SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 4/11/2009 CORSO |A| |B|

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

ESERCIZIO 1 (4 punti)

Si consideri un processore che dispone dei registri speciali PC (program counter) e PS (program status), di un banco di registri riservato allo stato utente, che comprende i registri generali R1, R2, R3, R4 e lo stack pointer SP, e di 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'.

Il sistema gestisce il processore con politica Round Robin e dispone anche di una primitiva per il rilascio spontaneo del processore da parte del processo in esecuzione, invocabile con la chiamata di sistema `Yield().` Al tempo t sono presenti nel sistema (tra gli altri) il processo Pi, in stato di esecuzione, e il processo Pj che è il primo processo in coda pronti. Sempre al tempo t il processo Pi invoca la chiamata di sistema `Yield().` Immediatamente dopo il tempo *t*, quando l'interruzione viene riconosciuta, i registri del processore, i descrittori di Pi e Pj e lo stack del nucleo hanno i contenuti mostrati in figura.

L'interruzione determina l'intervento del nucleo, che esegue una funzione di servizio. Supponendo che il vettore di interruzione associato alla chiamata `Yield()` sia **0425** 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 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 PI		DESCRITTORE DI PJ		STACK DEL NUCLEO		REG. STATO UTENTE		REG. STATO SUPERV.	
Stato	Esecuzione	Stato	Pronto	……	……	SP	EF00	SP'	1016
PC	4100	PC	A100	1016	AAAA	R1	1100	R'1	0000
PS	16F2	PS	16F2	1015		R2	2200	R'2	0001
SP	EF00	SP	F000	1014		R3	3300	R'3	0003
R1	1111	R1	A111	1013		R4	4400	R'4	0004
R2	2222	R2	A222	1012					
R3	3333	R3	A333	1011		PROCESSORE: Registri speciali			
R4	4444	R4	A444	1010		PC	4FFF	PS	16F2

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 PI		DESCRITTORE DI PJ		STACK DEL NUCLEO		REG. STATO UTENTE		REG. STATO SUPERV.	
Stato	Esecuzione	Stato	Pronto	……	……	SP	Invariato	SP'	1014
PC	Invariato	PC	Invariato	1016	AAAA	R1	Invariato	R'1	Invariato
PS	Invariato	PS	Invariato	1015	4FFF	R2	Invariato	R'2	Invariato
SP	Invariato	SP	Invariato	1014	16F2	R3	Invariato	R'3	Invariato
R1	Invariato	R1	Invariato	1013		R4	Invariato	R'4	Invariato
R2	Invariato	R2	Invariato	1012					
R3	Invariato	R3	Invariato	1011		PROCESSORE: Registri speciali			
R4	Invariato	R4	Invariato	1010		PC	0425	PS	275E

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 PI		DESCRITTORE DI PJ		STACK DEL NUCLEO		REG. STATO UTENTE		REG. STATO SUPERV.	
Stato	Pronto	Stato	Esecuzione	……	……	SP	F000	SP'	1014
PC	4FFF	PC	Invariato	1016	AAAA	R1	A111	R'1	??
PS	16F2	PS	Invariato	1015	A100	R2	A222	R'2	??
SP	EF00	SP	Invariato	1014	16F2	R3	A333	R'3	??
R1	1100	R1	Invariato	1013		R4	A444	R'4	??
R2	2200	R2	Invariato	1012					
R3	3300	R3	Invariato	1011		PROCESSORE: Registri speciali			
R4	4400	R4	Invariato	1010		PC	0425+ ??	PS	275E

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 PI		DESCRITTORE DI PJ		STACK DEL NUCLEO		REG. STATO UTENTE		REG. STATO SUPERV.	
Stato	Pronto	Stato	Esecuzione	……	……	SP	Invariato	SP'	1016
PC	Invariato	PC	Invariato	1016	AAAA	R1	Invariato	R'1	??
PS	Invariato	PS	Invariato	1015		R2	Invariato	R'2	??
SP	Invariato	SP	Invariato	1014		R3	Invariato	R'3	??
R1	Invariato	R1	Invariato	1013		R4	Invariato	R'4	??
R2	Invariato	R2	Invariato	1012					
R3	Invariato	R3	Invariato	1011		PROCESSORE: Registri speciali			
R4	Invariato	R4	Invariato	1010		PC	A100	PS	16F2

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 4/11/2009 CORSO |A| |B|

ESERCIZIO 2 (4 punti)

