SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO - 14/9/2006

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

ESERCIZIO A-1 (4 punti)

Si consideri un processore che dispone dei seguenti registri:

- i registri speciali PC (program counter), PS (program status) e SP(stack pointer);
- i registri generali R1, R2 e R3, utilizzati sia nello stato utente, sia nello stato supervisore

Il sistema operativo riserva un'area di memoria (appartenente al nucleo) per il vettore di interruzione e per lo stack del nucleo. Il vettore di interruzione specifica, per ogni interruzione, un indirizzo nell'area di memoria del nucleo e una parola di stato.

- Al riconoscimento di un'interruzione, l'hardware salva i registri generali e speciali nello stack del nucleo e salta alla funzione di servizio dell'interruzione.
- Nella fase di esecuzione dell'istruzione IRET, l'hardware ripristina **tutti** i registri dallo stack del nucleo.

A un certo tempo, mentre è in esecuzione il processo P1, viene riconosciuta l'interruzione INTD lanciata da un dispositivo di I/O. La funzione di servizio attivata con questo meccanismo sospende il processo P1 e fa passare in esecuzione il processo P2 che gestisce il dispositivo.
All'istante in cui viene riconosciuta l'interruzione, i registri del processore, i descrittori di P1 e P2 e lo stack del nucleo hanno i contenuti mostrati in figura, che mostra anche il contenuto dell'elemento del vettore di interruzione associato all'interruzione INTD.
Si chiede:

1) Lo stato del processore dopo il riconoscimento di INTD;
2) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito dopo il riconoscimento di INTD;
3) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito prima dell'estrazione dell'istruzione IRET dall'indirizzo 1A00;
4) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito dopo l'esecuzione dell'istruzione IRET;
5) Lo stato del processore subito dopo l'esecuzione dell'istruzione IRET

Descrittore di P1	
.....	.....
PC	AABB
PS	BBCC
SP	CCCC
R1	ABCD
R2	1100
R3	0011

Descrittore di P2	
.....	.....
PC	33AA
PS	22AB
SP	1111
R1	0000
R2	1000
R3	1111

Stack del nucleo	
1000	ABCD
1001	
1002	
1003	
1004	
1005	
1006	

Registri	
PC	2000
PS	3000
SP	4000
R1	5555
R2	6666
R3	7777

INDIRIZZO	1000
PAROLA DI STATO	00FF
Vettore di interruzione	

Stack pointer del nucleo	1000

SOLUZIONE

1) Stato del processore: SUPERVISORE

Nelle tabelle seguenti: riportare solo i contenuti che cambiano

2)

Descrittore di P1	
.....	.....
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato
R3	invariato

Descrittore di P2	
.....	.....
PC	invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	invariato
R3	invariato

Stack del nucleo	
1000	ABCD
1001	2000
1002	3000
1003	4000
1004	5555
1005	6666
1006	7777

Registri	
PC	1000
PS	00FF
SP	1006
R1	??
R2	??
R3	??

3)

Descrittore di P1	
.....	.....
PC	2000
PS	3000
SP	4000
R1	5555
R2	6666
R3	7777

Descrittore di P2	
.....	.....
PC	invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	invariato
R3	invariato

Stack del nucleo	
10000	ABCD
1001	33AA
1002	22AB
1003	1111
1004	0000
1005	1000
1006	1111

Registri	
PC	1A00
PS	00FF
SP	1006
R1	??
R2	??
R3	??

4)

Descrittore di P1	
.....	.....
PC	invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	invariato
R3	invariato

Descrittore di P2	
.....	.....
PC	invariato
PS	Invariato
SP	Invariato
R1	Invariato
R2	invariato
R3	invariato

Stack del nucleo	
1000	ABCD
1001	
1002	
1003	
1004	
1005	
1006	

Registri	
PC	33AA
PS	22AB
SP	1111
R1	0000
R2	1000
R3	1111

5) Stato del processore: UTENTE

SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO - 14/9/2006

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

ESERCIZIO A-2 (4 punti)

Un sistema con processi A, B, C, D e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [4, 4, 5, 5], ha raggiunto lo stato mostrato nelle tabelle seguenti.

