Sistemi Operativi, corsi A e B - Prima prova di verifica intermedia - 8/4/2009

CORSO |A| |B| , NOME:_____MATRICOLA:_____

ESERCIZIO 1 (4 punti)

Si consideri un processore che dispone dei seguenti registri:
- i registri speciali PC (program counter) e PS (program status)
- un banco di registri riservato allo stato utente, che comprende i registri generali R1, R2, R3 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 e lo stack pointer SP'.

Il sistema gestisce il processore con politica Round Robin. Al tempo t sono presenti nel sistema (tra gli altri) il processo Pi, in stato di esecuzione, e il processo Pj che è bloccato sul semaforo di mutua esclusione *mux*. Al tempo t il processo Pi invoca la chiamata di sistema *signal(mux)*. L'invocazione della chiamata di sistema viene effettuata tramite l'istruzione SVC che provoca un'interruzione. Immediatamente prima che l'interruzione venga riconosciuta, i registri del processore, i descrittori di Pi e Pj e gli stack di Pi, di Pj e del nucleo hanno i contenuti mostrati in tabella.

L'interruzione determina l'intervento del nucleo, che esegue la funzione di servizio dell'interruzione corrispondente alla chiamata di sistema. Supponendo che il vettore di interruzione associato all'interruzione generata da SVC sia 0425 e che la parola di stato del nucleo sia 275E, si chiede:

a) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio;
b) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;
c) il contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione eseguita subito dopo la IRET.

DESCRITTORE DI P_I			DESCRITTORE DI P_J			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esecuzione		Stato	Bloccato		……	……		SP	2997
PC	2E31		PC	A12C		1016	23BB		R1	2649
PS	26F2		PS	A6F2		1015			R2	22CE
SP	2873		SP	A275		1014			R3	2410
R1	2234		R1	A5CC		1013				
R2	26CC		R2	A000		1012			REG. STATO SUPERV.	
R3	2000		R3	A056		1011			SP'	1016
									R'1	0012
PROCESSORE: Registri speciali e stato									R'2	AACC
PC	2F00		PS	16F2		stato	Utente		R'3	2345

Soluzione

a) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio:

DESCRITTORE DI P_I			DESCRITTORE DI P_J			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esecuzione		Stato	Bloccato		……	……		SP	2997
PC	2E31		PC	A12C		1016	23BB		R1	2649
PS	26F2		PS	A6F2		1015	2F00		R2	22CE
SP	2873		SP	A275		1014	16F2		R3	2410
R1	2234		R1	A5CC		1013				
R2	26CC		R2	A000		1012			REG. STATO SUPERV.	
R3	2000		R3	A056		1011			SP'	1014
									R'1	0012
PROCESSORE: Registri speciali e stato									R'2	AACC
PC	0425		PS	275E		stato	Supervisore		R'3	2345

b) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;

DESCRITTORE DI P_I			DESCRITTORE DI P_J			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esecuzione		Stato	Pronto		……	……		SP	2997
PC	2E31		PC	A12C		1016	23BB		R1	2649
PS	26F2		PS	A6F2		1015	2F00		R2	22CE
SP	2873		SP	A275		1014	16F2		R3	2410
R1	2234		R1	A5CC		1013				
R2	26CC		R2	A000		1012			REG. STATO SUPERV.	
R3	2000		R3	A056		1011			SP'	1014

										R'1	?
PROCESSORE: Registri speciali e stato										R'2	?
PC	0425 + ??		PS	275E		stato	Supervisore			R'3	?

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

DESCRITTORE DI P_I			DESCRITTORE DI P_J			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato	Esecuzione		Stato	Pronto		……	……		SP	2997
PC	2E31		PC	A12C		1016	23BB		R1	2649
PS	26F2		PS	A6F2		1015			R2	22CE
SP	2873		SP	A275		1014			R3	2410
R1	2234		R1	A5CC		1013				
R2	26CC		R2	A000		1012			REG. STATO SUPERV.	
R3	2000		R3	A056		1011			SP'	1016
									R'1	?
PROCESSORE: Registri speciali e stato									R'2	?
PC	2F00		PS	16F2		stato	Utente		R'3	?

