Sistemi Operativi e Laboratorio, Prova del 5/4/2016

Nome: _____ Cognome: _____ Matricola:_____ fila:___ posto: ___ corso:____

Esercizio 1 (5 punti)

Si consideri un processore che dispone dei seguenti registri:
- i registri speciali PC (program counter) e PS (program status) e lo stack pointer SP
- un banco di registri riservato allo stato utente, che comprende i registri generali R1, R2, R3.
- un ulteriore banco di registri riservato allo stato supervisore, che comprende i registri generali R'1, R'2, R'3.

Il sistema riserva in memoria un'area per il vettore di interruzione e per lo stack del nucleo. Inoltre, il processore ha un comportamento standard e salva automaticamente il minimo indispensabile sullo stack.

Al tempo *t* il processo Pi in esecuzione invoca una chiamata di sistema yield() che provoca una commutazione di contesto a favore del processo Pj, che viene messo in esecuzione. Il vettore di interruzione associato alla chiamata di sistema ha il valore 0800 e la parola di stato del nucleo è 375E.

Quando l'interruzione viene riconosciuta, i registri del processore, i descrittori di Pi e Pj e lo stack del nucleo, che inizia alla locazione 8080, hanno i contenuti mostrati in figura.

L'interruzione determina l'intervento del nucleo, che esegue una funzione di servizio comprendente lo scheduler. Si chiede:

a) il contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione della prima istruzione della funzione di servizio;
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 funzione di servizio;
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 P_J	
Stato	Pronto
PC	BAB0
PS	16F2
SP	F1C0
R1	2200
R2	2211
R3	2233

DESCRITTORE DI P_I	
Stato	Esec
PC	AA11
PS	26F2
SP	DEC3
R1	1122
R2	1133
R3	1144

STACK del nucleo	
8080	
8079	
8078	
8077	
8076	
8075	
8074	.
8073	

REG. STATO UTENTE	
R1	1199
R2	1188
R3	1177
REG. STATO SUPERV.	
R'1	0012
R'2	00CC
R'3	0045

PROCESSORE: Registri speciali								
PC	AB00		PS	36F2		SP	DEC0	

Soluzione

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

DESCRITTORE DI P_J	
Stato	Pronto
PC	BAB0
PS	16F2
SP	F1C0
R1	2200
R2	2211
R3	2233

DESCRITTORE DI P_I	
Stato	Esec
PC	AA11
PS	26F2
SP	DEC3
R1	1122
R2	1133
R3	1144

STACK del nucleo	
8080	AB00
8079	36F2
8078	DEC0
8077	
8076	
8075	
8074	
8073	

REG. STATO UTENTE	
R1	1199
R2	1188
R3	1177
REG. STATO SUPERV.	
R'1	0012
R'2	00CC
R'3	0045

PROCESSORE: Registri speciali								
PC	0800		PS	375E		SP	8077	

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 funzione di servizio;

DESCRITTORE DI P_J	
Stato	Esec
PC	BAB0

DESCRITTORE DI P_I	
Stato	Pronto
PC	AB00

STACK del nucleo	
8080	BAB0
8079	16F2

REG. STATO UTENTE	
R1	2200
R2	2211

PS	16F2		PS	36F2		8078	F1C0		R3	2233
SP	F1C0		SP	DEC0		8077				
R1	2200		R1	1199		8076				
R2	2211		R2	1188		8075				
R3	2233		R3	1177		8074			REG. STATO SUPERV.	
						8073			R'1	??
									R'2	??
									R'3	??
PROCESSORE: Registri speciali										
PC	0800+??		PS	375E		SP	8077			

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 P_J			DESCRITTORE DI P_I			STACK del nucleo			REG. STATO UTENTE	
Stato	Esec		Stato	Pronto		8080			R1	2200
PC	BAB0		PC	AB00		8079			R2	2211
PS	16F2		PS	36F2		8078			R3	2233
SP	F1C0		SP	DEC0		8077				
R1	2200		R1	1199		8076				
R2	2211		R2	1188		8075				
R3	2233		R3	1177		8074			REG. STATO SUPERV.	
						8073			R'1	??
									R'2	??
									R'3	??
PROCESSORE: Registri speciali										
PC	BAB0		PS	16F2		SP	F1C0			

Esercizio 2 (5 punti)

Si consideri un processore che dispone dei registri speciali PC (program counter) e PS (program status), dello stack pointer SP e dei registri generali R1, R2 e R3. In stato utente, ogni processo dispone di uno stack ad uso generale e di uno stack riservato per gestire le upcall chiamato *Signal Stack* (per semplicità assumiamo che ogni processo abbia un unico thread).

Per predisporre all'esecuzione della upcall in un processo, il nucleo utilizza il Signal Stack per salvare il contesto del processo. La upcall si conclude con l'istruzione RETU che ripristina la normale esecuzione del processo.

Al tempo t, il nucleo invia una upcall al processo P che è associata alla funzione di gestione che inizia all'indirizzo 1000 (nello spazio di memoria del processo). Il signal stack del processo comincia alla locazione E0CA, il processo è in stato di pronto e il contenuto dei suoi registri speciali e generali è conservato nel suo descrittore come mostrato in tabella.

DESCRITTORE DI P	
Stato	Pronto
PC	3300
PS	36F2
SP	D800
R1	4031
R2	4042
R3	4053

Mostrare:

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 gestione della upcall;
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 RETU con la quale termina la upcall;
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 RETU.

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 gestione della upcall:

DESCRITTORE DI P	
Stato	Esecuzione

SIGNAL STACK DI P	
E0CA	3300

REGISTRI SP, R1, …, R4	
SP	E0C5

PC	3300		E0C9	D800		R1	??
PS	36F2		E0C8	4031		R2	??
SP	D800		E0C7	4042		R3	??
R1	4031		E0C6	4053			
R2	4042		E0C5				
R3	4053		E0C4				

PROCESSORE: Registri speciali e stato							
PC	1000		PS	36F2		Stato	Utente

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 RETU con la quale termina la upcall:

DESCRITTORE DI P			SIGNAL STACK DI P			REGISTRI SP, R1, …, R4	
Stato	Esecuzione		E0CA	3300		SP	E0C8
PC	3300		E0C9	D800		R1	4031
PS	36F2		E0C8			R2	4042
SP	D800		E0C7			R3	4053
R1	4031		E0C6				
R2	4042		E0C5				
R3	4053		E0C4				

PROCESSORE: Registri speciali e stato							
PC	1000+??		PS	36F2		Stato	Utente

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 RETU.

DESCRITTORE DI P			SIGNAL STACK DI P			REGISTRI SP, R1, …, R4	
Stato	Esecuzione		E0CA			SP	D800
PC	3300		E0C9			R1	4031
PS	36F2		E0C8			R2	4042
SP	D800		E0C7			R3	4053
R1	4031		E0C6				
R2	4042		E0C5				
R3	4053		E0C4				

PROCESSORE: Registri speciali e stato							
PC	3300		PS	36F2		Stato	Utente

Esercizio 3 (3 punti)

In un sistema con thread realizzati a livello kernel, dove l'unità schedulabile è il thread, sono presenti i processi P1 con thread P11, P12, P13, e P2 con thread P21 e P22. Il processore viene gestito in time sharing con quanti di tempo e coda pronti gestita in modo FIFO (politica Round Robin).

Immediatamente prima del tempo *t* è in esecuzione il thread P13, il thread P21 è sospeso sul semaforo SA e i rimanenti thread sono pronti, con il seguente ordinamento nella coda pronti:

(primo) P22->P12->P11 (ultimo).

Inoltre nel sistema è presente il semaforo SB con valore 2. Riempire la seguente tabella indicando quale thread è in esecuzione se dal tempo t si verificano (in sequenza) i seguenti eventi:

	Evento	Thread in esecuzione	Valore di SA / thread sospesi su SA	Valore di SB / thread sospesi su SB
(a)	Il thread in esecuzione esegue P(SA)			

(b)	Il thread in esecuzione esegue V(SB)			
(c)	Scade il quanto di tempo			
(d)	Il thread in esecuzione esegue P(SB)			

Soluzione

	Evento	Thread in esecuzione	Valore di SA / thread sospesi su SA	Valore di SB e thread sospesi su SB
(a)	Il thread in esecuzione esegue P(SA)	P22	0 / P21, P13	2 / -
(b)	Il thread in esecuzione esegue V(SB)	P22	0 / P21, P13	3 / -
(c)	Scade il quanto di tempo	P12	0 / P21, P13	3 / -
(d)	Il thread in esecuzione esegue P(SB)	P12	0 / P21, P13	2 / -

