SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

COGNOME ..... NOME ..... MATRICOLA ..... CORSO |A| |B|

ESERCIZIO A.1 (4 punti)

In un sistema che implementa i thread a livello del nucleo, all'interno di un processo sono definiti i thread `R, A1 e A2` che cooperano per un'attività di controllo industriale. Il thread `R` (rilevatore) effettua un ciclo nel quale rileva e analizza alcuni dati dell'impianto e scrive i risultati in un buffer condiviso. Il dato prodotto dal rilevatore è acquisito dai thread attuatori `A1` e `A2` che periodicamente e in modo indipendente tra loro e dal rilevatore leggono dal buffer l'ultimo valore disponibile e pilotano di conseguenza alcuni attuatori.

Dato che i thread attuatori non modificano il contenuto del buffer, possono accedere al buffer contemporaneamente. D'altra deve essere garantita la mutua esclusione nell'accesso al buffer tra il thread rilevatore e i thread attuatori.

Per gestire il buffer sono impiegate le seguenti variabili condivise (è anche mostrato il valore col quale sono inizializzate).

`Att_attivi=0, Att_in_attesa=0; R_in_attesa=false; R_attivo=false.`

Inoltre sono usati i seguenti semafori:

- `mutex` (inizializzato ad 1)
- `sem_R` (inizializzato a 0) per sospendere il thread rilevatore nel caso uno o più thread attuatori stiano leggendo dal buffer
- `sem_Att` (inizializzato a 0) per sospendere i thread attuatori nel caso il thread rilevatore stia scrivendo nel buffer

Thread rilevatore:

```
1    While (true) {
2            <rileva e analizza i dati>
3            wait(mutex);
4              if Att_attivi>0 R_in_attesa=true; else {R_attivo=true;signal(sem_R);}
5            signal (mutex);
6            wait(sem_R);
7            <scrive nel buffer>
8            wait(mutex);
9              while (Att_in_attesa>0) {
                   signal(sem_Att); Att_in_attesa--; Att_attivi ++;}
10           R_attivo=false;
11           signal (mutex);
     }
```

Thread Attuatori:

```
1    While (true) {
2            wait(mutex);
3            if R_attivo att_in_attesa++; else {Att_attivi ++; signal(sem_att);}
3            signal (mutex);
5            wait(sem_att);
6            <legge dal buffer>
7            wait(mutex);
8            Att_attivi--;
9           if att_attivi= 0 and R_in_attesa {
                signal(sem_R); R_in_attesa=false; R_attivo=true;}
10           signal (mutex);
11           <controlla gli attuatori>
     }
```

I thread sono schedulati con politica round robin e quanto di tempo di 10 ms.

Al tempo 0 la coda pronti è `A1-> A2,` mentre il thread `R` è in esecuzione ed esegue la riga 1. Allo stesso tempo `A1` e `A2` devono entrambi eseguire la riga 1.

Si assuma che l'esecuzione delle righe 2 e 7 del rilevatore richieda in 8 ms, e che l'esecuzione delle righe 6 e 11 degli attuatori venga completata in 12 ms,, mentre il tempo di esecuzione di tutte le altre istruzioni di `R, A1` e `A2` è trascurabile.

Dire quanto tempo impiega il thread A1 per completare la riga 11 e mostrare come evolve lo stato di `R, A1` e `A2` in questo intervallo di tempo.

SOLUZIONE

Tempo	In esec	Coda pronti	Att bloccati	R bloccato?	Commenti
0	R	A1 – A2	∅	NO	
10	A1	A2 --> R	∅	NO	Esaurito QdT, R ha eseguito linea 7 per 2 msec
10	A2	R	A1	NO	A1 si sospende eseguendo linea 5
10	R	∅	A1, A2	NO	A2 si sospende eseguendo linea 5
16	R	A1 --> A2	∅	NO	R termina linea 7 e riattiva A1, A2
20	A1	A2 --> R	∅	NO	Esaurito QdT, R ha eseguito linea 2 per 4 msec
30	A2	R--> A1	∅	NO	Esaurito QdT, A1 ha eseguito linea 6 per 10 msec
40	R	A1 --> A2	∅	NO	Esaurito QdT, A2 ha eseguito linea 6 per 10 msec
44	A1	A2	∅	SI	R termina linea 2 e si sospende eseguendo linea 6
46	A1	A2	∅	SI	A1 termina linea 6 e inizia linea 11
54	A2	A1		SI	Esaurito QdT; A1 ha eseguito linea 11 per 8 msec
56	A2	A1 --> R		NO	A2 termina linea 6 e riattiva R
64	A1	R --> A2		NO	Esaurito QdT; A2 ha eseguito linea 11 per 8 msec
68	A1	R --> A2		NO	A1 termina linea 11