Soluzione

a) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione della prima istruzione della funzione di servizio:

DESCRITTORE DI P_I			DESCRITTORE DI P_J			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato			Stato			……	……		SP	
PC			PC			1016			R1	
PS			PS			1015			R2	
SP			SP			1014			R3	
R1			R1			1013				
R2			R2			1012			REG. STATO SUPERV.	
R3			R3			1011			SP'	
									R'1	
PROCESSORE: Registri speciali e stato									R'2	
PC			PS			stato			R'3	

b) contenuto dei descrittori, dei registri generali e speciali, dello stack del nucleo e lo stato del processore durante la fase di estrazione dell'istruzione IRET con la quale termina la chiamata di sistema;

DESCRITTORE DI P_I			DESCRITTORE DI P_J			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato			Stato			……	……		SP	
PC			PC			1016			R1	
PS			PS			1015			R2	
SP			SP			1014			R3	
R1			R1			1013				
R2			R2			1012			REG. STATO SUPERV.	
R3			R3			1011			SP'	
									R'1	
PROCESSORE: Registri speciali e stato									R'2	
PC			PS			stato			R'3	

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

DESCRITTORE DI P_I			DESCRITTORE DI P_J			STACK DEL NUCLEO			REG. STATO UTENTE	
Stato			Stato			……	……		SP	
PC			PC			1016			R1	
PS			PS			1015			R2	
SP			SP			1014			R3	
R1			R1			1013				
R2			R2			1012			REG. STATO SUPERV.	
R3			R3			1011			SP'	
									R'1	

Sistemi Operativi, corsi A e B - Prima prova di verifica intermedia - 8/4/2009

PROCESSORE: Registri speciali e stato									R'2	
PC			PS			stato			R'3	

ESERCIZIO 2 (4 punti)

Un word processor gestisce il documento di lavoro tramite un'unica struttura dati condivisa da 4 threads T1, T2, T3, T4. Il thread T1 gestisce l'input dei dati da tastiera, T2 gestisce la formattazione del documento, T3 visualizza il documento sul monitor, e T4 effettua periodicamente il salvataggio automatico del documento. Per le loro caratteristiche i thread T1 e T2 modificano il documento (e quindi la struttura dati condivisa), mentre i thread T3 e T4 accedono al documento in sola lettura. Per questo motivo T1 e T2 devono accedere alla struttura dati in mutua esclusione (tra loro stessi e nei confronti dei thread T3 e T4), mentre T3 e T4 possono accedere alla struttura dati concorrentemente ma in mutua esclusione rispetto a T1 e T2.

Per l'accesso alla struttura dati si adotta la seguente politica:

- Inizialmente accede il primo thread che ne fa richiesta;
- Quando il documento è in uso da T3 o da T4 o da entrambi, se i thread T1 e T2 decidono di accedere alla struttura dati vengono messi in attesa. I thread in attesa vengono riattivati solo quando sia T3 sia T4 hanno completato l'accesso alla struttura dati. Se T1 e T2 sono entrambi in attesa allora viene riattivato uno dei due, con politica FIFO. Se né T1 né T2 sono in attesa allora la struttura dati ritorna disponibile a tutti i thread e si torna alla situazione iniziale.
- Quando la struttura dati condivisa è utilizzata da T1, tutti gli altri thread che richiedono l'accesso vengono messi in attesa. Quando T1 termina il proprio accesso, verifica prima se T2 è in attesa e in caso affermativo lo riattiva. Altrimenti, se sono in attesa, vengono riattivati T3 e/o T4. Se nessun thread è in attesa allora la struttura dati ritorna disponibile a tutti e si torna alla situazione iniziale.
- Analogamente, quando la struttura dati condivisa è utilizzata da T2, tutti gli altri thread che richiedono l'accesso vengono messi in attesa. Quando T2 termina il proprio accesso, verifica prima se T1 è in attesa e in caso affermativo lo riattiva. Altrimenti, se sono in attesa, vengono riattivati T3 e/o T4. Se nessun thread è in attesa allora la struttura dati ritorna disponibile a tutti e si torna alla situazione iniziale.

