SISTEMI OPERATIVI, CORSI A e B PRIMA VERIFICA INTERMEDIA - 2/4 /2008 COMPITO 1

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

ESERCIZIO 1 (4 punti)

Si consideri un processore che dispone dei seguenti registri:

- i registri speciali PC (program counter) e PS (program status). L'hardware del processore provvede al salvataggio di questi registri nello stack del nucleo quando riconosce un'interruzione e al loro ripristino dallo stack del nucleo quando esegue un'istruzione IRET.
- un banco di registri riservato allo stato utente, che comprende i registri generali R1, R2, R3, R4 e lo stack pointer SP,
- un ulteriore banco di registri riservato allo stato supervisore, che comprende i registri generali R'1, R'2, R'3, R'4 e lo stack pointer SP'.

Tutti i processi hanno uguale priorità e il sistema gestisce il processore con politica Round Robin. Al tempo t, quando sono presenti nel sistema (tra gli altri) il processo A, in stato di esecuzione, e il processo B che è l'unico processo bloccato in attesa sul semaforo *Sem*, il processo A esegue l'istruzione SVC (Supervisor Call) per invocare la chiamata di sistema *signal(Sem)*.

Immediatamente dopo il tempo *t*, quando l'interruzione generata dalla SVC viene riconosciuta, i registri del processore, i descrittori di A e B e lo stack del nucleo hanno i contenuti mostrati in figura.

L'interruzione determina l'intervento del nucleo, che esegue la primitiva *signal(Sem)*. Supponendo che il vettore di interruzione associato all'interruzione generata dalla SVC sia 0280, che la parola di stato del nucleo sia A709 e che il processo A non esaurisca il proprio quanto di tempo, si chiede:

a) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione eseguita subito dopo la SVC;
b) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione IRET con la quale termina la primitiva *signal(Sem)*;
c) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esec		Stato	Bloccato		……	……		SP	3F01
PC	23F8		PC	409A		1016	0011		R1	0707
PS	16F2		PS	16F2		1015			R2	0808
SP	3EFF		SP	5001		1014			R3	0909
R1	1234		R1	9876		1013			R4	4567
R2	2345		R2	8765		1012				
R3	3456		R3	0643		1011			REG. STATO SUPERV.	
R4	4567		R4	1010		1010			SP'	1016
									R'1	AAAA
									R'2	BBBB
PROCESSORE: Registri speciali									R'3	0011
PC	2401		PS	16F2					R'4	000F

SOLUZIONE

a) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione eseguita subito dopo la SVC;

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esec		Stato	Bloccato		……	……		SP	3F01
PC	23F8		PC	409A		1016	0011		R1	0707
PS	16F2		PS	16F2		1015	2401		R2	0808
SP	3EFF		SP	5001		1014	16F2		R3	0909
R1	1234		R1	9876		1013			R4	4567
R2	2345		R2	8765		1012				
R3	3456		R3	0643		1011			REG. STATO SUPERV.	
R4	4567		R4	1010		1010			SP'	1014
									R'1	AAAA
									R'2	BBBB
	PROCESSORE: Registri speciali								R'3	0011
PC	0280		PS	A709					R'4	000F

b) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione IRET con la quale termina la primitiva *signal(Sem)*;

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esec		Stato	Pronto		……	……		SP	3F01
PC	Invariato		PC	409A		1016	0011		R1	0707
PS	Invariato		PS	16F2		1015	2401		R2	0808
SP	3F01		SP	5001		1014	16F2		R3	0909
R1	0707		R1	9876		1013			R4	4567
R2	0808		R2	8765		1012				
R3	0909		R3	0643		1011			REG. STATO SUPERV.	
R4	4567		R4	1010		1010			SP'	1014
									R'1	?
									R'2	?
	PROCESSORE: Registri speciali								R'3	?
PC	?		PS	A709					R'4	?

c) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET

DESCRITTORE DI A			DESCRITTORE DI B			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esec		Stato	Pronto		……	……		SP	3F01
PC	Invariato		PC	409A		1016	0011		R1	0707
SP	Invariato		PS	16F2		1015			R2	0808
PS	3F01		SP	5001		1014			R3	0909
R1	0707		R1	9876		1013			R4	4567
R2	0808		R2	8765		1012				
R3	0909		R3	0643		1011			REG. STATO SUPERV.	
R4	4567		R4	1010		1010			SP'	1016
									R'1	?
									R'2	?
	PROCESSORE: Registri speciali								R'3	?
PC	2401		PS	16F2					R'4	?

