

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

ESERCIZIO 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 R1 e R2, utilizzati sia nello stato utente che in quello supervisore.

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

- Al riconoscimento di un'interruzione, l'hardware salva tutti i registri nello stack del nucleo (azzerando contemporaneamente i registri R1 e R2) 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 è in esecuzione il processo P1 che esegue una chiamata di sistema con l'istruzione SVC. La primitiva eseguita con questo meccanismo sospende il processo P1 e fa passare in esecuzione il processo P2, che si trova nello stato di pronto.

All'istante in cui viene estratta l'istruzione SVC, 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 alle interruzioni da programma.

Si chiede:

- 1) Lo stato del processore dopo l'esecuzione della SVC;
- 2) Il contenuto dei registri, dei descrittori e dello stack del nucleo dopo l'esecuzione della SVC;
- 3) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito prima dell'esecuzione della IRET nell'ipotesi che la IRET sia contenuta nella locazione 3100;
- 4) Il contenuto dei registri, dei descrittori e dello stack del nucleo subito dopo l'esecuzione della IRET;
- 5) Lo stato del processore dopo l'esecuzione della IRET

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
.....	2000		PC	C097
PC	A908	PC	B012	2001		PS	7E55
PS	2F55	PS	CC87	2002		SP	B800
SP	B892	SP	E9E0	2003		R1	1021
R1	0101	R1	A2DE	2004		R2	0956
R2	AFFF	R2	BABA	2005			

INDIRIZZO	3030
PAROLA DI STATO	1100
Vettore di interruzione	

Stack pointer del nucleo	2000
--------------------------	------

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

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

Stack del nucleo	
2000	
2001	C097
2002	7E55
2003	B800
2004	1021
2005	0956

Registri	
PC	3030
PS	1100
SP	2005
R1	0
R2	0

3)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
.....	2000		PC	3131
PC	C097	PC	invariato	2001	B012	PS	1100
PS	7E55	PS	invariato	2002	CC87	SP	2005
SP	B800	SP	invariato	2003	E9E0	R1	?
R1	1021	R1	invariato	2004	A2DE	R2	?
R2	0956	R2	invariato	2005	BABA		

4)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
.....	2000		PC	B012
PC	invariato	PC	invariato	2001		PS	CC87
PS	invariato	PS	invariato	2002		SP	E9E0
SP	invariato	SP	invariato	2003		R1	A2DE
R1	invariato	R1	invariato	2004		R2	BABA
R2	invariato	R2	invariato	2005			

5) Stato del processore: UTENTE

ESERCIZIO 2 (4 punti)

Si considerino due thread POSIX A e B che condividono un buffer a n posizioni (numerato da 0 a n-1) sul quale scambiano messaggi secondo il paradigma produttore/consumatore. Ogni messaggio occupa una posizione del buffer. A tal fine A e B condividono inoltre un semaforo di mutua esclusione *mutex* (inizializzato ad 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. Correggere il codice di A e B marcando con una X le righe da eliminare ed eventualmente inserendo comandi aggiuntivi nelle righe vuote.

SOLUZIONE

Processo A:

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

Processo B:

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

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

ESERCIZIO 3 (4 punti)

In un sistema operativo ad ambiente globale, i processi A_1, A_2, \dots, A_m (clienti) e il processo B (servente) comunicano in base al modello produttore/consumatore tramite un buffer condiviso realizzato come un array circolare ad N posizioni.

Per la gestione del buffer si utilizzano due variabili *testa* e *coda* che rappresentano rispettivamente il primo elemento libero nell'array circolare (usato per l'inserzione) e il primo elemento occupato nell'array circolare (usato per l'estrazione). Le variabili *testa* e *coda* sono entrambe inizializzate al valore 0.

Inoltre sono usati i semafori *libero* (inizializzato al valore N-1) e *pieno* (inizializzato al valore 0) e un semaforo di mutua esclusione *mutex* (inizializzato ad 1)

Completare il codice del generico processo cliente A_i e del processo servente B.

SOLUZIONE

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

Processo Ai

```
while (true) {  
    mess = produce_mess();  
    wait(libero);  
    wait(mutex);  
    arr[testa]=mess;  
    testa=(testa+1)%N;  
    signal(mutex);  
    signal(pieno);  
}
```

Processo B

```
while (true) {  
    wait(pieno);  
    mess=arr[coda];  
    coda=(coda+1)%N;  
    signal(libero);  
    consuma_messaggio(mess);  
}
```

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

ESERCIZIO 4 (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 e tutti i processi sono sospesi.

