Sistemi Operativi e Laboratorio, Prova del 15/01/2014

Cognome e nome: _____ Matricola: _____ Corso [A/B] :___ fila: ___ posto: ___

Esercizio 1 (4 punti)

Si consideri un processore che dispone dei registri speciali PC (program counter), PS (program status) e SP (stack pointer), e di un banco di registri generali, utilizzati sia in stato utente, sia in stato supervisore, che comprende i registri R1, R2 e R3. Inoltre, il nucleo dispone di un proprio stack, che inizia all'indirizzo 1016 e si espande verso il basso. Al riconoscimento di un'interruzione, il processore salva sullo stack del nucleo i soli registri speciali.

Il sistema operativo realizza i thread a livello del nucleo e lo schedulatore gestisce esclusivamente i thread. La politica di scheduling è a priorità e con prerilascio (un valore maggiore della priorità corrisponde a priorità maggiore).

Al tempo *t* sono presenti nel sistema il thread T_{i1} del processo P_i, che si trova in stato di esecuzione, e il thread T_{j1} del processo P_j, che è in stato di attesa del completamento di un'operazione di I/O sul disco. In questo istante di tempo, il processore riceve un'interruzione generata dal disco che segnala il completamento dell'operazione richiesta.

Immediatamente dopo il riconoscimento dell'interruzione generata dal disco, i registri del processore, i descrittori di T_{i1} e T_{j1} hanno i contenuti mostrati in tabella. Lo stack del nucleo è vuoto.

Supponendo che il vettore di interruzione associato all'interruzione sia 2900 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 procedura di gestione dell'interruzione;
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 procedura di gestione dell'interruzione;
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 T_{i1}	
Stato	Esec
Priorità	5
PC	89AA
PS	16F2
SP	B000
R1	1000
R2	1100
R3	1110

DESCRITTORE DI T_{J2}	
Stato	Attesa
Priorità	9
PC	C011
PS	16F2
SP	E899
R1	2000
R2	2200
R3	2220

STACK DEL NUCLEO	
1016	
1015	
1014	
1013	
1012	
1011	
1010	
1009	
……	

REGISTRI GENERALI	
SP	AE98
R1	1001
R2	1011
R3	0011

PROCESSORE: Registri speciali				
PC	9100		PS	1682

Stato processore
UTENTE

Soluzione

a) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione della prima istruzione della procedura di gestione dell'interruzione:

DESCRITTORE DI T_{i1}	
Stato	
Priorità	
PC	
PS	
SP	
R1	
R2	
R3	

DESCRITTORE DI T_{J2}	
Stato	
Priorità	
PC	
PS	
SP	
R1	
R2	
R3	

STACK DEL NUCLEO	
1016	
1015	
1014	
1013	
1012	
1011	
1010	
1009	
……	

REGISTRI GENERALI	
SP	
R1	
R2	
R3	

PROCESSORE: Registri speciali				
PC			PS	

Stato processore

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 procedura di gestione dell'interruzione;

DESCRITTORE DI T_{i1}	
Stato	
Priorità	
PC	
PS	
SP	
R1	
R2	

DESCRITTORE DI T_{J2}	
Stato	
Priorità	
PC	
PS	
SP	
R1	
R2	

STACK DEL NUCLEO	
1016	
1015	
1014	
1013	
1012	
1011	
1010	

REGISTRI GENERALI	
SP	
R1	
R2	
R3	

R3			R3			1009	
						…….	
PROCESSORE: Registri speciali							Stato processore
PC			PS				

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 T_{i1}			DESCRITTORE DI T_{J2}			STACK DEL NUCLEO			REGISTRI GENERALI	
Stato			Stato			1016				
Priorità			Priorità			1015			SP	
PC			PC			1014			R1	
PS			PS			1013			R2	
SP			SP			1012			R3	
R1			R1			1011				
R2			R2			1010				
R3			R3			1009				
						…….				
PROCESSORE: Registri speciali									Stato processore	
PC			PS							

Soluzione

a) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione della prima istruzione della procedura di gestione dell'interruzione:

DESCRITTORE DI T_{i1}			DESCRITTORE DI T_{J2}			STACK DEL NUCLEO			REGISTRI GENERALI	
Stato	Esec		Stato	Attesa		1016	9100			
Priorità	5		Priorità	9		1015	1682		SP	1013
PC	89AA		PC	C011		1014	AE98		R1	1001
PS	16F2		PS	16F2		1013			R2	1011
SP	B000		SP	E899		1012			R3	0011
R1	1000		R1	2000		1011				
R2	1100		R2	2200		1010				
R3	1110		R3	2220		1009				
						…….				
PROCESSORE: Registri speciali									Stato processore	
PC	2900		PS	275E					SUPERVISORE	