Esercizio 4 (6 punti)

In un ufficio postale vi sono 20 sportelli per il servizio ai clienti, situati in una sala d'aspetto. I clienti entrano nella sala d'aspetto, attendono che vi sia uno sportello disponibile e, quando si verifica questo evento, lo individuano, lo raggiungono e ottengono il servizio voluto. Quindi liberano lo sportello e tornano in sala d'aspetto per uscire dall'ufficio.

Si consideri una formalizzazione del problema nel quale i clienti sono thread di uno stesso processo, realizzati a livello del nucleo, che si sincronizzano mediante la variabile *lock* per la mutua esclusione (sulla quale sono definite le operazioni `lock.acquire()` e `lock.release()`), e la variabile di condizione *SportelloDisponibile* (sulla quale sono definite le operazioni `SportelloDisponibile.wait(lock)` e `sportelloDisponibile.signal()`). I thread utilizzano inoltre la variabile *SportelliDisponibili*, di tipo intero e inizializzata al valore 20.

Si chiede di completare il seguente schema di soluzione e di indicare inoltre quale thread viene riattivato quando uno sportello viene liberato.

Soluzione

```
<il cliente entra nella sala d'aspetto>

_____________________
while (_____________________)
      _____________________

//esiste uno sportello disponibile: il cliente lo individua e lo raggiunge//
SportelliDisponibili --;

_____________________
<il cliente ottiene il servizio richiesto>

_____________________
<il cliente lascia lo sportello e torna in sala d'aspetto>
SportelliDisponibili ++;

_____________________

_____________________
<il cliente lascia la sala d'aspetto>
```

Viene riattivato il thread: _____

Soluzione

```
<il cliente entra nella sala d'aspetto>
lock.acquire();
while (SportelliDisponibili == 0) SportelloDisponibile.wait(lock);
//esiste uno sportello disponibile: il cliente lo individua e lo raggiunge//
SportelliDisponibili --;
lock.release();
<il cliente ottiene il servizio richiesto>
lock.acquire();
<il cliente lascia lo sportello e torna in sala d'aspetto>
SportelliDisponibili ++;
SportelloDisponibile.signal();
```

```
lock.release();
<il cliente lascia la sala d'aspetto>
```

Viene riattivato il thread: uno a caso tra quelli in attesa. Se nessun thread è in attesa la signal non ha effetto.

Esercizio 5 (3 punti)

Si consideri il seguente frammento di codice:

```
…
printf("X");
a = fork();
if (a!=0) {
        b=fork();
        if (b!= 0) { printf("Y"); exit(0);}
        else printf("Z");
        }
else printf("U");
printf("W");
…
```

Il processo che esegue l'intero frammento è quello che esegue le due fork, cioè il padre.
Il processo generato con la prima fork è il primo figlio.
Il processo generato con la seconda fork è il secondo figlio.
Dire cosa stampa il padre e i due figli nel caso in cui entrambe le fork abbiano successo.

Soluzione

Il padre stampa: _____
Il primo figlio stampa: _____
Il secondo figlio stampa: _____

Soluzione

Il padre stampa: "XY".
Il primo figlio stampa: "UW".
Il secondo figlio stampa: "ZW".

Esercizio 6 (3 punti)

In un sistema con thread *realizzati a livello kernel,* dove l'unità schedulabile è il thread, sono presenti i processi P1 con thread P11, P12, P13, e P2 con thread P21 e P22. Il processore viene gestito in time sharing con quanti di tempo e coda pronti gestita in modo FIFO (politica Round Robin).

.

Ad un certo istante di tempo è in esecuzione il thread P11, il thread P21 è sospeso sulla variabile di condizione *Condizione* (legata alla variabile mutex *mux*) e i rimanenti thread sono pronti, con il seguente ordinamento nella coda pronti:

(primo) P22->P12->P13 (ultimo).

