



**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 mercoledì 8 febbraio 2017**

**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 **1 h : 45 m**

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

**esercizio 1 (4 punti)** \_\_\_\_\_

**esercizio 2 (4 punti)** \_\_\_\_\_

**esercizio 3 (2 punti)** \_\_\_\_\_

**esercizio 4 (4 punti)** \_\_\_\_\_

**esercizio 5 (2 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):

```
pthread_mutex_t line, circle
sem_t point
int global = 0
```

---

```
void * ruler (void * arg) {
    pthread_mutex_lock (&line)
    sem_post (&point) /* statement A */
    pthread_mutex_unlock (&line)
    ... /* statement non rilevanti */
    pthread_mutex_lock (&circle)
    global = 1 /* statement B */
    sem_wait (&point)
    pthread_mutex_unlock (&circle)
    return NULL
} /* end ruler */
```

---

```
void * compass (void * arg) {
    pthread_mutex_lock (&line)
    global = 2
    sem_wait (&point)
    pthread_mutex_lock (&circle)
    sem_post (&point) /* statement C */
    global = 3
    pthread_mutex_unlock (&circle)
    pthread_mutex_unlock (&line)
    return NULL
} /* end compass */
```

---

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&point, 0, 0)
    pthread_create (&th_2, NULL, compass, NULL)
    pthread_create (&th_1, NULL, ruler, NULL)
    pthread_join (th_2, NULL) /* statement D */
    pthread_join (th_1, 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 – ruler	th_2 – compass
subito dopo stat. <b>A</b>	Esiste	Esiste
subito dopo stat. <b>B</b>	Esiste	Può esistere
subito dopo stat. <b>C</b>	Esiste	Esiste
subito dopo stat. <b>D</b>	Può esistere	Non esiste

**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>point</i>	<i>global</i>
subito dopo stat. <b>A</b>	1	0
subito dopo stat. <b>B</b>	1 / 0	1 / 2
subito dopo stat. <b>D</b>	1 / 0	1 / 3

**Il sistema può andare in stallo (*deadlock*)**, con uno o più *thread* che si bloccano, in **tre casi diversi** (con *deadlock* si intende anche un blocco dovuto a un solo *thread* che non potrà mai proseguire). Si indichino gli statement dove avvengono i blocchi:

caso	th_1 – ruler	th_2 – compass
1	pthread_mutex_lock(&line)	sem_wait(&point)
2	sem_wait(&point) (II)	pthread_mutex_lock(&circle)
3	X	sem_wait(&point)

## esercizio n. 2 – gestione dello stato dei processi

// programma <b>prova.c</b>	
main ( ) {	
pid1 = fork ( )	
fd = open ("/acso/esame", O_RDWR)	
if (pid1 == 0) {                   // codice eseguito solo da Q	
write (fd, vett, 50)	
exit (1)	
} else {	
pid2 = fork ( )	
if (pid2 == 0) {           // codice eseguito solo da R	
read (fd, vett, 5)	
exit (2)	
} else {	
nanosleep (1)	
} /* if */	
exit (0)	
} /* prova */	

// programma <b>prog_x.c</b>	
pthread_mutex_t GATE = PTHREAD_MUTEX_INITIALIZER	
sem_t CHECK	
void * SINGLE (void * arg) {	void * SEQUENCE (void * arg) {
(1) sem_wait (&CHECK)	for (num = 1; num <= 3; num++) {
(2) pthread_mutex_lock (&GATE)	(5) pthread_mutex_lock (&GATE)
(3) sem_wait (&CHECK)	(6) sem_post (&CHECK)
(4) pthread_mutex_unlock (&GATE)	(7) pthread_mutex_unlock (&GATE)
return NULL	} /* end_for */
} /* SINGLE */	return NULL
	} /* SEQUENCE */

main ( ) { // codice eseguito da <b>S</b>	
pthread_t TH_1, TH_2	
sem_init (&CHECK, 0, 0)	
pthread_create (&TH_1, NULL, SINGLE, (void *) 1)	
pthread_create (&TH_2, NULL, SEQUENCE, NULL)	
(8) pthread_join (TH_2, NULL)	
(9) pthread_join (TH_1, NULL)	
exit (1)	
} /* main */	

Un processo **P** esegue il programma **prova**. Un processo **S** esegue il programma **prog\_x**. Il processo **P** crea i processi **Q** e **R**. Il processo **S** crea i thread **TH1** e **TH2**.

