SISTEMI OPERATIVI, CORSI A e B PRIMA VERIFICA INTERMEDIA - 3/4 /2007 COMPITO 1

COGNOME E NOME ..... MATRICOLA ..... FILA POSTO

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.

Il sistema operativo utilizza uno scheduling a priorità (vai in esecuzione il processi con valore maggiore di priorità) e con prerilascio.

Al tempo T i soli processi presenti nel sistema sono P1 che è in esecuzione e il processo P2 che è bloccato in attesa del completamento di un'operazione di input/output sul disco. In un istante successivo arriva un'interruzione dal disco che segnala il completamento dell'operazione di input/output e che attiva la routine di gestione dell'interruzione, la quale riattiva il processo P2. Completata la gestione dell'interruzione la routine invoca lo schedulatore che termina il suo compito con l'istruzione IRET.

Nell'istante in cui arriva l'interruzione dal dispositivo che ha completato l'operazione di input/output, 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.

Descrittore di P1	
Stato	In esec.
priorità	2
PC	ABAB
PS	6544
SP	F122
R1	C100
R2	0002

Descrittore di P2	
Stato	bloccato
priorità	3
PC	6543
PS	9FE3
SP	7652
R1	1E34
R2	EE00

Stack del nucleo	
0FFF	……
1000	
1001	
1002	
1003	
1004	

Registri	
PC	AF92
PS	7676
SP	F012
R1	4617
R2	FE41

SP nucleo	0FFF

Vettore di interruzione	
INDIRIZZO	1010
PAROLA DI STATO	86FA

Siano:

1) T+x l'istante in cui viene estratta la prima istruzione della routine di gestione dell'interruzione
2) T+y l'istante in cui viene estratta l'istruzione IRET
3) T+z l'istante in cui termina l'esecuzione dell'istruzione IRET

Si chiede:

1) Lo stato del processore nell'istante T+x
2) Il contenuto dei registri, dei descrittori e dello stack del nucleo nell'istante T+x;
3) Il contenuto dei registri, dei descrittori e dello stack del nucleo nell'istante T+y;
4) Il contenuto dei registri, dei descrittori e dello stack del nucleo nell'istante T+z;
5) Lo stato del processore nell'istante T+z

SOLUZIONE

1) Stato del processore: SUPERVISORE

Nelle tabelle seguenti: riportare solo i contenuti che cambiano

2)

Descrittore di P1	
Stato	In esec.
priorità	2
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Descrittore di P2	
stato	bloccato
priorità	3
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Stack del nucleo	
0FFF	……
1000	AF92
1001	7676
1002	F012
1003	4617
1004	FE41

Registri	
PC	1010
PS	86FA
SP	F012
R1	0
R2	0
SP nucleo	1004

3)

Descrittore di P1	
Stato	Pronto
priorità	2
PC	AF92
PS	7676
SP	F012
R1	4617
R2	FE41

Descrittore di P2	
stato	In esec.
priorità	3
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Stack del nucleo	
0FFF	……
1000	6543
1001	9FE3
1002	7652
1003	1E34
1004	EE00

Registri	
PC	????
PS	????
SP	F012
R1	????
R2	????
SP nucleo	1004

4)

Descrittore di P1	
Stato	Pronto
priorità	2
PC	AF92
PS	7676
SP	F012
R1	4617
R2	FE41

Descrittore di P2	
stato	In esec.
priorità	3
PC	invariato
PS	invariato
SP	invariato
R1	invariato
R2	invariato

Stack del nucleo	
0FFF	……
1000	
1001	
1002	
1003	
1004	

Registri	
PC	6543
PS	9FE3
SP	7652
R1	1E34
R2	EE00
SP nucleo	0FFF

5) Stato del processore: UTENTE

ESERCIZIO 2 (4 punti)

In un sistema operativo che realizza i threads a livello utente, i thread T_1, T_2 e T_3 del processo P competono in un gioco di carte. I tre threads avanzano senza sincronizzarsi ed eseguono tutti lo stesso codice:

```
1. While (true) {
2.     lock (&Chiave);
3.           MescolaCarte(&Mazzo);
4.           ThreadYield();
5.           EstraiCarta(&Carta);
6.     unlock (&Chiave);
7.     ThreadYield();
8.     UtilizzaCarta(Carta);
9.     lock (&Chiave);
10.          InserisciCarta(&Mazzo,Carta);
11.    unlock (&Chiave);
12.    ThreadYield();
       }
```

Lo scheduling dei thread avviene senza prerilascio.