Per la soluzione del problema, si utilizzano i seguenti dati condivisi da tutti i thread:

- *T1T2Ammesso*: intero non negativo (esprime quanti tra i thread T1 e T2 sono ammessi) ; valore iniziale 0
- *T1T2InAttesa*: intero non negativo (esprime quanti tra i thread T1 e T2 sono in attesa); valore iniziale 0
- *T3T4Ammessi*: intero non negativo; (esprime quanti tra i thread T3 e T4 sono ammessi) valore iniziale 0
- *T3T4InAttesa*: intero non negativo (esprime quanti tra i thread T3 e T4 sono in attesa); valore iniziale 0

e i seguenti semafori:

- *mutex* (valore iniziale 1): semaforo utilizzato per la mutua esclusione sulle sezioni critiche con le quali i processi verificano la condizione di accesso;
- *AttesaT1T2 (*valore iniziale 0)*:* semaforo utilizzato per la sospensione di T1 e T2.
- *AttesaT3T4 (*valore iniziale 0)*:* semaforo utilizzato per la sospensione di T3 e T4;

Si chiede di completare lo pseudo-codice sotto riportato, inserendo opportunamente le operazioni sui semafori *mutex, AttesaT1T2* e *AttesaT3T4.*

Soluzione

Processi T1 e T2

```
   // prologo per l'accesso alla struttura dati da parte di T1 e T2

   OK = 0;
   wait(mutex);
   if T3T4Ammessi== 0 and T1T2Ammesso == 0    {
        T1T2Ammesso =1; OK= 1;
       }
   else T1T2InAttesa++ ;
   signal(mutex);
   if OK == 0 wait(AttesaT1T2);

  < esegue l'accesso alla struttura dati>

// epilogo dell'accesso alla struttura dati da parte di T1 e T2
   wait(mutex);
   T1T2Ammesso = 0;
   if T1T2InAttesa > 0       {
              T1T2Ammesso = 1; T1T2InAttesa --; signal(AttesaT1T2);
       }
```

```
    else while T3T4InAttesa >0       {
         T3T4InAttesa --; signal(AttesaT3T4); T3T4Ammessi ++;
         }
    signal(mutex) ;
```

Processi T3 e T4

```
    // prologo per l'accesso alla struttura dati da parte di T3 e T4

    OK = 0 ;
    wait(mutex);
    if T1T2InAttesa == 0 and T1T2Ammesso == 0     {
         T3T4Ammessi ++; OK= 1;
         }
    else T3T4InAttesa ++;
    signal(mutex);
    if OK == 0  wait(AttesaT3T4);

   < esegue l'accesso alla struttura dati>

    // epilogo dell'accesso alla struttura dati da parte di T3 e T4
    wait(mutex);
    T3T4Ammessi -- ;
    if T3T4Ammessi== 0  and  T1T2InAttesa > 0       {
         T1T2Ammesso = 1; signal(AttesaT3T4);
         }
    signal(mutex);
```

Soluzione

Processi T1 e T2

```
    // prologo per l'accesso alla struttura dati da parte di T1 e T2

    OK = 0;

    if T3T4Ammessi== 0 and T1T2Ammesso == 0     {
          T1T2Ammesso =1; OK= 1;
         }
    else T1T2InAttesa++ ;

    if OK == 0

   < esegue l'accesso alla struttura dati>

// epilogo dell'accesso alla struttura dati da parte di T1 e T2

    T1T2Ammesso = 0;
    if T1T2InAttesa > 0       {
                T1T2Ammesso = 1; T1T2InAttesa --;

         }
    else while T3T4InAttesa >0       {
         T3T4InAttesa --; T3T4Ammessi ++;

         }
```

Processi T3 e T4

```
    // prologo per l'accesso alla struttura dati da parte di T3 e T4
```

```
OK = 0 ;

if T1T2InAttesa == 0 and T1T2Ammesso == 0     {
      T3T4Ammessi ++; OK= 1;
      }
else T3T4InAttesa ++;

if OK == 0

< esegue l'accesso alla struttura dati>

// epilogo dell'accesso alla struttura dati da parte di T3 e T4

T3T4Ammessi -- ;
if T3T4Ammessi== 0  and  T1T2InAttesa > 0       {
      T1T2Ammesso = 1;

      }
```

ESERCIZIO 3 (4 punti)

Un sistema con processi A, B, C, D, E e risorse dei tipi R1, R2, R3, R4, ha raggiunto lo stato mostrato nelle tabelle seguenti:

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

Esigenza Massima (indipendente dall'assegnazione attuale)				
	R1	R2	R3	R4
A	3	1	1	3
B	3	3	2	3
C	2	3	2	3
D	1		1	2
E	1	1	3	3

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

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			
0	1	1	1

Successivamente, arrivano in sequenza le seguenti richieste:

1. C richiede 1 istanza di R4
2. B richiede 1 istanza di R3
3. B richiede 1 istanza di R1

Il gestore delle risorse applica l'algoritmo del banchiere per evitare lo stallo.
Dire, per ogni richiesta, se la risorsa viene assegnata e lo stato raggiunto dal sistema e dal processo richiedente dopo la richiesta.

