SISTEMI OPERATIVI, CORSI A e B Prova del 1/7/2008 CORSO |A| |B|

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

PARTE A

ESERCIZIO A-1 (4 punti)

Si consideri un processore che dispone dei registri speciali PC (program counter) e PS (program status), di 4 registri generali R1, R2, R3 e R4, dello stack pointer SP e dello stack pointer del nucleo SP'. L'hardware del processore provvede al salvataggio di tutti i registri nello stack del nucleo quando riconosce un'interruzione e al loro ripristino dallo stack del nucleo quando esegue un'istruzione IRET. Inoltre il sistema gestisce il processore con politica a priorità e con prerilascio.
Al tempo t, quando sono presenti nel sistema (tra gli altri) il processo A, in stato di esecuzione, e il processo B che è l'unico processo bloccato in attesa sul semaforo *Sem*, il processo A esegue l'istruzione SVC (Supervisor Call) per invocare la chiamata di sistema *signal(Sem)*.
Immediatamente dopo il tempo *t*, quando l'interruzione generata dalla SVC viene riconosciuta, i registri del processore, i descrittori di A e B e lo stack del nucleo hanno i contenuti mostrati in figura.
Supponendo che il vettore di interruzione associato all'interruzione generata dalla SVC sia 0121, che la parola di stato del nucleo sia AB1F e che il processo A non esaurisca il proprio quanto di tempo, si chiede:

a) 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 SVC;
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 primitiva *signal(Sem)*;
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 A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esec		Stato	Bloccato		……	……		SP	3F00
Priorità	7		Priorità	8		1016	0011		R1	1145
PC	3000		PC	7000		1015			R2	1156
PS	16F2		PS	16F2		1014			R3	1167
SP	4000		SP	8000		1013			R4	1178
R1	1010		R1	209A		1012				
R2	1092		R2	20AB		1011			REG. STATO SUPERV.	
R3	10AB		R3	20CD		1010			SP'	1016
R4	10CD		R4	20DE		1009				
						1008				
PROCESSORE: Registri speciali						1007				
PC	30F9		PS	A7F2						

SOLUZIONE

a) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione eseguita subito dopo la SVC;

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esec		Stato	Bloccato		……	……		SP	3F00
Priorità	7		Priorità	8		1016	0011		R1	1145
PC	3000		PC	7000		1015	30F9		R2	1156
PS	16F2		PS	16F2		1014	A7F2		R3	1167
SP	4000		SP	8000		1013	3F00		R4	1178
R1	1010		R1	209A		1012	1145			
R2	1092		R2	20AB		1011	1156		REG. STATO SUPERV.	
R3	10AB		R3	20CD		1010	1167		SP'	1009
R4	10CD		R4	20DE		1009	1178			
						1008				
PROCESSORE: Registri speciali						1007				
PC	0121		PS	AB1F						

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 primitiva *signal(Sem)*;

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Pronto		Stato	Esecuz.		……	……		SP	?
Priorità	7		Priorità	8		1016	0011		R1	?
PC	30F9		PC	7000		1015	7000		R2	?
PS	A7F2		PS	16F2		1014	16F2		R3	?
SP	3F00		SP	8000		1013	8000		R4	?
R1	1145		R1	209A		1012	209A			
R2	1156		R2	20AB		1011	20AB		REG. STATO SUPERV.	
R3	1167		R3	20CD		1010	20CD		SP'	1009
R4	1178		R4	20DE		1009	20DE			
						1008				
PROCESSORE: Registri speciali						1007				
PC	?		PS	AB1F						

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 A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Pronto		Stato	Esecuz.		……	……		SP	8000
Priorità	7		Priorità	8		1016	0011		R1	209A
PC	30F9		PC	7000		1015			R2	20AB
PS	A7F2		PS	16F2		1014			R3	20CD
SP	3F00		SP	8000		1013			R4	20DE
R1	1145		R1	209A		1012				
R2	1156		R2	20AB		1011			REG. STATO SUPERV.	
R3	1167		R3	20CD		1010			SP'	1016
R4	1178		R4	20DE		1009				
						1008				
PROCESSORE: Registri speciali						1007				
PC	7000		PS	16F2						