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 procedura di gestione dell'interruzione;

DESCRITTORE DI T_{i1}			DESCRITTORE DI T_{J2}			STACK DEL NUCLEO			REGISTRI GENERALI	
Stato	Pronto		Stato	Esecuzione		1016	C011			
Priorità	5		Priorità	9		1015	16F2		SP	1013
PC	9100		PC	C011		1014	E899		R1	2000
PS	1682		PS	16F2		1013			R2	2200
SP	AE98		SP	E899		1012			R3	2220
R1	1001		R1	2000		1011				
R2	1011		R2	2200		1010				
R3	0011		R3	2220		1009				
						…….				
PROCESSORE: Registri speciali									Stato processore	
PC	2900+??		PS	275E					SUPERVISORE	

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 T_{i1}			DESCRITTORE DI T_{J2}			STACK DEL NUCLEO			REGISTRI GENERALI	
Stato	Pronto		Stato	Esecuzione		1016				
Priorità	5		Priorità	9		1015			SP	E899
PC	9100		PC	C011		1014			R1	2000
PS	1682		PS	16F2		1013			R2	2200
SP	AE98		SP	E899		1012			R3	2220

Sistemi Operativi e Laboratorio, Prova del 15/01/2014

R1	1001		R1	2000	1011		
R2	1011		R2	2200	1010		
R3	0011		R3	2220	1009		
					……		
PROCESSORE: Registri speciali							Stato processore
PC	C011		PS	16F2			UTENTE

Esercizio 2 (4 punti)

Un laboratorio di informatica aperto a un numero illimitato di studenti dispone di 5 stazioni di lavoro. Ogni stazione è assegnabile di volta in volta ad un unico studente, che la utilizza per un tempo finito. Inoltre alcuni studenti devono svolgere un progetto per il quale necessitano di usare in modo esclusivo (e per un tempo finito) 3 stazioni di lavoro per volta.

Un generico studente richiede l'accesso per il numero di stazioni che gli servono (1 o 3), ed è ammesso a condizione che vi siano un numero di stazioni disponibili sufficienti a soddisfare la sua richiesta. Se queste condizioni non sono verificate, il richiedente attende fino a quando non ci sia un numero sufficiente di stazioni disponibili. Inoltre, per evitare l'attesa indefinita da parte degli studenti che devono lavorare al progetto (e che quindi hanno bisogno di 3 stazioni di lavoro), gli studenti si mettono in attesa su due condizioni distinte: la condizione *AttesaStudenti_S* per gli studenti in attesa di una sola stazione di lavoro disponibile, e la condizione *AttesaStudenti_T* per gli studenti in attesa di tre stazioni di lavoro disponibili.

Al rilascio di una o più stazioni di lavoro da parte di uno studente, le stazioni disponibili vengono assegnate secondo la seguente politica:

1. Se la coda T è vuota vengono assegnate agli studenti presenti in coda S.
2. Se la coda T è non vuota e ci sono almeno 3 stazioni disponibili, 3 di queste vengono assegnate al primo studente in attesa nella coda T e le restanti vengono assegnate agli studenti in attesa nella coda S
3. Se la coda T è non vuota ma non ci sono almeno 3 stazioni disponibili, allora non viene assegnata nessuna stazione di lavoro.

Gli studenti e il sistemista sono thread di uno stesso processo.

Per la soluzione del problema, si utilizzano i seguenti dati condivisi da tutti i thread:

- *StazioniDisponibili*: intero non negativo; valore iniziale 5
- *StudentiInAttesa_S*: intero non negativo; valore iniziale 0
- *StudentiInAttesa_T*: intero non negativo; valore iniziale 0
- *StudenteAttivo_T*: booleano; inizializzato a FALSE; assume valore TRUE quando uno studente che svolge il progetto sta usando 3 stazioni contemporaneamente.

Per gli studenti che sostengono la prova di verifica intermedia: Si chiede di completare la soluzione proposta utilizzando i meccanismi delle variabili di condizione (visti a lezione e sul libro), con la variabile *MutexLab,* di tipo *mutex* e le variabili *AttesaStudenti_T* e *AttesaStudenti_S*, di tipo *condition*, impiegate per la sospensione degli studenti.