Soluzione

1. Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R4 al processo C:

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

Esigenza Massima (indipendente dall'assegnazione attuale)				
	R1	R2	R3	R4
A	3	1	1	3
B	3	3	2	3
C	2	3	2	3
D	1		1	2
E	1	1	3	3

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

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			
0	1	1	0

Nessun processo può terminare, quindi lo stato non è sicuro e il processo C viene messo in stato di attesa.

2. Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R3 al processo B:

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

Esigenza Massima (indipendente dall'assegnazione attuale)				
	R1	R2	R3	R4
A	3	1	1	3
B	3	3	2	3
C	2	3	2	3
D	1		1	2
E	1	1	3	3

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

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			
0	1	0	1

- Il processo E può terminare
  La disponibilità di {R1, R2, R3, R4} diviene { 1,1,3,3 }
- Il processo A può terminare
  La disponibilità di {R1, R2, R3, R4} diviene { 3,2,3,3}
- Il processo D può terminare
  La disponibilità di {R1, R2, R3, R4} diviene { 3,2,4,4}
- Il processo B può terminare
  La disponibilità di {R1, R2, R3, R4} diviene { 4,4,5,4}
- Il processo C può terminare
  La disponibilità di {R1, R2, R3, R4} diviene { 4,5,5,6}

Quindi lo stato è sicuro e l'istanza di R3 può essere assegnata a B che resta in stato di esecuzione

3. Non esistono istanze libere di R1 quindi il processo B viene messo in stato di attesa.

Soluzione

1. Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R4 al processo C:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza Massima (indipendente dall'assegnazione attuale)				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

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

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			

- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}

Di conseguenza:
Stato sicuro [SI/NO]? _____; risorsa assegnata [SI/NO]?: _____; stato del processo C: _____

2. Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R3 al processo B:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				

Esigenza Massima (indipendente dall'assegnazione attuale)				
	R1	R2	R3	R4
A				
B				
C				

Esigenza residua (esclusa l'assegnazione attuale)				
	R1	R2	R3	R4
A				
B				
C				

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità

D				
E				

D				
E				

D				
E				

- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}

Di conseguenza:
Stato sicuro [SI/NO]? _____; risorsa assegnata [SI/NO]?: _____; stato del processo B: _____

3. Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R1 al processo B:

Assegnazione attuale				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

Esigenza Massima (indipendente dall'assegnazione attuale)				
	R1	R2	R3	R4
A				
B				
C				
D				
E				

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

Molteplicità			
R1	R2	R3	R4
4	5	5	6

Disponibilità			

- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}
- Il processo …… può terminare. La disponibilità di {R1, R2, R3, R4} diviene {……}

Di conseguenza:
Stato sicuro [SI/NO]? _____; risorsa assegnata [SI/NO]?: _____; stato del processo B: _____

ESERCIZIO 4 (4 PUNTI)

Nel problema dei "filosofi a cena", *N* filosofi, individuati da un indice i compreso nell'intervallo *[0, N)* si riuniscono per la cena in un ristorante cinese, dove occupano un tavolo circolare, apparecchiato con un piatto per ogni filosofo e un bastoncino interposto fra ogni coppia di piatti adiacenti. Per mangiare, il filosofo di indice i deve acquisire il bastoncino di indice *i* e quello di indice *(i+1) mod N*. Se il bastoncino di indice *i* è utilizzato dal filosofo di indice *(i-1) mod N*, oppure se bastoncino di indice *(i+1) mod N* è utilizzato dal filosofo di indice *(i+1) mod N*, il filosofo di indice *i* deve attendere la disponibilità di entrambi i bastoncini. Ogni filosofo alterna ciclicamente, con velocità arbitraria, tra uno stato in cui pensa, uno in cui desidera mangiare e uno in cui mangia.
Se i filosofi sono thread di uno stesso processo, realizzati a livello kernel, il problema può essere risolto con semafori nel modo seguente:

ProgrammaDelFilosofo_i

```
{  i= IndiceDelFilosofo; stato[i]= pensa;
while (true)        {
   OK= 0;
   wait(mutex);
   stato[i]= HaFame;
   //la transizione nello stato "HaFame" corrisponde alla richiesta dei due bastoncini//
   if  stato[(i- 1) mod N]<> mangia and stato[(i+ 1) mod N]<> mangia      { stato[i]= mangia; OK= 1}
      // la transizione nello stato "mangia" corrisponde all'assegnazione dei due bastoncini //
   signal(mutex);
   if OK= 0  wait(filosofo[i]);    II il filosofo si sospende nello stato "pensa"; la richiesta rimane pendente//
   < il filosofo di indice i mangia >
  //il filosofo di indice i  ha finito di mangiare e torna nello stato "pensa" //
  wait(mutex);
   stato[i]=pensa;  // la transizione nello stato "pensa" corrisponde al rilascio dei due bastoncini //
   if stato[(i- 1) mod N]== HaFame and stato[(i- 2) mod N]<> mangia      {
         IIil filosofo (i-1) mod N ha a disposizione entrambi i bastoncini //
```

```
        stato[(i+ 1) mod N]= mangia ; signal(filosofo[i-1 mod N]));
        }
   if stato[(i+ 1) mod N]== HaFame and stato[(i+ 2) mod N]<> mangia    {
        IIil filosofo (i+1) mod N ha a disposizione entrambi i bastoncini //
        stato[(i+ 1) mod N]= mangia; signal(filosofo[i+1 mod N]));
        }
   signal(mutex)
   }
}
```

dove *filosofo[i]* è l'elemento di indice i di un vettore di semafori, associato al filosofo con lo stesso indice e inizializzato al valore 0;
mutex è un semaforo utilizzato per la mutua esclusione, con valore iniziale 1;
OK è una variabile locale;
stato[i] è l'elemento di indice *i* di un vettore condiviso di dimensione *N*. I possibili valori di *stato[i]* sono *"pensa"*, *"HaFame"* e *"mangia"*, con i seguenti significati:

- *stato[i]= "pensa"* significa che il filosofo di indice *i* non è in possesso di nessun bastoncino;
- *stato[i]= "HaFame"* significa che il filosofo di indice *i* richiede i due bastoncini che gli sono necessari per mangiare;
- *stato[i]= "mangia"* significa che il filosofo di indice *i* è in possesso dei due bastoncini

Si chiede di trasformare la precedente soluzione utilizzando, invece dei semafori, i meccanismi POSIX *pthread_mutex_lock, pthread_mutex_unlock, pthread_cond_wait e pthread_cond_signal.* Utilizzare la variabile *MutexTavolo*, di tipo *mutex* e un vettore *filosofo*, dove *filosofo[i]* è una variabile di tipo *condition* associata al filosofo di indice *i*. Si suggerisce di riflettere se in questa soluzione è necessario conservare la variabile locale *OK*.

Soluzione

```
ProgrammaDelFilosofo_i
{
i= IndiceDelFilosofo; stato[i]= pensa;
while (true)        {
   pthread_mutex_lock(&MutexTavolo);
   stato[i]= HaFame;
   //la transizione nello stato "HaFame" corrisponde alla richiesta dei due bastoncini//
   while  stato[(i- 1) mod N] == mangia or stato[(i+ 1) mod N]== mangia
        pthread_cond_wait(&filosofo[i]),&MutexTavolo)
        II il filosofo si sospende nello stato "pensa"; la richiesta rimane pendente//
    stato[i]= mangia;
   // la transizione nello stato "mangia" corrisponde all'assegnazione dei due bastoncini //
   pthread_mutex_unlock(&MutexTavolo)
   < il filosofo di indice i mangia >
   //il filosofo di indice i  ha finito di mangiare e torna nello stato "pensa" //
   pthread_mutex_lock(&MutexTavolo)
   stato[i]=pensa;
   // la transizione nello stato "pensa" corrisponde al rilascio dei due bastoncini //
   if stato[(i- 1) mod N]== HaFame and stato[(i- 2) mod N]<> mangia     {
        stato[[i- 1] mod N]= mangia;
        IIil filosofo (i-1) mod N ha a disposizione entrambi i bastoncini //
        pthread_cond_signal(&filosofo[(i+1) mod n];
        }
   if stato[(i+ 1) mod N]== HaFame and stato[(i+ 2) mod N]<> mangia    {
        stato[(i+ 1) mod N]= mangia;
        II il filosofo (i+1) mod N ha a disposizione entrambi i bastoncini //
        pthread_cond_signal(&filosofo[[i+1) mod n];
        }
   pthread_mutex_unlock(&MutexTavolo)
   }
}
```