ESERCIZIO A-2 (4 punti)

In una banca vi sono 10 sportelli per il servizio ai clienti, situati in una sala d'aspetto. I clienti entrano nella sala d'aspetto, attendono che vi sia uno sportello libero e, quando si verifica questo evento, lo individuano, lo raggiungono e ottengono il servizio voluto. Quindi liberano lo sportello e tornano in sala d'aspetto per uscire dalla banca. Si considerino le seguenti formalizzazioni del problema:

Ipotesi A). I clienti sono thread di uno stesso processo, conformi allo standard POSIX, che si sincronizzano mediante la variabile *mux* di tipo *mutex* (sulla quale sono definite le operazioni `pthread_mutex_lock` e `pthread_mutex_unlock`), e la variabile *SportelloLibero* di tipo *condition* (sulla quale sono definite le operazioni `pthread_cond_wait` e `pthread_cond_signal`). I thread utilizzano inoltre la variabile *SportelliLiberi*, di tipo intero e inizializzata al valore 10.

Ipotesi B). I clienti sono processi che si sincronizzano mediante il semaforo *SportelliLiberi,* inizializzato al valore 10, con le operazioni *wait* e *signal.*

Nelle due ipotesi, inserire le operazioni di sincronizzazione nei due schemi di soluzione riportati di seguito.

SOLUZIONE

Ipotesi a)

```
<il cliente entra nella sala d'aspetto>
pthread_mutex_lock(&Mutex)
while SportelliLiberi == 0 pthread_cond_wait(&SportelloLibero,&Mutex);
//esiste uno sportello libero: il cliente lo individua e lo raggiunge//
SportelliLiberi - -
pthread_mutex_unlock(&Mutex);
<il cliente ottiene il servizio richiesto>
pthread_mutex_lock(&Mutex);
<il cliente lascia lo sportello e torna in sala d'aspetto>
SportelliLiberi ++
pthread_cond_signal(&SportelloLibero);
pthread_mutex_unlock(&Mutex);
<il cliente lascia la sala d'aspetto>
```

Ipotesi b)

```
<il cliente entra nella sala d'aspetto>
wait(SportelliLiberi);
//esiste uno sportello libero: il cliente lo individua e lo raggiunge//
<il cliente ottiene il servizio richiesto>
<il cliente lascia lo sportello e torna in sala d'aspetto>
signal(SportelliLiberi)
<l'utente lascia la sala d'aspetto>
```

ESERCIZIO A-3 (3 punti)

In un sistema con thread *realizzati a livello del nucleo*, dove l'unità schedulabile dal kernel è il thread, sono presenti i processi P1 (con thread T11 e T12) e P2 (con thread T21 e T22). Lo schedulatore schedula i thread secondo la politica round robin, ma può essere invocato esplicitamente dai thread tramite la chiamata *ThreadYield*.

Al tempo *t* è in esecuzione il thread T11, la coda pronti contiene T21->T12 (T21 è in testa alla coda), e il thread T22 è sospeso sul semaforo *Sem*. Inoltre il processo P1 ha una variabile di mutex *Mux* inizializzata a 0 (per cui la mutex è bloccata).

Successivamente si verifica la seguente sequenza di eventi:

a. il thread in esecuzione esegue *lock*(*Mux*)
b. Scade il quanto di tempo
c. Il thread in esecuzione esegue *signal(Sem)*
d. Il thread in esecuzione esegue *unlock*(*Mux*)
e. Il thread in esecuzione esegue *ThreadYield*
f. il thread in esecuzione esegue *lock*(*Mux*)

Si chiede qual'è il thread in esecuzione, la coda pronti, il valore di *Mux* e di *Sem*, e il contenuto della coda di *Sem* dopo ogni evento.