Si simuli l'esecuzione dei processi (fino a **udt = 150**) così come risulta dal codice dato, dagli eventi indicati e ipotizzando che il processo **S** non abbia ancora eseguito la prima *pthread create*. **Si completi** la tabella riportando quanto segue:

- $\langle PID, TGID \rangle$  di ciascun processo che viene creato
- $\langle \text{identificativo del processo-chiamata di sistema / libreria} \rangle$  nella prima colonna, dove necessario e in funzione del codice proposto
- in ciascuna riga lo stato dei processi **al termine del tempo indicato**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

**TABELLA DA COMPILARE** (numero di colonne non significativo)

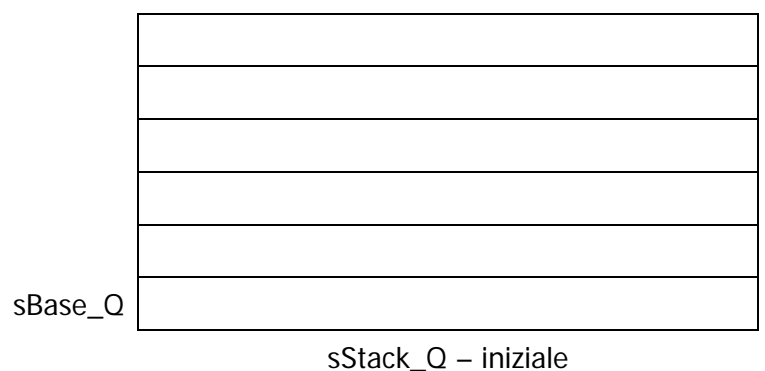
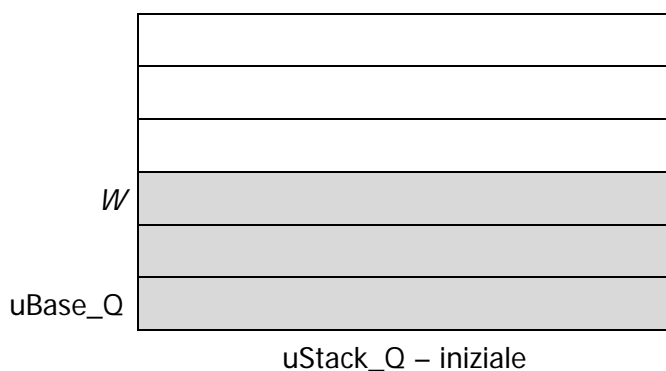
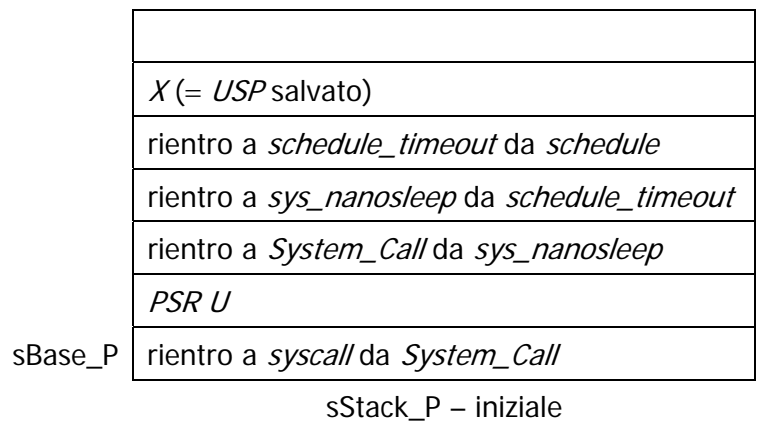
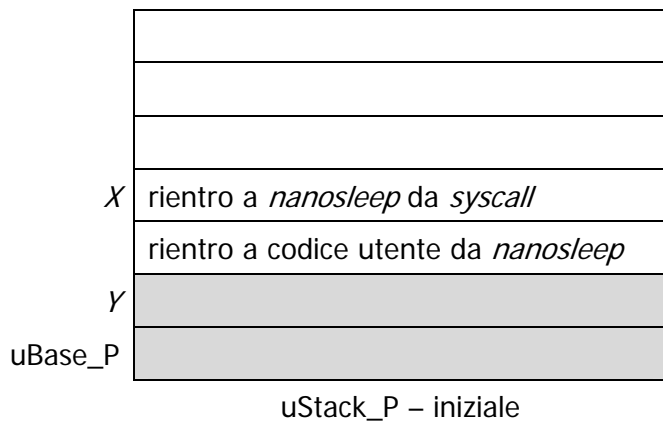
<i>identificativo simbolico del processo</i>		<i>IDLE</i>	<i>S</i>	<i>P</i>	<i>Q</i>	<i>TH1</i>	<i>R</i>	<i>TH2</i>	
<i>evento processo-chiamata</i>	<i>PID</i>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	
	<i>TGID</i>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>2</b>	<b>6</b>	<b>2</b>	
<b>P –pid1=fork</b>	<b>0</b>	<b>pronto</b>	<b>pronto</b>	<b>esec</b>	<b>pronto</b>	<b>NE</b>	<b>NE</b>	<b>NE</b>	
<b>P - open (sys_open)</b>	10	pronto	ESEC	attesa open	pronto	NE	NE	NE	
<b>S - pthread_create(TH1) (sys_clone)</b>	20	pronto	ESEC	attesa open	pronto	pronto	NE	NE	
