

COGNOME E NOME ..... MATRICOLA .....;..... AULA A FILA POSTO

**ESERCIZIO A-1 (4 punti)**

Un parcheggio per auto della capacità di 100 posti è dotato di un unico ingresso e un'unica uscita. L'ingresso e l'uscita sono controllate da sbarre. Le auto sono thread di uno stesso processo, realizzati a livello utente, che si sincronizzano mediante *lock* e *unlock* con chiave *MutexParcheggio* inizializzata al valore 1, e condividono la variabile *PostiDisponibili*. Per ogni thread è definita la variabile privata *SbarraAbbassata*, con valore iniziale 1.

Lo scheduling dei thread avviene con politica *Round Robin*, con quanto di tempo uguale a 10. Inoltre i thread possono rilasciare il processore invocando la funzione *ThreadYield*.

I protocolli *lock* e *unlock* sono definiti nel modo seguente (il valore 0 della chiave K instaura la mutua esclusione)

<i>lock(K)</i>	<i>unlock(K)</i>
loop      TSL K, R1 JNZ R1, SezCrit CALL ThreadYield JMP loop SezCr      .....	MOV K, #1

Al tempo 0 sono presenti i thread A e B, che eseguono il codice riportato in tabella. Per ogni thread e per ogni riga di codice, l'ultima colonna specifica il tempo di esecuzione della riga.

Op	Thread A	t	Op	Thread B	t
A1	while SbarraAbbassata== 1 {	1	B1	while SbarraAbbassata== 1 {	1
A2	lock (MutexParcheggio);	6	B2	lock (MutexParcheggio);	6
A3	if PostiDisponibili> 0 {	1	B3	if PostiDisponibili> 0 {	1
A4	PostiDisponibili --;	1	B4	PostiDisponibili --;	1
A5	SbarraAbbassata== 0;	1	B5	SbarraAbbassata== 0;	1
A6	} unlock (MutexParcheggio); }	1	B6	} unlock (MutexParcheggio); }	1
A7	<auto parcheggiata>	1000	B7	<auto parcheggiata>	5000
A8	lock (MutexParcheggio);	6	B8	lock (MutexParcheggio);	6
A9	PostiDisponibili ++;	1	B9	PostiDisponibili ++;	1
A10	unlock (MutexParcheggio);	1	B10	unlock (MutexParcheggio);	1

Al tempo 0 il thread A è in esecuzione e il thread B è pronto. Il thread A deve eseguire la riga A1 e la variabile *SbarraAbbassata* è inizializzata ad 1, mentre la prossima istruzione del thread B è la B1 e anche per il thread B vale *SbarraAbbassata=1*. Inoltre, sempre al tempo 0, si ha *PostiDisponibili = 100* e *MutexParcheggio= 1*.

Supponendo che per un tempo sufficientemente lungo non vengano creati altri thread, si chiede:

- a quale tempol'auto controllata dal thread B occupa un posto nel parcheggio (inizio dell'esecuzione della linea B7);
- l'evoluzione del sistema fino al tempo in cui l'auto controllata dal thread B occupa un posto nel parcheggio. Utilizzare la tabella, dove la prima colonna indica *il tempo di inizio* dell'esecuzione di una linea di codice.

**SOLUZIONE**

Evoluzione del sistema a partire dal tempo  $t=0$  e fino al tempo in cui l'auto controllata dal thread B occupa un posto nel parcheggio (inizio dell'esecuzione della linea B7):

Tempo	PostiDisponibili=	MutexParcheggio=	Thread A		Thread B	
			Stato	Esegue riga	Stato	Esegue riga
0	100	1	Esec	A1	Pronto	
1	100	1	Esec	A2	Pronto	
7	100	0	Esec	A3	Pronto	
8	100	0	Esec	A4	Pronto	
9	99	0	Esec	A5	Pronto	
10	99	0	Pronto		Esec	B1
11	99	0	Pronto		Esec	B2
17	99	0	Esec	A6	Pronto	
18-26	99	1	Esec	A7	Pronto	
27	99	1	Pronto		Esec	B2
33	99	0	Pronto		Esec	B3
34	99	0	Pronto		Esec	B4
35	98	0	Pronto		Esec	B5
36	98	0	Pronto		Esec	B6
37-46	98	1	Esec	A7	Pronto	
47	98	1	Pronto		Esec	B7

L'auto controllata dal thread B occupa un posto nel parcheggio al tempo  $t=47$

**ESERCIZIO A-2 (4 punti)**

Un sistema con 4 processi A, B, C, D e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [5, 6, 3, 6] adotta nei confronti dello stallo la politica di riconoscimento ed eliminazione. Al tempo  $t$  dopo aver raggiunto lo stato mostrato nelle seguenti tabelle, i processi A, C e D si sono sospesi per aver richiesto una risorsa di tipo R4 e il processo B si è sospeso per aver richiesto una risorsa di tipo R3. Pertanto il sistema è in stallo.

Assegnazione attuale					Esigenza Massima					Esigenza residua					Molteplicità			
	R1	R2	R3	R4		R1	R2	R3	R4		R1	R2	R3	R4	R1	R2	R3	R4
A	2	0	0	4	A	4	0	0	5	A	2	0	0	1	5	6	3	6
B	1	1	0	0	B	5	6	3	0	B	4	5	3	0				
C	0	0	3	1	C	0	0	4	3	C	0	0	1	2	Disponibilità			
D	0	3	0	1	D	1	3	0	5	D	1	0	0	4	2	2	0	0

Per l'eliminazione dello stallo si adottano, in alternativa, i seguenti provvedimenti (da considerarsi in alternativa) elimina lo stallo:

- si sopprime il processo C
- il processo D viene forzato a rilasciare 1 risorsa di tipo R4

**SOLUZIONE**

- Stato raggiunto dopo l'eliminazione del processo C:

Assegnazione attuale					Esigenza Massima					Esigenza residua					Molteplicità			
	R1	R2	R3	R4		R1	R2	R3	R4		R1	R2	R3	R4	R1	R2	R3	R4
A	2	0	0	4	A	4	0	0	5	A	2	0	0	1	5	6	3	6
B	1	1	0	0	B	5	6	3	0	B	4	5	3	0				
C					C					C					Disponibilità			
D	0	3	0	1	D	1	3	0	5	D	1	0	0	4	2	2	3	1

# SISTEMI OPERATIVI, CORSI A e B Prova del 17/7/2008 CORSO [A] [B]

La soppressione del processo C elimina lo stallo. Infatti esiste una sequenza di assegnazioni che fa terminare tutti i processi a partire dallo stato della precedente tabella:

- 1) Il processo A può terminare  
La disponibilità di {R1, R2, R3, R4} diviene {4, 2, 3, 5}
- 2) Il processo D può terminare  
La disponibilità di {R1, R2, R3, R4} diviene {4, 5, 3, 6}
- 3) Il processo B può terminare  
La disponibilità di {R1, R2, R3, R4} diviene {5, 6, 3, 6}

b) Stato raggiunto dopo che si è forzato il rilascio di 1 risorsa di tipo R4 da parte del processo D

Assegnazione attuale					Esigenza Massima					Esigenza residua					Molteplicità			
	R1	R2	R3	R4		R1	R2	R3	R4		R1	R2	R3	R4	R1	R2	R3	R4
A	2	0	0	4	A	4	0	0	5	A	2	0	0	1	5	6	3	6
B	1	1	0	0	B	5	6	3	0	B	4	5	3	0				
C	0	0	3	1	C	0	0	4	3	C	0	0	1	2	Disponibilità			
D	0	3	0	0	D	1	3	0	5	D	1	0	0	5	2	2	0	1

Forzando il rilascio di una risorsa di tipo R4 da parte del processo D non elimina lo stallo. Infatti non esiste una sequenza di assegnazioni che faccia terminare tutti i processi a partire dallo stato della precedente tabella:

- 1) Il processo A può terminare  
La disponibilità di {R1, R2, R3, R4} diviene {4, 2, 0, 5}
- 2) Il processo D può terminare  
La disponibilità di {R1, R2, R3, R4} diviene {4, 5, 0, 5}
- 3) Il processo B non può terminare; il processo C non può terminare.

## ESERCIZIO A-3 (3 punti)

In un sistema vengono generati i processi A,B,C,D, con i tempi di arrivo e le durate (tempo di utilizzo del processore necessario per arrivare alla terminazione) sotto specificate:

Processo	Durata	Tempo di generazione
A	25	0
B	40	1
C	5	2
D	15	3
E	10	4

Tutti i tempi sono espressi in millisecondi

La politica di gestione del processore è la Round-Robin, con quanto di tempo pari a 10 msec e con tempo di commutazione di contesto trascurabile. Se un processo si sospende, alla riattivazione ottiene un intero quanto di tempo, indipendentemente dalla frazione consumata prima della sospensione.

E' inoltre definito il semaforo *sem*, con valore iniziale 0 e coda inizialmente vuota, sul quale vengono eseguite le seguenti operazioni:

- al tempo 12 il processo in esecuzione esegue *wait(sem)*;
- al tempo 28 il processo in esecuzione esegue *wait(sem)*;
- al tempo 36 il processo in esecuzione esegue *signal(sem)*
- al tempo 40 il processo in esecuzione esegue *signal(sem)*.
- al tempo 50 il processo in esecuzione esegue *signal(sem)*.

Si chiede di compilare la tabella riportata nello schema di soluzione fino al tempo 50, riportando in ogni riga lo stato raggiunto dal sistema (processo in esecuzione, contenuto della Coda Pronti, valore di *sem*, contenuto della coda di *sem*) subito dopo aver gestito l'evento che si verifica al tempo indicato nella prima colonna.

## SOLUZIONE

(indicare tra parentesi tonde il tempo residuo di esecuzione del processo)

Tempo	Evento	Proc. in esecuzione	Coda Pronti	Valore di <i>sem</i>	Coda di <i>sem</i>
0	Generato A	A(25)	∅	0	∅

10	Scade QdiT	B(40)	C(5)→D(15)→E(10)→A(15)	0	∅
12	wait(sem)	C(5)	D(15)→E(10)→A(15)	0	B(38)
17	Ternina C	D(15)	E(10)→A(15)	0	B(38)
27	Scade QdiT	E(10)	A(15)→D(5)	0	B(38)
28	wait(sem)	A(15)	D(5)	0	B(38)→E(9)
36	signal(sem)	A(7)	D(5)→B(38)	0	E(9)
38	Scade QdiT	D(5)	B(38)→A(5)	0	E(9)
40	signal(sem)	D(3)	B(38)→A(5)→E(9)	0	∅
43	Termina D	B(38)	A(5)→E(9)	0	∅
50	signal(sem)	B(31)	A(5)→E(9)	1	∅

### ESERCIZIO A.4 (2 punti)

Per ognuna delle seguenti informazioni indicare la struttura dati di un sistema Unix che la include (marcare la casella corrispondente con una X):

	signal handler array	pending signal bitmap	text structure	process structure	user structure
Associazione tra segnali e handler	X				X
Copia dei registri della CPU					X
PID del processo				X	
Memoria allocata per il codice del processo			X		
Segnali pendenti		X		X	
Stato del processo				X	

### ESERCIZIO A.5 (2 punti)

In un sistema UNIX, un processo genera un figlio eseguendo il seguente frammento di codice:

```
printf("UNO");
pid=fork();
if (pid==0)
{
    execl("xxx",NULL);
    printf("DUE");
    exit(1);
}
else if (pid>0)
{
    pid=wait(&status);
    printf("%n",status);
    exit(2);
}
printf("TRE");
```

Che cosa stampano per effetto di questo codice il processo padre e il processo figlio se la *fork* ha successo e la *exec* fallisce?

**SOLUZIONE**

Il padre stampa: "UNO1"


Il figlio stampa: "DUE"

**ESERCIZIO B-1 (4 punti)**

Un sistema operativo che gestisce la memoria con paginazione a domanda, utilizza l'algoritmo di sostituzione *Second Chance* in una versione che tiene conto, oltre che del bit di riferimento, anche del bit di modifica delle pagine caricate in memoria. Gli elementi della *CoreMap* hanno i seguenti campi:

- *Proc*: processo a cui è assegnato il blocco; il campo è vuoto se il blocco è libero;
- *Pag* indice della pagina caricata nel blocco;
- *Rif*: bit di pagina riferita: valore 1 assegnato dall'hardware quando la pagina viene riferita in lettura o scrittura, valore 0 assegnato da *Second Chance*;
- *Mod*: bit di pagina modificata: valore 1 assegnato dall'hardware quando la pagina viene riferita in scrittura, valore 0 assegnato dal sistema operativo quando completa il salvataggio della pagina in memoria secondaria, a seguito di un comando generato da *Second Chance*.

Al tempo  $t$  la *Core Map* ha la configurazione mostrata in figura, con il puntatore dell'algoritmo di *SecondChance* posizionato sul blocco 11. Sono ignorati i blocchi di memoria occupati dal sistema operativo.



Proc	D	D	A	A	A	B	C	A	B	A	B	B	C	D	D	C	D	B	A	B	C	C	D	C
Pag	0	8	5	10	12	7	0	1	0	2	9	6	3	1	2	10	6	2	7	3	12	7	11	2
Rif	1	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	0	
Mod	1	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	1	1	0	0	0	1	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

Core Map al tempo  $t$ 

L'algoritmo *Second Chance* percorre ciclicamente gli elementi della *CoreMap* a partire da quello sul quale è inizialmente posizionato il puntatore; esamina i bit *Rif* e *Mod* dell'elemento corrente, e:

- se *Rif* = 1 assegna *Rif* = 0; se *Mod* = 1 comanda il salvataggio in memoria secondaria della pagina caricata nel blocco corrente; quindi posiziona il puntatore sull'elemento successivo della *CoreMap* e continua la scansione;
- se *Rif* = 0 e *Mod* = 0 sceglie la pagina corrente come *vittima*; quindi posiziona il puntatore sull'elemento successivo della *CoreMap* e termina.


E' da notare che le pagine per le quali è stato richiesto il salvataggio in memoria secondaria restano marcate come modificate finchè l'operazione di salvataggio non è stata completata.

A partire dal tempo  $t$  si verifica la seguente sequenza di eventi:

Tempo	Evento	Page Fault	Tempo	Evento	Page Fault
$t+1$	Proc C esegue lettura di pag 2		$t+5$	Proc A esegue lettura di pag 2	
$t+2$	Proc C esegue scrittura di pag 0		$t+6$	Termina salvataggio di D-1, D-2, D-8	
$t+3$	Proc C esegue scrittura di pag 3		$t+7$	Proc A esegue scrittura di pag 10	
$t+4$	Proc C esegue lettura di pag 5	SI	$t+8$	Proc A esegue scrittura di pag 4	SI


Si chiede la configurazione della *CoreMap* subito dopo la gestione degli eventi che si verificano ai tempi  $t+3$ ,  $t+4$ ;  $t+7$  e  $t+8$ .

## SOLUZIONE




Proc	D	D	A	A	A	B	C	A	B	A	B	B	C	D	D	C	D	B	A	B	C	C	D	C
Pag	0	8	5	10	12	7	0	1	0	2	9	6	3	1	2	10	6	2	7	3	12	7	11	2
Rif	1	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1
Mod	1	1	0	0	1	0	1	1	0	0	0	1	1	1	1	0	1	1	0	0	0	1	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

Core Map dopo la gestione dell'evento che si verifica al tempo t+3




Proc	D	D	A	A	A	B	C	A	B	A	B	B	C	D	D	C	D	B	A	C	C	C	D	C
Pag	0	8	5	10	12	7	0	1	0	2	9	6	3	1	2	10	6	2	7	5	12	7	11	2
Rif	1	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1
Mod	1	1	0	0	1	0	1	1	0	0	0	1	1	1	1	0	1	1	0	0	0	1	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

Core Map dopo la gestione dell'evento che si verifica al tempo t+4



Proc	D	D	A	A	A	B	C	A	B	A	B	B	C	D	D	C	D	B	A	C	C	C	D	C
Pag	0	8	5	10	12	7	0	1	0	2	9	6	3	1	2	10	6	2	7	5	12	7	11	2
Rif	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1
Mod	1	0	0	1	1	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	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

Core Map dopo la gestione dell'evento che si verifica al tempo t+7



Proc	D	A	A	A	A	B	C	A	B	A	B	B	C	D	D	C	D	B	A	C	C	C	D	C
Pag	0	4	5	10	12	7	0	1	0	2	9	6	3	1	2	10	6	2	7	5	12	7	11	2
Rif	0	1	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0
Mod	1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	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

Core Map dopo la gestione dell'evento che si verifica al tempo t+8

## ESERCIZIO B-2 (4 punti)

In un File System UNIX gli i-node contengono 10 indirizzi di blocchi diretti e 2 indirizzi indiretti, rispettivamente per l'indirizzamento indiretto semplice e per l'indirizzamento indiretto doppio.

I blocchi hanno ampiezza di 1024 byte ( $= 2^{10}$  byte); i puntatori ai blocchi sono codificati con 3 byte.

I puntatori indiretti e quelli contenuti in ciascun blocco indiretto sono indicizzati a partire da 0.

Ricordando che la chiamata *seek(fd, off)* posiziona il puntatore di lettura del file *fd* sul carattere di indice *off* del file. Si chiede:

- il numero di blocchi indirizzabili e la massima capacità del disco;
- il numero di puntatori contenuti in ogni blocco indiretto;
- a quali blocchi fisici si deve accedere per eseguire le seguenti operazioni sul file con *fd= 15*:
  - seek(15, 7168); read(15, &buffer, 10)*
  - seek(15, 15360); read(15, &buffer, 100)*
  - seek(15, 718848); read(15, &buffer, 40)*

## SOLUZIONE

- Possono essere indirizzati  $2^{24}$  blocchi di  $2^{10}$  byte ;  
pertanto la massima capacità disco è di  $2^{34}$  byte (16 Gbyte);
- Ogni blocco indiretto contiene  $\lfloor 2^{10} / 3 \rfloor = 341$  indirizzi ;
- Si deve accedere ai seguenti blocchi fisici:
  - per eseguire le operazioni *seek(7168); read(15, &buffer, 10)*, il puntatore di lettura viene posizionato sul carattere 0 del blocco logico 7  
questo blocco logico è indirizzato (considerare una delle seguenti alternative):
    - dal puntatore diretto di indice 7
    - dal puntatore indiretto semplice e dall'indirizzo di indice 5 del blocco indiretto
    - dal puntatore indiretto doppio, e dall'indirizzo di indice 10 del blocco indiretto di secondo livello,

a sua volta indirizzato dall'indirizzo 0 del blocco indiretto di primo livello

- 3-b) per eseguire le operazioni *seek(15360); read(15, &buffer, 100)*, il puntatore di lettura viene posizionato sul carattere 0 del blocco logico 15  
questo blocco logico è indirizzato (considerare una delle seguenti alternative):
- dal puntatore diretto di indice
  - **dal puntatore indiretto semplice e dall'indirizzo di indice 5 del blocco indiretto**
  - dal puntatore indiretto doppio, e dall'indirizzo di indice del blocco indiretto di secondo livello, a sua volta indirizzato dall'indirizzo del blocco indiretto di primo livello
- 3-c) per eseguire le operazioni *seek(718848); read(15, &buffer, 40)*, il puntatore di lettura viene posizionato sul carattere 0 del blocco logico 702  
questo blocco logico è indirizzato (considerare una delle seguenti alternative):
- dal puntatore diretto di indice
  - dal puntatore indiretto semplice e dall'indirizzo di indice del blocco indiretto
  - **dal puntatore indiretto doppio, e dall'indirizzo di indice 10 del blocco indiretto di secondo livello, a sua volta indirizzato dall'indirizzo 1 del blocco indiretto di primo livello.**

**ESERCIZIO B-3 (3 punti)**

Si consideri un sistema dove gli indirizzi logici hanno la lunghezza di 32 bit e le pagine logiche e fisiche hanno ampiezza di 8 kByte. Per la gestione della memoria con paginazione dinamica si utilizzano tabelle delle pagine a 2 livelli. La tabella di primo livello comprende  $2^8$  elementi.

Gli elementi di ogni tabella di primo o secondo livello hanno una lunghezza di 4 byte. Un byte è riservato agli indicatori (pagina caricata, riferita, modificata, ecc.) mentre i rimanenti codificano un indice di blocco fisico.