Per tutti gli altri studenti: Si chiede di completare la soluzione proposta utilizzando i meccanismi della libreria ***pthread***, con la variabile *MutexLab,* di tipo *mutex* e le variabili *AttesaStudenti_T* e *AttesaStudenti_S*, di tipo *condition*, impiegate per la sospensione degli studenti.

Soluzione

//codice dello studente che richiede una stazione per volta

{

……

_____

StudentiInAttesa_S ++;

while (StazioniDisponibili == 0 **||** (StudentiInAttesa_T > 0 **&&** StudenteAttivo_T == FALSE)

_____

//lo studente si mette in attesa //

StudentiInAttesa_S – –; StazioniDisponibili – –;

_____

< utilizza la stazione per un tempo finito>

_____

StazioniDisponibili ++;

if (StudentiInAttesa_T == 0)

if (StudentiInAttesa_S > 0)

_____

else if (StazioniDisponibili>=3) {

_____

```
_____________________________________________
……..
}


//codice dello studente che richiede tre stazioni per volta
{
……..
_____________________________________________
StudentiInAttesa_T ++;
while (StazioniDisponibili < 3)

        _____________________________________________
        //lo studente si mette in attesa //
StudentiInAttesa_T – –; StazioniDisponibili –= 3;
StudenteAttivo_T = TRUE;
if (StudentiInAttesa_S > 0 && StazioniDisponibili > 0)

        _____________________________________________

_____________________________________________
< utilizza la stazione per un tempo finito>

_____________________________________________
StazioniDisponibili +=3;
StudenteAttivo_T = FALSE;
if (StudentiInAttesa_T == 0)
        if (StudentiInAttesa_S > 0)

                _____________________________________________

else if (StazioniDisponibili>=3) {

        _____________________________________________

_____________________________________________
……..
}
```

Soluzione- Per gli studenti che sostengono la prova di verifica intermedia

```
//codice dello studente che richiede una stazione per volta
{
……..
MutexLab. Acquire();
StudentiInAttesa_S ++;
while (StazioniDisponibili == 0 || (StudentiInAttesa_T > 0 && StudenteAttivo_T == FALSE )
        AttesaStudenti_S.wait(&MutexLab);
        //lo studente si mette in attesa //
StudentiInAttesa_S – –; StazioniDisponibili – –;
MutexLab. Release();

< utilizza la stazione per un tempo finito>

MutexLab. Acquire();
StazioniDisponibili ++;
if (StudentiInAttesa_T == 0)
        if (StudentiInAttesa_S > 0)
                        AttesaStudenti_S.broadcast(&MutexLab);

else if (StazioniDisponibili>=3) {
        AttesaStudenti_T.signal(&MutexLab);
MutexLab. Release();
……..
```

```
}

//codice dello studente che richiede tre stazioni per volta
{
……..
MutexLab. Acquire();
StudentiInAttesa_T ++;
while (StazioniDisponibili < 3)
        AttesaStudenti_T.wait(&MutexLab);
        //lo studente si mette in attesa //
StudentiInAttesa_T – –; StazioniDisponibili –= 3;
StudenteAttivo_T = TRUE;
if (StudentiInAttesa_S > 0 && StazioniDisponibili > 0)
                AttesaStudenti_S.broadcast(&MutexLab);

MutexLab. Release();
< utilizza la stazione per un tempo finito>
MutexLab. Acquire();
StazioniDisponibili +=3;
StudenteAttivo_T = FALSE;
if (StudentiInAttesa_T == 0)
        if (StudentiInAttesa_S > 0)
                AttesaStudenti_S.broadcast(&MutexLab);

else if (StazioniDisponibili>=3) {
        AttesaStudenti_T.signal(&MutexLab);
MutexLab. Release();
……..
}
```