<i>interrupt da DMA_in, tutti i blocchi richiesti trasferiti</i>	30	pronto	pronto	ESEC	pronto	pronto	NE	NE	
<b>P - fork (sys_clone)</b>	40	pronto	pronto	ESEC	pronto	pronto	pronto	NE	
<i>interrupt da RT_clock e scadenza quanto di tempo</i>	50	pronto	pronto	pronto	ESEC	pronto	pronto	NE	
<b>Q - write (sys_write)</b>	60	pronto	pronto	pronto	attesa write	ESEC	pronto	NE	
<b>TH1 - sem_wait (sys_wait)</b>	70	pronto	ESEC	pronto	attesa write	attesa sem_wait	pronto	NE	
<b>S - pthread_create(TH2) (sys_clone)</b>	80	pronto	ESEC	pronto	attesa write	attesa sem_wait	pronto	pronto	
<b>S - pthread_join(TH1) (sys_wait)</b>	90	pronto	attesa join(TH1)	pronto	attesa write	attesa sem_wait	ESEC	pronto	
<b>R - read (sys_read)</b>	100	pronto	attesa join(TH1)	ESEC	attesa write	attesa sem_wait	attesa read	pronto	
<b>P - exit (sys_exit)</b>	110	pronto	attesa join(TH1)	attesa nanosleep	attesa write	attesa sem_wait	attesa read	ESEC	
<b>TH2 - mutex_lock(GATE)</b>	120	pronto	attesa join(TH1)	attesa nanosleep	attesa write	attesa sem_wait	attesa read	ESEC	
<b>TH2 - sem_post(GATE)</b>	130	pronto	attesa join(TH1)	attesa nanosleep	attesa write	attesa sem_wait	attesa read	ESEC	
<b>TH1 - mutex_lock(GATE) (sys_wait)</b>	140	pronto	attesa join(TH1)	attesa nanosleep	attesa write	attesa lock	attesa read	ESEC	
<b>TH2 - mutex_unlock(GATE)</b>	150	pronto	attesa join(TH1)	attesa nanosleep	attesa write	ESEC	attesa read	pronto	

