



Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa Anna Antola

prof. Luca Breveglieri

prof. Roberto Negrini

prof. Giuseppe Pelagatti

prof.ssa Donatella Sciuto

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) _____

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	<i>thread</i>	
	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	
	<i>glance</i>	<i>global</i>
subito dopo stat. A	0	1
subito dopo stat. B	1 - 0	1 - 3
subito dopo stat. C	1 - 0	1 - 2 - 9
subito dopo stat. D	1 - 0	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	<i>global</i>
1	sem_wait(&glance)	mutex_lock(&far)	0
2	sem_wait(&glance)	-	3
3	mutex_lock(&near)	sem_wait(&glance)	2

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:

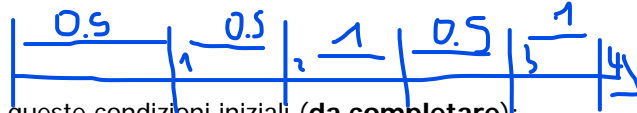
- $\langle PID, TGID \rangle$ di ciascun processo che viene creato
- $\langle \text{identificativo del processo-chiamata di sistema} / \text{libreria} \rangle$ nella prima colonna, dove necessario e in funzione del codice proposto
- in ciascuna riga lo stato dei processi **al termine dell'evento o della 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		<i>IDLE</i>	<i>S</i>	<i>P</i>	<i>Q</i>	TH2	TH1		
evento oppure processo-chiamata	<i>PID</i>	1	2	3	4	5	6		
	<i>TGID</i>	1	2	3	4	4	4		
Q – nanosleep (7)	0	exec	A open	A write su stdout	A nano	NE	NE		
interrupt da <i>DMA_in</i> , tutti i blocchi richiesti trasferiti	1	pronto	ESEC	A write	A nano	NE	NE		
interrupt da <i>RT_clock</i> e scadenza timer di <i>nanosleep</i>	2	pronto	pronto	A write	ESEC	NE	NE		
Q - execl	3	pronto	pronto	A write	ESEC	NE	NE		
Q - pthread_create(TH2)	4	pronto	pronto	A write	ESEC	pronto	NE		
Interrupt da <i>RT_clock</i> e scadenza del q.d.t	5	pronto	exec	A write	pronto	pronto	NE		
S - write	6	pronto	A write	A write	pronto	ESEC	NE		
TH2 - sem_wait(&NEW)	7	pronto	A write	A write	ESEC	A sem	NE		
Q - pthread_create(TH1)	8	pronto	A write	A write	ESEC	A sem	pronto		
interrupt da <i>RT_clock</i> e scadenza quanto di tempo	9	pronto	A write	A write	pronto	A sem	ESEC		
TH1 - mutex_lock(&BLACK)	10	pronto	A write	A write	pronto	A sem	ESEC		
TH1 - sem_post(&NEW)	11	pronto	A write	A write	pronto	exec	pronto		
TH2 - mutex_lock(&BLACK)	12	pronto	A write	A write	ESEC	A lock	pronto		
Q - pthread_join(TH2)	13	pronto	A write	A write	A join	A lock	ESEC		
25 interrupt da <i>stdout</i> , tutti i caratteri trasferiti	14	pronto	A write	ESEC	A join	A lock	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; ¹ WAKEUP after 0,5; ²
 WAIT at 0,5; ³ WAKEUP after 1,0; ⁴

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

Simulare l'evoluzione del sistema per **5 eventi** riempiendo le seguenti tabelle (per indicare le condizioni di rescheduling delle due wakeup, 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	100.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	1/3	2	1	20	100
RB	T2	2	2/3	4	0.5	30	101
WAITING	T1	1				10.5	100.5

$$T1 \rightarrow VRT = 100 + 0.5 * 1 = 100.5$$

EVENTO 2		TIME	TYPE	CONTEXT	RESCHED		
		1	wake 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							

$$T3 \rightarrow VRT = 100 + 0.5 * 1 = 100.5$$

$$T1 \rightarrow VRT = 100.5$$

$$RESCHED: 100.5 + 1 * 0.25 = 100.75 > 100.75 \Rightarrow FALSE$$

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED	T3 -> VRT = 100.5 + 1 * 1 = 101.5	
		2	s. QdT	T3	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	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	T1 -> VRT = 100.5 + 0.5 * 1 = 101	
		2.5	wait	T1	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	2	T2	101.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	2/3	4	0.5	30	101
RB	T3	1	1/3	2	1	21.5	101.5
WAITING	T1	1				11	101