ESERCIZIO 2 (4 punti)

SISTEMI OPERATIVI, CORSI A e B PRIMA VERIFICA INTERMEDIA - 2/4 /2008 COMPITO 1

In un sistema che implementa i thread a livello del nucleo, un processo comprende due gruppi di thread cooperanti: S1,..,Sn e T1,…,Tm. I thread cooperano tramite due strutture dati condivise (`A` e `B`) e tramite due buffer (`buffer1` e `buffer2`) entrambi a 10 posizioni ed utilizzati secondo il paradigma produttore consumatore.
In particolare:

- i thread S1,…,Sn usano in sola lettura la struttura `A` (a questo scopo invocano la funzione `fun1(A)`), agiscono da consumatori nei confronti di `buffer1`, e agiscono da produttori nei confronti di `buffer2`.
- i thread T1,…,Tn usano in scrittura `B` (tramite la funzione `fun3(B,int)`) , agiscono da produttori nei confronti di `buffer1`, e agiscono da consumatori nei confronti di `buffer2`.

Per la sincronizzazione sui due buffer condivisi si usano i seguenti semafori:

- `Buffer1Pieno`, inizializzato al valore 10, *(il valore esprime il numero di elementi disponibili nel buffer)*
- `Buffer1Vuoto`, inizializzato al valore 0, *(il valore esprime il numero di elementi occupati nel buffer)*
- `Buffer2Pieno`, inizializzato al valore 10, *(il valore esprime il numero di elementi disponibili nel buffer)*
- `Buffer2Vuoto`, inizializzato al valore 0, *(il valore esprime il numero di elementi occupati nel buffer)*

Inoltre i thread utilizzano i semafori di mutua esclusione necessari per interagire correttamente.
Il codice eseguito dai thread S1,..,Sn e dai thread T1,…,Tm è il seguente:

Thread S1,…,Sn:
```
int v,d;
while(True) {
      ……
      Legge v da buffer1
      ……
      d=fun1(A); //legge da A
      ……
      d=fun2(d,v);
      ……
      Scrive d su buffer2
      ……
}
```

Thread T1,…,Tn:
```
int v,d;
d=…;
while(True) {
      ……
      Scrive d su buffer1
      ……
      Legge v da buffer2
      ……
      d=fun3(B,v); //scrive su B
      ……
      d=fun4(d,v);
      ……
}
```

Si chiede di definire i necessari semafori di mutua esclusione e di completare lo pseudo-codice dei thread aggiungendo le chiamate alle primitive wait e signal necessarie per garantire un uso corretto delle strutture dati e una corretta sincronizzazione tra i thread.

SOLUZIONE
Si utilizzano i seguenti semafori di mutua esclusione:

- `MutexBuf1` per `buffer1`
- `MutexBuf2` per `buffer2`
- `MutexB` per la struttura `B`

Thread S1,…,Sn:
```
int v,d;
while(True) {
      wait(Buffer1Vuoto);
      wait(MutexBuf1);
      Legge v da buffer1
      signal(MutexBuf1);
      signal(Buffer1Pieno);
      d=fun1(A); //legge da A
      d=fun2(d,v);
      wait(Buffer2Pieno);
      wait(MutexBuf2);
      Scrive d su buffer2
```

Thread T1,…,Tn:
```
int v,d;
d=…;
while(True) {
      wait(Buffer1Pieno);
      wait(MutexBuf1);
      Scrive d su buffer1
      signal(MutexBuf1);
      signal(Buffer1Vuoto);
      wait(Buffer2Vuoto);
      wait(MutexBuf2);
      Legge v da buffer2
      signal(MutexBuf2);
```

```
        signal(MutexBuf2);
        signal(Buffer2Vuoto);

}
```

```
        signal(Buffer2Pieno);
        wait(MutexB);
        d=fun3(B,v); //scrive su B
        signal(MutexB);
        d=fun4(d,v);
}
```

ESERCIZIO 3 (4 punti)

In un sistema sono presenti i processi P1, P2 e P3 che cooperano tramite cinque sezioni critiche per l'accesso a risorse condivise.