Si considerino le chiamate in **prog\_x** contrassegnate dai numeri d'ordine da **1** a **9**. Con riferimento alla loro implementazione tramite *futex* e alla simulazione effettuata, si indichino quelle eseguite:

- senza invocare *System\_Call*: **5** (num = 1)
- con invocazione di *System\_Call*: **1 6** (num = 1) **7 8 9 2**

### esercizio n. 3 – struttura e moduli del nucleo

Sono dati due processi **P** e **Q**. Lo stato iniziale delle pile di sistema e utente dei due processi è riportato qui sotto.



- **Si indichi** lo stato dei processi così come deducibile dallo stato iniziale delle pile:

**P**    In attesa della scadenza di un time-out (*sys\_nanosleep*)

**Q**    Attualmente in esecuzione

- Per l'evento indicato **si mostrino** le invocazioni di tutti i **moduli** (e eventuali relativi ritorni) per la gestione dell'evento stesso (precisando processo e modo) e – come specificato nella descrizione – il **contenuto delle pile** utente e di sistema.

NOTAZIONE da usare per i moduli:   > (invocazione),   nome\_modulo (esecuzione),   < (ritorno)

**Evento:** *interrupt* da *real-time clock* e scadenza di **timeout** (il processo **P** ha maggiori diritti di esecuzione del processo **Q**).

**Si mostri** lo stato delle pile di **Q** al termine della gestione dell'evento.

**invocazione moduli** (num. di righe vuote non signif.)

**contenuto della pila**

<i>processo</i>	<i>modo</i>	<i>modulo</i>
Q	U	"codice utente di Q"
Q	U -> S	>R_int_clock
Q	S	>task_tick <
Q	S	>Controlla_timer
Q	S	>wake_up_process
Q	S	>enqueue_task <
Q	S	>check_preempt_curr
Q	S	>resched (T.N.R = 1) <
Q	S	check_preempt_curr <
Q	S	wake_up_process <
Q	S	Controlla_timer <
Q	S	>schedule
Q	S	>pick_next_task <
Q -> P	S	schedule: "context_switch"
P	S	schedule <
P	S	schedule_timeout <
P	S	sys_nanosleep <
P	S -> U	SYSRET (system_call <)
P	U	syscall <
P	U	nanosleep <
P	U	"codice utente di P"

(W)  
uBase\_Q

Pop =>

Pop =>

Pop =>

Pop =>

Pop =>

Pop =>

Pop =>

sBase\_Q


USP = (W) uStack\_Q

I. rientro a schedule da pick\_next\_task

I. rientro R\_int\_clock da schedule

I. rientro a check\_preempt\_curr da resched

I. rientro a wake\_up\_process da  
check\_preempt\_curr

I. rientro a wake\_up\_process da  
enqueue\_task

I. rientro a Controlla time da  
wake\_up\_process

I. rientro a R\_int\_clock da Controlla\_timer

I. rientro a R\_int\_clock da task\_tick

PSR (U)

I. rientro a codice utente Q da R\_int\_clock

sStack\_Q

## esercizio n. 4 – gestione della memoria – 1

È dato un sistema di memoria caratterizzato dai seguenti parametri generali: **MAXFREE = 3**, **MINFREE = 2**.  
Si consideri la seguente **situazione iniziale** (raggiunta tramite la seguente sequenza di eventi partendo da memoria vuota: *exec* (2, 0, 1, 3, 1, "X"), *read* (Ps0, Pd0), *write* (Pp1, Pd1, Pd2)).