EVENTO 5		TIME	TYPE	CONTEXT	RESCHED	T2 -> VRT = 101 + 1 * 0.5 = 101.5 T1 -> VRT = 101	
		3.5	wake 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	T2	2	0.5	3	0.5	31	101.5
	T3	1	0.25	1.5	1	21.5	101.5
WAITING							

Condizioni di rescheduling al wake_up del task t1:

1° wake_up:	RESCHED: $100.5 + 1 * 0.25 = 100.75 > 100.75 \Rightarrow \text{FALSE}$
2° wake_up:	RESCHED: $101 + 1 * 0.25 = 101.25 < 101.5 \Rightarrow \text{TRUE}$

Nel caso di una wake up in resched non è asserito, si considera come quanto di tempo quello che il task correntemente in esecuzione aveva prima della wake up stessa

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:

evento 1: write (Pp0, Pp1)

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:

LRU INACTIVE: pp1, pm10, pm00, qc0

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

LRU ACTIVE: PS0, PC0

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:

LRU INACTIVE: pp1, pm10, qc0

evento 3: write (Pp2, Pp3)

LRU ACTIVE: PP3, PP2, PS0, PC0

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: Pp1
s2: Pp0	s3:
s4:	s5:

LRU INACTIVE: qc0

evento 4: write (Pp4)

LRU ACTIVE: PP4, PP3, PP2, PS0, PC0

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / < X, 0>
02: Ps0	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

0 ---- 4096 ---- 8192 ---- 12288 ---- 16384 ---- 20480 ---- 24576
0 1 2 3 4 5

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) $7000 + 10000 = 17000$

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)

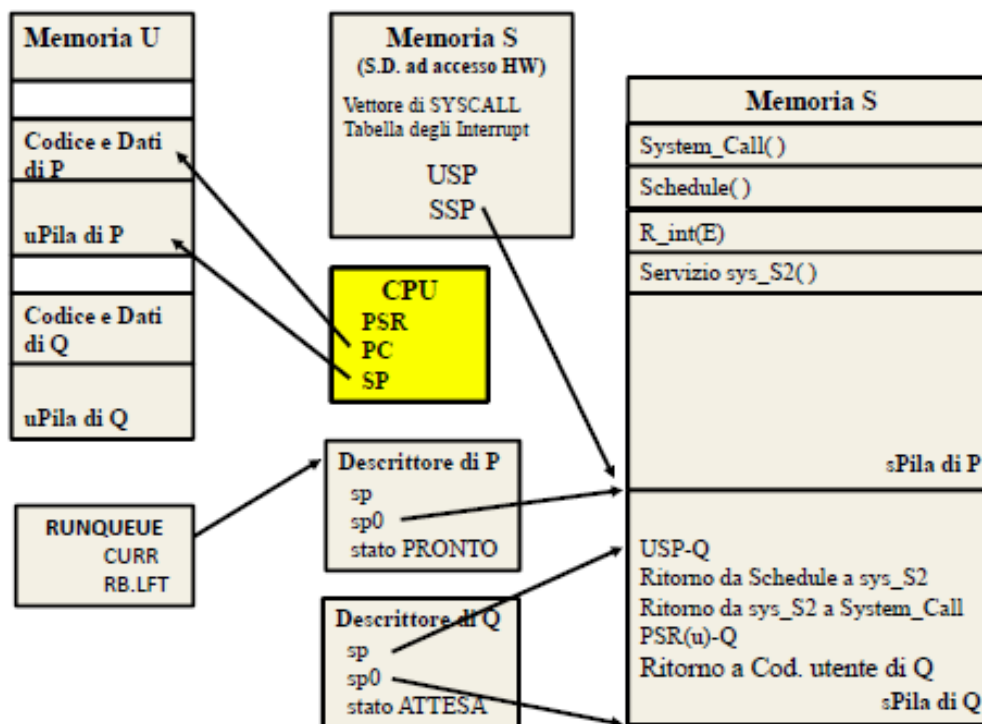
	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	----	0	5	6

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
SP	W
SSP	Z
USP	n.s.
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	
SP	
s_Pila di P a (Z - 3)	
s_Pila di P a (Z - 2)	
s_Pila di P a (Z - 1)	
s_Pila di P a (Z)	
SSP	
USP	
Descrittore di P.sp	
Descrittore di P.sp0	
Descrittore di P.stato	

// non di interesse

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	
SP	
SSP	
USP	
Descrittore di Q.sp	
Descrittore di Q.sp0	
Descrittore di Q.stato	
RUNQUEUE	
CURR	
RB.LFT	

// subito prima del ritorno da *schedule*

// non di interesse