		MOLTEPLICITA' →								
			R1	R2	R3	R4	R1	R2	R3	R4
							4	4	5	5
ASSEGNAZIONE ATTUALE →	A	2	1	1	1					
	B	2	0	1	2					
	C	0	1	0	0					
	D	0	2	2	2					
		DISPONIBILTA' →								
			R1	R2	R3	R4				
			0	0	1	0				
		ESIGENZA ATTUALE								
			R1	R2	R3	R4				
			0	0	1	0				
							0	1	0	0
							0	0	0	1
							0	0	0	2
							0	0	3	0

Poichè il sistema è in stallo, si forza il processo D a prerilasciare 1 risorsa di tipo R4 (quindi l'esigenza di R4 da parte di D viene incrementata di 1). Questa soppressione elimina lo stallo?

SOLUZIONE

a) Stato raggiunto dopo il prerilascio forzato di 1 risorsa di tipo R4 da parte di D:

		MOLTEPLICITA' →				R1R2R3R4				
			R1	R2	R3	R4	4455			
ASSEGNAZIONE ATTUALE →	A	2	1	1	1					
	B	2	0	1	2					
	C	0	1	0	0					
	D	0	2	2	1					
		DISPONIBILTA' →				R1R2R3R4				
			R1	R2	R3	R4	0011			
		ESIGENZA ATTUALE								
			R1	R2	R3	R4	0100			
			R1	R2	R3	R4	0001			
			R1	R2	R3	R4	0002			
			R1	R2	R3	R4	0031			

b) Per la verifica delle eliminazione dello stallo:

b1) Il processo B può terminare

La disponibilità di {R1, R2, R3, R4} diviene { 2, 0, 2, 3 }

b2) Il processo C può terminare

La disponibilità di {R1, R2, R3, R4} diviene { 2, 1, 2, 3 }

b3) Il processo A può terminare

La disponibilità di {R1, R2, R3, R4} diviene { 4, 2, 3, 4 }

b4) Il processo D può terminare

La disponibilità di {R1, R2, R3, R4} diviene { 4, 4, 5, 5 }

Di conseguenza: stallo eliminato? SI

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

ESERCIZIO 5 (4 punti)

Un sistema che gestisce il processore combinando le politiche a priorità e RoundRobin con la tecnica delle code multiple (una coda FIFO per ogni valore di priorità; i processi pronti di uguale priorità sono inseriti in una stessa coda; il processore viene assegnato al processo che occupa la prima posizione nella coda non vuota di massima priorità; ai processi pronti di uguale priorità si applica la politica Round Robin).

Il quanto di tempo è di 5 msec.

La politica prevede il prerilascio, che avviene immediatamente dopo l'evento che lo provoca, senza attendere l'esaurimento del quanto di tempo corrente. Quando un processo va in esecuzione gli viene assegnato un intero quanto di tempo, indipendentemente dal tempo consumato nel precedente turno di esecuzione.

Al tempo T, nel sistema sono presenti i seguenti processi:

- Processo A, con priorità 2, che al tempo t è pronto;
- Processo B, con priorità 3, che al tempo t è in stato di attesa;
- Processo C, con priorità 2, che è in esecuzione dal tempo t;
- Processo D, con priorità 2, che al tempo t è pronto;
- Processo E, con priorità 1, che al tempo t è pronto.

La composizione delle tre code al tempo t è la seguente:

coda 1: E coda 2: D->A coda 3: Ø

Si chiede quale è il processo in esecuzione e la composizione delle 3 code al tempo T+15

se si verificano le seguenti sequenze di eventi (**da considerare in alternativa**):

1. Al tempo T+8 si sospende il processo in esecuzione, quindi al tempo T+12 si sospende il processo in esecuzione;
2. Al tempo T+9 viene riattivato il processo B, quindi al tempo T+14 si sospende il processo in esecuzione;
3. Al tempo T+3 si sospende il processo in esecuzione, al tempo T+5 si sospende il processo in esecuzione, al tempo T+8 si sospende il processo in esecuzione, quindi al tempo T+12 viene riattivato il processo A;
4. Al tempo T+4 viene riattivato il processo B, quindi al tempo T+10 si sospende il processo in esecuzione, quindi al tempo T+13 vengono riattivati tutti i processi sospesi (se esistono).

