SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 1

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;
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	
.....	.....
PC	7E31
PS	16F2
SP	7873
R1	1234
R2	56CC

Descrittore di P2	
.....	.....
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

Stack del nucleo	
0FFF	
1000	
1001	
1002	
1003	
1004	

Registri	
PC	1880
PS	16F2
SP	2880
R1	4500
R2	CD31

INDIRIZZO	2000
PAROLA DI STATO	0045
Vettore di interruzione	

Stack pointer del nucleo	0FFF

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	
0FFF	
1000	1880
1001	16F2
1002	2880
1003	4500
1004	CD31

Registri	
PC	2000
PS	0045
SP	1004
R1	0
R2	0

3)

Descrittore di P1	
.....	.....
PC	1880
PS	16F2
SP	2880
R1	4500
R2	CD31

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

Stack del nucleo	
0FFF	
1000	A12C
1001	16F2
1002	A275
1003	25CC
1004	F012

Registri	
PC	??
PS	0045
SP	1004
R1	??
R2	??

4)

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	
0FFF	-
1000	-
1001	-
1002	-
1003	-
1004	-

Registri	
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

5) Stato del processore: UTENTE

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 1

ESERCIZIO 2 (4 punti)

Si considerino due thread POSIX A e B che condividono un buffer a n posizioni (numerate 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 *mux* (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.
Completare il codice di A e B inserendo i comandi **pthread_mutex_lock**(&mux) e **pthread_mutex_unlock**(&mux).

SOLUZIONE

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);

}

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 3 (4 punti)

In un sistema operativo ad ambiente locale, i processi A1, A2, ... ,Am (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 Ai e del processo servente B.

SOLUZIONE

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

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

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 1

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
4	4	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

ESIGENZA ATTUALE

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

DISPONIBILTA' ATTUALE

R1	R2	R3	R4
0	0	1	0

Poichè il sistema è in stallo, si sopprime il processo B. Questa soppressione elimina lo stallo?

SOLUZIONE

a) Stato raggiunto dopo l'eliminazione del processo B:

MOLTEPLICITA' →

R1	R2	R3	R4
4	4	5	5

ASSEGNAZIONE ATTUALE →

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

ESIGENZA ATTUALE

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

DISPONIBILTA' ATTUALE

R1	R2	R3	R4
2	0	2	2

b) Per la verifica delle eliminazione dello stallo:

b1) Il processo C può terminare

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

b2) Il processo A può terminare

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

b3) 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 1

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 al verificarsi dell'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 è in esecuzione dal tempo T;
- Processo B, con priorità 3, che al tempo T è in stato di attesa;
- Processo C, con priorità 2, che al tempo T è pronto;
- Processo D, con priorità 1, 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: D -> E coda 2: C 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, quindi al tempo T+5 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 C	Coda 1: D,E	Coda 2: A	Coda 3: Ø
Al tempo T+ 8 : in esecuzione A	Coda 1: D,E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 12: in esecuzione D	Coda 1: E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 15: in esecuzione D	Coda 1: E	Coda 2: Ø	Coda 3: Ø
2. Al tempo T+ 5: in esecuzione C	Coda 1: D,E	Coda 2: A	Coda 3: Ø
Al tempo T+ 9: in esecuzione B	Coda 1: D,E	Coda 2: A,C	Coda 3: Ø
Al tempo T+ 14: in esecuzione A	Coda 1: D,E	Coda 2: C	Coda 3: Ø
Al tempo T+ 15: in esecuzione A	Coda 1: D,E	Coda 2: C	Coda 3: Ø
3. Al tempo T+ 3: in esecuzione C	Coda 1: D,E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 5: in esecuzione D	Coda 1: E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 10: in esecuzione E	Coda 1: D	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 12: in esecuzione A	Coda 1: D,E	Coda 2: Ø	Coda 3: Ø
Al tempo T+ 15: in esecuzione A	Coda 1: D,E	Coda 2: Ø	Coda 3: Ø
4. Al tempo T+ 4: in esecuzione B	Coda 1: D,E	Coda 2: C,A	Coda 3: Ø
Al tempo T+ 9: in esecuzione B	Coda 1: D,E	Coda 2: A	Coda 3: Ø
Al tempo T+ 10: in esecuzione C	Coda 1: D,E	Coda 2: A	Coda 3: Ø
Al tempo T+ 13: in esecuzione B	Coda 1: D,E	Coda 2: A,C	Coda 3: Ø
Al tempo T+ 15: in esecuzione B	Coda 1: D,E	Coda 2: A,C	Coda 3: Ø

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 1

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 P4 → P5
Sem2: valore 2, coda Ø
Allo stesso tempo, la CodaPronti ha la configurazione P2 → P3 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 *wait (Sem1)*	P2	P3	0, P4→P5→P1	2, Ø
a-2	Il processo in esecuzione esegue *wait(Sem2)*	P2	P3	0, P4→P5→P1	1, Ø
b-1	P1 esegue *signal (Sem1)*	P1	P2→P3→P4	0, P5	2, Ø
b-2	Il processo in esecuzione esegue *signal(Sem2)*	P1	P2→P3→P4	0, P5	3, 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
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 *wait* sul semaforo *sem3* con valore 0; quindi il thread in esecuzione esegue una *signal* sul semaforo *sem2* con valore 2;
b) il thread in esecuzione esegue una *wait* sul semaforo *sem2* con valore 2; il thread in esecuzione esegue una operazione di *thread_yield*; quindi il thread in esecuzione esegue una *signal* sul semaforo *sem1*.

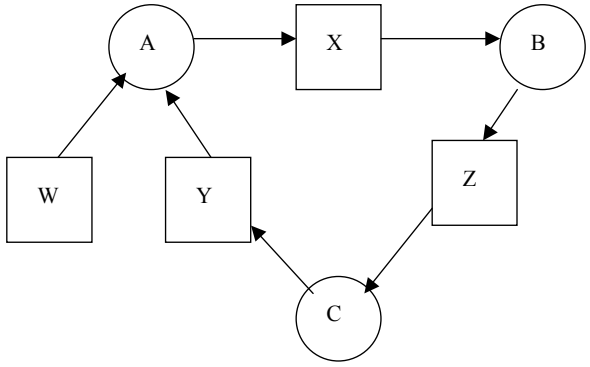
SOLUZIONE

	Sequenze di eventi	Thread in esecuzione dopo l'evento
a-1	Il thread in esecuzione esaurisce il quanto di tempo	T12
a-2	il thread in esecuzione esegue una *wait* sul semaforo *sem3* con valore 0	T21
a-3	il thread in esecuzione esegue una *signal* sul semaforo *sem2* con valore 2	T21
b-1	il thread in esecuzione esegue una *wait* sul semaforo *sem2* con valore 2	T11
b-2	il thread in esecuzione esegue una operazione di *thread_yield*	T12
b-3	il thread in esecuzione esegue una *signal* sul semaforo *sem1*	T12

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 1

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 i processi A,B e C sono in attesa circolare

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

1) A richiede Z Risultato: Z assegnato ad A
2) C richiede Z Risultato: C attende Z
3) A richiede W Risultato: W assegnato ad A
4) B richiede X Risultato: X assegnato a B
5) A rilascia Z Risultato: Z assegnato a C
6) C richiede Y Risultato: Y assegnato a C
7) A richiede Y Risultato: A attende Y
8) A richiede X Risultato: A sospeso non può richiedere

Possibile o impossibile? Impossibile

Motivo se impossibile: Y assegnato a C, non può essere assegnato ad A. A non può chiedere X perché già in attesa di Y. B non richiede Z.

SISTEMI OPERATIVI, CORSI A e B - PRIMA VERIFICA INTERMEDIA - 6/4 /2006
COMPITO 1

ESERCIZIO 9 (2 punti)

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

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

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

SOLUZIONE

Se la fork ha successo, il processo padre esegue la exec1, che fallisce, e stampa "uno" "due" "cinque"
Altrimenti non esegue la exec1 e stampa "uno" "quattro" "cinque"

ESERCIZIO 10 (2 punti)

In seguito alla chiamata di sistema 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			Si
Heap			Si
Informazioni sui segnali pendenti (signal bitmap)	Si		Si
Informazioni sulla gestione dei segnali (signal handler array)		Si	Si
Handler dei segnali			Si
Program counter (PC)		Si	Si
Stack pointer (SP)		Si	Si
Registri generali e PS		Si	Si
PID del processo	Si		No
PID del padre del processo	Si		No