Si fanno le seguenti ipotesi:

- `lock(&Chiave)` esegue `ThreadYield()` nel caso in cui il valore di `chiave` sia `0` (occupato), mentre `unlock(&Chiave)` non esegue `ThreadYield()` quando assegna il valore 1 alla `chiave`.
- le operazioni `MescolaCarte, EstraiCarta, UtilizzaCarta` e `InserisciCarta` richiedono 10 msec. Tutte le altre operazioni richiedono un tempo trascurabile.
- Al tempo t T_2 è in stato di pronto e deve eseguire l'istruzione 1.
- Al tempo t T_3 è in stato di pronto e deve eseguire l'istruzione 8.
- al tempo t , l'ordinamento dei thread in coda pronti è T_2 -> T_3 ;

Se il thread T_1 esegue l'istruzione 4 al tempo t , quanto tempo impiega T_3 per completare l'istruzione 10?

SOLUZIONE

(riempire una riga della tabella ad ogni nuova assegnazione del processore ai thread)

Tempo	Thread in esecuzione	Prossima istruzione T1	Prossima istruzione T2	Prossima istruzione T3	Coda pronti
t	T2	5	1	8	T3 -> T1
t	T3	5	2	8	T1 -> T2
t+10	T1	5	2	9	T2 -> T3
t+20	T2	8	2	9	T3 -> T1
t+30	T3	8	5	9	T1 ->T2
t+30	T1	8	5	9	T2 -> T3
t+40	T2	9	5	9	T3 -> T1
t+50	T3	9	8	9	T1 -> T2
t+60	T1	9	8	1	T2 -> T3

T3 completa l'istruzione 10 al tempo t+60.

ESERCIZIO 3 (4 punti)

Un parcheggio per auto della capacità di 10 posti è dotato di un unico ingresso I e un'unica uscita U. L'ingresso e l'uscita sono controllate da sbarre. Le auto sono thread di uno stesso processo, conformi allo standard POSIX, che si sincronizzano mediante il meccanismo *mutex* (sul quale sono definite le operazioni `pthread_mutex_lock` e `pthread_mutex_unlock`), e il meccanismo *condition* (sul quale sono definite le operazioni `pthread_cond_wait` e `pthread_cond_signal`).
Per l'utilizzo del parcheggio, i thread condividono la variabile *PostiDisponibili,* con valore iniziale 10, la variabile *MutexParcheggio* di tipo *mutex*, e la variabile *PostoLibero,* di tipo *condition.*
A meno delle necessarie operazioni di scincronizzazione, ogni thread che utilizza il parcheggio esegue la seguente sequenza di operazioni:

```
<l'auto arriva all'ingresso I>
if PostiDisponibili = 0
//l'auto attende la disponibilità di un posto//
<la sbarra si alza e l'auto entra nel parcheggio>
<l'auto sceglie un posto libero e lo occupa >
PostiDisponibili - -
<auto parcheggiata>
<l'auto libera il posto che aveva occupato e si dirige all'uscita U>
PostiDisponibili ++
<la sbarra si alza e l'auto esce dal parcheggio>
```

Si chiede di inserire le corrette operazioni di sincronizzazione con i meccanismi *mutex* e *condition*.

