SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 9/9/2010

Esercizio A-1 (4 punti)

In un sistema Unix sono presenti i Pthread `A`, `B` e `C` che comunicano tramite due pile P1 e P2.
Il Pthread `A` effettua un ciclo nel quale produce una sequenza infinita di numeri interi e li deposita in P1.
Il Pthread `B` effettua un ciclo nel quale preleva due valori da P1 e deposita la loro somma in P2.
Il Pthread `C` effettua un ciclo infinito nel quale preleva un valore da P2 e lo consuma.
La pila P1 è definita come un vettore di 10 elementi puntato dalla variabile p1_top. Se la pila è vuota p1_top è pari a 0, mentre quando la pila è piena p1_top è pari a 10. In modo simile la pila P2 è definita come un vettore di 5 elementi puntato dalla variabile p2_top (p2_top==0 implica che P2 è vuota, mentre p2_top==5 implica che P2 è piena).
Inoltre sono usati:

- La variabile `mutexP1` per gestire la mutua esclusione su P1.
- La variabile `mutexP2` per gestire la mutua esclusione su P2.
- La variabile di condizione `condA` per la sospensione del Pthread A
- La variabile di condizione `condB` per la sospensione del Pthread B
- La variabile di condizione `condC` per la sospensione del Pthread C

Si chiede di completare con le opportune operazioni di sincronizzazione della libreria p_thread lo pseudo codice dei PThread A, B e C.

Soluzione

Pthread A:

```
while (true) {
      <produce un intero d>
      Pthread_mutex_lock(&mutexP1);
      while (p1_top == 10) pthread_cond_wait(&condA,&mutexP1);
      p1_top ++; P1[p1_top] = d;    //inserisce l'elemento d nella pila
      if (p1_top==2) pthread_cond_signal(&condB);
      Pthread_mutex_unlock(&mutexP1);
}
```

Pthread B:

```
while (true) {
      Pthread_mutex_lock(&mutexP1);
      while (p1_top <= 1) pthread_cond_wait(&condB,&mutexP1);
      x=P1[p1_top]; p1_top --; x=x+P1[p1_top]; p1_top --;
      //preleva dalla pila i primi due elementi e li somma
      pthread_cond_signal(&condA);
      Pthread_mutex_unlock(&mutexP1);
      Pthread_mutex_lock(&mutexP2);
      while (p2_top == 5) pthread_cond_wait(&condB,&mutexP2);
      p2_top ++; P2[p2_top] = x;    //inserisce la somma in testa a P2
      pthread_cond_signal(&condC);
      Pthread_mutex_unlock(&mutexP2);
}
```

Pthread C:

```
while (true) {
      Pthread_mutex_lock(&mutexP2);
      while (p2_top == 0) pthread_cond_wait(&condC,&mutexP2);
      x=P2[p2_top]; //preleva il primo elemento dalla pila P2
      p2_top --;
      pthread_cond_signal(&condB);
      Pthread_mutex_unlock(&mutexP2);
      <consuma x>
}
```

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 9/9/2010 CORSO |A| |B|

Esercizio A-2 (4 punti)

In un sistema con scheduling round robin e quanto di tempo di 5 msec, nel quale sono definite le risorse Q, R e S singole, non prerilasciabili e utilizzabili in mutua esclusione, sono presenti i processi A, B e C. Al tempo t le tre risorse Q, R e S sono disponibili, è in esecuzione il processo A e i processi B e C sono pronti (B è in testa alla coda).
All'istante t, i programmi dei tre processi specificano le seguenti azioni:

Proc. A	Chiede Q	Avanza per 7 msec	Chiede R	Avanza per 9 msec	Rilascia Q,R	Termina		
Proc. B	Chiede S	Avanza per 12 msec	Chiede Q	Avanza per 1 msec	Chiede R	Avanza per 4 msec	Rilascia Q, R, S	Termina
Proc. C	Chiede S	Avanza per 3 msec	Chiede R	Avanza per 1 msec	Chiede Q	Avanza per 8 msec	Rilascia Q, R, S	Termina

I processi sono concorrenti e avanzano con velocità determinate dalle transizioni di stato e dallo scheduling.
Il processo che richiede una risorsa l'ottiene immediatamente se è disponibile; altrimenti si sospende in attesa di ottenerla.
Al rilascio di una risorsa viene riattivato con politica FIFO, se esiste, uno dei processi in attesa di quella risorsa,
Si chiede:

- l'evoluzione del sistema a partire dal tempo t e fino alla terminazione di tutti i processi o fino al raggiungimento di uno stallo;
- Si verifica uno stallo?

Soluzione

	Evento	Stato dei processi			Stato delle risorse		
		Processo in esecuzione	Coda pronti	Processi sospesi	Q	R	S
t	A richiede Q	A	B → C	-	Ass. a A	libera	libera
t+5	Esaurito QdT, B richiede S	B	C → A(2)	-	Ass. a A	libera	Ass. a B
t+10	C si sospende in attesa di S	A(2)	B(7)	C	Ass. a A	libera	Ass. a B
t+12	A richiede R	A(9)	B(7)	C	Ass. a A	Ass. a A	Ass. a B
t+15	Esaurito QdT	B(7)	A(6)	C	Ass. a A	Ass. a A	Ass. a B
t+20	Esaurito QdT	A(6)	B(2)	C	Ass. a A	Ass. a A	Ass. a B
t+25	Esaurito QdT	B(2)	A(1)	C	Ass. a A	Ass. a A	Ass. a B
t+27	B richiede Q e si sospende	A(1)	-	C,B	Ass. a A	Ass. a A	Ass. a B
t+28	A termina	B	-	C	Ass. a B	libera	Ass. a B
t+29	B richiede R	B	-	C	Ass. a B	Ass. a B	Ass. a B
t+33	B termina	C	-	-	libera	libera	Ass. a C
t+36	C richiede R	C	-	-	libera	Ass. a C	Ass. a C
t+37	C richiede Q	C	-	-	Ass. a C	Ass. a C	Ass. a C
t+45	C termina	-	-	-	libera	libera	libera

Si verifica uno stallo? NO

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 9/9/2010

Esercizio A-3 (3 punti)

In un sistema UNIX, vengono eseguite in sequenza le seguenti operazioni:

a) il processo P esegue con sucesso una chiamata *pipe,* che restituisce i descrittori *fa[0]* e *fa[1];*
b) il processo P esegue con successo una *fork,* generando il processo P1 che immediatamente esegue con successo una *exec;*
c) processo P esegue con sucesso una chiamata *pipe,* che restituisce i descrittori *fb[0]* e *fb[1];*
d) il processo P esegue con successo una *fork,* generando il processo P2 che immediatamente esegue con successo una *exec;*
e) il processo P1 scrive nel pipe *fa* la sequenza di caratteri *'apice'* con l'operazione *write(fa[1],'apice', 5);*
f) il processo P2 esegue con successo una *exec;*
g) il processo P2 scrive nel pipe *fb* la sequenza di caratteri *'pedice'* con l'operazione *write(fb[1],'pedice', 6)*
h) il processo P legge 6 caratteri dal pipe *fb* con l'operazione *read(fb[0], &temp, 6)*
i) il processo P scrive nel pipe *fa* la sequenza di caratteri contenuta nel proprio buffer *temp* con l'operazione *write(fa[1], &temp, 6)*
j) il processo P2 legge 9 caratteri dal pipe *fa* con l'operazione *read(fa[0], &dest2, 9)*
k) il processo P1 legge 9 caratteri dal pipe *fa* con l'operazione *read(fa[0], &dest1, 9)*

Dire:
- Qual è il valore della variabile *dest2* del processo P2 dopo l'operazione *j)*?
- Qual è il valore della variabile *dest1* del processo P1 dopo l'operazione *k)*?

Soluzione

Dopo l'operazione *j)*, il contenuto di *dest2* è *'apicepedi'*.
Dopo l'operazione *k)*, il contenuto di *dest1* è *'ce'*.

Esercizio A-4 (2 punti)

In un sistema con thread *realizzati a livello del nucleo*, dove l'unità schedulabile dal kernel è il thread, sono presenti:

a. il processo P1 con i thread T11 (di priorità 3) e T12 (di priorità 4)
b. il processo P2 con i thread T21 (di priorità 5) e T22 (di priorità 1)

Il sistema operativo schedula i thread con una politica a code multiple con 5 valori di priorità (da 1 a 5): una coda FIFO per ogni valore di priorità; i thread pronti di uguale priorità sono inseriti in una stessa coda; il processore viene assegnato al thread che occupa la prima posizione nella coda non vuota di massima priorità; ai thread pronti di uguale priorità si applica la politica Round Robin. Inoltre, se un thread permane in stato di pronto per 20 quanti di tempo , il sistema applica la politica dell'inversione di priorità assegnando a quel thread il valore 5 di priorità per due quanti di tempo.
Nel sistema operativo è definito il semaforo *sem* che, immediatamente prima dell'istante *t* ha valore 0 e ha in coda il thread T21.
Allo stesso istante di tempo è in esecuzione il thread T12 e i thread T11 e T22 sono in stato di pronto da 18 e 19 quanti di tempo, rispettivamente.
Dire quale thread va in esecuzione a partire dal tempo *t* dopo ciascuno degli eventi elencati in tabella che si verificano in sequenza:

Soluzione

	Evento che si verifica al tempo t	Stato dei thread dopo l'evento		
		Tread In esecuzione	Tread Pronti	In attesa su *sem*
(a)	Il thread in esecuzione esegue una wait sul semaforo *sem*	T11	T22	T21 → T12
(b)	Il thread in esecuzione esaurisce il quanto di tempo	T22	T11	T21 → T12
(c)	Il thread in esecuzione esaurisce il quanto di tempo	T22	T11	T21 → T12
(d)	Il thread in esecuzione esegue signal(*sem*)	T22	T21, T11	T12
(e)	Il thread in esecuzione esaurisce il quanto di tempo	T21	T11, T22	T12

Esercizio A-5 (2 punti)

In un sistema tipo Unix, al tempo *t* è presente il solo processo P1, con priorità 4, che è in esecuzione.
Immediatamente prima del tempo *t* il processo P1 ha eseguito una chiamata *pipe,* che ha restituito i descrittori *f[0]* e *f[1]*.
Dopo il tempo *t* si verifica la seguente sequenza di eventi:

1) P1 esegue la chiamata di sistema *fork()* che genera il processo P2;
2) Scade il quanto di tempo per il processo in esecuzione;
3) Il processo in esecuzione esegue la chiamata *read[f[0], &dest, 10);*
4) Il processo in esecuzione esegue la chiamata *write[f[1], &src, 15);*
5) Il processo in esecuzione esegue la chiamata *wait;*
6) Scade il quanto di tempo per il processo in esecuzione;
7) Il processo in esecuzione esegue la chiamata *exit*

Si chiede di specificare lo stato dei processi P1 e P2 dopo ogni evento.

Soluzione

Sequenza di eventi	Stato di P1	Stato di P2
1) P1 esegue *fork()*	Esecuzione	Pronto
2) Scade il quanto di tempo	Pronto	Esecuzione
3)Il processo in esecuzione esegue *read[f[0], &dest, 10*)	Esecuzione	Attesa
4) Il processo in esecuzione esegue *write[f[1], &src, 15)*	Esecuzione	Pronto
5) Il processo in esecuzione esegue *wait*	Attesa	Esecuzione
6) Scade il quanto di tempo	Attesa	Esecuzione
7) Il processo in esecuzione esegue *exit*	Esecuzione	Terminato

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 9/9/2010

Esercizio B-1 (4 punti)

Si consideri un sistema che gestisce la memoria con paginazione dinamica con le seguenti caratteristiche:

- indirizzi logici di 32 bit e ampiezza dello spazio logico di ogni processo pari a 2^{32} byte;
- pagine logiche e blocchi fisici di 1 KByte;
- tabelle delle pagine a due livelli; la tabella di primo livello comprende 2^{10} elementi;
- tutti gli elementi della tabella di primo e di secondo livello hanno lunghezza pari a 32 bit, di cui 10 sono indicatori e i restanti 22 codificano l'indice di un blocco fisico;

In questo sistema è presente il processo P, che ha allocato nello spazio virtuale due aree di memoria:

- Area 1: 2 Mbyte a partire dalla locazione 100* 1024
- Area 2: 4 Mbyte a partire dalla locazione 4096* 1024

Si chiede:

1. La lunghezza, in numero di bit, delle componenti dell'indirizzo logico che indirizzano, rispettivamente, la tabella di primo livello, di secondo livello, e l'offset;
2. Lo spazio occupato in memoria dalla tabella di primo livello e da ogni tabella di secondo livello (in byte);
3. La massima dimensione della memoria fisica (numero di blocchi e di byte, espressi come potenze di 2);
4. Lo spazio indirizzabile da una tabella di secondo livello;
5. Quali elementi della tabella di primo livello sono usati per indirizzare l'Area 1;
6. Quali elementi della tabella/ delle tabelle di secondo livello sono usate per indirizzare l'Area 1;
7. Quali elementi della tabella di primo livello sono usati per indirizzare l'Area 2;
8. Quali elementi della tabella/ delle tabelle di secondo livello sono usate per indirizzare l'Area 2;

Soluzione

1. Lunghezza del campo che indirizza la tabella di 1° livello: 10 bit
   Lunghezza del campo che indirizza la tabella di 2° livello: 12 bit
   Lunghezza del campo offset: 10 bit
2. Spazio occupato in memoria dalla tabella di 1° livello: 4 kByte
   Spazio occupato in memoria dalla tabella di 2° livello: 16 kByte
3. Massima dimensione della memoria fisica: 2^{22} blocchi = 2^{22+10} Bytes
4. Spazio indirizzabile da una tabella di 2° livello: 2^{12+10} Bytes = 4 MBytes
5. L'Area 1 occupa le locazioni comprese tra 100*1024 e (100+2048) *1024 – 1, quindi tra 100 Kbyte (estremo incluso) e 2Mbyte+100Kbyte (estremo escluso). Si tratta di locazioni comprese nei primi 4Mbyte della memoria virtuale, che sono indirizzabili dalla prima tabella di secondo livello. Quindi l'Area 1 è indirizzata dall'elemento 0 della tabella di primo livello.
6. Ogni elemento di una tabella di 2° livello indirizza 1 Kbyte, quindi l'Area 1 è indirizzata dagli elementi della 1° tabella di secondo livello compresi tra: 100 e 100+2048-1
7. L'Area 2 occupa le locazioni comprese tra 4096* 1024 e (4096+4096) *1024 – 1, quindi tra 4 MByte (estremo incluso)e 8MByte (estremo escluso). Si tratta di locazioni comprese nei secondi 4Mbyte della memoria virtuale, che sono indirizzabili dalla seconda tabella di secondo livello. Quindi l'Area 2 è indirizzata dall'elemento 1 della tabella di primo livello.
8. Ogni elemento di una tabella di 2° livello indirizza 1 Kbyte, quindi l'Area 2 è indirizzata dagli elementi della 1° tabella di secondo livello compresi tra 0 e 4096- 1. In altri termini, per indirizzare l'Area 2 si utilizzano tutti gli elementi della tabella di secondo livello di indice 1.

Esercizio B-2 (4 punti)

Un sistema operativo che gestisce la memoria con paginazione a domanda con pagine e blocchi fisici di 1 Kbyte, è dotato di un File System FAT 16 con le seguenti caratteristiche:

- il File System è ospitato da un disco della capacità di 64 Mbyte, con blocchi di 1 Kbyte;
- gli elementi della FAT hanno una lunghezza di 2 byte;
- la FAT risiede stabilmente sul disco ed è logicamente suddivisa in pagine di 1 Kbyte, che vengono caricate in memoria a domanda.

Lo stato di occupazione della memoria è descritto dalla *Core Map,* i cui elementi hanno i campi *Id* (indice di un processo o identificatore della FAT, o *null* se il blocco è libero) *Pag* (pagina del processo o della *FAT* caricata nel blocco), *Rif* (indicatore di pagina riferita) e *Mod* (indicatore di modifica). L'algoritmo di sostituzione utilizzato è il *Second Chance.*
Al tempo *t* sono caricate in memoria pagine dei processi A, B, C e della FAT, e la *CoreMap* ha la configurazione mostrata in figura. Ad esempio il blocco 10 è occupato dalla pagina 98 della *FAT,* con indicatore di riferimento uguale a 0 e indicatore di modifica uguale a 0. I primi 6 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione. Al tempo *t* il puntatore dell'algoritmo *Second Chance* è posizionato sul blocco 10.
Al tempo *t* è in esecuzione il processo A, che esegue una chiamata di sistema per leggere i blocchi logici 1 e 2 del file *filename.* Il file *filename* risiede nei blocchi fisici 7980, 4786, 2430, 5670 e 10000 (corrispondenti ai blocchi logici 0,1,2,3 e 4, rispettivamente). La chiamata di sistema esegue codice del nucleo, che non riferisce pagine soggette a caricamento dinamico, eccetto quelle della *FAT.*

↓

Id							B	C	A	C	FAT	A	B	C	A	FAT	A	C	B	A	FAT	B	FAT	A
Pag							0	1	0	2	98	6	3	8	9	29	12	2	7	3	120	7	77	2
Rif							0	1	0	0	0	1	1	0	1	1	0	0	1	0	1	1	0	0
Mod							0	1	1	0	0	0	1	1	1	0	0	0	0	1	0	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*

Si chiede:

1) il numero di elementi della FAT;
2) il numero di byte e il numero di pagine che compongono la FAT;
3) quali pagine della FAT vengono riferite per leggere i blocchi logici 0 e 1 di *filename;*
4) la configurazione della *Core Map* dopo la lettura del blocco logico 0 di *filename;*
5) la configurazione della *Core Map* dopo la lettura del blocco logico 1 di *filename;*
6) il numero di accessi al disco che vengono eseguiti per trasferire in memoria i due blocchi logici.

Soluzione

1. numero di elementi della FAT: uguale al numero di blocchi del disco → $64*2^{20} / 2^{10} = 64*2^{10}$ elementi;
2. numero di byte e il numero di pagine che compongono la FAT: $64*2^{10} * 2 = 2^{17}$ byte; $2^{7} = 128$ pagine
3. il blocco logico 0 risiede nel blocco fisico 7980, il suo indirizzo è contenuto nell'elemento 7980 della FAT;
   il blocco logico 1 risiede nel blocco fisico 4786, e il suo indirizzo è contenuto nell'elemento 4786 della FAT.
   Quindi è necessario leggere gli elementi 7980 e 4786 della FAT che sono contenuti nelle pagine
   (7980 **div** 1024)=7 e (4786 **div** 1024)=4 della FAT.
4. configurazione della *Core Map* dopo la lettura del blocco logico 0:

↓

Id							B	C	A	C	FAT	A	B	C	A	FAT	A	C	B	A	FAT	B	FAT	A
Pag							0	1	0	2	7	6	3	8	9	29	12	2	7	3	120	7	77	2
Rif							0	1	0	0	1	1	1	0	1	1	0	0	1	0	1	1	0	0
Mod							0	1	1	0	0	0	1	1	1	0	0	0	0	1	0	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

5. configurazione della *Core Map* dopo la lettura del blocco logico 1:

															↓								
Id							B	C	A	C	FAT	A	B	FAT	A	FAT	A	C	B	A	FAT	B	FAT	A
Pag							0	1	0	2	7	6	3	4	9	29	12	2	7	3	120	7	77	2
Rif							0	1	0	0	1	0	0	1	1	1	0	0	1	0	1	1	0	0
Mod							0	1	1	0	0	0	1	0	1	0	0	0	0	1	0	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

6. numero di accessi al disco che vengono eseguiti per eseguire la chiamata di sistema: 5
   Precisamente:
   - 1 accesso per caricare da disco la pagina 7 della FAT
   - 1 accesso per salvare sul disco la pagina 8 del processo C
   - 1 accesso per caricare da disco la pagina 4 della FAT
   - 2 accessi per trasferire in memoria i blocchi fisici 4786 e 2430 di *filename.*

Esercizio B-5 (2 punti)

In un disco con capacità di 100 Gbyte (=100* 2^{30} byte) e blocchi di 2 Kbyte (= 2^{11} byte), è definito un file system di tipo Unix. Ogni i-node occupa 256 byte e contiene, oltre agli attributi del file, 10 puntatori diretti e tre puntatori indiretti (indiretto singolo, doppio e triplo). I puntatori dello i-node e quelli dei blocchi indiretti hanno una lunghezza di 32 bit (4 byte). Lo lunghezza dello i/o pointer è quella necessaria a indirizzare completamente un file della massima lunghezza ammissibile. Il massimo numero di file definibili nel file system è pari a 2^{15}= 32768.
Il boot block e il superblocco occupano, rispettivamente, il blocco 0 e 1 del disco, e la i-list è memorizzata a partire dal blocco 2.
Si chiede:

1. L'indice dell'ultimo blocco del disco occupato dalla i-list;
2. La dimensione della i-list in byte:
3. La massima dimensione di un file in numero di blocchi;
4. La lunghezza dello i/o pointer.