MOLTEPLICITA' →

R1	R2	R3	R4
4	5	5	5

ASSEGNAZIONE ATTUALE →

	R1	R2	R3	R4
A	2	1	1	1
B	2	0	1	2
C	0	1	0	0
D	0	2	2	2

	R1	R2	R3	R4
A	0	2	0	0
B	0	0	0	1
C	0	0	0	2
D	0	0	3	0

ESIGENZA ATTUALE

DISPONIBILTA' ATTUALE

R1	R2	R3	R4
0	1	1	0

A partire da questo stato A richiede in sequenza due risorse di R2, B richiede una risorsa di R4, C richiede una risorsa di R4 e D richiede in sequenza due risorse di R3.Si chiede:

1) Dopo queste richieste, il sistema è in stallo?
2) Se si sopprime il processo A, qual e' lo stato raggiunto dal sistema immediatamente dopo la soppressione?
3) Questo stato è sicuro?

SOLUZIONE

1) Il sistema è in stallo.
2) Stato raggiunto dopo l'eliminazione del processo A:

MOLTEPLICITA' →

R1	R2	R3	R4
4	5	5	5

ASSEGNAZIONE ATTUALE →

	R1	R2	R3	R4
A	-	-	-	-
B	2	0	1	2
C	0	1	0	0
D	0	2	3	2

	R1	R2	R3	R4
A	-	-	-	-
B	0	0	0	1
C	0	0	0	2
D	0	0	2	0

ESIGENZA ATTUALE

DISPONIBILTA' ATTUALE

R1	R2	R3	R4
2	2	1	1

3) Lo stato raggiunto eliminando il processo A è sicuro. Infatti:
   - Il processo B può terminare
     La disponibilità di {R1, R2, R3, R4} diviene { 4, 2 , 2 , 3 }
   - Il processo C può terminare
     La disponibilità di {R1, R2, R3, R4} diviene { 4 , 2 , 2 , 3}
   - Il processo D può terminare
     La disponibilità di {R1, R2, R3, R4} diviene { 4 , 5 , 5 , 5 }

SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO - 14/9/2006

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

ESERCIZIO A-3 (3 punti)

Si considerino i due thread A (produttore) e B (consumatore), realizzati a livello kernel, che condividono un buffer a n posizioni (numerate da 0 a n-1) sul quale, rispettivamente, depositano e prelevano messaggi. Ogni messaggio occupa una posizione del buffer. A tal fine A e B condividono un semaforo di mutua esclusione *mux* (inizializzato a 1), due variabili di condizione *libero* e *pieno* e una variabile *n_elem* inizializzata a 0 che rappresenta il numero di posizioni occupate nel buffer.
Completare il codice del thread B inserendo opportunamente i comandi **pthread_mutex_lock**, **pthread_mutex_unlock**. **pthread_cond_wait, pthread_cond_signal** con i rispettivi parametri.

SOLUZIONE

Thread A:

```
while (true) {
<produce un valore v>
pthread_mutex_lock(&mux);
if (n_elem == n-1) pthread_cond_wait(&libero,&mux);
<deposita v nel buffer>
if (n_elem == 0) pthread_cond_signal(&pieno);
n_elem++;
pthread_mutex_unlock(&mux)
}
```

Thread B:

```
while (true) {
pthread_mutex_lock(&mux);
if (n_elem == 0) pthread_cond_wait(&pieno,&mux);
<preleva un valore v dal buffer>
if (n_elem == n-1) pthread_cond_signal(&libero);
n_elem--;
pthread_mutex_unlock(&mux)
<consuma v>
}
```

ESERCIZIO A-4 (2 punti)