SOLUZIONE

```
<l'auto arriva all'ingresso I>
pthread_mutex_lock(&MutexParcheggio)
if PostiDisponibili = 0 pthread_cond_wait(&PostoLibero,&MutexParcheggio);
//l'auto attende la disponibilità di un posto//
<la sbarra si alza e l'auto entra nel parcheggio>
<l'auto sceglie un posto libero e lo occupa >
PostiDisponibili - -
pthread_mutex_unlock(&MutexParcheggio);
<auto parcheggiata>
pthread_mutex_lock(&MutexParcheggio);
<l'auto libera il posto che aveva occupato e si dirige all'uscita U>
PostiDisponibili ++
pthread_cond_signal(&PostoLibero);
pthread_mutex_unlock(&MutexParcheggio);
<la sbarra si alza e l'auto esce dal parcheggio>
```

ESERCIZIO 4 (4 punti)

Un sistema con processi A, B, C, D, E e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [4, 5, 4, 6], utilizza l'algoritmo del banchiere per evitare lo stallo. Il sistema ha raggiunto lo stato (sicuro) mostrato nelle tabelle seguenti.

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

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

Molteplicità			
R1	R2	R3	R4
4	5	4	6

Disponibilità			
1	1	0	1

Si considerino ora i seguenti casi (in alternativa):

a. il processo A richiede un'istanza della risorsa R2
b. Il processo B richiede un'istanza della risorsa R4

Dire in ognuno dei due casi se la richiesta è accettata dal sistema operativo.

SOLUZIONE

a) Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R2 al processo A:

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

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

Molteplicità			
R1	R2	R3	R4
4	5	4	6

Disponibilità			
1	0	0	1

Di conseguenza: risorsa assegnata? NO

b) Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R4 al processo B:

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

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

Molteplicità			
R1	R2	R3	R4
4	5	4	6

Disponibilità			
1	1	0	0

b1) Il processo C…… può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 1,2,0,1…… }
b2) Il processo D…… può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 1,4,2,1…… }
b3) Il processo A…… può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 1,4,3,2……}
b4) Il processo E…… può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 3,5,4,3…… }
b5) Il processo B…… può terminare
La disponibilità di {R1, R2, R3, R4} diviene { 4,5,4,6…… }
Di conseguenza: risorsa assegnata? SI ……

ESERCIZIO 5 (4 punti)

In un sistema che gestisce il processore con la politica RoundRobin e quanto di tempo di 10 msec. vengono generati 5 processi (A,B,C), con i tempi di arrivo e le durate (in millisecondi) sotto specificate:

Processo	Durata	Tempo di arrivo
A	15	0
B	25	7
C	15	12

Calcolare il tempo di permanenza nel sistema di ogni processo (definito come differenza tra il tempo di completamento dell'esecuzione e il tempo di arrivo nel sistema), supponendo che si verifichino (in sequenza) i seguenti eventi:

1. Al tempo T+17 si sospende il processo in esecuzione
2. Al tempo T+31 viene riattivato il processo sospeso

E supponendo che nessun altro processo si sospenda.

Mostrare come si evolve il sistema a ogni riassegnazione del processore e comunque al tempo T+31 (dopo la riattivazione del processo sospeso).

SOLUZIONE

Tempo T+	proc. in esecuzione	coda pronti	Processi sospesi
0	A (15)	-	-
7	A (8)	B (25)	-
10	B (25)	A(5)	-
12	B (23)	A(5) -> C (15)	-
17	A (5)	C (15)	B (18)
22	C(15) *	-	B (18)
31	C (6)	B (18)	-
32	B (18)	C (5)	-
42	C (5)	B (8)	-
47	B (8) **	-	-
55	***	-	-

*A Termina all'istante 22
** C termina all'istante 47
*** B termina all'istante 55

Tempi di permanenza nel sistema:
A : 22- 0= 22; B: 55- 7= 48; C: 47- 12= 35.

ESERCIZIO 6 (2 punti)
Si consideri un sistema nel quale è definito il semaforo *sem1* e i processi P1, P2 e P3.
Al tempo *t* il semaforo *sem1* ha la seguente configurazione:
Sem1: valore 0, coda P3
Allo stesso tempo, la CodaPronti contiene solo P2 e il processo P1 è in esecuzione.
Lo scheduler dei processi non prevede il prerilascio del processore.
Come si modificano il semaforo *sem1* 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 *signal(Sem1);*
b) P1 esegue *signal (Sem1)* e successivamente il processo in esecuzione esegue *signal(Sem1);*