PROCESSO: P \*\*\*\*\*

```
VMA:  C  000000400,  2 , R  ,  P  ,  M  ,  <X,0>
      S  000000600,  1 , W  ,  P  ,  M  ,  <X,2>
      D  000000601,  3 , W  ,  P  ,  A  ,  <-1,0>
      P  7FFFFFFFC,  3 , W  ,  P  ,  A  ,  <-1,0>
```

```
PT:  <c0 :- ->    <c1 :1  R>
      <s0 :3  R>
      <d0 :0  R>  <d1 :5  W>  <d2 :6  W>
      <p0 :2  W>  <p1 :4  W>  <p2 :- ->
```

processo P - NPV di PC e SP: c1, p1

MEMORIA FISICA (pagine libere: 5)

00 : Pd0 / <ZP>	01 : Pc1 / <X,1>
02 : Pp0	03 : Ps0 / <X,2>
04 : Pp1	05 : Pd1
06 : Pd2	07 : ----
08 : ----	09 : ----
10 : ----	11 : ----

Si rappresenti l'effetto dei seguenti quattro eventi consecutivi sulle strutture dati della memoria compilando esclusivamente le tabelle fornite per ciascun evento (l'assenza di una tabella significa che non è richiesta la compilazione della corrispondente struttura dati).

**ATTENZIONE:** nella rappresentazioni delle TP, le PTE di una stessa VMA sono scritte su una riga; le PTE di VMA diverse sono scritte su righe diverse, come esemplificato nello stato iniziale.

### evento 1: *fork* (Q)

PT del processo: P				
<c0 :- ->	<c1 :1 R>			
<s0 :3 R>				
<d0 :0 R>	<d1 :5 R>	<d2 :6 R>		
<p0 :2 R>	<p1 :7 W>	<p2 :- ->		
PT del processo: Q (indicare solo le PTE relative alle VMA D e P)				
<d0 :0 R>	<d1 :5 R>	<d2 :6 R>		
<p0 :2 R>	<p1 :4 W D>	<p2 :- ->		

MEMORIA FISICA	
00: Pd0 / Qd0 / <ZP>	01: Pc1 / Qc1 / <X, 1>
02: Pp0 / Qp0	03: Ps0 / Qs0 / <X, 2>
04: Qp1 (D)	05: Pd1 / Qd1
06: Pd2 / Qd2	07: Pp1
08: ----	09: ----
10: ----	11: ----



**evento 2: write (Ps0)**

PT del processo: P				
<c0 :- ->	<c1 :1 R>			
<s0 :8 W>				
<d0 :0 R>	<d1 :5 R>	<d2 :6 R>		
<p0 :2 R>	<p1 :7 W>	<p2 :- ->		

MEMORIA FISICA	
00: Pd0 / Qd0 / <ZP>	01: Pc1 / Qc1 / <X, 1>
02: Pp0 / Qp0	03: Qs0
04: Qp1 (D)	05: Pd1 / Qd1
06: Pd2 / Qd2	07: Pp1
08: Ps0	09: ----
10: ----	11: ----

**evento 3: mmap (0x 000030000000, 2, W, P, M, "F", 2 ), read (Pm00)**

VMA del processo P (compilare solo la riga relativa alla nuova VMA creata)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
M0	0x 0000 3000 0000	2	W	P	M	F	2

PT del processo: P (compilare solo la riga relativa alla nuova VMA creata)				
<Pm00 :9 R>	<Pm01 :- ->			

MEMORIA FISICA	
00: Pd0 / Qd0 / <ZP>	01: Pc1 / Qc1 / <X, 1>
02: Pp0 / Qp0	03: Qs0
04: Qp1 (D)	05: Pd1 / Qd1
06: Pd2 / Qd2	07: Pp1
08: Ps0	09: Pm00 / <F, 2>
10: ----	11: ----