Soluzione- Per gli altri studenti

```
//codice dello studente che richiede una stazione per volta
{
……..
pthread_mutex_lock(&MutexLab);
StudentiInAttesa_S ++;
while (StazioniDisponibili == 0 || (StudentiInAttesa_T > 0 && StudenteAttivo_T == FALSE )
        pthread_cond_wait(&AttesaStudenti_S, &MutexLab);
        //lo studente si mette in attesa //
StudentiInAttesa_S – –; StazioniDisponibili – –;
pthread_mutex_unlock(&MutexLab);

< utilizza la stazione per un tempo finito>

pthread_mutex_lock(&MutexLab);
StazioniDisponibili ++;
if (StudentiInAttesa_T == 0)
        if (StudentiInAttesa_S > 0)
                pthread_cond_broadcast(&AttesaStudenti_S);

else if (StazioniDisponibili>=3) {
        pthread_cond_signal(&AttesaStudenti_T);
pthread_mutex_unlock(&MutexLab);
……..
}

//codice dello studente che richiede tre stazioni per volta
{
……..
pthread_mutex_lock(&MutexLab);
StudentiInAttesa_T ++;
while (StazioniDisponibili < 3)
        pthread_cond_wait(&AttesaStudenti_T, &MutexLab);
        //lo studente si mette in attesa //
```

```
StudentiInAttesa_T – –; StazioniDisponibili –= 3;
StudenteAttivo_T = TRUE;
if (StudentiInAttesa_S > 0 && StazioniDisponibili > 0)
        pthread_cond_broadcast(&AttesaStudenti_S);

pthread_mutex_unlock(&MutexLab);
< utilizza la stazione per un tempo finito>
pthread_mutex_lock(&MutexLab);
StazioniDisponibili +=3;
StudenteAttivo_T = FALSE;
if (StudentiInAttesa_T == 0)
        if (StudentiInAttesa_S > 0)
                pthread_cond_broadcast(&AttesaStudenti_S);

else if (StazioniDisponibili>=3) {
        pthread_cond_signal(&AttesaStudenti_T);
pthread_mutex_unlock(&MutexLab);
……
}
```

Esercizio 3 (4 punti)

Un sistema operativo simile a UNIX, che gestisce la memoria con paginazione a domanda, utilizza il processo *PageDaemon,* con parametri *lotsfree= 7 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; 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 11. I primi 2 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione.

												↓												
Proc			A	D	A	C	D		D	C	A	A	C		B	B		A	C		B	D	B	A
Pag			2	6	4	0	4		9	3	5	6	4		2	3		8	10		5	11	7	10
Rif			1	1	1	1	0		1	1	0	1	1		0	1		0	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*

PageDaemon interviene al tempo *t* 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 (per arrivare a *lotsfree* pagine libere) se il numero di pagine libere è almeno *minfree.* Se invece il numero di pagine libere è minore di *minfree,* allora esegue lo *swapout* di uno o più processi fino ad arrivare ad almeno *lotsfree* pagine libere.
La selezione dei processi candidati allo *swapout* avviene in ordine di dimensione (per primi i processi più grandi). In caso di errori di pagina, i blocchi liberi vengono assegnati in ordine crescente di indice.
Considerare la seguente evoluzione del sistema (i tempi sono espressi in millisecondi):

1. Dal tempo *t* al tempo *t*+1 avanza il processo *PageDaemon*;
2. Dal tempo *t*+1 al tempo *t*+10 i processi A, B, C e D riferiscono nell'ordine le pagine: A10, C3, C0, C10, C2, C6, B3, B2, D10, D11, C9
3. Dal tempo *t*+10 al tempo *t*+11 avanza il processo *PageDaemon*;
4. Dal tempo *t*+11 al tempo *t*+20 i processi A, B, C e D riferiscono nell'ordine le pagine: A2, A5, A9, A11, A4, B0, B5, B6, B7, D6, D10
5. Dal tempo *t*+20 al tempo *t*+21 avanza il processo *PageDaemon.*

Mostrare la configurazione della *CoreMap* ai tempi 1, 10, 11, 20 e 21

Soluzione

(Modificare incrementalmente la configurazione iniziale della *CoreMap*, aggiornando anche la posizione del puntatore)

Proc																								
Pag																								
Rif																								
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

Pagine scaricate: _____

Proc																								
Pag																								
Rif																								
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

Sistemi Operativi e Laboratorio, Prova del 15/01/2014

Proc																								
Pag																								
Rif																								
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

Pagine scaricate: _____

Proc																								
Pag																								
Rif																								
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

Proc																								
Pag																								
Rif																								
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*+21

Pagine scaricate: _____

Soluzione

(Modificare incrementalmente la configurazione iniziale della *CoreMap*, aggiornando anche la posizione del puntatore)

								↓																
Proc			A	D	A	C	-		D	C	A	A	C		-	B		-	C		B	D	B	A
Pag			2	6	4	0	-		9	3	5	6	4		-	3		-	10		5	11	7	10
Rif			0	0	0	0	-		1	1	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	22	23

Core Map al tempo *t*+1

Pagine scaricate: B2, A8, D4

												↓												
Proc			A	D	A	C	C	C	D	C	A	A	C	B	D	B	C		C		B	D	B	A
Pag			2	6	4	0	2	6	9	3	5	6	4	2	10	3	9		10		5	11	7	10
Rif			0	0	0	1	1	1	1	1	0	0	0	1	1	1	1		1		0	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	22	23

Core Map al tempo *t*+10

											↓													
Proc			A	D	A	-	-	-	D	-	A	A	-	B	D	B	-		-		B	D	B	A
Pag			2	6	4	-	-	-	9	-	5	6	-	2	10	3	-		-		5	11	7	10
Rif			0	0	0	-	-	-	1	-	0	0	-	1	1	1	-		-		0	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	22	23

Core Map al tempo *t*+11

Pagine scaricate: C0, C2, C6, C3, C4, C9, C10 (intero processo C)

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

															↓									
Proc			A	D	A	A	A	B	D	B	A	-		-	D	B					B	D	B	A
Pag			2	6	4	9	11	0	9	6	5	-		-	10	3					5	11	7	10
Rif			0	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	22	23

Core Map al tempo *t*+21

Pagine scaricate: A6, B2

Esercizio 4 (4 punti)

Sistemi Operativi e Laboratorio, Prova del 15/01/2014

Un sistema operativo gestisce la memoria con paginazione dinamica con pagine di 1 Kbyte e utilizza un file system di tipo FAT-32 (con indirizzi a 32 bit) con blocchi di 1 Kbyte per gestire un disco di 1 Gbyte. La FAT è allocata sul disco a partire dal blocco 3.
Nel file system il file Cervino appartenente alla directory Alpi occupa 5 blocchi logici allocati come segue:

Blocco logico:	0	1	2	3	4	5
Blocco fisico:	7600	7400	7900	7901	7902	8000

Ad un dato istante di tempo, quando nessun blocco della FAT è caricato in memoria principale, la directory Alpi è caricata in memoria e nessun blocco del file Cervino è caricato in memoria (pertanto il puntatore al primo blocco fisico del file Cervino è già disponibile in memoria principale), il sistema operativo riceve una richiesta di lettura dei primi 4200 caratteri del file Cervino (compresi tra il byte 0 e il byte 4199, estremi inclusi).
Si chiede:

1. Quali blocchi logici del file Cervino vengono letti dal sistema operativo
2. Quali blocchi fisici del file Cervino vengono letti dal sistema operativo
3. Quali blocchi fisici contenenti la FAT vengono letti dal sistema operativo
4. Quanti page fault causa l'operazione di lettura nell'ipotesi che le pagine libere in memoria principale siano sufficienti ad accogliere tutti i blocchi da caricare.

Soluzione

1. Blocchi logici del file Cervino letti dal sistema operativo: _____
2. Blocchi fisici del file Cervino letti dal sistema operativo: _____
3. Blocchi fisici letti dal sistema operativo in totale:

Si devono leggere:

a. i blocchi dati _____ del file Cervino,

b. i blocchi del disco che contengono gli elementi: _____ della FAT (l'ultimo elemento contiene il puntatore al blocco 7902). Infatti per ipotesi nessuno dei blocchi che contengono la FAT è caricato in memoria principale.

Considerando che ogni blocco della FAT contiene _____ indirizzi, e che la FAT comincia dal blocco 3, si ha che:

l'elemento _____ è contenuto nel blocco _____

l'elemento _____ è contenuto nel blocco _____

l'elemento _____ è contenuto nel blocco _____

l'elemento _____ è contenuto nel blocco _____

Pertanto i blocchi del disco da leggere per percorrere la FAT sono: _____

4. Page fault causati dall'operazione di lettura: _____

Soluzione

1. Blocchi logici del file Cervino letti dal sistema operativo: 0,1,2,3,4;
2. Blocchi fisici del file Cervino letti dal sistema operativo: 7600, 7400, 7900, 7901, 7902;
3. Blocchi fisici letti dal sistema operativo in totale:

Si devono leggere:

a. i blocchi dati 7600, 7400, 7900, 7901, 7902 del file Cervino,

b. i blocchi del disco che contengono gli elementi: 7600, 7400, 7900, 7901 della FAT (l'ultimo elemento contiene il puntatore al blocco 7902). Infatti per ipotesi nessuno dei blocchi che contengono la FAT è caricato in memoria principale.