SOLUZIONE

Evento che si verifica al tempo t	Thread in esecuzione	Coda pronti	Valore di *Mux*	Valore di *Sem*	Coda di *Sem*
a. il thread in esecuzione esegue *lock*(*Mux*)	T11	T21->T12	0	0	T22

b. Scade il quanto di tempo	T21	T12->T11	0	0	T22
c. Il thread in esecuzione esegue *signal(Sem)*	T21	T12->T11->T22	0	0	-
d. Il thread in esecuzione esegue *unlock*(*Mux*)	T21	T12->T11->T22	1	0	-
e. Il thread in esecuzione esegue *ThreadYield*	T12	T11->T22->T21	1	0	-
f. il thread in esecuzione esegue *lock*(*Mux*)	T12	T11->T22->T21	0	0	-

ESERCIZIO A-4 (2 punti)

In un sistema con risorse singole R1, R2, R3, R4 e R5 sono presenti i processi P1, P2 e P3, che inizialmente non possiedono risorse e successivamente avanzano utilizzando risorse dell'insieme {R1, R2, R3, R4, R5}.
Ogni processo conosce le risorse che utilizzerà nel corso della propria esistenza.
Nella propria sequenza di avanzamento, che si sovrappone in modo arbitrario con quelle degli altri processi, ciascuno dei processi dell'insieme {P1, P2 , P3} utilizza le risorse secondo le seguenti sequenze:
P1 inizia a utilizzare R3; quindi inizia a utilizzare R1; quindi inizia a utilizzare R4; infine rilascia tutte le risorse.
P2 inizia a utilizzare R1; quindi inizia a utilizzare R5; quindi inizia a utilizzare R3; quindi inizia a utilizzare R4; infine rilascia tutte le risorse.
P3 inizia a utilizzare R4; quindi inizia a utilizzare R2; quindi inizia a utilizzare R5; quindi inizia a utilizzare R3; infine rilascia tutte le risorse.
Si chiede in quale ordine i tre processi devono richiedere le risorse per evitare lo stallo.

SOLUZIONE

Sequenza di richieste che evita lo stallo:

Processo	Prima richiesta	Seconda richiesta	Terza richiesta	Quarta richiesta
P1	R1	R3	R4	
P2	R1	R3	R4	R5
P3	R2	R3	R4	R5

ESERCIZIO A-5 (2 punti)

In un sistema con processi ad ambiente locale sono presenti i processi P1 (con priorità 1), P2 (con priorità 2), e P3 (con priorità 3), I processi comunicano con le primitive *send(&mess, destinatario)* e *receive(&buffer, mittente),* che specificano esplicitamente mittente e destinatario (modello *uno-a-uno).* La primitiva *send* è asincrona; la primitiva *receive* è bloccante.

Al tempo *t* il processo P3 è in esecuzione e la coda pronti contiene (nell'ordine) i processi P2 e P1. Nel canale di comunicazione non vi sono messaggi giacenti.
Lo scheduling del processore avviene con una politica a priorità (va in esecuzione il processo con priorità più elevata).
Si chiede come si modifica lo stato dei processi se si verificano (in alternativa) le seguenti sequenze di eventi:

a) P3 esegue *receive(&buffer, P1);* quindi il processo in esecuzione esegue *receive(&buff, P3);* quindi il processo in esecuzione esegue *send(&mess, P3).*
b) P3 esegue *send(&mess, P2));* quindi il processo in esecuzione esegue *receive(&buffer, P2);* quindi il processo in esecuzione esegue *receive(&buffer, P3).*

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti
a-1	P3 esegue *receive(&buffer, P1)*	P2	P1
a-2	Il processo in esecuzione esegue *receive(&buff, P3)*	P1	∅
a-3	Il processo in esecuzione esegue *send(&mess, P3)*	P3	P1
b-1	P3 esegue *send(&mess, P2)*	P3	P2 → P1
b-2	Il processo in esecuzione esegue *receive(&buffer, P2)*	P2	P1
b-3	Il processo in esecuzione esegue *receive(&buffer, P3)*	P2	P1

PARTE B

ESERCIZIO B-1 (4 punti)

Un sistema simile a Windows gestisce la memoria con paginazione a domanda mediante un *Working Set Manager*. A un certo tempo lo stato di occupazione della memoria (senza considerare i blocchi riservati al sistema operativo) è quello descritto nella seguente Core Map, i cui elementi hanno valore nullo se il blocco è libero, o altrimenti identificano il processo e la pagina a cui il blocco è assegnato.

