Sistemi Operativi e Laboratorio, autovalutazione del 2/4/2020

Esercizio 1 (5 punti)

Si consideri un processore che dispone dei registri speciali PC (program counter) e PS (program status), di un banco di registri generali, utilizzati sia in stato utente, sia in stato supervisore, che comprende i registri R1, R2, R3, R4 e lo stack pointer SP. Inoltre, il nucleo dispone di un proprio stack, che inizia all'indirizzo 1016 e si espande verso il basso, e di un proprio stack pointer SP' (valore 1016 quando lo stack è vuoto).
Il sistema gestisce il processore con una politica a priorità (valore maggiore di priorità corrisponde a priorità maggiore). Al tempo t sono presenti nel sistema il processo P_i, che si trova in stato di esecuzione, e i processi P_j e P_k che sono entrambi in stato di pronto. Tutti gli altri processi sono in stato bloccato. Al tempo t, il processo P_i esegue l'istruzione *SVC* per invocare il comando *exit*, con il quale termina l'esecuzione di un processo. Immediatamente dopo il riconoscimento dell'interruzione generata dalla *SVC*, i registri del processore, i descrittori di P_i, P_j e P_k hanno i contenuti mostrati in figura. Lo stack del nucleo è vuoto e il puntatore SP' ha il valore 1016.
L'interruzione determina l'intervento del nucleo, che esegue *exit*. Supponendo che il vettore di interruzione associato alla *SVC* sia 2900 e che la parola di stato del nucleo sia 275E, 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 SVC che realizza *exit*;
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 tale SVC;
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_i	
Stato	Esec
Priorità	5
PC	89AA
PS	16F2
SP	B000
R1	1000
R2	1100
R3	1110
R4	1111

DESCRITTORE DI P_J	
Stato	Pronto
Priorità	2
PC	C011
PS	16F2
SP	E899
R1	2000
R2	2200
R3	2220
R4	2222

DESCRITTORE DI P_k	
Stato	Pronto
Priorità	4
PC	6011
PS	16F2
SP	7899
R1	3000
R2	3300
R3	3330
R4	3333

STACK DEL NUCLEO	
1016	
1015	
1014	
1013	
1012	
1011	
1010	
1009	
1008	
….	

REGISTRI GENERALI	
SP	AE98
R1	1001
R2	1011
R3	0011
R4	0001

PROCESSORE: Registri speciali						
PC	9100		PS	1682	SP'	1016

Stato processore
UTENTE

Soluzione

a) contenuto dei descrittori, dei registri generali e speciali e dello stack del nucleo durante la fase di estrazione della prima istruzione della exit (in grigio i campi modificati): **vengono salvati i registri speciali di P_i anche se non serviranno più perché la fase firmware del cambio contesto (cioè durante la fase di estrazione della prima istruzione della exit) salva automaticamente tali valori nello stack del kernel. Sarà poi la gestione (software) della SVC (o interruzione o eccezione) – in questo caso della exit - a stabilire cosa farne di tali valori.**

DESCRITTORE DI P_i	
Stato	Esec
Priorità	5
PC	89AA
PS	16F2
SP	B000
R1	1000
R2	1100
R3	1110
R4	1111

DESCRITTORE DI P_J	
Stato	Pronto
Priorità	2
PC	C011
PS	16F2
SP	E899
R1	2000
R2	2200
R3	2220
R4	2222

DESCRITTORE DI P_k	
Stato	Pronto
Priorità	4
PC	6011
PS	16F2
SP	7899
R1	3000
R2	3300
R3	3330
R4	3333

STACK DEL NUCLEO	
1016	9100
1015	1682
1014	
1013	
1012	
1011	
1010	
1009	
1008	
….	