Considerando che ogni blocco della FAT contiene 1024/4=256 indirizzi, e che la FAT comincia dal blocco 3, si ha che:
l'elemento 7600 è contenuto nel blocco 7600 div 256 + 3 = 29 + 3 = 32
l'elemento 7400 è contenuto nel blocco 7400 div 256 + 3 = 28 + 3 = 31
l'elemento 7900 è contenuto nel blocco 7900 div 256 + 3 = 30 + 3 = 33
l'elemento 7901 è contenuto nel blocco 7901 div 256 + 3 = 30 + 3 = 33
Pertanto i blocchi del disco da leggere per percorrere la FAT sono: 32, 31 e 33

4. Page fault causati dall'operazione di lettura: 3

Esercizio 5 (3 punti)

In un file system UNIX i blocchi del disco hanno ampiezza di 8 Kbyte e gli indici di blocco fisico sono codificati con 8 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 512 byte.
Si chiede:

1. il numero di i-node contenuti in un blocco: _____
2. il numero di puntatori che possono essere contenuti in un blocco: _____
3. il numero di blocchi indirizzabili con indirizzamento indiretto singolo: _____

4. il numero di blocchi indirizzabili con indirizzamento indiretto doppio: _____
5. il numero di blocchi indirizzabili con indirizzamento indiretto triplo: _____
6. massima dimensione di un file: _____

Soluzione

1. numero di i-node contenuti in un blocco: 8192 Byte / 512 Byte = 16
2. numero di puntatori che possono essere contenuti in un blocco: 8 KBytes / 8 Bytes = 1024
3. numero di blocchi indirizzabili con indirizzamento indiretto singolo: 1024
4. numero di blocchi indirizzabili con indirizzamento indiretto doppio: 1024 x 1024 = 1048576
5. numero di blocchi indirizzabili con indirizzamento indiretto triplo: 1024 x 1024 x 1024
6. massima dimensione di un file: 10 + 1024 + 1024^2 + 1024^3 blocchi = 1 TByte circa

Esercizio 6 (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, che è uno stato sicuro:

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

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

Molteplicità			
R1	R2	R3	R4
6	4	7	6

Disponibilità			
0	1	1	3

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

1. A richiede 2 istanze di R4 (richiesta multipla: deve essere soddisfatta integralmente)
2. C richiede 1 istanza di R2

Il gestore delle risorse applica l'algoritmo del banchiere per evitare lo stallo. Verificare se il gestore assegna le risorse richieste.

Soluzione

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

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza Residua				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Molteplicità			
R1	R2	R3	R4
6	4	7	6

Disponibilità			

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

- La richiesta lascia il sistema in uno stato sicuro? …… Può essere accettata? ……..

Stato raggiunto dopo l'assegnazione di 1 istanza di R2 al processo C:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				

Esigenza Residua				
	R1	R2	R3	R4
A				
B				

Molteplicità			
R1	R2	R3	R4
6	4	7	6

C				
D				
E				

C				
D				
E				

Disponibilità			

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

Il processo ……. [può/non può] terminare, il vettore disponibilità diventa: ……..

- La richiesta lascia il sistema in uno stato sicuro? …… Può essere accettata? ……..

Soluzione

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

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

Esigenza Residua				
	R1	R2	R3	R4
A	0	0	0	0
B	2	1	0	2
C	2	4	4	3
D	2	1	0	1
E	2	2	4	2

Molteplicità			
R1	R2	R3	R4
6	4	7	6

Disponibilità			
0	1	1	1

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

- La richiesta lascia il sistema in uno stato sicuro? SI Può essere accettata? SI

Stato raggiunto dopo l'assegnazione di 1 istanza di R2 al processo C:

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

Esigenza Residua				
	R1	R2	R3	R4
A	0	0	0	0
B	2	1	0	2
C	2	3	4	3
D	2	1	0	1
E	2	2	4	2

Molteplicità			
R1	R2	R3	R4
6	4	7	6

Disponibilità			
0	0	1	1

Il processo A può terminare, il vettore disponibilità diventa: 2,0,1,4

Nessun altro processo può terminare. Se accettata, la richiesta lascerebbe il sistema in uno stato non sicuro, quindi non viene accordata e il processo C viene sospeso.

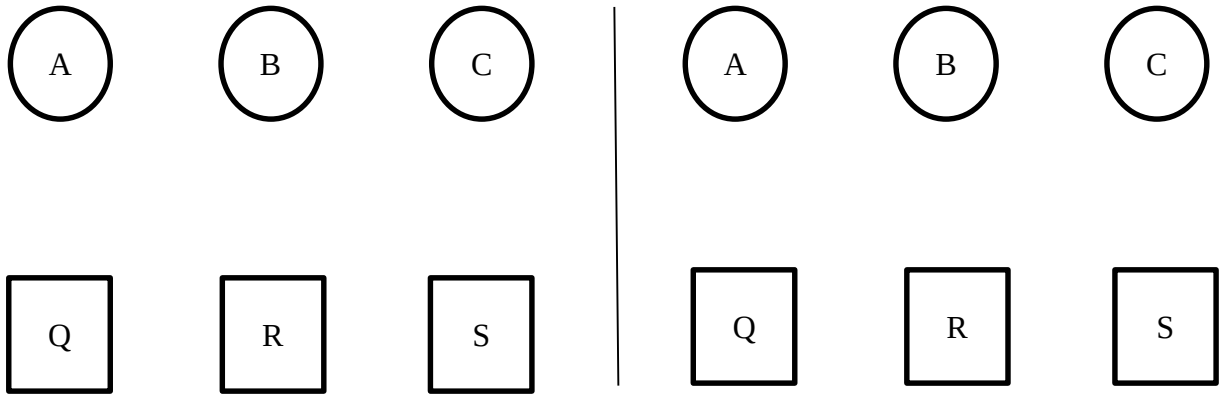
Sistemi Operativi e Laboratorio, Prova del 15/01/2014

Esercizio 7 (2 punti)

Dati tre processi A,B,C che osservano il paradigma di "wait while holding" (detto anche di "possesso e attesa") e tre risorse singole Q, R, S, da utilizzare in mutua esclusione e senza possibilità di prerilascio, supponiamo che il gestore assegni le risorse al processo richiedente alla sola condizione che la risorsa sia disponibile. Inizialmente tutte le risorse sono disponibili.

Si considerino le seguenti sequenze di richieste e rilasci:

Sequenza S1:	Sequenza S2:
1) B richiede Q;	1) B richiede Q;
2) B richiede R;	2) B richiede R;
3) A richiede S;	3) A richiede S;
4) A richiede R;	4) A richiede R;
5) C richiede Q;	5) B rilascia R;
6) B rilascia R;	6) A richiede Q;
7) B richiede S;	7) C richiede Q;
8) A richiede Q;	8) B rilascia Q;



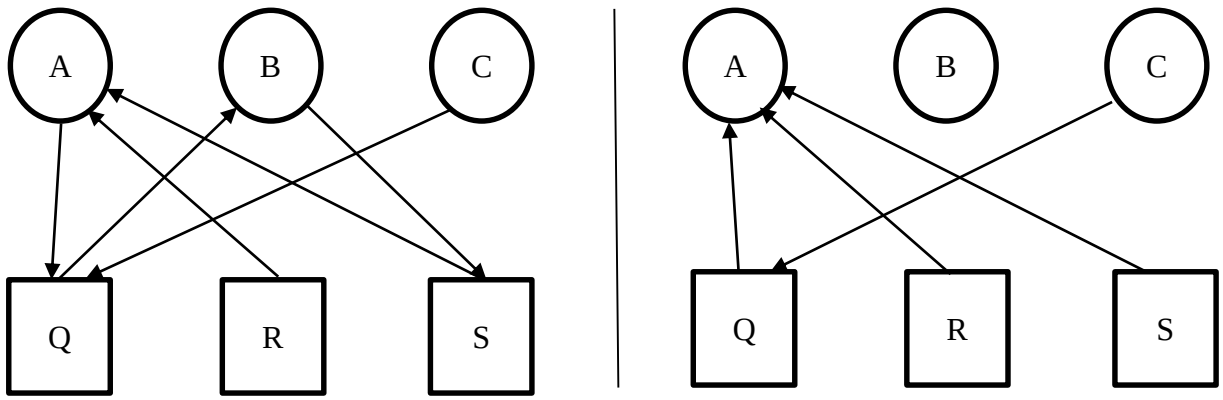
Queste sequenze provocano stallo? Motivare le risposte rappresentando nel grafo lo stato (assegnazioni e richieste pendenti) al termine di ciascuna sequenza.

Sequenza S1: Stallo[SI/NO]_____; Motivo: _____

Sequenza S2: Stallo[SI/NO]_____; Motivo: _____

Soluzione

Sequenza S1:	Sequenza S2:
1) B richiede Q;	1) B richiede Q;
2) B richiede R;	2) B richiede R;
3) A richiede S;	3) A richiede S;
4) A richiede R;	4) A richiede R;
5) C richiede Q;	5) B rilascia R;
6) B rilascia R;	6) A richiede Q;
7) B richiede S;	7) C richiede Q;
8) A richiede Q;	8) B rilascia Q;



La sequenza S1 provoca uno stallo: nel grafo si forma il ciclo A-Q-B-S-A

La sequenza S2 non provoca stallo: nel grafo risultante non sono presenti cicli.