**evento 4: *exec*(4, 0, 3, 1, 3, "Y")**

VMA del processo P							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	0x 0000 0040 0000	4	R	P	M	Y	0
S	0x 0000 0060 0000	3	W	P	M	Y	4
D	0x 0000 0060 0003	1	W	P	A	/	/
P	0x 7FFF FFFF FFFF	3	W	P	A	/	/

PT del processo P				
<c0 :- ->	<c1 :- ->	<c2 :- ->	<c3 :7 R>	
<s0 :- ->	<s1 :- ->	<s2 :- ->		
<d0 :- ->				
<p0 :8 W>	<p1 :- ->	<p2 :- ->		

processo P – NPV di PC e SP: Pc3 - Pp0

MEMORIA FISICA	
00: <ZP>	01: Qc1 / <X, 1>
02: Qp0	03: Qs0 / <X, 2>
04: Qp1 (D)	05: Qd1
06: Qd2	07: Pc3 / <Y, 3>
08: Pp0	09: <F, 2>
10: ----	11: ----



## esercizio n. 5 – gestione della memoria – 2

È dato un sistema di memoria caratterizzato dai seguenti parametri generali: **MAXFREE = 3**, **MINFREE = 1**.  
**ATTENZIONE: MINFREE è diverso rispetto all'esercizio precedente.**

Si consideri la seguente **situazione iniziale**:

```
PROCESSO: P *****
PT: <c0 :1 R>
    <s0 :4 R> <s1 :- ->
    <d0 :5 R> <d1 :- ->
    <p0 :2 R> <p1 :7 W> <p2 :3 W> <p3 :8 W> <p4 :- ->
process P - NPV of PC and SP: c0, p3
```

```
PROCESSO: Q *****
PT: <c0 :1 R>
    <s0 :4 R> <s1 :- ->
    <d0 :5 R> <d1 :- ->
    <p0 :2 R> <p1 :6 D W> <p2 :- ->
process Q - NPV of PC and SP: c0, p1
```

```
_____MEMORIA FISICA_____ (pagine libere: 1)_____
00 : <ZP> | | 01 : Pc0 / Qc0 / <X,0> | |
02 : Pp0 / Qp0 | | 03 : Pp2 | |
04 : Ps0 / Qs0 | | 05 : Pd0 / Qd0 | |
06 : Qp1 D | | 07 : Pp1 | |
08 : Pp3 | | 09 : ---- | |
```

```
_____STATO del TLB_____
Pc0 : 01 - 0: 1: | | Pp0 : 02 - 1: 0: | |
Ps0 : 04 - 1: 0: | | Pd0 : 05 - 1: 0: | |
Pp1 : 07 - 1: 1: | | Pp2 : 03 - 1: 1: | |
Pp3 : 08 - 1: 1: | | ----- | |
```

SWAP FILE: ----, ----, ----, ----, ----, ----

LRU ACTIVE: PP3, PP2, PC0, PP1

LRU INACTIVE: pp0, pd0, ps0, qp1, qd0, qs0, qp0, qc0

Si rappresenti l'effetto del seguente evento sulle strutture dati della memoria compilando esclusivamente le tabelle fornite per ciascun evento (l'assenza di una tabella significa che non è richiesta la compilazione della corrispondente struttura dati).

**ATTENZIONE:** nella rappresentazioni delle TP, le PTE di una stessa VMA sono scritte su una riga (andando a capo se necessario); le PTE di VMA diverse sono scritte su righe diverse, come esemplificato nello stato iniziale.

**evento: *write* (Pp4)**

VMA del processo P (compilare solo la riga relativa alla VMA della pila)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
P							

PT del processo P				

PT del processo Q				

MEMORIA FISICA	
00:	01:
02:	03:
04:	05:
06:	07:
08:	09:

TLB							
NPV	NPF	D	A	NPV	NPF	D	A

SWAP FILE	
s0:	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: \_\_\_\_\_

LRU INACTIVE: \_\_\_\_\_