Nel problema del "barbiere dormiglione" si considera un negozio gestito da un unico barbiere, con una poltrona per il servizio dei clienti e un numero illimitato di sedie per l'attesa. Il barbiere e i clienti sono processi ad ambiente globale e la poltrona è una risorsa, che può essere assegnata a un cliente per il taglio dei capelli, oppure utilizzata dal barbiere per dormire. All'apertura del negozio e quando non ci sono clienti in attesa di servizio, il barbiere occupa la poltrona per dormire, fino all'arrivo del primo cliente.
Quando entra nel negozio, il generico cliente ha il seguente comportamento:

- se il barbiere è addormentato, lo risveglia provocando il rilascio della poltrona, che occupa immediatamente;
- altrimenti (ciò avviene quando il barbiere è attivo per eseguire il taglio dei capelli ad un altro cliente) attende il suo turno accomodandosi su una delle sedie;
- quando arriva il suo turno, è risvegliato dal barbiere ed occupa la poltrona;
- dopo il taglio dei capelli, paga ed esce dal negozio.

Dopo che un cliente ha occupato la poltrona, il barbiere esegue il taglio dei capelli e al termine:

- se ci sono clienti in attesa del proprio turno, riattiva il primo;
- altrimenti occupa la poltrona e si addormenta.

Il problema viene risolto utilizzando i seguenti dati condivisi:

- *BarbiereAddormentato: booleano;*
- *ClientiInAttesa: intero; valore iniziale 0;*

e i seguenti semafori:

- *mutex: valore iniziale 1 (per la mutua esclusione);*
- *AttesaBarbiere: valore iniziale 0 (utilizzato dal barbiere per addormentarsi sulla poltrona. L'attesa su questo semaforo implica l'assegnazione della poltrona al barbiere);*
- *AttesaTurno: valore iniziale 0 (per i clienti che attendono il taglio. Le modalità di utilizzo garantiscono che il valore di questo semaforo sia sempre 0)*
- *TaglioCapelli: valore iniziale 0 (il cliente si sospende su questo semaforo all'inizio del taglio dei capelli e viene riattivato dal barbiere alla fine. L'attesa su questo semaforo implica l'assegnazione della poltrona al cliente).*

Per ogni cliente è inoltre definita la variabile privata *attende* per il controllo della sospensione a seguito della decisione presa in mutua esclusione. Allo stesso fine il barbiere utilizza la variabile privata *dorme.*
Notare che la risorsa poltrona non viene gestita esplicitamente, perché le assegnazioni e i rilasci sono implicite nelle operazioni sui semafori AttesaBarbiere e TaglioCapelli.
Si chiede di completare il programma del barbiere e il frammento di codice che controlla i clienti che entrano nel negozio, inserendo le opportune operazioni sui semafori.

SOLUZIONE

```
Barbiere
{
BarbiereAddormentato= true; wait(AttesaBarbiere);
// la riga precedente inizializza il negozio ed è eseguita prima
della generazione dei processi "cliente" //
while(true)          {
        <esegue il taglio dei capelli>
        signal(taglioCapelli);
        wait(mutex);
        if ClientiInAttesa> 0          {
                // riattiva un cliente in attesa del turno e
                libera la poltrona //
                ClientiInAttesa –;  dorme= false
                signal(AttesaTurno);                    }
                }
        else { BarbiereAddormentato= true; dorme= true; }
        // occupa la poltrona per dormire, sospendendosi //
        signal(mutex);
        if dorme= true  wait(AttesaBarbiere);
        }
}
```

```
Cliente
//Frammento di codice//
< entra nel negozio >
wait(mutex);
if BarbiereAddormentato== true {
        BarbiereAddormentato= false; attende= false;
        signal(AttesaBarbiere);
        // sveglia il barbiere che libera la poltrona //
        }
else                            {
        // il barbiere è sveglio e sta servendo un cliente //
        ClientiInAttesa ++; attende= true;
        }
signal(mutex);
if attende == true
        wait(AttesaTurno);
        // Il cliente attende il suo turno. Sarà riattivato dal
        barbiere e troverà la poltrona libera //
wait(TaglioCapelli);
//siede sulla poltrona e attende il taglio dei capelli. Al
termine…//
<paga ed esce dal negozio>
// Fine del frammento di codice //
```

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 4/11/2009 CORSO |A| |B|

ESERCIZIO 3 (4 punti)