In un sistema con thread realizzati a livello utente, sono presenti i processi P1 con thread T11, T12, il processo P2 con i thread T21 e T22 e il processo P3 con i thread T31 e T32. I thread alternano tra lo stato di esecuzione, assegnato allo scheduler del processo, e quello di pronto, raggiunto quando il thread esegue l'operazione *thread-yield*. Ogni processo gestisce i suoi thread con politica FIFO.
I processsi sono gestiti con priorità e il processo riattivato con una *signal* assume priorità massima.
Al tempo *T* è in esecuzione il processo P1, il processo P2 è pronto e il processo P3 è sospeso sul semaforo *sem1*. Nel processo P1 è in esecuzione il thread T11 e i rimanenti thread dei tre processi sono pronti, con il seguente ordinamento nelle rispettive code:
Processo P1: T12 Processo P2: T21->T22 Processo P3: T31->T32.
Quale thread è in esecuzione dopo ogni evento della seguente sequenza:

1. Il thread in esecuzione esegue una operazione di *thread_yield*;
2. il thread in esecuzione esegue *wait(sem1)*;
3. il thread in esecuzione esegue una *signal* sul semaforo *sem1*;
4. il thread in esecuzione esaurisce il quanto di tempo;

SOLUZIONE

	Sequenze di eventi	Thread in esecuzione	Coda di P1	Coda di P2	Coda di P3
1	Il thread in esecuzione esegue l'operazione *thread_yield*	T12	T11	T21->T22	T31->T32
2	il thread in esecuzione esegue *wait(sem1)*	T21	T12->T11	T22	T31->T32
3	il thread in esecuzione esegue *signal*(*sem1)*	T31	T11->T12	T21->T22	T32
4	il thread in esecuzione esaurisce il quanto di tempo	T32	T11->T12	T21->T22	T31

SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO - 14/9/2006

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

ESERCIZIO A-5 (2 punti)

In un sistema operativo ad ambiente locale, i processi A_1, A_2, ... ,A_m (clienti) e il processo B (servente) comunicano mediante primitive di comunicazione. Per l'invio è disponibile la primitiva *send(&mess, destinatario)*, che è asincrona e specifica il nome del processo destinatario. Per la ricezione sono disponibili le primitive bloccanti *receive(mittente, &M),* che specifica il nome del processo dal quale si vuole ricevere un messaggio, e *receive (&M)*, che non specifica il mittente e riceve da un qualsiasi processo. I messaggi contengono i campi *mitt* e *info*, il primo dei quali è il nome del mittente.
Il generico processo cliente si sincronizza con il servente attendendo una risposta. Il processo servente riceve messaggi dai clienti e si sincronizza con il mittente di ciascun messaggio (protocollo *rendez-vous esteso*).
Completare il codice del generico processo cliente A_i e del processo servente B inserendo le opportune primitive di comunicazione.

SOLUZIONE

Processo A_i

```
while (true) {
        mess = produce_mess();
        send(&mess,B);
        receive(B,&risp);
        <consuma risposta>
}
```

Processo B

```
while (true) {
        receive(&mess);
        mittente= estrazione_mittente(&mess);
        risposta= produci_risposta();
        send(&risposta,mittente);
}
```

SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO - 14/9/2006

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

ESERCIZIO B-1 (4 PUNTI)

Un disco con 2 facce, 50 settori per traccia e 200 cilindri ha un tempo di seek (proporzionale al numero di cilindri attraversati) pari a 0,5 ms per ogni cilindro. Il periodo di rotazione è di 5 msec: di conseguenza il tempo impiegato per percorrere un settore è 0,1 msec.
Al tempo 0 termina l'esecuzione dei comandi relativi al cilindro 170 e sono pendenti le seguenti richieste di lettura o scrittura:

- cilindro 150 : settori 20 e 30 della faccia 0
- cilindro 190 : settore 44 della faccia 1
- cilindro 165 : settori 30 della faccia 0 e 30 della faccia 1

Successivamente arrivano i seguenti comandi:

- al tempo 20: cilindro 180 settore 7 della faccia 1.
- al tempo 50: cilindro 50 settore 12 della faccia 0.
- al tempo 80: cilindro 80 settore 40 della faccia 0.

Lo scheduling è effettuato con politica SCAN e al tempo iniziale è attiva la fase di salita. Calcolare il tempo necessario per eseguire tutte le operazioni. Il tempo di esecuzione di ogni operazione è uguale alla somma dell'eventuale tempo di *seek*, del ritardo rotazionale (*dopo ogni operazione di seek, da calcolare considerando il valore di caso peggiore, pari a 5 msec*) e del tempo di percorrenza del settore indirizzato. Quando si raggiunge un cilindro, i comandi pendenti devono essere eseguiti nell'ordine in cui sono elencati.

SOLUZIONE

op. su cilindro:	190	; settore:	44						
inizio:	0	; seek:	10	; rotazione:	5	; percorrenza:	0,1	; fine:	15,1
op. su cilindro:	165	; settore:	30						
inizio:	16,1	; seek:	12,5	; rotazione:	5	; percorrenza:	0,1	; fine:	32,7
Arrivato comando su cilindro 180									
op. su cilindro:	165	; settore:	30						
inizio:	32,7	; seek:	0	; rotazione:	5- 0,1	; percorrenza:	0,1	; fine:	37,7
op. su cilindro:	150	; settore:	20						
inizio:	37,7	; seek:	7,5	; rotazione:	5	; percorrenza:	0,1	; fine:	50,3
Arrivato comando su cilindro 50									
op. su cilindro:	150	; settore:	30						
inizio:	50,3	; seek:	0	; rotazione:	1- 0,1	; percorrenza:	0,1	; fine:	51,3
op. su cilindro:	50	; settore:	12						
inizio:	51,3	; seek:	50	; rotazione:	5	; percorrenza:	0,1	; fine:	106,4
Arrivato comando su cilindro 80									
op. su cilindro:	80	; settore:	40						
inizio:	106,4	; seek:	15	; rotazione:	5	; percorrenza:	0,1	; fine:	126,5
op. su cilindro:	180	; settore:	7						
inizio:	126,5	; seek:	50	; rotazione: 5		percorrenza 0,1		; fine	181,6

SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO - 14/9/2006

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

ESERCIZIO B-2 (4 punti)

Un sistema operativo simile a UNIX, che gestisce la memoria con paginazione a domanda, utilizza il processo *PageDaemon,* con parametri *lotsfree= 5 e minfree=2,* 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 6 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall'algoritmo di sostituzione.

Proc							C	A	B	A		B	C	D	D		D	B	A	B		C		C
Pag							0	1	0	2		6	3	1	2		6	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 occupando in modo esclusivo il processore e scarica fino a 3 pagine o, in alternativa, esegue lo *swapout* di un processo.
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, B2, B5, C2, C0, C4
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 D e B, che riferiscono nell'ordine le pagine D3, D6, D2, D1, B0, B2, B6, B5;
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							C	A	B	A		B	C		D		D	B	A	B		C		C
Pag							0	1	0	2		6	3		2		6	2	7	3		7		2
Rif							0	1	0	0		0	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+1*

Proc							C	A	B	A	B	B	C	B	D	C	D	B	A	B		C		C
Pag							0	1	0	2	1	6	3	5	2	4	6	2	7	3		7		2
Rif							1	1	1	0	1	1	0	1	1	1	0	1	1	0		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+10*

Proc							C	A	B		B	B	C	B	D	C		B	A			C		C
Pag							0	1	0		1	6	3	5	2	4		2	7			7		2
Rif							0	0	0		1	1	0	1	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+11*

Proc							C	A	B	D	B	B	C	B	D	C	D	B	A	D		C		C
Pag							0	1	0	3	1	6	3	5	2	4	6	2	7	1		7		2
Rif							0	0	1	1	1	1	0	1	1	0	1	1	0	1		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+20*