Soluzione

```
ProgrammaDelFilosofo_i
{ i= IndiceDelFilosofo; stato[i]= pensa;
while (true)        {
```

```
   stato[i]= HaFame;
   while                                                    {

    stato[i]= mangia;

< il filosofo di indice i mangia >

 stato[i]=pensa;

   if                                                       {
        stato[[i- 1] mod N]= mangia;

        }
   if                                                       {
        stato[[i+ 1] mod N]= mangia;

        }


   }
}
```

Sistemi Operativi, corsi A e B - Prima prova di verifica intermedia - 8/4/2009

ESERCIZIO 5 (4 punti)

Un sistema simile a UNIX gestisce il processore con una politica a code multiple, con più Code Pronti associate alle diverse classi di priorità. I processi pronti sono inseriti nella coda della classe corrispondente alla propria priorità; il processore viene assegnato al processo che occupa la prima posizione nella coda non vuota della classe di **minima priorità**; ai processi pronti della stessa classe di priorità si applica la politica Round Robin con Quanto di Tempo (QdT) di *10 msec*. Quando un processo va in esecuzione gli viene assegnato un intero quanto di tempo, indipendentemente dal tempo consumato nel precedente turno di esecuzione.
La politica prevede la revoca del processore, che avviene immediatamente al verificarsi di uno dei seguenti eventi:

- il processo in esecuzione esaurisce il quanto di tempo;
- la priorità del processo in esecuzione viene aumentata ed esiste una coda non vuota di classe minore del nuovo valore di priorità;
- la priorità di un processo pronto viene diminuita e tutte le code di classe minore o uguale del nuovo valore sono vuote.

La priorità di un processo pronto assume il valore -5 quando il processo viene riattivato dopo essersi sospeso in stato supervisore;
La priorità del processo in esecuzione viene aumentata quando:

- il processo in esecuzione esegue la chiamata di sistema *nice(delta)*: la priorità è aumentata del valore *delta*;
- il processo in esecuzione esce dallo stato supervisore instaurato dalla chiamata di sistema all'interno della quale la sua priorità era stata decrementata: la priorità torna al valore che aveva prima del decremento.

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

- Processo A, con priorità 2, che è in esecuzione e ha ottenuto il processore al tempo *t*;
- Processo B, con priorità 2, che al tempo t è pronto;
- Processo C, con priorità 4, che al tempo t è pronto;
- Processo D, con priorità 4, che al tempo t è pronto;

La composizione delle code al tempo t è la seguente:

- coda 2: → B
- coda 4: → C → D

A partire dal tempo *t* si verificano i seguenti eventi:

1. t+ 15: il processo in esecuzione si sospende dopo aver eseguito una chiamata di sistema per la lettura da terminale;
2. t+ 20: il processo in esecuzione esegue la chiamata di sistema *nice(delta)* con *delta= 2;*
3. t+ 40: un segnale riattiva il processo sospeso in attesa del completamento della lettura da terminale
4. t+ 45: termina la gestione della chiamata di sistema con la quale al tempo t+ 15 il processo in esecuzione aveva richiesto la lettura da terminale
5. t+ 48: termina il processo in esecuzione.

Si chiede di compilare la seguente tabella definendo lo stato dei processi A, B, C e D a seguito per ogni intervento dello scheduler che si verifica tra il tempo 0 e il tempo 48.