SOLUZIONE

1. Al tempo T+ 5: in esecuzione	D	Coda 1: E	Coda 2: A-> C	Coda 3: Ø
Al tempo T+ 8 :	in esecuzione A	Coda 1: E	Coda 2: C	Coda 3: Ø
Al tempo T+ 12: in esecuzione	C	Coda 1: E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 15: in esecuzione	C	Coda 1: E	Coda 2: Ø	Coda 3: Ø
2. Al tempo T+ 5: in esecuzione	D	Coda 1: E	Coda 2: A-> C	Coda 3: Ø
Al tempo T+ 9: in esecuzione	B	Coda 1: E	Coda 2: A-> C-> D	Coda 3: Ø
Al tempo T+ 14: in esecuzione	A	Coda 1: E	Coda 2: C-> D	Coda 3: Ø
Al tempo T+ 15: in esecuzione	A	Coda 1: E	Coda 2: C-> D	Coda 3: Ø
3. Al tempo T+ 3: in esecuzione	D	Coda 1: E	Coda 2: A	Coda 3: Ø
Al tempo T+ 5: in esecuzione	A	Coda 1: E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 8: in esecuzione	E	Coda 1: Ø	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 12: in esecuzione	A	Coda 1: E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 15: in esecuzione	A	Coda 1: E	Coda 2: Ø	Coda 3: Ø
4. Al tempo T+ 4: in esecuzione	B	Coda 1: E	Coda 2: D-> A-> C	Coda 3: Ø
Al tempo T+ 9: in esecuzione	B	Coda 1: E	Coda 2: D-> A-> C	Coda 3: Ø
Al tempo T+ 10: in esecuzione	D	Coda 1: E	Coda 2: A-> C	Coda 3: Ø
Al tempo T+ 13: in esecuzione	B	Coda 1: E	Coda 2: A-> C-> D	Coda 3: Ø
Al tempo T+ 15: in esecuzione	B	Coda 1: E	Coda 2: A-> C-> D	Coda 3: Ø

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

ESERCIZIO 6 (2 punti)

Si consideri un sistema nel quale sono definiti i semafori *sem1* e *sem2* e I processi P1, P2, P3, P4 e P5.

Al tempo *T* i semafori hanno la seguente configurazione:

Sem1: valore 0, coda P2 → P3

Sem2: valore 1, coda Ø

Allo stesso tempo, la CodaPronti ha la configurazione P4 → P5 e il processo P1 è in esecuzione.

Lo scheduler dei processi non prevede il prerilascio del processore.

Come si modificano i semafori e la CodaPronti e quale processo è in esecuzione se si verificano (in alternativa) le due seguenti sequenze di eventi:

- a) P1 esegue *wait (Sem1)* e successivamente il processo in esecuzione esegue *wait(Sem2)*;
- b) P1 esegue *signal (Sem1)* e successivamente il processo in esecuzione esegue *signal(Sem2)*;

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti	Sem1	Sem2
a-1	P1 esegue <i>wait (Sem1)</i>	P4	P5	0, P2->P3->P1	1, Ø
a-2	Il processo in esecuzione esegue <i>wait(Sem2)</i>	P4	P5	0, P2->P3->P1	0, Ø
b-1	P1 esegue <i>signal (Sem1)</i>	P1	P4->P5->P2	0, P3	1, Ø
b-2	Il processo in esecuzione esegue <i>signal(Sem2)</i>	P1	P4->P5->P2	0, P3	2, Ø

ESERCIZIO 7 (2 punti)

In un sistema con thread realizzati a livello utente, sono presenti i processi P1 con thread T11, T12, T13, il processo P2 con thread T21 e T22 e il processo P3 con il solo thread T31. Ogni processo gestisce i suoi thread con politica Round Robin. La politica di scheduling dei processi non prevede il prerilascio.