REGISTRI GENERALI	
SP	AE98
R1	1001
R2	1011
R3	0011
R4	0001

PROCESSORE: Registri speciali						
PC	2900		PS	275E	SP'	1014

Stato processore
SUPERVISORE

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 exit (in grigio i campi modificati):

DESCRITTORE DI -	
Stato	
Priorità	
PC	
PS	
SP	
R1	
R2	
R3	
R4	

DESCRITTORE DI P_J	
Stato	Pronto
Priorità	2
PC	C011
PS	16F2
SP	E899
R1	2000
R2	2200
R3	2220
R4	2222

DESCRITTORE DI P_k	
Stato	Esec.
Priorità	4
PC	6011
PS	16F2
SP	7899
R1	3000
R2	3300
R3	3330
R4	3333

STACK DEL NUCLEO	
1016	6011
1015	16F2
1014	
1013	
1012	
1011	
1010	
1009	
1008	
….	

REGISTRI GENERALI	
SP	7899
R1	3000
R2	3300
R3	3330
R4	3333

PROCESSORE: Registri speciali						
PC	2900+		PS	275E	SP'	1014

Stato processore
SUPERVISORE

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 (in grigio i campi modificati):

DESCRITTORE DI -	
Stato	
Priorità	
PC	
PS	
SP	
R1	
R2	
R3	
R4	

DESCRITTORE DI P_J	
Stato	Pronto
Priorità	2
PC	C011
PS	16F2
SP	E899
R1	2000
R2	2200
R3	2220
R4	2222

DESCRITTORE DI P_k	
Stato	Esec.
Priorità	4
PC	6011
PS	16F2
SP	7899
R1	3000
R2	3300
R3	3330
R4	3333

STACK DEL NUCLEO	
1016	
1015	
1014	
1013	
1012	
1011	
1010	
1009	
1008	
….	

REGISTRI GENERALI	
SP	7899
R1	3000
R2	3300
R3	3330
R4	3333

PROCESSORE: Registri speciali						
PC	6011		PS	16F2	SP'	1016

Stato processore
UTENTE

Esercizio 2 (5 punti)

In un sistema uniprocessore, 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 10, e il processo P2 con priorità 5. Lo scheduling dei processi avviene con una politica a priorità e prerilascio (va in esecuzione il processo pronto con il massimo valore di priorità).
Per ogni processo, i thread alternano tra lo stato di esecuzione e quello di pronto. Le transizioni avvengono quando il thread esegue la funzione *ThreadYield*: Il thread che passa dallo stato di esecuzione a quello di pronto viene inserito nell'ultima posizione della coda dei thread pronti. Lo scheduling dei thread avviene con politica FIFO.
I processi comunicano tra loro usando chiamate di sistema del kernel che implementano funzioni di tipo send(&mess, P) (che permette di inviare un messaggio al processo P) e receive(&buff) (che permette di ricevere un messaggio da un qualsiasi mittente. Le send sono **non** bloccanti, quindi deposita il messaggio nel buffer di ricezione del destinatario e non blocca il processo mittente. Le receive sono invece bloccanti e, se non è presente un messaggio nel buffer di ricezione, il processo che la invoca resta in attesa che arrivi un nuovo messaggio.
Al tempo *T* lo stato dei processi e dei thread è il seguente:

P1: Priorità 10; Stato di P1: esecuzione;
Thread di P1: T11, T12, T13; Thread in esecuzione: T11; CodaPronti dei thread → T12 → T13
P2: Priorità 5; Stato di P2: pronto;
Thread di P2: T21, T22, T23; Thread in esecuzione: T21; CodaPronti dei thread → T22→ T23
A partire dal tempo *t* si verifica la seguente sequenza di eventi :

1. il thread in esecuzione (del processo in esecuzione) esegue la primitiva (bloccante) receive(&buff1)
2. il thread in esecuzione esegue la primitiva (non bloccante) send(&mess1, P2)
3. il thread in esecuzione esegue ThreadYield;
4. il thread in esecuzione esegue la primitiva (non bloccante) send(&mess2, P1)
5. il thread in esecuzione esegue ThreadYield;
6. il thread in esecuzione esegue la primitiva (bloccante) receive(&buff1)

Sistemi Operativi e Laboratorio, autovalutazione del 2/4/2020

Supponendo che prima del tempo *T* non siano state eseguite primitive *send* o *receive*, si chiede quali sono gli eventi che determinano la ricezione di messaggi e quali sono i thread che li ricevono. (compilare la tabella)

Soluzione

Lo stato in tabella è quello **immediatamente dopo** l'evento

Evento	Processo P1 (priorità 10)			Processo P2 (priorità 5)			Messaggio inviato o ricevuto
	Stato	Thread in esecuzione	CodaPronti dei thread	Stato	Thread in esecuzione	CodaPronti dei thread	
Subito prima di 1	Esecuzione	T11	→T12→T13	Pronto	T21	→T22→T23	
1	Bloccato	T11	→T12→T13	Esecuzione	T21	→T22→T23	-
2	Bloccato	T11	→T12→T13	Esecuzione	T21	→T22→T23	Inviato mess1 a P2
3	Bloccato	T11	→T12→T13	Esecuzione	T22	→T23→T21	-
4	Esecuzione	T11	→T12→T13	Pronto	T22	→T23→T21	P2 invia *mess2* a P1; P1 riceve *mess2* e quindi può ripartire
5	Esecuzione	T12	→T13→T11	Pronto	T22	→T23→T21	-
6	Bloccato	T12	→T13→T11	Esecuzione	T22	→T23→T21	-

Esercizio 3 (3 punti)

Si consideri un sistema nel quale sono definiti i semafori *mutex* e *sem*. Lo scheduling avviene con una politica a priorità, che prevede il prerilascio e assegna il processore al processo pronto con valore più elevato di priorità (a pari priorità applica la politica FIFO). La politica applicata ai semafori è la FIFO.
Al tempo *T* sono presenti i seguenti processi:

- P1, con priorità 1, in esecuzione;
- P2, con priorità 1, pronto;
- P3, con priorità 2, sospeso sul semaforo *sem.*

Inoltre, sempre al tempo *T*, il semaforo *mutex* ha valore 0.
Dopo il tempo *T* si verificano **in sequenza** gli eventi riportati in tabella.
Si chiede di completare la tabella indicando qual è, subito dopo ogni evento, il processo in esecuzione, il contenuto della Coda Pronti, il valore e il contenuto della coda dei due semafori.

Soluzione

prima degli eventi P1 P2 (0,P3)

Eventi	In esecuzione	Coda Pronti	Senafori (valore, coda)	
			Sem *mutex*	Sem *sem*
1) P1 esegue *P (sem)*	P2	∅	(0, ∅)	(0, P3,P1)
2) Il processo in esecuzione esegue *V(mutex)*	P2	∅	(1, ∅)	(0, P3,P1)
3) Il processo in esecuzione esegue *V(sem)*	P3	P2	(1, ∅)	(0, P1)
4) Il processo in esecuzione esegue *V(sem)*	P3	P2, P1	(1, ∅)	(0, ∅)
5) Il processo in esecuzione esegue *V(sem)*	P3	P2, P1	(1, ∅)	(1, ∅)
6) Il processo in esecuzione esegue *P(mutex)*	P3	P2, P1	(0, ∅)	(1, ∅)

Sistemi Operativi e Laboratorio, autovalutazione del 2/4/2020

Esercizio 4 (3 punti)

In un sistema operativo che prevede scheduling senza prerilascio del processore, sono presenti i processi P1, P2, P3 e P4, e sono definiti i semafori *sem1* e *sem2*.
Al tempo *t* i semafori hanno la seguente configurazione:

Sem1: valore 0, coda P2, P4
Sem2: valore 1, coda Ø

Allo stesso tempo, la CodaPronti contiene P1 e il processo P3 è in esecuzione.
Dire 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:

1) P3 esegue *V (Sem1)* e successivamente il processo in esecuzione esegue *P(Sem1);*
2) P3 esegue *P (Sem2)* e successivamente il processo in esecuzione esegue *P(Sem2);*

Soluzione

prima degli eventi — P3 — P1 — 0, P2→ P4 — 1, vuota

	Sequenze di eventi	In Esecuzione	Coda Pronti	Sem1	Sem2
a-1	P3 esegue *V(Sem1)*	P3	P1, P2	0,P4	1, Ø
a-2	Il processo in esecuzione esegue *P(Sem1)*	P1	P2	0, P4, P3	1, Ø
b-1	P3 esegue *P (Sem2)*	P3	P1	0, P2, P4	0, Ø
b-2	Il processo in esecuzione esegue *P (Sem2)*	P1	Ø	0, P2, P4	0, P3

Esercizio 5 (5 punti)

Un sistema operativo con 5 processi (A, B, C, D, E) e risorse dei tipi R1, R2, R3, R4, rispettivamente di molteplicità [6, 6, 7, 3], 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
B	1	3	2	
C	2	2		
D			1	
E	2		3	2

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

Molteplicità			
R1	R2	R3	R4
6	6	7	3

Disponibilità			
1	1	1	0

Si considerino ora i seguenti casi (**in alternativa**):

a. il processo A richiede un'istanza della risorsa R1
b. Il processo D richiede un'istanza della risorsa R3

In ognuno dei due casi, dire se la risorsa viene assegnata dal sistema operativo.

Soluzione

Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R1 al processo A:

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

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

Molteplicità			
R1	R2	R3	R4
6	6	7	3

Disponibilità			
0	1	1	0

a1) Il processo A può terminare; la disponibilità di {R1, R2, R3, R4} diviene { 1,1,1,1}
a2) Il processo E può terminare ; la disponibilità di {R1, R2, R3, R4} diviene { 3,1,4,3}
a questo punto, un ordine qualunque di selezione dei processi va bene. Per semplicità, qui si prendono in ordine alfabetico
a3) Il processo B può terminare; la disponibilità di {R1, R2, R3, R4} diviene { 4,4,6,3}
a4) Il processo C può terminare ; la disponibilità di {R1, R2, R3, R4} diviene { 6,6,6,3}
a5) Il processo D può terminare ; la disponibilità di {R1, R2, R3, R4} diviene {6,6,7,3}

Di conseguenza: risorsa assegnata? SI

Stato raggiunto dopo l'ipotetica assegnazione di un'istanza di R3 al processo D:

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

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

Molteplicità			
R1	R2	R3	R4
6	6	7	3

Disponibilità			
1	1	0	0

b1) Il processo A può terminare; la disponibilità di {R1, R2, R3, R4} diviene { 1,1,0,1}
b2) Il processo E può terminare; la disponibilità di {R1, R2, R3, R4} diviene { 3,1,3,3}
a questo punto, un ordine qualunque di selezione dei processi va bene. Per semplicità, qui si prendono in ordine alfabetico
b3) Il processo B può terminare ; la disponibilità di {R1, R2, R3, R4} diviene { 4,4,5,3}
b4) Il processo C può terminare ; la disponibilità di {R1, R2, R3, R4} diviene { 6,6,5,3}
b5) Il processo D può terminare; la disponibilità di {R1, R2, R3, R4} diviene {6,6,7,3}

Di conseguenza: risorsa assegnata? SI

Esercizio 6 (5 punti)

In un sistema che implementa i thread a livello del nucleo, un processo che gestisce documenti ha due thread T1 e T2. In particolare, il thread T1 gestisce la visualizzazione, mentre il thread T2 gestisce la stampa di pagine del documento. La comunicazione tra i due Thread avviene tramite un buffer condiviso *Buf:* il buffer è realizzato come un vettore di *N* caratteri. Il Buffer è gestito tramite il puntatore *Testa* che indirizza il primo carattere libero nel buffer, e la variabile *NCaratteri* che indica il numero di caratteri presenti nel buffer.

- La scrittura nel buffer avviene da parte del thread T1 per pagine di *dim* caratteri (con *dim*<*N*), tramite la funzione *Deposita*(*&pagina*, *dim*). In questa funzione, se il numero di posizioni libere nel buffer è maggiore di *dim*, il contenuto di *&pagina* (di *dim* caratteri) viene ricopiato nel buffer a partire dalla posizione *Testa*, e *NCaratteri* viene incrementato di un valore pari a *dim.* Altrimenti, se il numero di posizioni libere nel buffer è inferiore a *dim*, il thread T1 viene sospeso senza modificare il buffer, e verrà riattivato in seguito dalla funzione *Preleva* eseguita dal thread T2 per ripetere il tentativo di scrittura.
- La lettura dei dati da stampare da parte del thread T2 avviene tramite la funzione *int Preleva*(*&pagine*). Se il buffer *Buf* non è vuoto, questa funzione svuota completamente il buffer, ricopiandone il contenuto dentro il buffer *pagine* di T2 (che si suppone di lunghezza illimitata), restituisce il numero di caratteri prelevati e se il thread T1 è sospeso in attesa di eseguire una scrittura, lo riattiva. Altrimenti, se il buffer *Buf* è vuoto, il thread T2 viene sospeso e verrà riattivato in seguito dalla funzione *Deposita* per ripetere il tentativo di lettura.

Per la sincronizzazione, i thread utilizzano le variabili di condizione *SospesoInScrittura* e *SospesoInLettura*, e la lock *mux*. Si chiede di completare lo pseudo codice delle funzioni *Deposita* e *Preleva*, parzialmente definito nello schema di soluzione.

Soluzione

```
void function Deposita(&pagina, dim)
{
    lock_acquire(mux);
    while (N-NCaratteri < dim)        // non ci sono sufficienti posizioni disponibili //
          SospesoInScrittura.wait(mux);

    for (i= 0; i< dim; i ++) {
        Buf[testa] = pagina[i];
        testa++;
    }

    NCaratteri=NCaratteri+dim;
    SospesoInLettura.signal();
    lock_release(mux);
}
```

```
int function Preleva(&pagine)
{
    lock_acquire(mux);
    Int prelevati=NCaratteri;
    while (NCaratteri== 0) SospesoInLettura.wait(mux);

    for (i= 0; i< NCaratteri; i ++) pagine[i]= Buf[i];

    NCaratteri=0;
    Testa=0;
    SospesoInScrittura.signal();
    lock_release(mux);
    return (prelevati);
}
```

Esercizio 7 (5 punti)

Si consideri un sistema che gestisce il processore con politica *RoundRobin*, con quanto di tempo di *5 msec.* Lo scheduler è attivato 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	1	20	Avanza fino alla terminazione senza sospendersi
B	0	25	Avanza fino alla terminazione senza sospendersi
C	3	40	Si sospende dopo aver utilizzato il processore per 18 msec; viene riattivato dopo 4 msec
D	2	25	Si sospende dopo aver utilizzato il processore per 16 msec; viene riattivato dopo 3 msec

Si chiede il tempo di terminazione dei processi A e B.

Soluzione

Al tempo 0 passa in esecuzione il processo B.
Dal tempo 3 (quando è in esecuzione il processo B), la coda pronti contiene ordinatamente i processi A, D, C.

Il processo A ottiene per la prima volta il processore al tempo 5 (scadenza del primo quanto di tempo del processo B) e lo utilizza per 5 msec. Dopo il primo turno di esecuzione, cioè quando sono stati eseguiti tutti i processi per un quanto, la situazione residua è: A=(15,-), B=(20,-), C=(35,13), D=(20,11), dove la notazione (x,y) indica che il processo deve ancora essere eseguito per x msec, e che tra y msec si sospenderà (- se non si sospenderà mai). Alla fine dei due turni successivi si avrà: A=(10,-), B=(15,-), C=(30,8), D=(15,6) e A=(5,-), B=(10,-), C=(25,3), D=(10,1). A questo punto, A deve attendere un quanto di B ed il suo ultimo quanto. Siccome ogni turno dura 5*4=20 msec, il processo A termina al tempo T_A= 3*20+5+5 = 70 msec.

A questo punto, quando il processo A termina, la situazione è la seguente: A=(-,-), B=(5,-), C=(25,3), D=(10,1) e al processo B mancano ancora 5 msec per la terminazione. Subito dopo la terminazione di A passa in esecuzione D, che si sospende dopo 1 msec; quindi passa in esecuzione C, che si sospende dopo 3 msec; infine passa in esecuzione B che termina dopo 5 msec. Pertanto, il processo B termina al tempo T_B= T_A+1+3+5= 70+1+3+5=79 msec.