L'accesso alle sezioni critiche avviene tramite cinque semafori di mutua esclusione, che sono rispettivamente A, B, C, D ed E.

Nel corso della propria esistenza, ciascun processo esegue una propria sequenza di sezioni critiche, intercalate con codice non critico. I processi avanzano alternandosi nello stato di esecuzioni con sequenza e velocità di avanzamento arbitrarie.

Ogni processo accede a una sezione critica quando supera la *wait* sul corrispondente semaforo e la rilascia quando esegue la *signal* sul medesimo semaforo. Tutte le sezioni critiche sono inizialmente disponibili.

Si considerino, in alternativa, le due sequenze di avanzamento dei processi sotto riportate (sono evidenziate solo le operazioni sui semafori che controllano le sezioni critiche) :

Sequenza 1)

	1	2	3	4	5	6	7	8
P1	Wait(D)	Signal(D)	Wait(C)	Wait(B)	Wait(A)	Signal(C)	Signal(B)	Signal(A)
P2	Wait(B)	Wait(A)	Wait(D)	Signal(D)	Signal(A)	Signal(B)		
P3	Wait(C)	Wait(A)	Wait(D)	Signal(A)	Signal(C)	Signal(D)		

Sequenza 2)

	1	2	3	4	5	6	7	8
P1	Wait(E)	Wait(A)	Wait(C)	Wait(D)	Signal(D)	Signal(C)	Signal(A)	Signal(E)
P2	Wait(B)	Wait(C)	Wait(D)	Wait(E)	Signal(E)	Signal(D)	Signal(C)	Signal(B)
P3	Wait(A)	Wait(B)	Wait(C)	Wait(D)	Signal(D)	Signal(C)	Signal(B)	Signal(A)

Per ogni sequenza, si chiede se i processi evitano la possibilità di stallo. Si chiede anche di motivare la risposta e, se questa è negativa (i processi non evitano la possibilità di stallo), di individuare una sequenza di esecuzione che provoca lo stallo.

SOLUZIONE

Sequenza 1) La sequenza evita la possibilità di stallo? SI

Motivazione della risposta: Nel richiedere l'accesso alle sezioni critiche, tutti i processi rispettano l'ordimanento C, B, A , D. La iniziale richiesta della sezione critica D da parte del processo P1, subito seguita dal rilascio, non è rilevante per chè in questo intervallo il processo non osserva il paradigma di "possesso e attesa"

Sequenza 2) La sequenza evita la possibilità di stallo? NO

Motivazione della risposta:

Nelle proprie richieste di accesso alle sezioni critiche i processi non rispettano un ordinamento comune.. Una sequenza che provoca lo stallo è: P1-1, P3-1, P2-1, P2-2, P3-2, P2-2, P2-3, P2-4

ESERCIZIO 4 (4 punti)

Si consideri la seguente variante della politica SJF (Shortest Job First) per la gestione del processore:

- Alla generazione di ogni processo è nota la sua *durata,* definita come il tempo di utilizzo del processore necessario per la terminazione del processo;
- per ogni processo è noto ad ogni tempo *t* il *tempo residuo di esecuzione,* definito come la differenza tra la *durata* e il tempo di utilizzo del processore fino al tempo *t.* Il *tempo residuo di esecuzione* del processo in esecuzione è noto allo scheduler ed è aggiornato con continuità dal sistema operativo.
- lo scheduler garantisce che il processo in esecuzione sia sempre *il processo attivo* (cioè presente nel sistema e non sospeso) con il valore minimo di *tempo residuo di esecuzione.* La politica prevede il prerilascio. A parità di tempo residuo di esecuzione, prevale il processo che era già in esecuzione.
- Un processo termina quando si azzera il suo tempo residuo di esecuzione.