Nel problema dei "filosofi a cena" N filosofi si riuniscono per la cena in un ristorante cinese, occupando 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* ($0 \le i \le N-1$) deve avere a disposizione i due bastoncini posti rispettivamente alla sua sinistra e alla sua destra (cioè quelli di indice *i* e *(i+1) mod N)*, entrando pertanto in competizione con i due filosofi che siedono ai suoi lati.
Ogni filosofo alterna periodi in cui pensa a periodi in cui vuole mangiare. Quando decide di mangiare richiede con richiesta multipla i due bastoncini che gli sono necessari, applicando quindi una strategia di prevenzione statica dello stallo, e si sospende se non li ottiene entrambi. Dopo aver mangiato torna a pensare e rilascia i due bastoncini.
La riattivazione di un filosofo in attesa dei bastoncini può avvenire quando uno dei filosofi adiacenti rilascia quelli in suo possesso. La riattivazione è subordinata alla condizione che sia disponibile, oltre al bastoncino appena rilasciato dal filosofo che siede alla sua destra (o alla sua sinistra), anche quello interposto tra lui medesimo e il filosofo che siede alla sua sinistra (o alla sua destra).
In questo problema i filosofi sono processi ad ambiente locale e i bastoncini sono risorse. Si utilizza inoltre un processo *Server,* e le variabili che descrivono lo stato dei processi e delle risorse sono dati privati di questo processo. Per le interazioni tra il generico filosofo e il *Server* sono disponibili le seguenti primitive:

- *send(proc, &buffer)*: asincrona; invia al processo di indice *proc* il messaggio contenuto in *buffer*
- *send(mailbox, &buffer)*: asincrona; invia a *maibox* il messaggio contenuto in *buffer,*
- *receive(proc, &buffer)*: bloccante; riceve un messaggio dal processo di indice *proc* e lo trasferisce in *buffer*
- *receive(mailbox , &mitt, &buffer)*: bloccante; riceve un messaggio da *mailbox* e lo trasferisce in *buffer*; inoltre riceve l'indice del mittente e lo restituisce con variabile *mitt*.

Lo stato dei filosofi e dei bastoncini è definito implicitamente dal vettore *stato*, la cui componente *stato[i]* è associata al filosofo di indice *i*. I possibili valori di *stato[i]* sono *pensa, HaFame* e *mangia*, con i seguenti significati: .

- *stato[i]= pensa:* il filosofo i non possiede i bastoncini di indice *i* e *(i+ 1) mod N*
- *stato[i]= mangia:* il filosofo i possiede i bastoncini di indice *i* e *(i+ 1) mod N*
- *stato[i]= HaFame:* il filosofo i è in attesa di ottenere i bastoncini di indice *i* e *(i+ 1) mod N*

Inotre la transizione *mangia* → *pensa* di *stato[i]* corrisponde al rilascio dei bastoncini di indice i *e (i+ 1) mod N* .
Il generico filosofo esegue il seguente programma:

```
while true      {
        BufferUscita= "voglio_mangiare" ; send(mailbox, &BufferUscita) // richiede il bastoncino alla sua destra e quello alla sua sinistra//
        receive(server, &BufferIngresso);                              // la ricezione del messaggio conferma l'assegazione //
        <mangia per una durata di tempo casuale>
        BufferUscita= "voglio_pensare"; send(mailbox, &BufferUscita)   // rilascia i bastoncini senza attendere la risposta //
        <pensa per una durata di tempo casuale >
        }
```

Si chiede di completare il programma del processo *Server*, inserendo in particolare le primitive di comunicazione e le espressioni che condizionano l'assegnazione dei bastoncini e la riattivazione dei filosofi sospesi.

SOLUZIONE

```
Server
{
while true      {
receive(maibox, &mitt, &mess)
if mess= "voglio mangiare"          {
        stato[mitt]= HaFame;
        if stato[(mitt-1) mod N]<> mangia and stato[(mitt+1) mod N]<> mangia     {
        stato[mitt]= mangia; OutBuff= "buon appetito";
        send(mitt, &OutBuff);
        }
else            {
        // ricevuto il messaggio "voglio pensare" //
        stato[mitt]= "pensa";
        if (stato[(mitt-1) mod N]== HaFame) and (stato[(mitt-2) mod N]<> mangia) {
                stato[(mitt-1) mod N]= mangia; OutBuff= "buon appetito";
                send((mitt-1) mod N, &OutBuff));
                // il filosofo di indice (mitt-1) mod N ottiene i bastoncini di indice
                 (mitt-1) mod N e di indice (mitt-2) mod N, e pertanto viene riattivato //
                }
        if (stato[(mitt+1) mod N]== HaFame) and (stato[(mitt+2) mod N]<> mangia) {
                stato[(mitt+1) mod N]= mangia; OutBuff= "buon appetito";
                send((mitt+1)mod N, &OutBuff));
                // il filosofo di indice (mitt+1) mod N ottiene l bastoncini di indice
                (mitt+1) mod N e di indice (mitt+2) mod N e pertanto viene riattivato //
                }
        }
}
```

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 4/11/2009 CORSO |A| |B|

ESERCIZIO 4 (4 punti)

In un sistema con risorse Q, R e S singole, non prerilasciabili e utilizzabili in mutua esclusione, viene generato inizialmente (t= 0) il processo A, con priorità 1, che chiede immediatamente la risorsa Q.
Al tempo t= 5 viene generato il processo B, con priorità 3; al tempo t= 6 viene generato il processo C, con priorità 2. Tutti i tempi sono espressi in msec. All'atto della generazione, i processi si trovano in uno stato attivo (pronto o esecuzione, in base alle decisioni dello scheduler).
A partire dai rispettivi tempi di generazione, i programmi dei singoli processi specificano le seguenti azioni:

PROCESSO A	Chiede Q	Avanza per 10 msec	Chiede S	Avanza per 10 msec	Termina		
PROCESSO B	Chiede R	Avanza per 10 msec	Chiede Q	Avanza per 10 msec	Chiede S	Avanza per 10 msec	Termina
PROCESSO C	Chiede Q	Avanza per 10 msec	Chiede R	Avanza per 10 msec	Chiede S	Avanza per 10 msec	Termina

Si intende che ogni processo mantenga il possesso delle risorse ottenute fino alla sua terminazione.
I processi sono concorrenti e avanzano con velocità determinate dalle transizioni di stato e dallo scheduling.
Lo scheduler assegna il processore al processo attivo con il massimo valore di priorità e prevede il prerilascio. A pari priorità applica la politica FIFO.
Il processo che richiede una risorsa l'ottiene immediatamente se è disponibile; altrimenti si sospende in attesa di ottenerla. Al rilascio di una risorsa viene riattivato, se esiste, uno dei processi in attesa di quella risorsa, selezionato con politica FIFO.
Si chiede:

- l'evoluzione del sistema a partire dal tempo t= 0 e fino alla terminazione di tutti i processi o fino al raggiungimento di uno stallo;
- Si verifica uno stallo?

SOLUZIONE

T=	Evento	Stato dei processi (in parentesi priorità)			Stato delle risorse		
		A (1)	B (3)	C (2)	Q	R	S
0	Generato A.	Esec	-	-	→ A	libera	libera
5	A ha avanzato per 5 msec; Generato B che chiede R	Pronto	Esec	-	→ A	→ B	libera
6	Generato C.	Pronto	Esec	Pronto	→ A	→ B	libera
15	B richiede Q e si sospende	Pronto	Attende Q	Esec	→ A	→ B	libera
15	C richiede Q e si sospende	Esec	Attende Q	Attende Q	→ A	→ B	libera
20	A chiede S	Esec	Attende Q	Attende Q	→ A	→ B	→ A
30	A termina. B ottiene Q	-	Esec	Attende Q	→ B	→ B	libera
40	B chiede S	-	Esec	Attende Q	→ B	→ B	→ B
50	B termina C ottiene Q	-	-	Esec	→ C	libera	libera
60	C chiede R	-	-	Esec	→ C	→ C	libera
70	C chiede S	-	-	Esec	→ C	→ C	→ C
80	C termina	-	-	-	libera	libera	libera

Si verifica uno stallo? NO

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 4/11/2009 CORSO |A| |B|

ESERCIZIO 5 (4 punti)

Un traghetto svolge un servizio di attraversamento di un fiume alternativamente da nord a sud e viceversa. Le persone che vogliono attraversare il fiume aspettano sulla riva l'arrivo del traghetto. Quando il traghetto arriva ad una riva (nord o sud) fa scendere tutti i passeggeri, quindi carica i nuovi passeggeri uno per volta fino al limite della sua capienza e riparte per la riva opposta. Se non sono presenti passeggeri il traghetto riparte comunque per la sponda opposta.

Il traghetto e i passeggeri sono pthread che si sincronizzano per realizzare l'interazione descritta. Per l'interazione tra i thread si utilizzano le operazioni della libreria standard *p_thread*, operando sulla variabile *mux* di tipo mutex, e sulle variabili *attesaPasseggeri[Nord]*, *attesaPasseggeri[Sud]*, *attesa Arrivo*, *attesaSbarco* e *attesaImbarco* di tipo condition. In particolare le condition *attesaSbarco* e *attesa Imbarco* sono usate dal pthread traghetto per far sbarcare ed imbarcare i passeggeri uno alla volta, mentre le altre condition sono usate per la sospensione dei passeggeri.

Lo stato del pthread traghetto è rappresentato dalla variabile *Stato* che assume i valori *inViaggio*, *sbarcoPasseggeri* e *imbarco Passeggeri*, dalla variabile *direzione Traghetto* che assume i valori *Nord* o *Sud*, e la variabile *passeggeriImbarcati* che contiene il numero di passeggeri imbarcati sul traghetto.

Nella tabella sottostante è riportato il codice del traghetto. Si chiede di completare il codice del passeggero.

```
thread traghetto
{
    direzioneTraghetto = Nord ; Stato=inViaggio;passeggeri Imbarcati =0;   // Inizializzazione
    while true {
        pthread_mutex_lock(&mux);
        If (direzioneTraghetto == Nord) direzioneTraghetto = Sud          // arriva ad una sponda e inverte direzione
        else direzioneTraghetto =  Nord
        Stato = sbarcoPasseggeri;                                          // fa scendere i passeggeri
        for(i=0, i<passeggeri Imbarcati, i++) {
            pthread_condition_signal (&attesaArrivo);                      // attiva un passeggero per lo sbraco
            pthread_cond_wait(&attesaSbarco, &mux);                        // attende che sia sbarcato
        }
        passeggeri Imbarcati=0;
        passeggeriImbarcati = min(10, passeggeriInAttesa[direzioneTraghetto]);  // numero di passeggeri da imbarcare, fino ad un massimo di 10
        Stato = imbarcoPasseggeri;
        for(i=0, i<passeggeri Imbarcati, i++) {
            pthread_condition_signal (&attesaPasseggeri[direzioneTraghetto]);    // attiva un passeggero per l'imbarco
            pthread_cond_wait(&attesaImbarco, &mux);                             // attende che sia imbarcato
            }
        Stato = InViaggio;
        pthread_mutex_unlock(&mux);
        <attraversa il fiume>
    }
```

SOLUZIONE

```
thread passeggero
// Frammento di codice eseguito da un passeggero che si imbarca sulla sponda Sud, in direzione Nord
// Analogamente se il passeggero si imbarca in direzione Sud
    spondaDiPartenza= sud;direzione= Nord;
    pthread_mutex_lock(&mux);                                          // si presenta per l'imbarco
    passeggeriInAttesa[spondaDiPartenza] ++;
    while (Stato != imbarcoPasseggeri OR DirezioneTraghetto == spondaDiPartenza)   // deve aspettare
        pthread_condition_wait (&attesaPasseggeri[direzione], &Mutex);
    passeggeriInAttesa[direzione] --;
    // il passeggero si imbarca
    pthread_cond_signal(&attesaImbarco);                               // il passeggero si è  imbarcato
    while (Stato != sbarcoPasseggeri)
        pthread_condition_wait (&attesaArrivo, &Mutex);                // attende lo sbarco
    pthread_cond_signal(&attesaSbarco);                                // il passeggero è sbarcato
    pthread_mutex_unlock(&mux);
    // Fine del frammento di codice
```

ESERCIZIO 6 (2 punti)

In un sistema UNIX, un processo genera due processi figli eseguendo il seguente frammento di codice:

```
printf("UNO ");
pid1=fork();
if(pid1<0) printf("DUE ");
else if (pid1>0) {
      pid2=fork();
      if (pid2!=0) printf("TRE ");
      else printf("QUATTRO ");
      exec("progr",NULL);
      }
      else printf("CINQUE ");
printf("SEI ");
```

Si chiede che cosa stampa per effetto di questo codice il processo **padre**.
Rispondere discutendo i casi nei quali le due fork e la exec falliscono o hanno successo.

