SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 16/1/2010 CORSO |A| |B|

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

ESERCIZIO A-1 (4 punti)

In una strada urbana è presente un semaforo, che consente di alternare il transito delle auto e l'attraversamento dei pedoni. Le auto e i pedoni sono thread realizzati a livello kernel e conformi allo standard POSIX. I thread si sincronizzano mediante il meccanismo *mutex* (sul quale sono definite le operazioni pthread_mutex_lock e pthread_mutex_unlock), e il meccanismo *condition* (sul quale sono definite le operazioni pthread_cond_wait e pthread_cond_signal).

Si adotta la seguente politica:

- il generico pedone che giunge al semaforo può iniziare ad attraversare se non vi sono auto in transito (semaforo verde per i pedoni e rosso per le auto); altrimenti si mette in attesa. Quando il pedone ha attraversato, se non vi sono altri pedoni che hanno iniziato ad attraversare e vi sono auto in attesa di transitare, tutte le auto in attesa iniziano a transitare (il semaforo diviene rosso per i pedoni e verde per le auto).
- la generica auto che giunge al semaforo può iniziare a transitare se non vi sono pedoni che stanno attraversando e il numero di pedoni in attesa non supera il valore *MaxCoda*. (semaforo rosso per i pedoni e verde per le auto); altrimenti si mette in attesa. Quando l'auto è transitata, se non vi sono altre auto che hanno iniziato a transitare e vi sono pedoni in attesa di attraversare, tutti i pedoni in attesa iniziano ad attraversare (il semaforo diviene verde per i pedoni e rosso per le auto).

I thread condividono la variabile *AutoInTransito, AutoInAttesa, PedoniInTransito* e *PedoniInAttesa,* tutte con valore iniziale *0*, e inoltre la variabile *MutexAttraversamento* di tipo *mutex*, e le variabili *AttesaPedoni* e *AttesaAuto,* di tipo *condition.*

Si chiede di risolvere il problema introducendo le opportune operazioni di sincronizzazione nello pseudo codice del generico pedone e in quello della generica auto (vi sono un numero arbitrario di istanze di *auto* e di *pedone)*.

SOLUZIONE

```
Pedone
{
    ……..
    // il pedone giunge al semaforo //
    pthread_mutex_lock(&MutexAttraversamento);
    PedoniInAttesa++;
    while  AutoInTransito> 0        pthread_cond_wait(&AttesaPedoni, &MutexAttraversamento);
    // semaforo rosso per i pedoni //
    PedoniInAttesa - -; PedoniInTransito ++;
    // ora il semaforo è verde //
    pthread_mutex_unlock(&MutexAttraversamento);
   < il pedone attraversa la strada >
    // il pedone ha finito di attraversare: ci sono altri pedoni che attraversano? //
    pthread_mutex_lock(&MutexAttraversamento);
    PedoniInTransito - -;
    if (PedoniInTransito == 0)
        for (i= 0, i<  AutoiInAttesa, i ++) pthread_cond_ signal(&AttesaAuto);
    // se AutoInAttesa> 0,  il semaforo diviene rosso per i pedoni e verde per le auto. Tutte le auto in attesa iniziano a transitare //
    pthread_mutex_unlock(&MutexAttraversamento);
    ……..
}

Auto
{
    ……..
    // l'auto giunge al semaforo //
    pthread_mutex_lock(&MutexAttraversamento);
    AutoInAttesa++;
    while  (PedoniInTransito> 0 or PedoniInAttesa>MaxCoda)
        pthread_cond_wait(&AttesaAuto, &MutexAttraversamento);
    // semaforo rosso per le auto //
    AutoiInAttesa - -; AutoInTransito ++;
    // ora il semaforo è verde //
    pthread_mutex_unlock(&MutexAttraversamento);
   < l'auto transita >
    //l'auto ha finito di transitare: ci sono altre auto in transito? //
    pthread_mutex_lock(&MutexAttraversamento);
    AutoInTransito - -;
    if (AutoInTransito == 0)
        for (i= 0, i<  PedoniInAttesa, i ++) pthread_cond_ signal(&AttesaPedoni);
    // il semaforo diviene rosso per le auto e verde per i pedoni. Tutti i pedoni  in attesa iniziano ad attraversare //
     pthread_mutex_unlock(&MutexAttraversamento);
    ……..
}
```

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 16/1/2010 CORSO |A| |B|

ESERCIZIO A-2 (4 punti)

Un sistema con risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [5, 6, 3, 6] non adotta alcuna politica di prevenzione dello stallo, ma lo riconosce e lo elimina dopo che si sia eventualmente verificato. Al tempo *t* sono presenti e attivi i processi A, B, C e D, e si è raggiunto lo stato mostrato nelle tabelle.

Assegnazione (tempo t+k)				
	R1	R2	R3	R4
A	2		1	3
B	1	1		
C			2	1
D	1	3		2

Esigenza residua (tempo t+k)				
	R1	R2	R3	R4
A	2	0	0	1
B	4	2	3	0
C	0	0	0	2
D	1	0	0	1

Disponibilità di risorse (tempo t+k)			
R1	R2	R3	R4
1	2	0	0

Al tempo *t+k* si sono verificati i seguenti eventi:

- A ha richiesto 1 istanza di R4 e si è sospeso;
- B ha richiesto 1 istanza di R3 e si è sospeso;
- C ha richiesto 1 istanza di R4 e si è sospeso;
- D ha richiesto 1 istanza di R4 e si è sospeso.

Ciascuno dei processi sospesi sarà riattivato se e quando la sua richiesta potrà essere soddisfatta.
Il sistema rileva che tutti i processi sono sospesi e che si è raggiunto uno stallo. Allo scopo di eliminarlo, forza il processo D a rilasciare 2 istanze della risorsa R4 e modifica la sua richiesta in una richiesta (multipla) di tre istanze di R4.
Si chiede se la sottrazione di queste risorse è sufficiente ad eliminare lo stallo.
Si tenga presente che lo stallo è eliminato se il rilascio permette di riattivare immediatamente almeno un processo e se, con un'opportuna sequenza di avanzamento, tutti i processi sono in grado di ottenere le risorse di cui hanno esigenza e conseguentemente giungere alla terminazione.

SOLUZIONE

Stato raggiunto dopo il rilascio di 2 istanze di R4 da parte del processo D

Assegnazione (dopo il rilascio)				
	R1	R2	R3	R4
A	2		1	3
B	1	1		
C			2	1
D	1	3		0

Esidenza residua (dopo il rilascio)				
	R1	R2	R3	R4
A	2	0	0	1
B	4	2	3	0
C	0	0	0	2
D	1	0	0	3

Disponibilità di risorse (dopo il rilascio)			
R1	R2	R3	R4
1	2	0	2

Dopo il rilascio:

1) I processi A e C sono riattivati immediatamente. Pr4ocessi attivi A, C
   Se avanza C, può ottenere tutte le risorse di cui ha esigenza e terminare;
   dopo la terminazione di C la disponibilità diviene [1, 2, 2, 3]
2) Sono riattivati anche i processi B e D. Sono attivi i processi A, B e D.
   Se avanza D, può ottenere tutte le risorse di cui ha esigenza e terminare
   dopo la terminazione di D la disponibilità diviene [2, 5, 2, 3]
3) Nessun processo riattivato. Sono attivi i processi A e B.
   Se avanza A, può ottenere tutte le risorse di cui ha esigenza e terminare
   dopo la terminazione di A la disponibilità diviene [4, 5, 3 6]
4) Nessun processo riattivato. E' attivo il solo processo B.
   B può avanzare, ottenere tutte le risorse di cui ha esigenza e quindi terminare
   Alla fine la disponibilità diviene [5, 6, 3, 6], uguale alla molteplicità.

Di conseguenza:

- lo stallo è eliminato.
  Infatti tuti i processi possono essere riattivati e ottenere tutte risorse di cui hanno esigenza.

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 16/1/2010 CORSO |A| |B|

ESERCIZIO A-3 (3 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 e lo stack pointer SP,
- un ulteriore banco di registri riservato allo stato supervisore, che comprende i registri generali R'1, R'2 e lo stack pointer SP'.

Il sistema gestisce il processore con politica a priorità (il processore è assegnato al processo attivo con il massimo valore di priorità). Al tempo t sono presenti nel sistema, tra gli altri, il processo Pi (con priorità 6), in stato di esecuzione, il processo Pj (con priorità 4) e il processo Pk (con priorità 5) entrambi in stato di pronto. Al tempo t il processo Pi esegue l'istruzione SVC per invocare la chiamata di sistema *exit,* che ne determina la terminazione immediata.. L'interruzione generata dalla SVC determina l'intervento del nucleo, che salva sullo stack del nucleo i registri PC e PS e lancia la funzione di servizio dell'interruzione corrispondente alla chiamata *exit*. Il vettore di interruzione associato a questa interruzione è 0600 e la corrispondente parola di stato è 275E. Immediatamente prima che l'interruzione venga riconosciuta, i registri del processore, i descrittori di Pi, Pj e Pk e lo stack del nucleo hanno i contenuti mostrati in tabella.

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.

Descrittori					
Processo	Pi	Processo	Pj	Processo	Pk
Stato	Esec	Stato	Pronto	Stato	Pronto.
Priorità	6	Priorità	4	Priorità	5
PC	2FF0	PC	7000	PC	A000
PS	16F2	PS	16F2	PS	16F2
SP	4010	SP	9000	SP	C000
R1	00AB	R1	1100	R1	22CC
R2	00CD	R2	11AA	R2	22DD

Stack del nucleo	
……	……
1016	0000
1015	
1014	
1013	
1012	
1011	

Registri stato utente	
SP	4000
R1	0011
R2	0012

Registri stato superv.	
SP	1016
R'1	AAAA
R'2	BBBB

Processore: registri speciali e di 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:

Descrittori					
Processo	Pi	Processo	Pj	Processo	Pk
Stato	Esec	Stato	Pronto	Stato	Pronto.
Priorità	5	Priorità	4	Priorità	6
PC	Invar.	PC	Invar.	PC	Invar.
PS	Invar.	PS	Invar.	PS	Invar.
SP	Invar.	SP	Invar.	SP	Invar.
R1	Invar.	R1	Invar.	R1	Invar.
R2	Invar.	R2	Invar.	R2	Invar.

Stack del nucleo	
……	……
1016	0000
1015	2F00
1014	16F2
1013	
1012	
1011	

Registri stato utente	
SP	Invar.
R1	Invar.
R2	Invar.

Registri stato superv.	
SP	1014
R'1	Invar.
R'2	Invar.

Processore: registri speciali e di stato						
PC	0600	PS	275E	stato	supervisore	

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;

Descrittori					
Processo	--	Processo	Pj	Processo	Pk
Stato	--	Stato	Pronto	Stato	Esecuz.
Priorità	--	Priorità	4	Priorità	6
PC	--	PC	Invar.	PC	Invar.
PS	--	PS	Invar.	PS	Invar.
SP	--	SP	Invar.	SP	Invar.
R1	--	R1	Invar.	R1	Invar.
R2	--	R2	Invar.	R2	Invar.

Stack del nucleo	
……	……
1016	0000
1015	A000
1014	16F2
1013	
1012	
1011	

Registri stato utente	
SP	C000
R1	22CC
R2	22DD

Registri stato superv.	
SP	1014
R'1	??
R'2	??

Processore: registri speciali e di stato						
PC	0600+ ??	PS	275E	stato	supervisore	

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.

Descrittori					
Processo	--	Processo	Pj	Processo	Pk
Stato	--	Stato	Pronto	Stato	Esecuz.
Priorità	--	Priorità	4	Priorità	6
PC	--	PC	Invar.	PC	Invar.
PS	--	PS	Invar.	PS	Invar.
SP	--	SP	Invar.	SP	Invar.
R1	--	R1	Invar.	R1	Invar.
R2	--	R2	Invar.	R2	Invar.

Stack del nucleo	
……	……
1016	0000
1015	
1014	
1013	
1012	
1011	

Registri stato utente	
SP	Invar.
R1	Invar.
R2	Invar.

Registri stato superv.	
SP	1016
R'1	Invar.
R'2	Invar.

Processore: registri speciali e di stato						
PC	A000	PS	16F2	stato	utente	

ESERCIZIO A-4 (2 punti)

Si consideri un sistema nel quale sono definiti il semaforo *sem* e i (soli) processi P1 (con priorità 2), P2 (con priorità 2) e P3 (con priorità 1). Lo scheduling avviene con una politica a priorità, che prevede il prerilascio e assegna il processore al processo attivo di priorità più elevata (a pari priorità applica la politica Round Robin). La politica applicata al semaforo è la FIFO.
Al tempo *t* il processo P1 è in esecuzione, il processo P2 è bloccato sul semaforo *sem* e P3 è in stato di pronto. Dopo il tempo *t* si verifica la seguente sequenza di eventi:

1) P1 esegue *wait (sem)*
2) il processo in esecuzione esegue *signal(sem);*
3) scade il quanto di tempo del processo in esecuzione
4) il processo in esecuzione esegue *signal(sem);*

Si chiede di specificare quale processo è in esecuzione dopo ogni evento e inoltre come si modificano il valore e la coda del semaforo *sem* e la *CodaPronti*.

SOLUZIONE

Sequenza di eventi	In Esecuzione	Coda Pronti Priorità 1	Coda Pronti Priorità 2	Valore di *sem*	Coda di *sem*
1) P1 esegue *wait (sem)*	P3	-	-	0	P2,P1
2) il processo in esecuzione esegue *signal(sem)*	P2	P3	-	0	P1
3) scade il quanto di tempo del processo in esecuzione	P2	P3	-	0	P1
4) Il processo in esecuzione esegue *signal(sem)*	P2	P3	P1	0	-

ESERCIZIO A-5 (2 punti)
In un sistema operativo simile ad Unix che implementa i thread a livello utente dire quali dei seguenti dati devono essere contenute nel nucleo:

	Struttura dati
a)	Contesto del thread
b)	Priorità del thread
c)	Identificatore del thread
d)	Stack del processo
e)	Informazioni sui segnali pendenti
f)	Contesto del processo

SOLUZIONE

	Struttura dati	Nel nucleo? [SI/NO]
a)	Contesto del thread	NO
b)	Priorità del thread	NO
c)	Identificatore del thread	NO
d)	Stack del processo	NO
e)	Informazioni sui segnali pendenti	SI
f)	Contesto del processo	SI

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 16/1/2010 CORSO |A| |B|

ESERCIZIO B-1 (4 punti)

Un sistema operativo simile a UNIX, che gestisce la memoria con paginazione a domanda, utilizza il processo *PageDaemon,* con parametri *lotsfree=6 e minfree=2*, e l'algoritmo di sostituzione *Second Chance*. Il modello di generazione dei processi non prevede l'allocazione di pagine fisiche ai processi appena creati, in quanto è lo stesso processo che, quando va in esecuzione, provoca il caricamento delle pagine tramite i page fault risultanti dai suoi riferimenti alla memoria.

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* (pagina del processo caricata nel blocco), *Rif* (bit di pagina riferita utilizzato da *Second Chance*). Al tempo *t* sono presenti i processi A, B, C, D e la *Core Map* ha la configurazione mostrata in figura, con il puntatore dell'algoritmo di sostituzione posizionato sul blocco 18. I primi 2 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione.

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

Core Map al tempo *t*

Il *PageDaemon* interviene al tempo *t*+5 e successivamente ogni 10 msec. Ad ogni intervento, *PageDaemon* avanza per 1 msec occupando in modo esclusivo il processore e scarica fino a 4 pagine o, in alternativa, esegue lo *swapout* di un processo. La selezione dei processi candidati allo *swapout* avviene in ordine di occupazione di memoria (per primi i processi che occupano più spazio) e, in caso di parità, in ordine alfabetico.

In caso di errori di pagina, i blocchi liberi vengono assegnati in ordine crescente di indice.

Considerare i seguenti eventi che si susseguono tra

1. dal tempo t al tempo t+5 avanzano i processi D e C che riferiscono nell'ordine le pagine: D4, D9, D10, C3, C7, C4, C8, C6;
2. dal tempo t+5 al tempo t+6 avanza il processo *PageDaemon*;
3. dal tempo t+6 al tempo t+15 viene generato il processo F che riferisce le pagine F1, F2, F3
4. dal tempo t+15 al tempo t+16 avanza il processo *PageDaemon*;
5. al tempo t+17 termina il processo A
6. dal tempo t+17 al tempo t+20 avanzano i processi D e F che riferiscono nell'ordine le pagine: D4, D10, D1, F1, F4

Mostrare la configurazione della *CoreMap* ai tempi 5, 6, 15, 16 e 20

SOLUZIONE

																			↓		
Proc			A	B	B	D	C	A	B	A	**D**	B	C	D	D	**C**	D	B	A	B	C	C
Pag			1	0	1	4	3	7	3	10	**10**	6	4	9	14	**6**	16	11	13	12	7	8
Rif			1	1	1	**1**	**1**	1	1	1	**1**	1	1	**1**	1	**1**	0	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

Core Map al tempo *t*+5

																		↓			
Proc			A			D	C	A		A	D		C	D	D	C	D		A		C	C
Pag			1			4	3	7		10	10		4	9	14	6	16		13		7	8
Rif			1			1	1	1		1	1		1	1	1	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

Core Map al tempo *t*+6

Eseguito lo swap out del processo B

																		↓			
Proc			A	**F**	**F**	D	C	A	**F**	A	D		C	D	D	C	D		A		C	C
Pag			1	**1**	**2**	4	3	7	**3**	10	10		4	9	14	6	16		13		7	8
Rif			1	**1**	**1**	1	1	1	**1**	1	1		1	1	1	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

Core Map al tempo *t*+15

																					↓
Proc			A	F	F	D	C	A	F	A	D		C	D	D	C						C
Pag			1	1	2	4	3	7	3	10	10		4	9	14	6						8
Rif			**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

Core Map al tempo *t*+16

Scaricate le pagine D16, A13, C7 Eseguito lo swap out del processo -----

																					↓
Proc			**D**	F	F	D	C	**F**	F		D		C	D	D	C						C
Pag			**1**	1	2	4	3	**4**	3		10		4	9	14	6						8
Rif			**1**	**1**	0	**1**	0	**1**	0		**1**		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

Core Map al tempo *t*+20

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 16/1/2010 CORSO |A| |B|

ESERCIZIO B-2 (4 punti)

Un disco con 2 facce, 200 settori per traccia e 1000 cilindri ha un tempo di seek pari ad un valore costante di 40 msec più 0,1 msec per ogni cilindro attraversato. Alla conclusione di ogni operazione di seek le testine del disco sono posizionate all'inizio del settore 0 del cilindro (il costo di questo posizionamento è già incluso nel tempo di seek).

Il periodo di rotazione è di 20 msec: conseguentemente il tempo impiegato per percorrere un settore è di 0,1 msec.

A un certo tempo (convenzionalmente indicato come t=0) termina l'esecuzione dei comandi sul cilindro 860 e sono pervenute, nell'ordine, le seguenti richieste di lettura o scrittura:

- clindro 890, faccia 0, settore 55
- clindro 630, faccia 0, settore 100
- clindro 47, faccia 1, settore 120
- cilindro 850, faccia 1, settore 189
- clindro 345, faccia 1, settore 5
- cilindro 930, faccia 0, settore 76

Sucessivamente, all'istante 40 arrivano anche le seguenti richieste:

- clindro 630, faccia 0, settore 88
- clindro 47, faccia 0, settore 50

Calcolare il tempo necessario per eseguire tutte queste operazioni supponendo che si adotti la politica di scheduling SCAN e supponendo che al tempo 0 il disco sia in fase di discesa.

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.

Il controllore è dotato di sufficiente capacità di buffering ed è sempre in grado di accettare senza ritardo i dati letti dal disco o quelli da scrivere sul disco. Si assuma inoltre che i comandi sullo stesso cilindro vengano eseguiti nell'ordine che minimizza il ritardo rotazionale.

SOLUZIONE

	op. su cilindro:	850	settore:	189					
Tempi	inizio:	0	seek:	41	rotazione:	18,9	percorrenza:	0,1	fine:	60
	op. su cilindro:	630	settore:	88					
Tempi	inizio:	60	seek:	62	rotazione:	8,8	percorrenza:	0,1	fine:	130,9
	op. su cilindro:	630	settore:	100					
Tempi	inizio:	130,9	seek:	0	rotazione:	1,1	percorrenza:	0,1	fine:	132,1
	op. su cilindro:	345	settore:	5					
Tempi	inizio:	132,1	seek:	68,5	rotazione:	0,5	percorrenza:	0,1	fine:	201,2
	op. su cilindro:	47	settore:	50					
Tempi	inizio:	201,2	seek:	69,8	rotazione:	5	percorrenza:	0,1	fine:	276,1
	op. su cilindro:	47	settore:	120					
Tempi	inizio:	276,1	seek:	0	rotazione:	6,9	percorrenza:	0,1	fine:	283,1
	op. su cilindro:	890	settore:	55					
Tempi	inizio:	283,1	seek:	124,3	rotazione:	5,5	percorrenza:	0,1	fine:	413
	op. su cilindro:	930	settore:	76					
Tempi	inizio:	413	seek:	44	rotazione:	7,6	percorrenza:	0,1	fine:	464,7

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 16/1/2010 CORSO |A| |B|

ESERCIZIO B-3 (3 punti)

In un file system UNIX i blocchi del disco hanno ampiezza di 4Kbyte e gli indici di blocco fisico sono codificati con 4 byte. Gli i-node contengono, oltre agli altri attributi, 10 puntatori diretti e 3 puntatori indiretti. I puntatori sono indici di blocco fisico. Ogni i-node occupa 1 Kbyte.
Il file system è ospitato in una partizione di un disco, la quale contiene nell'ordine:

- il boot block;
- il super block, comprensivo della bitmap degli i-node e di quella dei blocchi dati,
- 1000 blocchi riservati alla i-list;
- i blocchi dati, che occupano la parte rimanente della partizione..

Si chiede:

1. l'ampiezza della partizione, commisurata alla capacità di indirizzamento dei blocchi;
2. il massimo numero di file che possono essere definiti;
3. la massima estensione del file system, commisurata al numero di blocchi dati disponibili
4. il numero di puntatori che possono essere contenuti in un blocco indiretto;
5. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con puntatori diretti;
6. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con indirizzamento indiretto semplice;
7. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con indirizzamento indiretto doppio;
8. l'indice logico del primo blocco e dell'ultimo blocco indirizzabili con indirizzamento indiretto triplo;
9. la massima estensione di un singolo file.

SOLUZIONE

1. Ampiezza della partizione 2^{32} blocchi → $2^{32} * 2^2 = 2^{44}$ byte= 16 Tbyte;
2. massimo numero di file che possono essere definiti: 1000*4= 4000 (numero di i-nodes);
3. massima estensione del file system: 2^{32}- 1002 blocchi → $(2^{32}- 102)* 2^{12}$ byte ~ 16 Tbyte;
4. numero di puntatori che possono essere contenuti in un blocco indiretto: 4096 div 4 = 2^{10};
5. indice logico del primo blocco indirizzabile con puntatori diretti: 0;
   indice logico dell'ultimo blocco indirizzabile con puntatori diretti: 9;
6. indice logico del primo blocco indirizzabile con indirizzamento indiretto semplice: 10;
   indice logico dell'ultimo blocco indirizzabile con indirizzamento indiretto semplice: $10+ 2^{10}- 1$;
7. indice logico del primo blocco indirizzabile con indirizzamento indiretto doppio: $10+ 2^{10}$;
   indice logico dell'ultimo blocco indirizzabile con indirizzamento indiretto doppio: $10+ 2^{10} + 2^{20}- 1$;
8. indice logico del primo blocco indirizzabile con indirizzamento indiretto triplo; $10+ 2^{10} + 2^{20}$;
   indice logico dell'ultimo blocco indirizzabile con indirizzamento indiretto triplo: $10+ 2^{10} + 2^{20} + 2^{30}- 1$;
9. massima estensione di un singolo file $10+ 2^{10} + 2^{20} + 2^{30}$ blocchi → $> 2^{30} * 2^{12}$ byte = 4 Tbyte.

ESERCIZIO B-4 (2 punti)
Dire quali delle seguenti strutture dati di un processo Unix possono essere modificate nell'esecuzione dalla chiamata di sistema Read:

Struttura dati
Tabella dei file aperti dal processo:
Tabella dei file aperti di sistema
Tabella dei file attivi:

SOLUZIONE

Struttura dati	**SI/NO**
Tabella dei file aperti dal processo:	NO
Tabella dei file aperti di sistema	SI
Tabella dei file attivi:	NO

ESERCIZIO B-5 (2 punti)
In un sistema che gestisce la memoria con paginazione a domanda, la massima dimensione dello spazio virtuale dei processi è pari a 2^{40} byte, la massima dimensione della memoria fisica è pari a 2^{36} byte e le pagine hanno dimensione di 4 Kbyte. Inoltre in ogni descrittore di pagina il campo di controllo (che contiene tutti i bit utili per la gestione della paginazione) occupa 1 byte.
Si chiede:

1. Il massimo numero di pagine fisiche
2. La dimensione di un descrittore di pagina (in bytes)
3. Il massimo numero di pagine logiche
4. La dimensione dell'intera tabella delle pagine

SOLUZIONE

1. Massimo numero di pagine fisiche: 2^{36} byte/2^{12} byte = 2^{24}
2. Dimensione di un descrittore di pagina (in bytes): 1 byte (per il campo di controllo) + 24 bit (per l'indice della pagina fisica) = 4 bytes
3. Massimo numero di pagine logiche: 2^{40} byte/2^{12} byte = 2^{28}
4. La dimensione dell'intera tabella delle pagine= $4*2^{28}$ byte = 1 Gbyte