Soluzione

Tempo	Evento	Processo (*) in Esecuzione	Coda 4	Coda 2	Coda (-5)	Sospesi
t+ 0	--	A (2)	→ C → D	→ B	Ø	--
t+ 10	Esaurito QdT	B (2)	→ C → D	→ A	Ø	--
t+ 15	Evento 1	A (2)	→ C → D	Ø	Ø	B
t+ 20	Evento 2	A (4)	→ C → D	Ø	Ø	B
t+ 25	Esaurito QdT	C (4)	→ D → A	Ø	Ø	B
t+ 35	Esaurito QdT	D (4)	→ A→ C	Ø	Ø	B
t+ 40	Evento 3	B(-5)	→A→C→D	Ø	Ø	--
t+ 45	Evento 4	B(2)	→A→C→D	Ø	Ø	--
t+ 48	Evento 5	A (4)	→C→D	Ø	Ø	--

(*) in parentesi la sua priorità

Sistemi Operativi, corsi A e B - Prima prova di verifica intermedia - 8/4/2009

Soluzione

Tempo	Evento	Processo (*) in Esecuzione	Coda 4	Coda 2	Coda (-5)	Sospesi
t+ 0	--	A (2)	→ C → D	→ B	Ø	--
t+						
t+						
t+						
t+						
t+						
t+						
t+						
t+						

(*) in parentesi la sua priorità

ESERCIZIO 6 (2 punti)
Si consideri un processi Unix A che ha due processi figli B e C. I tre processi condividono una pipe con un buffer di dimensione 2000 byte, e tutti i processi hann accesso in lettura e scrittura alla pipe.
In un dato istante di tempo la pipe è vuota ed avvengono le seguente richieste di scrittura e lettura dalla pipe. Dire, in seguito ad ognuna di queste richieste, quali processi vengono riattivati e quali processi vengono bloccati.

1. Il processo B effettua una richiesta di lettura di 1000 byte dalla pipe;
2. il processo A scrive un messaggio di dimensione 1500 byte sulla pipe
3. il processo C scrive un messaggio di dimensione 2000 byte sulla pipe
4. il processo B legge un messaggio di dimensione 500 byte dalla pipe

Soluzione
1. processi bloccati: B; processi riattivati: -
2. processi bloccati:- ; processi riattivati: B
3. processi bloccati:C ; processi riattivati: -
4. processi bloccati:- ; processi riattivati: C

Soluzione

1. processi bloccati:_____; processi riattivati:_____
2. processi bloccati:_____; processi riattivati:_____
3. processi bloccati:_____; processi riattivati:_____
4. processi bloccati:_____; processi riattivati:_____

ESERCIZIO 7 (2 punti)
In un sistema con risorse R1, R2, R3 e R4, tutte con molteplicità 2, sono presenti i processi P1, P2 e P3 che inizialmente non possiedono risorse e successivamente avanzano senza interagire reciprocamente e alternandosi nello stato di esecuzione con velocità arbitrarie.
Nel corso della propria esistenza, ciascun processo esegue una propria sequenza di richieste, che si intercalano in modo arbitrario con quelle degli altri processi. Dopo aver ottenuto e utilizzato le risorse che richiede, ogni processo termina rilasciando tutte le risorse ottenute.
Si consideri la sequenza di richieste sotto riportate:

Processo	Prima richiesta	Seconda richiesta	Terza richiesta	Quarta richiesta	Terminazione
P1	1 istanza di R1	1 istanza di R2	1 istanza di R3	1 istanza di R4	Rilascia tutto
P2	1 istanza di R3	1 istanza di R4	1 istanza di R3	1 istanza di R4	Rilascia tutto
P3	1 istanza di R1	1 istanza di R3	1 istanza di R3	1 istanza di R4	Rilascia tutto

Dire se i processi evitano la possibilità di stallo e motivare la risposta.

Soluzione
I processi possono andare in blocco perché non acquisiscono le risorse in modo ordinato.
Una sequenza che porta allo stallo è:

Sistemi Operativi, corsi A e B - Prima prova di verifica intermedia - 8/4/2009

P2 P3 P3 P2 (bloccato) P3 (bloccato) P1 P1 P1(bloccato)

Soluzione
I processi possono andare in blocco [SI/NO]?: _____
Motivazione:

ESERCIZIO 8 (2 punti)
In un sistema con thread *realizzati a livello utente*, dove l'unità schedulabile dal kernel è il processo, sono presenti

a. il processo P1 di priorità 2 e thread T11, T12,
b. il processo P2 di priorità 2 e thread T21 e T22,
c. il processo P3 di priorità 1 e thread T31 e T32.