Soluzione

1. Indice dell'ultimo blocco del disco occupato dalla i-list;: 32.769. Infatti la i-list occupa i blocchi di indici compresi tra 2 e 32768- 1 (estremi inclusi).
2. Dimensione della i-list in byte: 2^{15}*2 KByte = 64 MByte
3. Massima dimensione di un file in numero di blocchi: 10+ 2^{11} + 2^{11*2} + 2^{11*3} blocchi < 2^{34} blocchi.
4. Lunghezza dello i/o pointer: 34 bit.

Esercizio B-3 (3 punti)

Un disco con 2 facce, 1000 settori per traccia e 10.000 cilindri ha un tempo di seek uguale alla somma di una componente costante k=0,5 ms e di una componente variabile v= 0,1*x msec, dove x è lil valore assoluto della differenza tra l'indice del cilindro di partenza e quello del cilindro di arrivo. Il periodo di rotazione è di 20 msec: conseguentemente il tempo impiegato per percorrere un settore è di 0,02 msec.
A un certo tempo (convenzionalmente indicato come t=0) termina l'esecuzione dei comandi sul cilindro 4750 e sono pervenute, nell'ordine, le seguenti richieste di lettura o scrittura:

- clindro 9800, faccia 0, settore 555
- clindro 2100, faccia 1, settore 56
- clindro 5670, faccia 0, settore 100
- clindro 6009, faccia 1, settore 920

Inoltre:
Al tempo 183 arriva un comando di lettura sul cilindro 5999, faccia 1, settore 656
Al tempo 184 arrivano un comando di lettura sul cilindro 7210, faccia 1, settore 111

Ricordando che il tempo di esecuzione di ogni operazione è uguale alla somma dell'eventuale tempo di *seek,* del ritardo rotazionale e del tempo di percorrenza del settore indirizzato, calcolare il tempo necessario per eseguire tutte queste operazioni supponendo che si adotti la politica di SCAN e che la fase attiva al tempo t=0 sia quella di salita.
Il controllore è dotato di sufficiente capacità di buffering ed è sempre in grado di accettare senza ritardo i dati da leggere o da scrivere sul disco.
Per il ritardo rotazionale dopo un'operazione di *seek* si assume sempre il valore di caso peggiore, pari a un intero periodo di rotazione (20 msec).

Soluzione

op. su cilindro:	5670	settore:	100						
inizio:	0	seek:	92,5	rotazione:	20	percorrenza:	0,02	fine:	112,52
op. su cilindro:	6009	settore:	920						
inizio:	112,52	seek:	34,4	rotazione:	20	percorrenza:	0,02	fine:	166,94
op. su cilindro:	9800	settore:	555						
inizio:	166,94	seek:	379,6	rotazione:	20	percorrenza:	0,02	fine:	566,56
op. su cilindro:	7210	settore:	111						
inizio:	566,56	seek:	259,5	rotazione:	20	percorrenza:	0,02	fine:	846,08
op. su cilindro:	5999	settore:	656						
inizio:	846,08	seek:	121,6	rotazione:	20	percorrenza:	0,02	fine:	987,7
op. su cilindro:	2100	settore:	56						
inizio:	987,7	seek:	390,4	rotazione:	20	percorrenza:	0,02	fine:	1398,12

Esercizio B-4 (2 punti)

Quando viene eseguita una chiamata di sistema *read* per la lettura da un file di Unix, quali tabelle del sistema possono essere modificate?

Soluzione

ABELLA DI UNIX: MODIFICATA?	**SI/NO**
process structure:	NO
user structure:	NO
Tabella dei file aperti dal processo:	NO
Tabella dei file aperti di sistema:	SI
Tabella dei file attivi:	NO