Proc							C	A	B	D	B	B		B	D		D	B		D		C		C
Pag							0	1	0	3	1	6		5	2		6	2		1		7		2
Rif							0	0	1	1	0	0		0	0		0	0		1		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

SISTEMI OPERATIVI, CORSI A e B - SESTO APPELLO - 14/9/2006

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

ESERCIZIO B-3 (3 punti)

In un file system di tipo Unix i blocchi hanno dimensione di 2 KB e i puntatori ai blocchi sono codificati con 32 bit (4 byte). Gli i-node contengono 10 puntatori diretti, un puntatore indiretto singolo, un puntatore indiretto doppio e un puntatore indiretto triplo.
Calcolare:

1. il massimo numero di blocchi e di byte indirizzabili con i soli puntatori diretti;
2. il massimo numero di blocchi e di byte indirizzabili con i puntatori diretti e il puntatore indiretto singolo;
3. il massimo numero di blocchi e di byte indirizzabili con i puntatori diretti e i puntatori indiretti singolo e doppio;
4. il massimo numero di blocchi e di byte indirizzabili con i puntatori diretti e i puntatori indiretti singolo, doppio e triplo;
5. la massima dimensione di un file, espressa in blocchi e in byte.

SOLUZIONE

I blocchi indiretti contengono $2^{11}/4 = 2^9 = 512$ indirizzi.

1. 10 blocchi; $10 * 2^{11} = 20480$ byte;
2. $10 + 2^9$ blocchi = $10 * 2^{11} + 2^{20}$ byte
3. $10 + 2^9 + 2^{18}$ blocchi = $10 * 2^{11} + 2^{20} + 2^{29}$ byte
4. $10 + 2^9 + 2^{18} + 2^{27}$ blocchi = $10 * 2^{11} + 2^{20} + 2^{29} + 2^{38}$ byte
5. massima dimensione di un file: $10 + 2^9 + 2^{18} + 2^{27}$ blocchi = $10 * 2^{11} + 2^{20} + 2^{29} + 2^{38}$ byte

ESERCIZIO B-4 (2 PUNTI)

In un sistema che gestisce la memoria con rilocazione dinamica, spazio logico segmentato (con segmento codice e segmento dati) e caricamento in partizioni variabili, i registri base e limite del segmento codice hanno a un certo istante i valori B_1= 15.000 e L_1= 35.000, mentre i registri base e limite del segmento dati hanno i valori B_2= 55.000 e L_2= 20.000.
Supponendo che il processo in esecuzione estragga istruzioni con indirizzi logici 16.000, 40.000 e 5.000, e riferisca dati con indirizzi logici 10.000, 20.000 e 60.000, dire se ognuno dei precedenti indirizzi logici è legittimo e, in caso affermativo, calcolare il corrispondente indirizzo fisico.

SOLUZIONE

Indirizzo logico	Legittimo?	Indirizzo fisico
16.000	SI	15.000 +16.000=31.000
40.000	NO	
5.000	SI	15.000+5.000=20.000
10.000	SI	55.000+10.000=65.000
20.000	NO	
60.000	NO	

ESERCIZIO B-5 (2 PUNTI)

In un file system UNIX dove ogni i-node occupa 1 blocco, si consideri il file */usr/filippo/appunti/SistemiOperativi*
Supponendo che ogni directory di questo path occupi 1 blocco e che lo i-node della directory *usr* sia caricato 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 leggere l'intero file *SistemiOperativi,* che occupa 3 blocchi.

SOLUZIONE

1. 1 accesso per leggere la directory *usr*
2. 1 accesso per leggere lo i-node di *filippo*
3. 1 accesso per leggere la directory *filippo*
4. 1 accesso per leggere lo i-node di *appunti*
5. 1 accesso per leggere la directory *appunti*
6. 1 accesso per leggere lo i-node di *SistemiOperativi*
7. 3 accesso per leggere i 3 blocchi di *SistemiOperativi*

In totale : 9 accessi.