Il sistema operativo schedula i processi con una politica a 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);
Ogni processo gestisce i suoi thread con politica Round Robin.
Immediatamente prima del tempo *t* è in esecuzione il processo P1 e i processi P2 e P3 sono pronti. Nel processo P1 è in esecuzione il thread T11 e tutti i rimanenti thread dei tre processi sono pronti, con il seguente ordinamento nelle rispettive code:
processo P1: T12.
processo P2: T21->T22 (ultimo).
processo P3: T31->T32 (ultimo).

Dire quale dei thread va in esecuzione dopo ciascuno degli eventi elencati in tabella, che si verificano in sequenza a partire dal tempo *t*.

	Evento che si verifica al tempo t	Thread in esecuzione dopo l'evento:
(a)	Il processo in esecuzione esaurisce il proprio quanto di tempo	T21
(b)	Il thread in esecuzione esegue una wait su un semaforo con valore 0	T11
(c)	Il thread in esecuzione esaurisce il proprio quanto di tempo	T12
(d)	Il processo in esecuzione esaurisce il proprio quanto di tempo	T12

	Evento che si verifica al tempo t	Thread in esecuzione dopo l'evento:
(a)	Il processo in esecuzione esaurisce il proprio quanto di tempo	
(b)	Il thread in esecuzione esegue una wait su un semaforo con valore 0	
(c)	Il thread in esecuzione esaurisce il proprio quanto di tempo	
(d)	Il processo in esecuzione esaurisce il proprio quanto di tempo	

ESERCIZIO 9 (2 punti)
In un sistema che definisce come unità di schedulazione i processi, dire quali delle seguenti operazioni possono essere eseguite solo in stato supervisore (scrivere SI o NO) ?

	Solo Supervisore
Modificare il vettore di interruzione	SI
Modificare la priorità di un thread	NO
Leggere il contenuto della coda pronti	SI
Modificare il contenuto della coda pronti	SI

ESERCIZIO 10 (2 punti)

In un sistema operativo che realizza i thread a livello kernel, i thread T1 e T2 del processo P si scambiano messaggi attraverso una pila di 10 celle, ciascuna della capacità di 1 messaggio. La pila è indicizzata dalla variabile condivisa `puntatore` (inizializzata al valore `0)`, e il numero di celle libere è contenuto nella variabile condivisa `CelleLibere`, inizializzata al valore 10. Si consideri la seguente soluzione realizzata con il semaforo `Mutex`, inizializzato ad `1`, e i semafori `SemThread` e `SemThread2`, inizializzati a `0` e utilizzati rispettivamente per la sospensione del thread T1 e del thread T2.

Thread T1:	Thread T2:

`ProduttoreSospeso= 0;`
`while (True) {`
`  messaggio = produce messaggio()`
	`ConsumatoreSospeso= 0;`
	`while (True) {`
	`  wait(&Mutex);`

```
        wait(&Mutex);
        if CelleLibere== 0 {
             ProduttoreSospeso= 1;
             wait(&SemThread1);
             }
        Pila[puntatore]= messaggio;
        puntatore ++;
        Celle Libere --;
        if ConsumatoreSospeso== 1 {
             ConsumatoreSospeso= 0;
             signal(&SemThread2);
             }
        signal(&Mutex);

}
```

```
        if CelleLibere== 10  {
        ConsumatoreSospeso= 1;
        wait(&SemThread2);
        }
        puntatore --;
        mess= Pila[puntatore];
        Celle Libere ++;
        if ProduttoreSospeso== 1  {
             ProduttoreSospeso= 0
             signal(&SemThread1);
             }
        signal(&Mutex);
        consuma_messaggio(mess);

}
```

La soluzione è corretta? Motivare la risposta.

Soluzione

La soluzione è errata
Motivo: si può raggiungere uno stallo. Infatti, se e il thread T1 trova Celle Libere= 0 si sospende senza rilasciare la mutua esclusione; analogamente, se il thread T2 trova Celle Libere= 10 si sospende senza rilasciare la mutua esclusione

Soluzione

La soluzione è …….
Motivo: