Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa	**Anna**	**Antola**	**prof.**	**Giuseppe**	**Pelagatti**
prof.	**Luca**	**Breveglieri**	**prof.ssa**	**Donatella**	**Sciuto**
prof.	**Roberto**	**Negrini**	**prof.ssa**	**Cristina**	**Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 22 gennaio 2019

Cognome_____ **Nome** _____

Matricola_____ **Firma**_____

Istruzioni

- Si scriva solo negli spazi previsti nel testo della prova e non si separino i fogli.
- Per la minuta si utilizzino le pagine bianche inserite in fondo al fascicolo distribuito con il testo della prova. I fogli di minuta se staccati vanno consegnati intestandoli con nome e cognome.
- È vietato portare con sé libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque fosse trovato in possesso di documentazione relativa al corso – anche se non strettamente attinente alle domande proposte – vedrà annullata la propria prova.
- Non è possibile lasciare l'aula conservando il tema della prova in corso.
- Tempo a disposizione 2 h : 00 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (4 punti) _____

esercizio 2 (5 punti) _____

esercizio 3 (6 punti) _____

esercizio 4 (1 punti) _____

voto finale: (16 punti) _____

CON SOLUZIONI (in corsivo)

esercizio n. 1 – programmazione concorrente

Si consideri il programma C seguente (gli "`#include`" e le inizializzazioni dei *mutex* sono omessi, come anche il prefisso `pthread` delle funzioni di libreria NPTL):

```
pthread_mutex_t near, far
sem_t glance
int global = 0
```

```
void * me (void * arg) {
   mutex_lock (&far)
   sem_wait (&glance)
   mutex_lock (&near)
   global = 1                                   /* statement A */
   mutex_unlock (&near)
   sem_post (&glance)                           /* statement B */
   mutex_unlock (&far)
   return (void * 9)
} /* end me */
```

```
void * you (void * arg) {
   mutex_lock (&far)
   global = 2
   sem_post (&glance)
   mutex_unlock (&far)                          /* statement C */
   mutex_lock (&near)
   sem_wait (&glance)
   global = 3
   mutex_unlock (&near)
   return NULL
} /* end you */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&glance, 0, 0)
   create (&th_1, NULL, me, NULL)
   create (&th_2, NULL, you, NULL)
   join (th_1, &global)                         /* statement D */
   join (th_2, NULL)
   return
} /* end main */
```

Si completi la tabella qui sotto **indicando lo stato di esistenza del *thread*** nell'istante di tempo specificato da ciascuna condizione, così: se il *thread* **esiste**, si scriva ESISTE; se **non esiste**, si scriva NON ESISTE; e se può essere **esistente** o **inesistente**, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede lo stato che il *thread* assume tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	***thread***	
	th_1 – me	**th_2 – you**
subito dopo stat. **A**	*ESISTE*	*ESISTE*
subito dopo stat. **B**	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **C**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **D**	*NON ESISTE*	*PUÒ ESISTERE*

Si completi la tabella qui sotto, **indicando i valori delle variabili globali** (sempre esistenti) nell'istante di tempo specificato da ciascuna condizione. Il **valore** della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	**variabili globali**	
	glance	*global*
subito dopo stat. **A**	*0*	*1*
subito dopo stat. **B**	*0 / 1*	*1 / 3*
subito dopo stat. **C**	*0 / 1*	*1 / 2 / 9*
subito dopo stat. **D**	*0 / 1*	*3 / 9*

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in (almeno) **due casi diversi**. Si chiede di precisare il comportamento dei thread (in due casi), indicando gli statement dove avvengono i blocchi e i possibili valori della variabile *global*:

caso	**th_1 – me**	**th_2 – you**	*global*
1	***wait glance***	***lock far***	***0***
2	***lock near***	***wait glance***	***2***
3	***wait glance***	**-**	***3***

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma prova.c
main ( ) {
   pid1 = fork ( )
   nanosleep (7)
   if (pid1 == 0) {          // codice eseguito dal figlio Q
      execl ("/acso/prog_x", "prog_x", NULL)
      exit (-1)
   } else {
      write (stdout, msg, 25)
      pid1 = wait (&status)
   } /* if */
   exit (0)
}  /* prova */
```

```
// programma prog_x.c
mutex_t BLACK = PTHREAD_MUTEX_INITIALIZER
sem_t NEW, OLD
```

```
void * LEFT (void * arg) {
  mutex_lock (&BLACK)
  sem_post (&NEW)
  mutex_unlock (&BLACK)
  sem_wait (&OLD)
  return NULL
}  /* LEFT */
```

```
void * RIGHT (void * arg) {
  sem_wait (&NEW)
  mutex_lock (&BLACK)
  sem_wait (&OLD)
  sem_post (&OLD)
  mutex_unlock (&BLACK)
  return NULL
}  /* RIGHT */
```

```
main ( ) {  // codice eseguito da Q
    pthread_t TH_1, TH_2
    sem_init (&NEW, 0, 0)
    sem_init (&OLD, 0, 1)
    create (&TH_2, NULL, RIGHT, NULL)
    create (&TH_1, NULL, LEFT, NULL)
    join (TH_2, NULL)
    join (TH_1, NULL)
    exit (1)
}  /* main */
```

```
// programma esempio.c
main ( ) {
    fd = open ("/acso/dati", O_RDWR)
    write (fd, vett, 512)
    exit (1)
}  /* main */
```

Un processo ***P*** esegue il programma ***prova*** e crea un figlio ***Q*** che esegue una mutazione di codice (programma ***prog_x***). La mutazione di codice va a buon fine e vengono creati i thread ***TH1*** e ***TH2***. Un processo ***S*** esegue il programma ***esempio***.

Si simuli l'esecuzione dei processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale e dagli eventi indicati. Si completi la tabella riportando quanto segue:

- ⟨ *PID*, *TGID* ⟩ di ciascun processo che viene creato
- ⟨ *identificativo del processo-chiamata di sistema / libreria* ⟩ nella prima colonna, dove necessario e in funzione del codice proposto
- in ciascuna riga lo stato dei processi **al termine dell'evento o della chiamata associata alla riga stessa**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

TABELLA DA COMPILARE (numero di colonne non significativo)

identificativo simbolico del processo		***IDLE***	***S***	***P***	**Q**	*TH2*	*TH_1*		
evento oppure *processo-chiamata*	*PID*	**1**	**2**	*3*	*4*	*5*	*6*		
	TGID	**1**	**2**	*3*	*4*	*4*	*4*		
Q – nanosleep (7)	**0**	**exec**	**A open**	**A write su stdout**	*A nanosleep*	*NE*	*NE*		
interrupt da ***DMA_in***, tutti i blocchi richiesti trasferiti	1	*pronto*	*exec*	*A write*	*A nanosleep*	*NE*	*NE*		
interrupt da *RT_clock* e scadenza timer di *nanosleep*	2	*pronto*	*pronto*	*A write*	*exec*	*NE*	*NE*		
Q – execl	3	*pronto*	*pronto*	*A write*	*exec*	*NE*	*NE*		
Q – create TH_2	4	*pronto*	*pronto*	*A write*	*exec*	*pronto*	*NE*		
interrupt da *RT_clock* e scadenza *quanto di tempo*	5	pronto	exec	A write	pronto	pronto	NE		
S – write	6	*pronto*	*A write*	*A write*	*pronto*	*exec*	*NE*		
TH2 – sem_wait NEW	7	*pronto*	*A write*	*A write*	*exec*	*A s_wait NEW*	*NE*		
Q – create TH_1	8	*pronto*	*A write*	*A write*	*exec*	*A s_wait NEW*	*pronto*		
interrupt da RT_clock e scadenza quanto di tempo	9	*pronto*	*A write*	*A write*	*pronto*	*A s_wait NEW*	*exec*		
TH1 – lock BLACK	10	*pronto*	*A write*	*A write*	*pronto*	*A s_wait NEW*	*exec*		
TH1 – post NEW	11	pronto	A write	A write	pronto	exec	pronto		
TH2 – lock BLACK	12	*pronto*	*A write*	*A write*	*exec*	*A lock BLACK*	*pronto*		
Q – join TH2	13	*pronto*	*A write*	*A write*	*A join*	*A lock BLACK*	*esec*		
25 interrupt da *stdout,* tutti i caratteri trasferiti	14	*pronto*	*A write*	*exec*	*A join*	*A lock BLACK*	*pronto*		

seconda parte – scheduling dei processi

Si consideri uno scheduler CFS con **3 task** caratterizzato da queste condizioni iniziali (**da completare**):

CONDIZIONI INIZIALI (da completare)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*4*	t1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	t1	1	*0,25*	*1,5*	*1*	10	100
RB	t3	1	*0,25*	*1,5*	*1*	20	100
	t2	2	*0,5*	*3*	*0,5*	30	101

Durante l'esecuzione dei task si verificano i seguenti eventi:

`Events of task t1: WAIT at 0,5; WAKEUP after 0,5;`
`WAIT at 0,5; WAKEUP after 1,0;`

Attenzione: la seconda wait si verifica **0,5ms <u>di esecuzione</u>** dopo la prima.

Simulare l'evoluzione del sistema per **5 eventi** riempiendo le seguenti tabelle (per indicare le condizioni di rescheduling delle due makeup, utilizzare la tabella finale):

EVENTO 1	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
	0,5	*wait*	*t1*	*true*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	2	*6*	*3*	*t3*	*100*	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t3*	*1*	*0,33*	*2*	*1*	*20*	*100*
RB	*t2*	*2*	*0,67*	*4*	*0,5*	*30*	*101*
WAITING	*t1*	*1*				*10,5*	*100,5*

EVENTO 2	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
	1	*w_up*	*t3*	*false*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	3	*6*	*4*	*t3*	*100,5*	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t3*	*1*	*0,25*	*1,5*	*1*	*20,5*	*100,5*
RB	*t1*	*1*	*0,25*	*1,5*	*1*	*10,5*	*100,5*
	t2	*2*	*0,5*	*3*	*0,5*	*30*	*101*
WAITING							

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
		2	*Q_scade*	*t3*	*true*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	*6*	*4*	*t1*	*100,5*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t1*	*1*	*0,25*	*1,5*	*1*	*10,5*	*100,5*
RB	*t2*	*2*	*0,5*	*3*	*0,5*	*30*	*101*
	t3	*1*	*0,25*	*1,5*	*1*	*21,5*	*101,5*
WAITING							

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
		2,5	*wait*	*t1*	*true*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	*6*	*3*	*t2*	*101*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t2*	*2*	*0,67*	*4*	*0,5*	*30*	*101*
RB	*t3*	*1*	*0,33*	*2*	*1*	*21,5*	*101,5*
WAITING	*t1*	*1*				*11*	*101*

EVENTO 5		TIME	TYPE	CONTEXT	RESCHED		
		3,5	*w_up*	*t2*	*true*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	*6*	*4*	*t1*	*101,5*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t1*	*1*	*0,25*	*1,5*	*1*	*11*	*101*
RB	*t3*	*1*	*0,25*	*1,5*	*1*	*21,5*	*101,5*
	t2	*2*	*0,5*	*3*	*0,5*	*31*	*101,5*
WAITING							

Condizioni di rescheduling al wake_up del task t1:

1° wake_up: *100,5 + 0,25 < 100,5 ? → false*
2° wake_up: *101 + 0,25 < 101,5 ? → true*

esercizio n. 3 – memoria e file system

prima parte – gestione dello spazio di memoria

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = **2** **MINFREE** = **1**

Situazione iniziale (esistono due processi P e Q)

```
PROCESSO: P  ********************************************************
    VMA : C  000000400,  1 , R  ,  P  ,  M  , <X, 0>
          S  000000600,  2 , W  ,  P  ,  M  , <X, 1>
          D  000000602,  2 , W  ,  P  ,  A  , <-1,0>
          M0 000001000,  4 , W  ,  S  ,  M  , <F, 0>
          M1 000002000,  4 , W  ,  P  ,  M  , <G, 0>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
    PT: <c0 :1 R> <s0 :- -> <s1 :- ->  <d0 :- -> <d1 :- -> <p0 :2 R>
        <p1 :4 R> <p2 :- -> <m00:5 W>  <m01:- -> <m02:- -> <m03:- ->
        <m10:3 W> <m11:- -> <m12:- ->  <m13:- ->
process P - NPV of PC and SP:  c0, p1

PROCESSO: Q  *** (i dettagli di questo processo non interessano) ****

____MEMORIA FISICA____(pagine libere: 2)_____________________________
      00 : <ZP>                ||  01 : Pc0 / Qc0 / <X,0>   ||
      02 : Pp0 / Qp0           ||  03 : Pm10                ||
      04 : Pp1                 ||  05 : Pm00 / <F,0>        ||
      06 : ----                ||  07 : ----                ||

____STATO del TLB_____________________________________________________
      Pc0  : 01 - 0: 1:   ||  Pp0  : 02 -  1: 0:   ||
      Pm10 : 03 - 1: 0:   ||  Pm00 : 05 -  1: 0:   ||
      Pp1  : 04 - 0: 0:   ||        -----          ||

SWAP FILE:    Qp1  , Pp1  , ----, ----, ----, ----, ----, ----,
LRU ACTIVE:   PC0,
LRU INACTIVE: pp1, pm10, pm00, pp0, qp0, qc0,
```

Domanda 1): Indicare i numeri delle pagine fisiche che appartengono alla Swap Cache:

pagina fisica 04 (il cui contenuto Pp1 è rappresentato in uno slot dello swap file)

evento 1: write (Pp0, Pp1)

(la pagina Pp0 causa COW e viene riallocata, la pagina Pp1 viene scritta e la pagina fisica 04 non occupa più lo swap file; la pagina Qp0 era prima condivisa con Pp0, che è dirty nel TLB, e dopo la separazione da Pp0 è marcata dirty nella PT e mostrata dirty nel descrittore di memoria fisica)

PT del processo: **P**				
M00: *5 W*	M10: *3 W*	P0: *6 W*	P1: *4 W*	P2: - -

MEMORIA FISICA	
00: <ZP>	01: *Pc0 / Qc0 / <X, 0>*
02: *Qp0 D*	03: *Pm10*
04: *Pp1*	05: *Pm00 / <F, 0>*
06: *Pp0*	07:

SWAP FILE	
s0: *Qp1*	s1:
s2:	s3:

evento 2: read (Ps0), write (Ps0)

(viene invocato PFRA che libera le pagine fisiche 02 e 06, poi si carica la pagina Ps0 in 02, poi la scrittura di Ps0 causa COW e Ps0 viene riallocata)

MEMORIA FISICA	
00: <ZP>	01: *Pc0 / Qc0 / <X, 0>*
02: *<X, 1>*	03: *Pm10*
04: *Pp1*	05: *Pm00 / <F, 0>*
06: *Ps0*	07:

SWAP FILE	
s0: *Qp1*	s1: *Qp0*
s2: *Pp0*	s3:

evento 3: write (Pp2, Pp3)

(la prima write attiva PFRA che libera le pagine fisiche 02 e 05; la pagina Pm00 non va in swap file, perché la VMA Pm0 è shared)

MEMORIA FISICA	
00: <ZP>	01: *Pc0 / Qc0 / <X, 0>*
02: *Pp2*	03: *Pm10*
04: *Pp1*	05: *Pp3*
06: *Ps0*	07:

SWAP FILE	
s0: *Qp1*	s1: *Qp0*
s2: *Pp0*	s3:
s4:	s5:

evento 4: write (Pp4)

(la write attiva PFRA che libera le pagine fisiche 03 e 04, le quali vanno entrambe in swap file, la 03 perché la VMA Pm1 è private)

MEMORIA FISICA	
00: <ZP>	01: *Pc0 / Qc0 / <X, 0>*

02: *Pp2*	03: *Pp4*
04:	05: *Pp3*
06: *Ps0*	07:

SWAP FILE	
s0: *Qp1*	s1: *Qp0*
s2: *Pp0*	s3: *Pm10*
s4: *Pp1*	s5:

seconda parte – memoria e file system

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = **2** **MINFREE** = **1**

Si consideri la seguente **situazione iniziale**:

```
____MEMORIA FISICA____(pagine libere: 5)_____________________________
      00 : <ZP>                ||  01 : Pc2 / <X, 2>        ||
      02 : Pp0                 ||  03 : ----                ||
      04 : ----                ||  05 : ----                ||
      06 : ----                ||  07 : ----                ||
____STATO del TLB_____________________________________________________
      Pc2 : 01 -  0: 1:   ||   Pp0 : 02 -  1: 1:   ||
            -----         ||         -----         ||
```

Per ognuno dei seguenti eventi compilare le Tabelle richieste con i dati relativi al contenuto della memoria fisica, delle variabili del FS relative al file F e al numero di accessi a disco effettuati in lettura e in scrittura.

È in esecuzione il processo **P**.

ATTENZIONE: il numero di pagine lette o scritte di un file è cumulativo, quindi è la somma delle pagine lette o scritte su quel file da tutti gli eventi precedenti oltre a quello considerato. Si ricorda che "close" scrive le pagine dirty di un file solo se fcount diventa = 0.

Eventi 1 e 2: fd = open ("F"), write (fd, 8000)

MEMORIA FISICA	
00: *<ZP>*	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<F, 0> D*
04: *<F, 1> D*	05: *----*
06: *----*	07: *----*

	f_pos	**f_count**	**numero pagine lette**	**numero pagine scritte**
file F	*8000*	*1*	*2*	*0*

Eventi 3: close (fd)

MEMORIA FISICA	
00: *<ZP>*	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<F, 0>*
04: *<F, 1>*	05: *----*
06: *----*	07: *----*

	f_pos	**f_count**	**numero pagine lette**	**numero pagine scritte**
file F	*--*	*0*	*2*	*2*

Eventi 4 e 5: fork (“Q”), context switch (“Q”)

MEMORIA FISICA	
00: *<ZP>*	01: *Pc2 / Qc2 / <X, 2>*
02: *Qp0 D*	03: *<F, 0>*
04: *<F, 1>*	05: *Pp0 D*
06: *----*	07: *----*

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	*--*	*0*	*2*	*2*

Evento 6: fd = open (“F”), read (7000)

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	*7000*	*1*	*2*	*2*

Evento 7: write (fd, 10000)

MEMORIA FISICA	
00: *<ZP>*	01: *Pc2 / Qc2 / <X, 2>*
02: *Qp0 D*	03: *<F, 3> D*
04: *<F, 4> D*	05: *Pp0 D*
06: *<F, 2> D*	07: *----*

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	*17000*	*1*	*5*	*3*

Evento 8: close (fd)

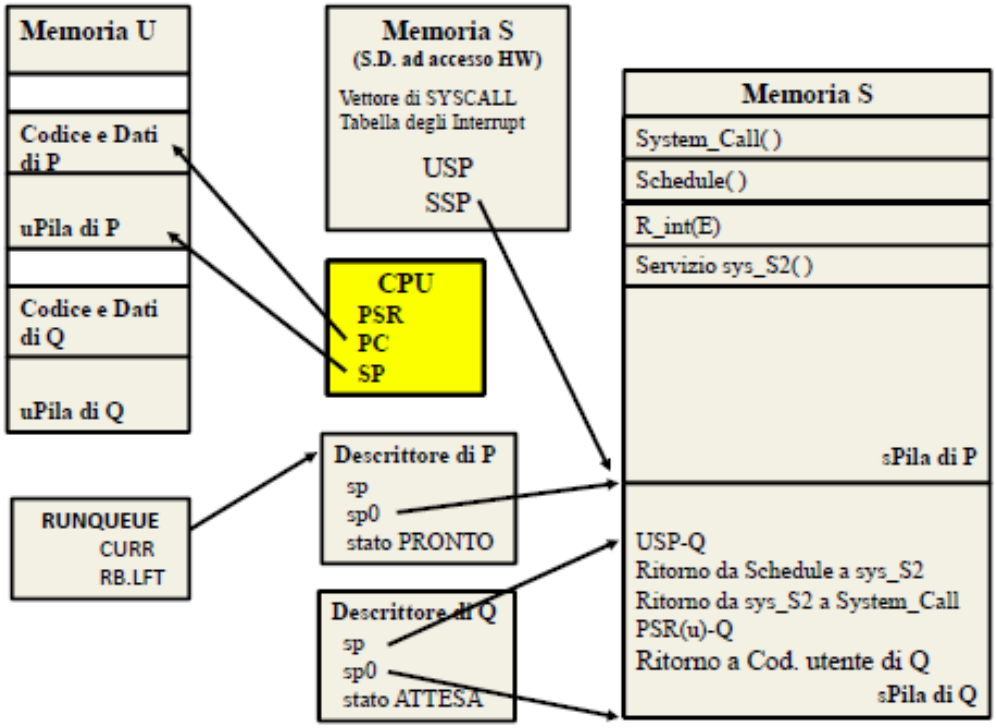
	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	*----*	*0*	*5*	*6*

esercizio n. 4 – moduli, pila e strutture dati HW

Si considerino due processi P e Q. La situazione iniziale considerata è la seguente:

- il processo P è in esecuzione in modo U
- il processo Q è in stato di ATTESA dell'evento **E**. La coda RB è da considerarsi vuota (si ricorda che IDLE non appartiene alla RB)

La figura sotto riportata e i valori nella tabella successiva descrivono compiutamente, ai fini dell'esercizio, il contesto di P e il contesto di Q.



I valori della situazione iniziale di interesse sono i seguenti, dove gli indirizzi rappresentati simbolicamente (X, W, .. A, B ..) sono indirizzi di parola.

Processo P		
PC	X	// è all'interno del codice utente
SP	W	// referenzia la uPila
SSP	Z	// base di sPila
USP	n.s.	// valore non significativo
Descrittore di P.stato	PRONTO	
Processo Q		
USP-Q	A	
Descrittore di Q.sp	B	
Descrittore di Q.sp0	C	
Descrittore di Q.stato	ATTESA	
RUNQUEUE		
CURR	P	
RB.LFT	NULL	

Si consideri il seguente **evento**:

si verifica l'interruzione associata all'evento E. La routine di risposta all'interrupt – *R_int(E)* – risveglia il processo Q che risulta avere un diritto di esecuzione maggiore di quello di P. All'interno di *R_int(E)* viene quindi invocato *schedule( )* per il *context switch*. Si supponga che l'invocazione di *schedule* avvenga all'indirizzo Y + 9, dove Y è l'indirizzo iniziale della routine di risposta all'interrupt.

Domanda 1 – **salvataggio** del contesto di **P** eseguito durante il *context switch*

Completare la tabella seguente con i valori assunti dagli elementi subito dopo il salvataggio del contesto di P, ma prima della commutazione della pila di sistema.

Processo P		
PC		// non di interesse
SP	*Z – 3*	
s_Pila di P a (Z - 3)	*W (= USP-P)*	
s_Pila di P a (Z - 2)	*Y + 9*	
s_Pila di P a (Z - 1)	*PSR(u)-P*	
s_Pila di P a (Z)	*X + 1*	
SSP	*Z*	
USP	*W*	
Descrittore di P.sp	*Z – 3*	
Descrittore di P.sp0	*Z*	
Descrittore di P.stato	*PRONTO*	

Domanda 2 – **caricamento** del contesto di **Q** eseguito durante il *context switch*

Completare la tabella seguente con i valori assunti dagli elementi subito prima del ritorno da *schedule* al servizio di sistema *sys_S2*.

Processo Q		
PC		// subito prima del ritorno da *schedule*
SP	*B + 1*	
SSP	*C*	
USP	*A*	
Descrittore di Q.sp		// non di interesse
Descrittore di Q.sp0	*C*	
Descrittore di Q.stato	*PRONTO*	
RUNQUEUE		
CURR	*Q*	
RB.LFT	*P*	

spazio libero per continuazione o brutta copia