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-> T13 Processo P2: T21->T22 Processo P3: T31

Dire il contenuto della coda pronti nel nucleo e quale thread è in esecuzione dopo che si sono verificate (in alternativa) le due seguenti sequenze di eventi:

- a) Il thread in esecuzione esaurisce il quanto di tempo; quindi il thread in esecuzione esegue una *signal* sul semaforo *sem1*; quindi il thread in esecuzione esegue una *wait* sul semaforo *sem2* con valore 0;
- b) il thread in esecuzione esegue una *wait* sul semaforo *sem1*; il thread in esecuzione esegue una operazione di *thread_yield*; quindi il thread in esecuzione esegue una *signal* sul semaforo *sem1*.

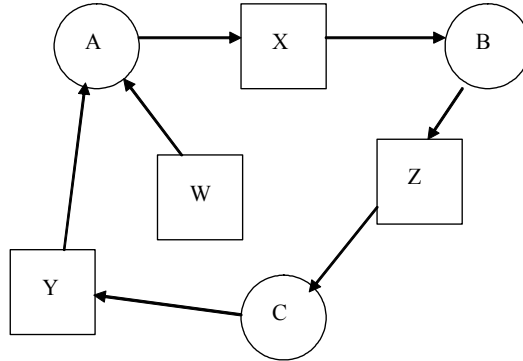
SOLUZIONE

	Sequenze di eventi	Thread in esecuzione dopo l'evento	Coda pronti
a-1	Il thread in esecuzione esaurisce il quanto di tempo	T12	P2
a-2	il thread in esecuzione esegue una <i>signal</i> sul semaforo <i>sem1</i>	T12	P2,P3
a-3	il thread in esecuzione esegue una <i>wait</i> sul semaforo <i>sem2</i> con valore 0	T21	P3
b-1	il thread in esecuzione esegue una <i>wait</i> sul semaforo <i>sem1</i>	T21	-
b-2	il thread in esecuzione esegue una operazione di <i>thread_yield</i>	T22	-
b-3	il thread in esecuzione esegue una <i>signal</i> sul semaforo <i>sem1</i>	T22	P3

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

ESERCIZIO 8 (2 punti)

Un sistema con 3 processi (A,B e C) e quattro risorse singole (W, X, Y e Z) ha raggiunto lo stato descritto dal grafo seguente:



Domande:

- a) Il sistema è in stallo?
- b) E' possibile che lo stato sia stato raggiunto con la seguente sequenza di assegnazioni e rilasci, a partire dallo stato iniziale in cui tutte le risorse sono disponibili?

SOLUZIONE

a) Il sistema è in stallo? SI perchè C'è un ciclo che coinvolge A,B e C

b) E' possibile che lo stato sia stato raggiunto con la seguente sequenza di assegnazioni e rilasci?

- | | |
|---|--|
| 1) A richiede Z Risultato: A ottiene Z | 6) A rilascia Z Risultato: C riattivato, ottiene Z |
| 2) C richiede Z Risultato: C in attesa di Z | 7) A richiede Y Risultato: A ottiene Y |
| 3) B richiede X Risultato: B ottieneX | 8) C richiede Y Risultato: C in attesa di Y |
| 4) B richiede Z Risultato: B in attesa di Z | 9) A richiede X Risultato: A in attesa di X |
| 5) A richiede W Risultato: A ottiene W | |

Possibile o impossibile? SI

Motivo se impossibile: -

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 2

ESERCIZIO 9 (2 punti)

Un processo che genera un figlio esegue il seguente frammento di codice.

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

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

SOLUZIONE

Si assume implicitamente che la fork abbia successo e che valgano tutte le altre condizioni (oltre a quella che il file /bin/pippo sia eseguibile) per il successo del comando `execl("/bin/pippo",NULL)`.

Con queste ipotesi il processo stampa "tre", oltre alle eventuali stampe del programma /bin/pippo

ESERCIZIO 10 (2 punti)

In seguito alla chiamata di sistema `exec`, eseguita dal processo figlio dopo essere stato generato con una `fork`, dire quali delle seguenti informazioni del processo figlio risiedono nella *process structure*, quali risiedono nella *user structure* e quali sono identiche a quelle del processo padre.

SOLUZIONE

INFORMAZIONE	Process structure	User Structure	Identica?
Stack			No
Codice			No
Heap			No
Informazioni sui segnali pendenti (signal bitmap)	Si		Si
Informazioni sulla gestione dei segnali (signal handler array)		Si	No
Handler dei segnali			No
Program counter (PC)		Si	No
Stack pointer (SP)		Si	No
Registri generali e PS		Si	No
PID del processo	Si		No
PID del padre del processo	Si		No