SOLUZIONE

Se la prima fork fallisce stampa: UNO DUE SEI
Se la prima fork ha successo, per qualunque esito della seconda fork:

- se la exec fallisce stampa: UNO TRE SEI
- se la exec ha successo stampa: UNO TRE

ESERCIZIO 7 (2 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 descrittori *fd[0]* e *fd[1];*
b) il processo P esegue con successo una *fork,* generando il processo P1;
c) il processo P esegue con successo una *exec;*
d) il processo P esegue con successo una *fork,* generando il processo P2;
e) il processo P esegue l'operazione *close (fd[1]);*
f) il processo P2 esegue con successo una *exec;*
g) il processo P1 esegue l'operazione *close (fd[1]);*

Si chiede quali tra le operazioni di lettura e di scrittura sul pipe possono eseguite dai processi P, P1 e P2 dopo l'evento g).

SOLUZIONE

PROCESSO	*read(fd[0] &buffer, NCaratteri)*	*write(fd[1], &buffer, NCaratteri)*
P	Eseguibile? SI	Eseguibile? NO
P1	Eseguibile? SI	Eseguibile? NO
P2	Eseguibile? SI	Eseguibile? SI

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 4/11/2009 CORSO |A| |B|

ESERCIZIO 8 (2 punti)

Si consideri un sistema nel quale sono definiti il semaforo *sem* e i processi P1, P2 e P3. Lo scheduling dei processi avviene con una politica Round Robin, e la politica applicata al semaforo è la FIFO.
Al tempo *t* il semaforo *sem* ha valore 0, e la sua coda contiene P3.
Allo stesso tempo, il processo P1 è in esecuzione il processo P2 è pronto.
Come si modificano il semaforo *sem* e la CodaPronti e quale processo è in esecuzione se si verifica la seguente sequenza di eventi:

a) P1 esegue *signal (sem)*
b) il processo in esecuzione esegue *wait(sem);*
c) arriva un'interruzione del timer
d) il processo in esecuzione esegue *signal(sem);*

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti	Valore di *sem*	Coda di *sem*
a	P1 esegue *signal (sem)*	P1	P2 -> P3	0	-
b	il processo in esecuzione esegue *wait(sem);*	P2	P3	0	P1
c	arriva un'interruzione del timer	P3	P2	0	P1
d	il processo in esecuzione esegue *signal(sem);*	P3	P2 -> P1	0	-

ESERCIZIO 9 (2 punti)

In un sistema con thread *realizzati a livello utente*, dove l'unità schedulabile dal kernel è il processo, sono presenti i processi P1 con thread T11 e T12 e P2 con thread T21 e T22. I processi sono schedulati con la politica round robin, e ogni processo gestisce i suoi thread con politica FIFO.
Immediatamente prima del tempo *t* è in esecuzione il processo P1 e il processo P2 è pronto. Nel processo P1 è in esecuzione il thread T11 e T12 è pronto, mentre nel processo P2 il thread T21 è in esecuzione (andrà realmente in esecuzione quando P2 passerà in stato esecuzione) e il thread T22 è pronto.

Quale thread va in esecuzione se al tempo t si verificano (in alternativa) i seguenti eventi:

(a)	T11 esegue la primitiva wait() su un semaforo con valore 0
(b)	T11 esegue la primitiva signal() su un semaforo con valore 0
(c)	T11 invoca la funzione ThreadYield()
(d)	Il timer di sistema genera un'interruzione

SOLUZIONE

	Evento che si verifica al tempo t	Thread in esecuzione dopo il tempo t
(a)	T11 esegue la primitiva wait() su un semaforo con valore 0	T21
(b)	T11 esegue la primitiva signal() su un semaforo con valore 0	T11
(c)	T11 invoca la funzione ThreadYield()	T12
(d)	Il timer di sistema genera un'interruzione	T21

SISTEMI OPERATIVI E LABORATORIO, CORSI A e B Prova del 4/11/2009 CORSO |A| |B|

ESERCIZIO 10 (2 punti)

Per ognuna delle seguenti strutture dati di Unix indicare se è swappable oppure no:
signal handler array, pending signal bitmap, text structure, process structure, user structure, contesto del processo.

Per ognuna delle seguenti strutture dati di Unix indicare se è swappable oppure no:

a)	signal handler array
b)	pending signal bitmap
c)	text structure
d)	process structure
e)	user structure
f)	Contesto del processo

SOLUZIONE

		Swappable? [SI/NO]
a)	signal handler array	SI
b)	pending signal bitmap	NO
c)	text structure	NO
d)	process structure	NO
e)	user structure	SI
f)	Contesto del processo	SI