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

Nel sistema sono presenti i processi A, B e C, le cui tabelle delle pagine sono le seguenti (il campo *TempoRif* registra il ***tempo virtuale del processo*** al quale è avvenuto l'ultimo riferimento alla pagina):

Pagina	Blocco	TempoRif
0	-	
1	15	7
2	-	
3	-	
4	11	5
5	23	9
6	-	
7	26	4
8		
9	10	8
Processo A		

Pagina	Blocco	TempoRif
0	16	10
1	-	
2	25	8
3	-	
4	-	
5	-	
6	19	9
7	-	
8		
9		
Processo B		

Pagina	Blocco	TempoRif
0	-	
1	17	5
2	-	
3	22	6
4	-	
5	24	7
6	-	
7	20	8
8		
9		
Processo C		

A ogni processo è assegnato un *WorkingSet Ammissibile* di 4 pagine. Per ogni pagina riferita da un processo che avanza, si procede nel modo seguente:

- se la pagina è presente in memoria, si scrive il valore attuale del tempo virtuale del processo nel campo *TempoRif* della tabella delle pagine;
- se si verifica un *PageFault*, la pagina riferita viene caricata in un blocco disponibile (anche se il numero di pagine caricate, o *insieme residente*, supera la dimensione del Woring Set Ammissibile), registrando il valore attuale del tempo virtuale del processo nel campo *TermpoRif*. I blocchi disponibili vengono assegnati in ordine crescente di indice.
- Il *Working Set Manager* viene attivato periodicamente o quando il numero di blocchi disponibili si riduce a 0, e si comporta nel modo seguente;
  - considera i soli processi la cui dimensione dell'*insieme residente* (numero di pagine caricate in memoria) superi quella del *WorkingSet Ammissibile*, e per questi processi ordina globalmente le pagine per valori decrescenti del parametro *TempoVirtuale- TempoRif,* dove *TempoVirtuale* è il valore attuale del tempo virtuale del processo;
  - scarica dalla memoria le pagine secondo questo ordinamento, fino a quando il numero di blocchi disponibili diventa uguale a 8. In caso di parità tra due o più pagine si considera l'ordine alfabetico dei processi.

A partire dal tempo considerato il sistema evolve nel modo seguente:

- avanza il processo B, che ai tempi virtuali 11, 12, 13 e 14 riferisce rispettivamente le pagine 0, 1, 2 e 6.
- quindi avanza il processo C, che ai tempi virtuali 9, 10, 11, 12 e 13 riferisce rispettivamente le pagine 0, 1, 2, 5 e 8.
- quindi si attiva il *Working Set Manager*. I tempi virtuali dei processi A, B e C sono rispettivamente 9, 14 e 13.

Si chiede:

1) La configurazione della *CoreMap* e delle tabelle delle pagine dei 3 processi al momento dell'attivazione del *Working Set Manager;*

2) il numero di blocchi liberi al momento dell'attivazione del *Working Set Manager* e il numero di pagine da scaricare;

3) quali sono i processi considerati dal *Working Set Manager* e i valori del parametro *TempoVirtuale- TempoRif* per le pagine di questi processi;

4) quali pagine vengono scaricate dal *Working Set Manager*.