Infine, le variabili di condizione sono realizzate con semantica MESA.
Dire come evolve lo stato del sistema se si verificano in sequenza i seguenti eventi:

	Evento	Thread in esecuzione	thread sospesi su Condizione	Stato di mux / thread sospesi su mux
(a)	Il thread in esecuzione esegue mux.release()			
(b)	Scade il quanto di tempo			
(c)	Il thread in esecuzione esegue mux.acquire()			
(d)	Il thread in esecuzione esegue Condizione.wait(mux)			

Soluzione

	Evento	Thread in esecuzione	thread sospesi su Condizione	Stato di mux / thread sospesi su mux
(a)	Il thread in esecuzione esegue mux.release()	P11	P21	Libero / -

(b)	Scade il quanto di tempo	P22	P21	Libero / -
(c)	Il thread in esecuzione esegue mux.acquire()	P22	P21	Bloccato / -
(d)	Il thread in esecuzione esegue Condizione.wait(mux)	P12	P21, P22	Libero / -

Esercizio 7 (5 punti)

In un sistema che implementa i thread a livello del nucleo, un processo comprende due gruppi di thread cooperanti: S1,..,Sn e T1,…,Tn. I thread cooperano tramite due vettori (`vet1` e `vet2`) entrambi a 20 posizioni ed utilizzati secondo il paradigma produttore consumatore.

In particolare:

- i thread S1,…,Sn agiscono da consumatori nei confronti di `Vet1`, e agiscono da produttori nei confronti di `Vet2`.
- i thread T1,…,Tn agiscono da produttori nei confronti di `Vet1`, e agiscono da consumatori nei confronti di `Vet2`.

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

- `Vet1Pieno`, inizializzato al valore 20, *(il valore esprime il numero di elementi disponibili in Vet)*
- `Vet1Vuoto`, inizializzato al valore 0, *(il valore esprime il numero di elementi occupati in Vet)*
- `Vet2Pieno`, inizializzato al valore 20, *(il valore esprime il numero di elementi disponibili in Vet)*
- `Vet2Vuoto`, inizializzato al valore 0, *(il valore esprime il numero di elementi occupati in Vet)*

Inoltre i thread utilizzano i semafori di mutua esclusione necessari per interagire correttamente.

A meno delle operazioni sui semafori, il codice eseguito dai thread S1,..,Sn e dai thread T1,…,Tm è riportato nello schema di soluzione.

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

Soluzione

Si utilizza/utilizzano il/i seguente/i semaforo/i di mutua esclusione:

- _____ per _____
- _____ per _____
- _____ per _____

Thread S1,…,Sn:

```
int v,d;
while(True) {
        _____________________________
        _____________________________
        _____________________________
        Legge v da Vet1
        _____________________________
        _____________________________
        _____________________________
        d=fun1(v);
        _____________________________
        _____________________________
        _____________________________

        Scrive d su Vet2
        _____________________________
        _____________________________
        _____________________________

}
```

Thread T1,…,Tn:

```
int v,d;
d=…;
while(True) {
        _____________________________
        _____________________________
        _____________________________
        Scrive d su Vet1
        _____________________________
        _____________________________
        _____________________________
        _____________________________
        _____________________________
        Legge v da Vet2
        _____________________________
        _____________________________
        _____________________________
        d=fun2(v);
}
```

Soluzione

Si utilizzano i seguenti semafori di mutua esclusione:

- `MutexVet1` per `Vet1`
- `MutexVet2` per `Vet2`

Thread S1,…,Sn:

```
int v,d;
```

Thread T1,…,Tn:

```
int v,d;
```

Sistemi Operativi e Laboratorio, Prova del 5/4/2016

```
while(True) {
      P(Vet1Vuoto);
      P(MutexVet1);
      Legge v da Vet1
      V(MutexVet1);
      V(Vet1Pieno);
      d=fun1(v);
      P(Vet2Pieno);
      P(MutexVet2);
      Scrive d su Vet2
      V(MutexVet2);
      V(Vet2Vuoto);
}
```

```
d=…;
while(True) {
      P(Vet1Pieno);
      P(MutexVet1);
      Scrive d su Vet1
      V(MutexVet1);
      V(Vet1Vuoto);
      P(Vet2Vuoto);
      P(MutexVet2);
      Legge v da Vet2
      V(MutexVet2);
      V(Vet2Pieno);
      d=fun2(v);
}
```