A1 completa l'esecuzione della riga 11 al tempo 68

SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

ESERCIZIO A.2 (4 punti)

In un sistema vengono generati 6 processi (A,B,C,D,E,F), con i tempi di arrivo e le durate (in millisecondi) sotto specificate:

Processo	Durata	Tempo di arrivo
A	15	0
B	15	1
C	10	21
D	35	23
E	5	37
F	15	43

La politica di scheduling è la Round Robin, con quanto di tempo pari a 10 msec. Si suppone che, dopo il passaggio in esecuzione, ogni processo avanzi senza sospendersi fino all'esaurimento del quanto di tempo o fino alla terminazione. Mostrare nella tabella come evolve l'utilizzo del processore e calcolare per ogni processo il tempo di permanenza nel sistema, definito come differenza tra il tempo di terminazione e il tempo di arrivo. Trascurare il tempo di commutazione di contesto.

SOLUZIONE

T=	Arriva	Termina	In esec.	Coda Pronti
0	A		A	∅
1	B		A	B
10			B	A
20			A	B
21	C		A	B --> C
23	D		A	B-->C-->D
25		A	B	C-->D
30		B	C	D
37	E		C	D-->E
40		C	D	E
43	F		D	E-->F
50			E	F-->D
55		E	F	D

T=	Arriva	Termina	In esec.	Coda Pronti
65			D	F
75			F	D
80		F	D	∅
90			D	∅
95		D	∅	∅

Tempo di completamento:
A : 25
B: 30- 1= 29
C: 40- 21= 19
D: 95- 23= 72
E: 55- 37= 18
F: 80- 43= 37

SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

ESERCIZIO A.3 (3 punti)

Un sistema con 4 processi A, B, C, D e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [5, 6, 3, 6] adotta nei confronti dello stallo la politica di riconoscimento ed eliminazione. Ad un dato istante di tempo *t* tutti i processi sono sospesi e si è raggiunto uno stallo, come mostrato nelle tabelle seguenti.

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

Esigenza Massima				
	R1	R2	R3	R4
A	4			4
B	5	6	3	
C			2	3
D	1	3		6

Esigenza residua (esclusa l'assegnazione attuale)				
	R1	R2	R3	R4
A	2	0	0	1
B	4	5	3	0
C	0	0	0	2
D	1	0	0	4

Molteplicità			
R1	R2	R3	R4
5	6	3	6

Disponibilità			
2	2	1	0

Dire quali delle seguenti azioni (da considerarsi in alternativa) elimina lo stallo:

a) il processo A viene forzato a rilasciare 1 risorsa di tipo R1
b) il processo C viene forzato a rilasciare 1 risorsa di tipo R4

SOLUZIONE

Ipotesi a): Stato raggiunto dopo il rilascio di un'istanza di R1 da parte di A:

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

Esigenza Massima				
	R1	R2	R3	R4
A	4			4
B	5	6	3	
C			2	3
D	1	3		6

Esigenza residua (esclusa l'assegnazione attuale)				
	R1	R2	R3	R4
A	3	0	0	1
B	4	5	3	0
C	0	0	0	2
D	1	0	0	4

Molteplicità			
R1	R2	R3	R4
5	6	3	6

Disponibilità			
3	2	1	0

Per la verifica dello stato sicuro:
Nessun processo può terminare
Di conseguenza: stallo eliminato? NO

b) Stato raggiunto dopo il rilascio di un'istanza di R4 da parte di C:

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

Esigenza Massima				
	R1	R2	R3	R4
A	4			4
B	5	6	3	
C			2	3
D	1	3		6

Esigenza residua (esclusa l'assegnazione attuale)				
	R1	R2	R3	R4
A	2	0	0	1
B	4	5	3	0
C	0	0	0	3
D	1	0	0	4

Molteplicità			
R1	R2	R3	R4
5	6	3	6

Disponibilità			
2	2	1	1

Per la verifica dello stato sicuro:
1) Il processo A può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 4,2,1,4}
2) Il processo C può terminare
La disponibilità di {R1, R2, R3, R4} diviene {4,2,3,4}
3) Il processo D può terminare
La disponibilità di {R1, R2, R3, R4} diviene {4,5,3,6}
4) Il processo B può terminare
La disponibilità di {R1, R2, R3, R4} diviene {5,6,3,6}

Di conseguenza: stallo eliminato? SI

SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

Esercizio A.4 (2 punti)

In un sistema operativo che realizza i thread a livello kernel, il thread T_{1i} del processo P_1 e il thread T_{2k} del processo P_2 si scambiano messaggi attraverso una pila di capacità illimitata. Si consideri la seguente soluzione realizzata con il semaforo Mutex, inizializzato ad 1.

Thread T_{1i}

```
While (True) {
      ……..
      messaggio = produce_messaggio();
      wait(&Mutex);
      puntatore ++;
      Pila[puntatore]= messaggio;
      signal(&Mutex);
      {
```

Thread T_{2k}

```
While (True) {
      wait(&Mutex);
      mess= Pila[puntatore];
      puntatore --;
      signal(&Mutex);
      consuma_messaggio(mess)
      }
```

La soluzione è corretta? Motivare la risposta.

SOLUZIONE

La soluzione è errata
Motivo: thread di processi diversi non possono condividere dati

ESERCIZIO A.5 (2 punti)

In un sistema UNIX il processo P esegue la chiamata di sistema **`int pipe (int fd[2]),`** dove **`fd[0]`** è il descrittore del lato di lettura e **`fd[1]`** quello del lato di scrittura, e successivamente esegue per due volte la chiamata di sistema **`fork`** generando i processi F1 e F2. I processi figli sostituiscono immediatamente il proprio codice mediante la chiamata **`exec`** e successivamente, al tempo t1, F1 esegue la chiamata di sistema **`close (fd[0])`**.
Si chiede quali delle seguenti operazioni sono eseguibili dopo il tempo t1:

1. scrittura di 10 caratteri eseguita da F1
2. lettura di 5 caratteri eseguita da F2
3. lettura di 2 caratteri eseguita da P
4. lettura di 10 caratteri eseguita dal processo Q, padre di P.

SOLUZIONE

OPERAZIONE	ESEGUIBILE?
1. scrittura di 10 caratteri eseguita da F1	SI
2. lettura di 5 caratteri eseguita da F2	SI
3. lettura di 2 caratteri eseguita da P	SI
4. lettura di 10 caratteri eseguita dal processo Q, padre di P	NO

SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

ESERCIZIO B.1 (4 PUNTI)

Un sistema operativo simile a UNIX gestisce la memoria con paginazione a domanda e utilizza il processo *PageDaemon,* con parametri *lotsfree= 6 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), *Pag* (pagina del processo caricata nel blocco), *Rif* (bit di pagina riferita, utilizzato da *Second Chance*). Tutti questi campi sono vuoti se il blocco è libero. Al tempo *t* sono presenti i processi A, B, C e la *Core Map* ha la configurazione mostrata in figura, con il puntatore dell'algoritmo di sostituzione posizionato sul blocco 11. I primi 6 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione.

Proc							C	A	B	A		B	C	A	C		C	B	A	B		C		C
Pag							0	1	0	2		6	3	8	9		12	2	7	3		7		2
Rif							0	1	0	0		1	1	0	1		0	0	1	0		1		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*

Il *PageDaemon* interviene al tempo *t* e successivamente ogni 10 msec. Ad ogni intervento, *PageDaemon* avanza per 1 msec utilizzando in modo esclusivo il processore e scarica fino a 3 pagine o, in alternativa, esegue lo *swapout* di un processo.
Ai fini dello *swapout,* la selezione del processo avviene in ordine decrescente di pagine caricate in memoria. In caso di errori di pagina, i blocchi liberi vengono assegnati in ordine crescente di indice.
Considerare la seguente evoluzione del sistema:

1. Dal tempo *t* al tempo *t+1* avanza il processo *PageDaemon*;
2. Dal tempo *t+1* al tempo *t+10* avanzano i processi B e C, che riferiscono nell'ordine le pagine B0, B1, B6, C1, C0, C4, B2, B5
3. Dal tempo *t+10* al tempo *t+11* avanza il processo *PageDaemon*;
4. Dal tempo *t+11* al tempo *t+20* avanzano i processi A e B, che riferiscono nell'ordine le pagine A3, A6, A2, A1, B0, B2, B3, B0;
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

Proc							C	A	B	A		B	C		C			B	A	B		C		C
Pag							0	1	0	2		6	3		9			2	7	3		7		2
Rif							0	1	0	0		0	0		0			0	1	0		1		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 (scaricate 2 pagine)*

Proc							C	A	B	A	B	B	C	C	C	C	B	B	A	B		C		C
Pag							0	1	0	2	1	6	3	1	9	4	5	2	7	3		7		2
Rif							1	1	1	0	1	1	0	1	0	1	1	1	1	0		1		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+10 (2 blocchi liberi)*

Proc								A	B	A	B	B					B	B	A	B				
Pag								1	0	2	1	6					5	2	7	3				
Rif								1	1	0	1	1					1	1	1	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+11 (swapout del processo C)*

Proc							A	A	B	A	B	B	A				B	B	A	B				
Pag							3	1	0	2	1	6	6				5	2	7	3				
Rif							1	1	1	1	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 (6 blocchi liberi)*

Proc							A	A	B	A	B	B	A				B	B	A	B				
Pag							3	1	0	2	1	6	6				5	2	7	3				
Rif							1	1	1	1	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+21 (scaricate 0 pagine)*

SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

ESERCIZIO B.2 (4 punti)

In un file system UNIX i blocchi del disco hanno ampiezza di 2Kbyte e i puntatori ai blocchi sono a 16 bit. Gli i-node contengono, oltre agli attributi, 10 indirizzi diretti e 3 indirizzi indiretti.
Si consideri il file rappresentato dal seguente i-node:

Indirizzo	0	1	2	3	4	5	6	7	8	9	10	11	12
Valore	70	340	412	44	609	610	611	612	981	172	179	198	211

Alcuni frammenti dei blocchi 179, 198, 211, 5001, 6121, 7002 e 7101 sono riportati nel seguito.
Blocco 179:

Indirizzo	…	11	12	13	14	15	16	17	18	19	…
Valore	…	9101	9122	9271	8987	9765	9810	8897	9456	9500	…

Blocco 198:

Indirizzo	0	1	2	3	4	5	6	7	8	9	…
Valore	7000	7001	7002	7003	7101	7102	7103	7107	7200	7210	…

Blocco 211:

Indirizzo	0	1	2	3	4	5	6	7	8	9	…
Valore	6121	6122	6123	6124	6125	6126	6100	6101	6102	6103	…

Blocco 5001:

Indirizzo	…	17	18	19	20	21	22	23	24	25	…
Valore	…	210	205	203	201	208	207	212	204	202	…

Blocco 6121:

Indirizzo	0	1	2	3	4	5	6	7	8	9	…
Valore	5001	5002	5003	5004	5005	5006	5007	5008	5009	5010	…

Blocco 7101:

Indirizzo	…	99	100	101	102	103	104	105	106	107	…
Valore	…	5800	5801	5802	5803	5987	5988	5989	5990	5991	…

Blocco 7002:

Indirizzo	…	156	157	158	159	160	161	162	163	164	…
Valore	…	6787	6788	6789	6700	6568	6569	6570	6571	6572	…

Utilizzando la tabella sotto riportata, dire quali blocchi fisici corrispondono ai seguenti blocchi logici del file:
9, 23, 3239, 3243, 1049635

SOLUZIONE

Ogni blocco indiretto contiene 2048/2= 1024 indirizzi
Per $0 \leq ind \leq 9$ al blocco logico di indice *ind* corrisponde il blocco fisico individuato dall'indirizzo diretto *ind*
Per $10 \leq ind < 1034$, al blocco logico di indice *ind* corrisponde il blocco fisico individuato dall'indirizzo *ind-10* del blocco indiretto semplice.
Per $1034 \leq ind < 1049610$, al blocco logico di indice *ind* corrisponde il blocco fisico individuato dall'indirizzo *(ind-1034)* ***mod*** *1024* del blocco indiretto doppio *(ind-1034)* ***div*** *1024*
Per $1049610 \leq ind$, al blocco logico di indice *ind* corrisponde il blocco fisico individuato dall'indirizzo *(ind-1049610)* ***mod*** *1024* del blocco indiretto triplo *(ind-1049610)* ***div*** *1024*

BLOCCO LOGICO	BLOCCO FISICO	Raggiunto attraverso:			
		INDIRIZZO DIRETTO	BLOCCO INDIRETTO PRIMO LIVELLO	BLOCCO INDIRETTO SECONDO LIVELLO	BLOCCO INDIRETTO TERZO LIVELLO
9	172	9	Blocco ..... Ind. nel blocco.....	Blocco ..... Ind. nel blocco.....	Blocco ..... Ind. nel blocco.....
23	9271		Blocco 179 Ind. nel blocco 13	Blocco ..... Ind. nel blocco.....	Blocco ..... Ind. nel blocco.....
3239	6788		Blocco 198 Ind. nel blocco 2	Blocco 7002 Ind. nel blocco 157	Blocco ..... Ind. nel blocco.....
3243	6569		Blocco 198 Ind. nel blocco 4	Blocco 7101 Ind. nel blocco 103	Blocco ..... Ind. nel blocco.....
1049635	NON RAPPRESENTABILE con puntatori a 16 bit		Blocco Ind. nel blocco	Blocco Ind. nel blocco	Blocco Ind. nel blocco

SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

ESERCIZIO B.3 (3 punti)

Si consideri un disco con 1000 cilindri, 4 facce e 1000 settori per traccia.
1. Tradurre i seguenti indici di blocco nelle rispettive terne <cilindro, faccia, settore>:
   - 2.081.678
   - 3.982.711
   - 60.078
2. Tradurre le seguenti terne <cilindro, faccia, settore> in indici di blocco:
   - <482; 0; 999>
   - <191; 1; 432>
   - <227; 1; 90>

SOLUZIONE

1. Traduzione degli indici di blocco in terne:
cilindro = IndiceDiBlocco **div** (4*1000)
faccia = (IndiceDiBlocco **mod** (4*1000)) **div** 1000
settore = (IndiceDiBlocco **mod** (4*1000)) **mod** 1000

Indice di blocco	cilindro	faccia	settore
2.081.678	520	1	678
3.982.711	995	2	711
60.078	15	0	78

2. Traduzione delle terne in indici di blocco:
blocco= cilindro*(4*1000) + faccia*100 + settore

cilindro	faccia	settore	Indice di blocco
482	0	999	1.928.999
191	1	432	765.432
227	1	90	909.090

ESERCIZIO B.4 (2 PUNTI)

In un sistema che gestisce la memoria con paginazione dinamica a domanda, gli indirizzi logici sono codificati con 32 bit e le pagine logiche e fisiche hanno ampiezza di 2 kByte.Per la traduzione degli indirizzi si utilizzano tabelle delle pagine a 2 livelli. La tabella di primo livello comprende 2^{11} elementi.
Gli elementi di ogni tabella di primo o secondo livello occupano 3 byte e riservano 2 bit agli indicatori di pagina caricata e di pagina riferita, mentre i rimanenti individuano un indice di blocco fisico.
Si chiede:
1. la lunghezza del campo offset, in numero di bit;
2. la massima dimensione della memoria fisica (numero di blocchi e di byte, espressi come potenze di 2).
3. la lunghezza delle tabelle di secondo livello (numero di elementi);
4. lo spazio occupato in memoria dalla tabella di primo livello (numero di byte)
5. lo spazio occupato in memoria da ogni tabella di secondo livello (numero di byte).

SOLUZIONE

1. lunghezza del campo offset : 11 bit
2. massima dimensione della memoria fisica : 2^{22} blocchi; $2^{22} * 2^{11} = 2^{33}$ byte (8 Gbyte)
3. lunghezza delle tabelle di secondo livello: 2^{10} elementi;
4. spazio occupato in memoria dalla tabella di primo livello: $3*2^{11}= 6*2^{10}$ byte
5. spazio occupato in memoria da ogni tabella di secondo livello: $3*2^{10}$ byte

SISTEMI OPERATIVI, CORSI A e B - TERZO APPELLO - 8/6//2007

ESERCIZIO B.5 (2 PUNTI)

In un file system UNIX dove ogni i-node occupa 1 blocco, si consideri la directory */home/caio/vecchi/dati/*
Supponendo che ogni directory di questo path occupi 1 blocco e che la directory radice sia caricata in memoria, mentre tutti gli altri i-node e tutte le directory interessate risiedono su disco, calcolare il numero di accessi al disco necessari per visualizzare il contenuto della directory *dati*.

SOLUZIONE

1. 1 accesso per leggere lo i-node di *home*
2. 1 accesso per leggere la directory *home*
3. 1 accesso per leggere lo i-node di *caio*
4. 1 accesso per leggere la directory *caio*
5. 1 accesso per leggere lo i-node di *vecchi*
6. 1 accesso per leggere la directory *vecchi*
7. 1 accesso per leggere lo i-node di *dati*
8. 1 accesso per leggere il primo (e unico) blocco di *dati*

In totale : 8 accessi.