Sistemi Operativi e Laboratorio, Prova del 15/01/2014

Esercizio 8 (2 punti)

Si consideri un sistema nel quale sono definiti il semaforo sem e i processi P1 (con priorità 3), P2 (con priorità 1) e P3 (con priorità 2). Lo scheduling avviene con una politica a priorità, che prevede il prerilascio e assegna il processore al processo pronto di priorità più. La politica applicata al semaforo è FIFO.
Al tempo t il processo P2 è in esecuzione e la coda pronti è vuota. Inoltre il semaforo sem ha valore 0 e la coda associata al semaforo contiene i processi P3->P1 (il processo P3 è in cima alla coda). Dopo il tempo t, si verificano in sequenza i seguenti di eventi:

1)	P2 esegue V(sem)	P2 esegue **signal(sem)** per gli studenti degli anni accademici precedenti l'A.A. 2013/14
2)	il processo in esecuzione esegue V(sem)	esegue **signal(sem)** per gli studenti degli anni accademici precedenti l'A.A. 2013/14
3)	scade il quanto di tempo	scade il quanto di tempo
4)	il processo in esecuzione esegue P(sem)	esegue **wait(sem)** per gli studenti degli anni accademici precedenti l'A.A. 2013/14

Si chiede di specificare quale processo è in esecuzione subito 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	Valore di sem	Coda di sem
1) P2 esegue V(sem) (o, equivalentemente, signal(sem)				
2) il processo in esecuzione esegue V(sem) (o, equivalentemente, signal(sem))				
3) scade il quanto di tempo				
4) Il processo in esecuzione esegue P(sem) (o, equivalentemente, wait(sem))				

Soluzione

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

NOTA: alla scadenza del quanto di tempo di P1 (evento 3) rimane in esecuzione P1, perché non esistono altri processi pronti con uguale priorità.

Esercizio 9 (2 punti)

In un sistema operativo che realizza i threads a livello kernel, i thread T11, T12 e T21 del processo P1 cooperano scambiando messaggi attraverso un buffer di una sola cella. Per l'interazione si utilizzano i semafori BufferVuoto (inizializzato ad 1), BufferPieno (inizializzato a 0) e Mutex (inizializzato ad 1). I thread T11 e T12 si comportano da produttori eseguendo la funzione deposito, e il thread T21 si comporta da consumatore eseguendo la funzione prelievo. Le funzioni sono definite nel modo seguente:

per gli studenti dell'anno accademico 2013/14:

```
void function deposito(&messaggio);
{
       P(&Mutex);
       P(&BufferVuoto);
       insert(messaggio);
       V(&Mutex);
       V(&BufferPieno);
}
```

```
void function prelievo(&mess);
{
       P(&Mutex);
       P(&BufferPieno);
       remove(&mess);
       V(&Mutex);
       V(&BufferVuoto);
}
```

per gli studenti degli anni accademici precedenti l'A.A. 2013/14:

```
void function deposito(&messaggio);
{
       wait(&Mutex);
       wait(&BufferVuoto);
```

```
void function prelievo(&mess);
{
       wait(&Mutex);
       wait(&BufferPieno);
```

```
        insert(messaggio);
        signal(&Mutex);
        signal(&BufferPieno);
}
```

```
        remove(&mess);
        signal(&Mutex);
        signal(&BufferVuoto);
}
```

La soluzione è corretta? Motivare la risposta.

_____

_____

Soluzione

La soluzione è sbagliata. I thread T11 e T12 possono sospendersi su P(&BufferVuoto) (o, equivalentemente su wait(&BufferVuoto)) senza rilasciare la mutua esclusione, causando stallo. In modo simile, il thread T21 può sospendersi su P(&BufferPieno) (o, equivalentemente su wait(&BufferPieno)) senza rilasciare la mutua esclusione, causando anch'esso stallo.

Esercizio 10 (2 punti)

Si consideri il seguente frammento di codice:

```
…
printf("uno");
a = fork();
if (a>0) {
       exec("/bin/pippo",NULL);
       printf("due ");
       }
else
   if (a==0)printf("tre");
   else printf("quattro");
printf("cinque");
…
```

Che cosa stampa il processo padre eseguendo questo frammento se il file /bin/pippo non è eseguibile?

Soluzione

Se la fork ha successo, il processo padre stampa: _____
Altrimenti stampa: _____

Soluzione

Se la fork ha successo, il processo padre esegue la exec, che fallisce, e stampa "uno" "due" "cinque"
Altrimenti non esegue la exec e stampa "uno" "quattro" "cinque"