SOLUZIONE

1-1) Configurazione della *CoreMap* al momento dell'attivazione del *Working Set Manager;*

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

1-2) Configurazione delle tabelle delle pagine dei 3 processi al momento dell'attivazione del *Working Set Manager:*

Pagina	Blocco	TempoRif
0		
1	15	7
2		
3		
4	11	5
5	23	9
6		
7	26	4
8		
9	10	8
Processo A		

Pagina	Blocco	TempoRif
0	16	11
1	12	12
2	25	13
3		
4		
5		
6	19	14
7		
8		
9		
Processo B		

Pagina	Blocco	TempoRif
0	13	9
1	17	10
2	14	11
3	22	6
4		
5	24	12
6		
7	20	8
8	18	13
9		
Processo C		

2) Numero di blocchi liberi al momento dell'attivazione del *Working Set Manager:* 4
numero di pagine da scaricare: 4

3-1) processi considerati dal *Working Set Manager:* A e C

3-2) valori del parametro *TempoVirtuale- TempoRif* per le pagine di questi processi;

- Processo A:
  (A,7): parametro *5* ; (A, 4) : parametro *4* ; (A, 1) : parametro *2* ;(A, 9) : parametro *1* ; (A, 5); : parametro *0* ;
- Processo C:
  (C, 3) : parametro *7* ; (C, 7) : parametro *5* ; (C, 0) : parametro *4* ; (C,1) : parametro *3* ; (C, 2) : parametro *2* ; (C, 5) : parametro *1* ; (C, 8) : parametro *0* ;

4) pagine scaricate dal *Working Set Manager:* (C,3), (A,7), (C,7), (A,4).

ESERCIZIO B.2 (4 punti)

Si consideri un RAID di livello 1 composto da 4 dischi fisici D0, D1, D2 e D3, dove i dischi D2 e D3 sono ridondanti. Ogni disco ha 2 facce, 200 settori per traccia e 200 cilindri. Per ogni disco il tempo di seek, proporzionale al numero di cilindri attraversati, è pari a 0,1 ms per ogni cilindro. Il periodo di rotazione è di 20 msec: conseguentemente il tempo impiegato per percorrere un settore è di 0,1 msec.

Quando riceve comandi di scrittura su un dato blocco logico, il controller del RAID invia ai dischi fisici due distinti comandi di scrittura del tipo <*scrittura*, *DiscoNonRidondante*, *cilindro*, *faccia*, *settore*> e <*scrittura*, *DiscoRidondante*, *cilindro*, *faccia*, *settore*>, dove *DiscoNonRidondante*∈{D0,D1} e *DiscoRidondante*= D2 se *DiscoNonRidondante*= D0 *DiscoRidondante*= D3 *se DiscoNonRidondante*= D1. Ogni disco fisico (ridondante o non) serve i comandi on politica SSTF (Shortest Seek Time First).

A un certo tempo (convenzionalmente indicato come t=0) le testine di lettura/scrittura dei dischi D0, D1, D2 e D3 sono posizionate rispettivamente sui cilindri 39, 48, 53 e 190. Inoltre al tempo t sono stati inviati ai dischi fisici, nell'ordine, i seguenti comandi di scrittura:

- disco D0: clindro 24, faccia 0, settore 180
- disco D1: clindro 18, faccia 1, settore 98
- disco D1: clindro 90, faccia 1, settore 54
- disco D0: clindro 36, faccia 0, settore 9
- disco D0: clindro 36, faccia 1, settore 122
- disco D0: clindro 81, faccia 1, settore 168
- disco D0: clindro 66, faccia 0, settore 45
- disco D1: clindro 65, faccia 1, settore 61
- disco D2: clindro 24, faccia 0, settore 180
- disco D3: clindro 18, faccia 1, settore 98
- disco D3: clindro 90, faccia 1, settore 54
- disco D2: clindro 36, faccia 0, settore 9
- disco D2: clindro 36, faccia 1, settore 122
- disco D2: clindro 81, faccia 1, settore 168
- disco D2: clindro 66, faccia 0, settore 45
- disco D3: clindro 65, faccia 1, settore 61

Per ciascun disco fisico, calcolare il tempo necessario per eseguire tutte queste operazioni, supponendo che durante il tempo di esecuzione non arrivino altri comandi.