In un sistema che adotta questa politica vengono generati in sequenza i processi A, B, C, D, E con i tempi di arrivo e le durate specificate in tabella:

PROCESSO	TEMPO DI ARRIVO	DURATA
A	0	10
B	3	8
C	5	4
D	7	3
E	18	5

E' inoltre presente il semaforo *sem,* con valore iniziale 0 e coda inizialmente vuota, sul quale vengono eseguite le seguenti operazioni:

- al tempo *t=6* il processo in esecuzione esegue *wait(sem)*;
- al tempo *t=12* il processo in esecuzione esegue *signal(sem)*;
- al tempo *t=16* il processo in esecuzione esegue *wait(sem)*;
- al tempo *t=25* il processo in esecuzione esegue *signal(sem)*.

Per ogni evento significativo (generazione o terminazione di un processo; esecuzione di *wait(sem)* o *signal(sem)*) si chiede di specificare, utilizzando una riga dello schema di soluzione, il nome del processo in esecuzione, il contenuto della Coda Pronti, il valore di *sem* e il contenuto della sua coda. Si suggerisce di associare sempre ad ogni processo il suo tempo residuo di esecuzione.

SOLUZIONE

TEMPO	EVENTO	PROC. IN ESEC.	CODA PRONTI	SEM.VALORE	SEM.CODA
0	Arriva A(10)	A(10)	Ø	0	∅
3	Arriva B(8)	A(7)	B(8)	0	∅
5	Arriva C(4)	C(4)	A(5), B(8)	0	∅
6	C(3) esegue wait(sem)	A(5)	B(8)	0	C(3)
7	Arriva D(3)	D(3)	A(4), B(8)	0	C(3)
10	Termina D	A(4)	B(8)	0	C(3)
12	A(2) esegue signal(sem)	A(2)	C(3), B(8)	0	∅
14	Termina A	C(3)	B(8)	0	∅
16	C(1) esegue wait(sem)	B(8)	Ø	0	C(1)
18	Arriva E(5)	E(5)	B(6)	0	C(1)
23	Termina E	B(6)	∅	0	C(1)
25	B(4) esegue signal(sem)	C(1)	B(4)	0	∅
26	Termina C	B(4)	∅	0	∅
30	Termina B	∅	∅	0	∅

ESERCIZIO 5 (4 punti)

In un sistema UNIX, vengono eseguite in sequenza le seguenti operazioni:

a) il processo P esegue con sucesso una chiamata *pipe,* che restituisce i descrttori *fd[0]* e *fd[1];*
b) il processo P esegue con successo una *fork,* generando il processo P1 che immediatamente esegue con successo una *exec;*
c) il processo P esegue con successo una *fork,* generando il processo P2 che immediatamente esegue con successo una *exec;*
d) il processo P1 scrive nel pipe la sequenza di caratteri *'abra'* con l'operazione *write(fd[1],'abra', 4)*
e) Il processo P1 esegue con successo una *fork,* generando il processo P3 che immediatamente esegue con successo una *exec;*
f) il processo P3 scrive nel pipe la sequenza di caratteri *'cad'* con l'operazione *write(fd[1],'cad', 3)*
g) il processo P chiude l'estremo di scrittura del pipe con l'operazione *close(fd[1])*
h) il processo P2 scrive nel pipe la sequenza di caratteri *'abra'* con l'operazione *write(fd[1],'abra', 4)*
i) il processo P1 chiude l'estremo di lettura del pipe con l'operazione *close(fd[0])*
j) il processo P3 legge 11 caratteri dal pipe con l'operazione *read(fd[0], &dest, 11)*

Qual è il valore della variabile *dest* del processo P3 dopo l'operazione *j)*?