SOLUZIONE

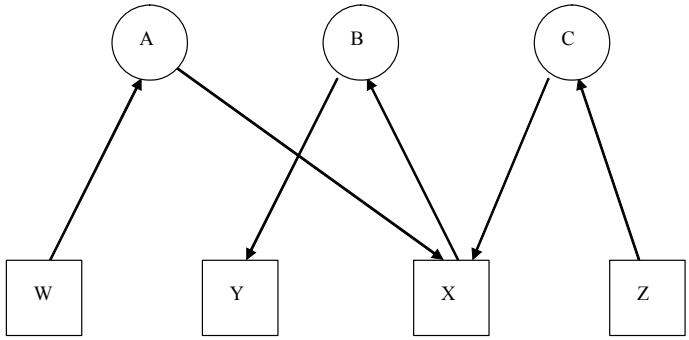
	Sequenze di eventi	In Esecuzione	Coda Pronti	Sem1
a-1	P1 esegue *wait (Sem1)*	P2	∅	0, P3→P1
a-2	Il processo in esecuzione esegue *signal(Sem1)*	P2	P3	0, P1
b-1	P1 esegue *signal (Sem1)*	P1	P2→P3	0, ∅
b-2	Il processo in esecuzione esegue *signal(Sem1)*	P1	P2→P3	1, ∅

ESERCIZIO 7 (2 punti)
Un sistema con 3 processi (A,B e C) e quattro risorse singole (W, X, Y e Z) esegue la seguente sequenza di assegnazioni e rilasci:

1) A richiede Y
2) C richiede Z
3) A richiede W
4) B richiede X
5) A rilascia Y
6) C richiede X
7) A richiede X
8) B richiede Y

Mostrare lo stato raggiunto dal sistema al termine della sequenza e dire se il sistema è in stallo.

SOLUZIONE



Il sistema è in stallo [si/no] ? NO

ESERCIZIO 8 (2 punti)
Quali delle seguenti operazioni possono essere eseguite solo in stato supervisore ?

	Solo Supervisore
Modificare il vettore delle interruzioni	SI
Abilitare le interruzioni	SI
Istruzione IRET	SI
Modificare il parametro *nice* di un processo in UNIX	
Eseguire la sostituzione di codice (EXEC) in Unix	

ESERCIZIO 9 (2 punti)
Nel sistema UNIX, si considerino i processi P, F1 e F2, dove F1 e F2 sono figli di P, che non ha altri figli.
In quale stato si trovano il processo P e il processo F1 al tempo 10 e al tempo 11 nelle seguenti ipotesi:

1. Il processo P esegue la chiamata di sistema `wait` unicamente al tempo 10 e i processi F1 ed F2 eseguono la chiamata di sistema `exit` rispettivamente al tempo 8 e al tempo 20;
2. Il processo P esegue la chiamata di sistema `wait` unicamente al tempo 10 e i processi F1 ed F2 eseguono la chiamata di sistema `exit` rispettivamente al tempo 8 e al tempo 5;

SOLUZIONE

1.	Tempo 10	Stato di P: ESECUZIONE	Stato di F1: ZOMBIE
	Tempo 11	Stato di P: ??	Stato di F1: TERMINATO
2.	Tempo 10	Stato di P: ESECUZIONE	Stato di F1: ZOMBIE
	Tempo 11	Stato di P: ??	Stato di F1: ZOMBIE

ESERCIZIO 10 (2 punti)
Si confrontino le primitive di sincronizzazione `wait` e `signal` con le omonime chiamate di sistema UNIX.
Si chiede se l'esecuzione di queste primitive o chiamate di sistema può produrre l'effetto di:

1) sospendere il processo in esecuzione
2) riattivare un processo
3) provocare la transizione di stato del processo in esecuzione
4) modificare la user structure del processo in esecuzione.

SOLUZIONE

	Può sospendere il processo in esecuzione?	Può riattivare un processo?	Può provocare commutazione di contesto?	Modifica in ogni caso la user structure?
Primitiva di sincronizzazione `wait`	SI	NO	SI	NO
Chiamata di sistema `wait di UNIX`	SI	NO	SI	NO
Primitiva di sincronizzazione `signal`	NO	SI	SI	NO
Chiamata di sistema `signal di UNIX`	NO	NO	NO	SI