Il tempo di esecuzione di ogni operazione è uguale alla somma dell'eventuale tempo di *seek,* del ritardo rotazionale (tempo necessario per raggiungere il settore indirizzato) e del tempo di percorrenza del settore indirizzato. Per il ritardo rotazionale dopo un'operazione di *seek* si assume sempre il valore di caso peggiore, pari a un intero periodo di rotazione.

Si assuma inoltre che il controllore sia dotato di capacità di buffering che gli consente di accettare senza ritardo i dati letti dal disco o quelli da scrivere sul disco, e inoltre che i comandi sullo stesso cilindro vengano eseguiti in ordine FIFO.

SOLUZIONE

Disco D0:

op. su cilindro:	36	settore:	9					
inizio:	0	seek:	0,3	rotazione:	20	percorrenza:	0,1	fine: 20,4
op. su cilindro:	36	settore:	122					
inizio:	20,4	seek:	0	rotazione:	11,2	percorrenza:	0,1	fine: 31,7
op. su cilindro:	24	settore:	180					
inizio:	31,7	seek:	1,2	rotazione:	20	percorrenza:	0,1	fine: 53
op. su cilindro:	66	settore:	45					
inizio:	53	seek:	4,2	rotazione:	20	percorrenza:	0,1	fine: 77,3
op. su cilindro:	81	settore:	168					
inizio:	77,3	seek:	1,5	rotazione:	20	percorrenza:	0,1	fine: 98,9

Disco D2:

op. su cilindro:	66	settore:	45					
inizio:	0	seek:	1,3	rotazione:	20	percorrenza:	0,1	fine: 21,4
op. su cilindro:	81	settore:	168					
inizio:	21,4	seek:	1,5	rotazione:	20	percorrenza:	0,1	fine: 43
op. su cilindro:	36	settore:	9					
inizio:	43	seek:	4,5	rotazione:	20	percorrenza:	0,1	fine: 67,6
op. su cilindro:	36	settore:	122					
inizio:	67,6	seek:	0	rotazione:	11,2	percorrenza:	0,1	fine: 78,9
op. su cilindro:	24	settore:	180					
inizio:	78,9	seek:	1,2	rotazione:	20	percorrenza:	0,1	fine: 100,2

Disco D1:

op. su cilindro:	65	settore:	61

inizio:	0	seek:	1,7	rotazione:	20	percorrenza:	0,1	fine:	21,8
op. su cilindro:	90	settore:	54						
inizio:	21,8	seek:	2,5	rotazione:	20	percorrenza:	0,1	fine:	44,4
op. su cilindro:	18	settore:	98						
inizio:	44,4	seek:	7,2	rotazione:	20	percorrenza:	0,1	fine:	71,7

Disco D3:

op. su cilindro:	90	settore:	54						
inizio:	0	seek:	10	rotazione:	20	percorrenza:	0,1	fine:	30,1
op. su cilindro:	65	settore:	61						
inizio:	30,1	seek:	2,5	rotazione:	20	percorrenza:	0,1	fine:	52,7
op. su cilindro:	18	settore:	98						
inizio:	52,7	seek:	4,7	rotazione:	20	percorrenza:	0,1	fine:	77,5

ESERCIZIO B.3 (3 punti)

Un file system rappresenta i file mediante la tecnica della lista concatenata, e ogni descrittore di file (conservato nella directory che include il file), contiene il nome del file e il puntatore al primo blocco del file. I blocchi hanno dimensione pari ad 1 Kbyte. Il file system permette sia l'accesso sequenziale che l'accesso diretto ai file, e offre tra le altre le seguenti chiamate di sistema:

- `fd=open(nomefile)` che apre il file `nomefile` e restituisce il descrittore `fd` necessario per accedere al file. La open posiziona il puntatore di lettura sul carattere di indice 0 del file.
- `read(fd, &buf, lung)` che legge dal file il cui descrittore è `fd` un numero di caratteri pari a `lung` e li copia nel buffer `buf`.
- `seek(fd, pos)` che posiziona il puntatore di lettura del file `fd` sul carattere di indice `pos` del file.

