elementi **4,5** della FAT
5. blocco fisico 6, corrispondente al blocco logico 8 e identificato dall'elemento 5 della FAT.

In totale 5 accessi al disco.