Soluzione

Dopo l'operazione j), il contenuto di *dest* è *'abracadabra.'*

ESERCIZIO 6 (2 punti)

Si consideri un sistema nel quale sono definiti il semaforo *sem* e i processi P1 (con priorità 3), P2 (con priorità 1) e P3 (con priorità 2). Lo scheduling avviene con una politica a priorità, che prevede il prerilascio e assegna il processore al processo pronto di priorità più elevata (a pari priorità applica la politica RoundRobin). La politica applicata al semaforo è la FIFO.
Al tempo *t* il processo P2 è in esecuzione e la coda pronti è vuota. Inoltre il semaforo *sem* ha valore 0 e la coda associata al semaforo contiene i processi P3->P1 (il processo P3 è in cima alla coda). Dopo il tempo *t* si verifica la seguente sequenza di eventi:

a) P2 esegue *signal(sem)*
b) il processo in esecuzione esegue *signal(sem);*
c) scade il quanto di tempo;
d) il processo in esecuzione esegue *wait(sem);*

Si chiede di specificare quale processo è in esecuzione dopo ogni evento e inoltre come si modificano il valore e la coda del semaforo *sem* e la *CodaPronti*.

SOLUZIONE

Sequenza di eventi	In Esecuzione	Coda Pronti	Valore di sem	Coda di sem
1) P2 esegue *signal(sem)*	P3	P2	0	P1
2) il processo in esecuzione esegue *signal(sem)*	P1	P3->P2	0	-
3) scade il quanto di tempo	P1	P3->P2	0	-
4) Il processo in esecuzione esegue *wait(sem)*	P3	P2	0	P1

ESERCIZIO 7 (2 punti)

In un sistema dove i processi operano in ambiente locale e dispongono di una libreria per la realizzazione dei thread a livello utente, sono presenti il processo P1 con priorità 2, il processo P2 con priorità 1 e il processo P3 con priorità 1. Lo scheduling dei processi avviene con una politica a priorità (va in esecuzione il processo pronto con il massimo valore di priorità, e a parità di priorità i processi si applica la politica FIFO) e prevede il prerilascio.

All'interno di ogni processo, i thread sono schedulati senza prerilascio e si alternano nello stato di esecuzione e di pronto. Quando un thread passa dallo stato di esecuzione a quello di pronto viene inserito nell'ultima posizione della coda dei thread pronti del processo.

Al tempo *T* è in esecuzione il processo P2, il processo P1 si è bloccato sulla primitiva *receive(P3, &mess)* e il processo P3 è pronto. Inoltre i processi contengono i seguenti thread:

Processo P1: thread T11 in stato di esecuzione; thread T12 in stato di pronto.

Processo P2 : thread T21 in stato di esecuzione; thread T22 e T23 in stato di pronto con il seguente ordinamento della coda: <primo> → T22 → T23.

Processo P3 : thread T31 in stato esecuzione; thread T32, T33 in stato di pronto con il seguente ordinamento della coda: <primo> → T32 → T33.

Si chiede qual è il thread in esecuzione dopo ciascuno degli eventi della sequenza riportata in tabella.

SOLUZIONE

	Sequenza di eventi	Thread in esecuzione dopo l'evento
a)	il thread in esecuzione esegue la primitiva (non bloccante) *send(&messaggio, P3)*	T21
b)	Scade il quanto di tempo	T31
c)	Il thread in esecuzione esegue l'operazione *thread_yield*	T32
d)	il thread in esecuzione esegue la primitiva (non bloccante) *send(&messaggio, P3)*	T11

ESERCIZIO 8 (2 punti)

Dati i processi A,B,C che osservano il paradigma di "possesso e attesa" e le risorse singole P, Q, R utilizzabili in mutua esclusione e senza possibilità di prerilascio e inizialmente disponibili, supponiamo le risorse siano assegnate ai processi richiedenti alla sola condizione di essere disponibili.
Si consideri la seguente sequenza di richieste e rilasci:

1) A richiede P; 2) B richiede Q; 3) A richiede Q; 4) B richiede R; 5) C richiede R;	6) B rilascia Q; 7) B richiede P; 8) A rilascia P; 9) A richiede R

Mostrare come si evolve il sistema utilizzando la tabella riportata nello schema di soluzione, e verificare se si raggiunge uno stallo.

SOLUZIONE

Evoluzione del sistema:

	Risorse disponibili prima dell' operazione	Dopo l'operazione					
		Processo A		Processo B		Processo C	
		Stato	Risorse Assegnate	Stato	Risorse Assegnate	Stato	Risorse Assegnate
1)	P, Q, R	Attivo	P	Attivo	Ø	Attivo	Ø
2)	Q, R	Attivo	P	Attivo	Q	Attivo	Ø
3)	R	Attende Q	P	Attivo	Q	Attivo	Ø
4)	R	Attende Q	P	Attivo	Q, R	Attivo	Ø
5)	Ø	Attende Q	P	Attivo	Q, R	Attende R	Ø
6)	Ø	Attivo	P, Q	Attivo	R	Attende R	Ø
7	Ø	Attivo	P, Q	Attende P	R	Attende R	Ø
8	Ø	Attivo	Q	Attivo	R, P	Attende R	Ø
9	Ø	Attende R	Q, R	Attivo	R, P	Attende R	Ø

Domande:

- Si raggiunge uno stallo? NO
- Perché? I processi A e C sono sospesi, ma non si ha attesa circolare: il processo B e la sua terminazione consentirà la riattivazione dei processi in attesa.

ESERCIZIO 9 (2 punti)

Si consideri un sistema che gestisce il processore con politica Round Robin, con quanto di tempo di *5 msec.* Lo scheduler è attivato per riassegnare il processore dai seguenti eventi (che possono anche verificarsi contemporaneamente):

- il processo in esecuzione esaurisce il quanto di tempo;
- il processo in esecuzione si sospende;
- il processo in esecuzione termina.

Nel sistema sono presenti i seguenti processi, che terminano dopo aver utilizzato il processore per la *durata* specificata in tabella:

Proc	Tempo di arrivo	Durata (msec)	Comportamento
A	0	30	Si sospende dopo aver utilizzato il processore per 20 msec; viene riattivato dopo 2 msec
B	1	25	Avanza fino alla terminazione senza sospendersi
C	2	40	Avanza fino alla terminazione senza sospendersi
D	3	25	Si sospende dopo aver utilizzato il processore per 10 msec; viene riattivato dopo 4 msec
E	4	35	Avanza fino alla terminazione senza sospendersi

Si chiede il tempo di terminazione del processo B.

SOLUZIONE

Il processo B termina dopo 5+ 5 + 4* (20 +5) = 110 msec.

Spiegazione:

Il processo B ottiene per la prima volta il processore dopo che lo ha ottenuto (per 5 msec) il processo A, che lo precede nella Coda Pronti: pertanto termina il primo turno dopo 5 +5 msec.

Per terminare, il processo deve ottenere altri 4 turni di esecuzione e ogni volta deve attendere i turni di esecuzione dei processi che sono presenti in Coda Pronti all'istante il cui B rilascia il processore. Il numero di questi processi è sempre uguale a 4, perché i processi A e D che si sospendono (rispettivamente per 2 e 4 msec) sono reinseriti in Coda Pronti prima che il processo in esecuzione abbia esaurito il suo quanto di tempo e quindi senza alterare l'ordinamento dei processi in coda. Quindi ognuno dei successivi 4 turni di esecuzione del processo B termina dopo 4*5+5 msec.

ESERCIZIO 10 (2 punti)

Dire che cosa viene stampato dal processo che esegue la *fork* all'interno del seguente frammento di codice:

```
…
printf("aaaa");
n = fork();
printf("bbbb");
if (n=0) {
        execl("prova",NULL);
        printf("cccc");
        }
else    {
        if (n>0) printf("dddd");
        else printf("eeee")
        }
…
```

Si consideri sia il caso in cui la *fork* ha successo, sia quello in cui la *fork* fallisce.

SOLUZIONE

- Se la fork ha successo, il processo che la esegue (processo padre) stampa in sequenza: "aaaa"; "bbbb", "dddd";
- Se la fork fallisce, il processo che la esegue (processo padre) stampa in sequenza: "aaaa"; "bbbb", "eeee".