Il file system contiene (tra gli altri) il file A allocato sui blocchi: 44->51->61->39->38->37.
Dire quanti accessi a blocchi del file sul disco sono richiesti dalle seguenti sequenze di operazioni (da considerare in alternativa):

1. `fd=open("A"); seek(fd,4600); read(fd, &buf, 10);`
2. `fd=open("A"); read(fd, &buf, 3000); seek(fd,0);`
3. `fd=open("A"); seek(fd,1500); read(fd, &buf, 10); seek(fd,2100); read(fd,&buf1,10);`

SOLUZIONE

1. `fd=open("A"); seek(fd,4600); read(fd, &buf, 10)`
Il primo carattere da leggere (che è il 4600) e l'ultimo carattere da leggere (che è il 4610) risiedono entrambi nel blocco 4, infatti la parte intera di 4600/1024 e di 4610/1024 è 4. Pertanto è necessario accedere ai blocchi 44,51,61,39,38 e sono quindi necessari 5 accessi al disco.

2. `fd=open("A"); read(fd, &buf, 3000); seek(fd,0);`
Il primo carattere da leggere è nel blocco 0 e l'ultimo carattere da leggere, il 2999, risiede nel blocco 2. Pertanto è necessario accedere ai blocchi 44,51,61 e sono quindi necessari 3 accessi al disco. L'ultima chiamata di seek non ha effetto sul numero di accessi.

3. `fd=open("A"); seek(fd,1500); read(fd, &buf, 10); seek(fd,2100); read(fd,&buf1,10);`
è necessari leggere i caratteri da 1500 a 1510 e da 2100 a 2110. pertanto i blocchi da leggere sono l'1 e il 2. Quindi è necessario accedere ai blocchi 44,51,61 e sono quindi necessari 3 accessi al disco.

ESERCIZIO B-4 (2 punti)
Dire quali tabelle di un processo Unix sono modificate dalle chiamate di sistema:
- `read(fd,&buf,n)` - nel caso in cui la chiamata riesca a leggere con successo `n` caratteri
- `close(fd)` - nel caso in cui il file da chiudere sia in uso dal solo processo che esegue la chiamata di sistema e che la chiamata termini con successo.

SOLUZIONE

	Read: SI/NO	**Close: SI/NO**
Tabella dei file aperti dal processo:	NO	SI
Tabella dei file aperti di sistema:	SI	SI
Tabella dei file attivi:	NO	SI

ESERCIZIO B-5 (2 punti)

In un disco con blocchi di 2 Kbyte (= 2^{11} byte), è definito un file system FAT 16. Ogni elemento ha lunghezza di 2 byte e indirizza un blocco del disco. La copia permanente della FAT risiede nel disco a partire dal blocco di indice 0 e una copia di lavoro viene caricata in memoria principale all'avviamento del sistema operativo.
Supponendo che la FAT sia dimensionata in base alla massima capacità di indirizzamento dei suoi elementi si chiede:

1) il numero di blocchi dati indirizzabili dalla FAT.
2) il numero di byte e di blocchi del disco occupati dalla FAT,
3) l'indice del primo blocco dati nel disco,
4) quale capacità (in blocchi e in byte deve avere il disco) per contenere tutti i blocchi dati indirizzabili.

SOLUZIONE:

1. Il numero di blocchi dati indirizzabili dalla FAT è 2^{16}
2. La FAT ha 2^{16} elementi di 2 byte
   pertanto occupa 2^{17} byte → $2^6 = 64$ blocchi
3. Il primo blocco dati nel disco è quello di indice 64 (i blocchi precedenti sono riservati alla FAT)
4. Per contenere tutti i blocchi indirizzabili, il disco deve avere una capacità di almeno $2^{16} + 2^6$ blocchi → $2^{27} + 2^{17}$ byte.