Si chiede:

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

Inoltre, supponendo che la tabella di primo livello sia caricata in memoria mentre nessuna delle tabelle di secondo livello sia caricata e che si riferiscano le seguenti pagine logiche, le cui informazioni per la traduzione non sono contenute nella cache della MMU:

- a) pagina 8202 =  $2^{13} + 10$
- b) pagina 24601 =  $3 \cdot 2^{13} + 25$
- c) pagina 8392 =  $2^{13} + 200$
- d) pagina 74178 =  $9 \cdot 2^{13} + 450$

si chiede quali tabelle di secondo livello devono essere caricate in memoria per portare a termine la traduzione degli indirizzi.

**SOLUZIONE**

1. lunghezza del campo offset : 13 bit
2. per la codifica degli indici di blocco sono disponibili 19 bit, di cui 8 indirizzano la tabella di primo livello  
numero di elementi contenuti in ogni tabella di secondo livello:  $2^{11}$  elementi
3. spazio occupato in memoria da ogni tabella di secondo livello :  $2^{11} \cdot 4 = 2^{13}$  byte
4. gli indici dei blocchi di memoria fisica sono codificati con 3 byte  $\rightarrow$  24 bit  
massima dimensione della memoria fisica :  $2^{24}$  blocchi  $\rightarrow 2^{24} \cdot 2^{13} = 2^{37}$  byte  $\rightarrow$  128 Gbyte

Le tabelle di secondo livello da caricare in memoria sono le seguenti:

- a) per il riferimento alla pagina 8202 =  $2^{13} + 10$  si deve caricare la tabella di secondo livello di indice 1
- b) per il riferimento alla pagina 24601 =  $3 \cdot 2^{13} + 25$  si deve caricare la tabella di secondo livello di indice 3
- c) per il riferimento alla pagina 8392 =  $2^{13} + 200$  la tabella di secondo livello di indice 1 è già caricata
- d) per il riferimento alla pagina 74178 =  $9 \cdot 2^{13} + 450$  si deve caricare la tabella di secondo livello di indice 9

**ESERCIZIO B.4 (2 punti)**

Si consideri un disco con 1024 cilindri, 4 facce e 800 settori per traccia.

- a) Quanti settori contiene il disco?
- b) Nell'ipotesi che un settore corrisponda ad un blocco, quale è la minima dimensione dei puntatori ai blocchi che permette l'indicizzazione di tutti i blocchi del disco?
- c) Nell'ipotesi che un settore corrisponda ad un blocco, tradurre i seguenti indici di blocco nelle rispettive terne <cilindro, faccia, settore>:  
2.111.800; 715.710; 19.918;

**SOLUZIONE**

- a) Il disco contiene 3.276.800 settori
- b) 22 bit in quanto la parte intera superiore del logaritmo in base 2 di 3.276.800 è 22
- c) Traduzione degli indici di blocco in terne:  
cilindro =  $\text{IndiceDiBlocco} \text{ div } (4 \cdot 800)$   
faccia =  $(\text{IndiceDiBlocco} \text{ mod } (4 \cdot 800)) \text{ div } 800$   
settore =  $(\text{IndiceDiBlocco} \text{ mod } (4 \cdot 800)) \text{ mod } 800$

Indice di blocco	cilindro	faccia	settore
2.111.800	659	3	600
715.710	223	2	510
19.918	6	0	718

**ESERCIZIO B.5 (2 punti)**

In un sistema UNIX il processo P esegue la chiamata di sistema `int pipe (int fd[2])`, dove `fd[0]` è il descrittore del lato di lettura e `fd[1]` quello del lato di scrittura, e successivamente esegue la chiamata di sistema `fork` generando il processo figlio F1. Successivamente sia P che F1 eseguono la chiamata di sistema `exec` ed infine, al tempo t, F1 esegue la chiamata di sistema `close (fd[0])`.

Date le seguenti istruzioni eseguite in sequenza dopo il tempo t:

1. F1 esegue `int a=write (fd[1], "compito", 7)`
2. P esegue `int a=read (fd[0], "&buf, 10)`
3. P esegue `int a=write (fd[1], "sistemi", 7)`
4. F1 esegue `int a=read (fd[0], "&buf, 10)`

Dire quali di queste sono eseguibili, e quale è il valore della variabile a al loro termine.

**SOLUZIONE**

OPERAZIONE	ESEGUIBILE?	Valore di a:
1.	SI	7
2.	SI	7
3.	SI	7
4.	NO	-