**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 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)** _____

**CON SOLUZIONI (in corsivo)**

## 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** | ***thread*** | |
|---|---|---|
| | **th_1 – `ruler`** | **th_2 – `compass`** |
| subito dopo stat. **A** | *ESISTE* | *ESISTE* |
| subito dopo stat. **B** | *ESISTE* | *PUÒ ESISTERE* |
| subito dopo stat. **C** | *ESISTE* | *ESISTE* |
| subito dopo stat. **D** | *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** | |
|---|---|---|
| | *point* | *global* |
| subito dopo stat. **A** | *1* | *0* |
| subito dopo stat. **B** | *0 / 1* | *1 / 2* |
| subito dopo stat. **D** | *0 / 1* | *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** | ***`lock line`*** | ***`wait point`*** |
| **2** | ***`wait point`*** | ***`lock circle`*** |
| **3** | **–** | ***`wait point`*** |

***Commenti alla soluzione***

*Esistenza:*

A. *lo statement appartiene a th_1*
    a. *th_1 → esiste*
    b. *th_2 è già stato creato (prima di th_1) e non ha già passata wait, th_2 → esiste*
B. *lo statement appartiene a th_1*
    a. *th_1 → esiste*
    b. *th2 è già stato creato (prima di th_1) e può terminare prima che th_1 entri in mutex circle, th_2 → può esistere*
C. *lo statement appartiene a th_2*
    a. *th_2 → esiste*
    b. *th_2 ha passata wait, dunque th_1 è già stato creato e ha passata post (semaforo inizialmente a 0), però ora th_2 è in mutex circle, dunque th_1 non è in mutex circle (se th_1 fosse entrato in circle, si sarebbe bloccato su wait e th_2 non sarebbe potuto entrare in circle) e non è ancora terminato, th_1 → esiste*
D. *lo statement appartiene a main e garantisce la terminazione di th_2*
    a. *th_2 → non esiste*
    b. *th_1 è già stato creato poiché th_2 ha passata wait (vedi caso C.b), e th_1 può essere già terminato, th_1 → può esistere*

*Variabili globali:*

A. *th_1 è in mutex line dopo post, th_2 è prima di lock line o su lock line, dunque point = 1 e global ha ancora valore iniziale 0*
B. *th_1 è in mutex circle, dopo post ma prima di wait, e ha eseguita global = 1, th_2 è già stato creato (prima di th_1): due sottocasi*
    a. *th_2 non è terminato e ha già eseguita o no wait e global = 2, dunque point = 0,1 e global = 1, 2*
    b. *th_2 è terminato, e th_1 ha eseguita global = 1 dopo che th_2 ha eseguita global = 3, dunque point = 1 e global = 1*

    *riassumendo: point = 0,1 e global = 1, 2*
C. *caso non richiesto*
D. *th_2 è terminato, th_1 è prima o dopo wait e global = 1, dunque point = 0, 1 e global = 1, 3*

*Deadlock:*

1. *deadlock classico su lock line in th_1 e wait in th_2 con semaforo inizialmente nullo*
2. *deadlock classico su wait in th_1 e lock circle in th_2: il semaforo da 0 iniziale era sailto a 1 (post in th_1) e poi era ridisceso a 0 (wait in th_2), ora si ripropone un deadlock simile a (1), ma relativo al mutex circle*
3. *deadlock di un solo thread (th_2) per esaurimento risorse (ma si noti che il deadlock coinvolge anche main)*

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

```
// programma prova.c
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 prog_x.c
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 S
    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:

- ⟨ *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 del tempo indicato**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

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

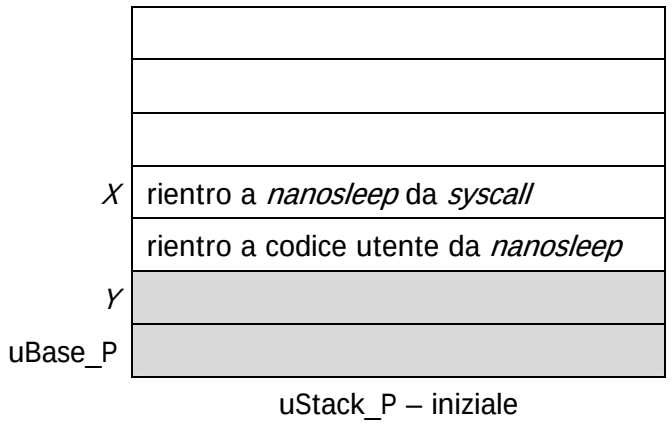
| *identificativo simbolico del processo* | | ***IDLE*** | ***S*** | ***P*** | *Q* | *TH1* | *R* | *TH2* | |
|---|---|---|---|---|---|---|---|---|---|
| | *PID* | **1** | **2** | **3** | *4* | *5* | *6* | *7* | |
| *evento processo-chiamata* | *TGID* | **1** | **2** | **3** | *4* | *2* | *6* | *2* | |
| **P –pid1=fork** | **0** | **pronto** | **pronto** | **esec** | *pronto* | *NE* | *NE* | *NE* | |
| *P – open* | 10 | *pronto* | *esec* | *A open* | *pronto* | *NE* | *NE* | *NE* | |
| *S – pthread_create TH1* | 20 | *pronto* | *esec* | *A open* | *pronto* | *pronto* | *NE* | *NE* | |
| *interrupt* da *DMA_in*, tutti i blocchi richiesti trasferiti | 30 | *pronto* | *pronto* | *esec* | *pronto* | *pronto* | *NE* | *NE* | |
| *P – pid2 = fork* | 40 | *pronto* | *pronto* | *esec* | *pronto* | *pronto* | *pronto* | *NE* | |
| *interrupt* da *RT_clock* e scadenza quanto di tempo | 50 | *pronto* | *pronto* | *pronto* | *esec* | *pronto* | *pronto* | *NE* | |
| *Q – open* | 60 | *pronto* | *pronto* | *pronto* | *A open* | *esec* | *pronto* | *NE* | |
| *TH1 – sem_wait* | 70 | *pronto* | *esec* | *pronto* | *A open* | *A sem_wait* | *pronto* | *NE* | |
| *S – pthread_create TH2* | 80 | *pronto* | *esec* | *pronto* | *A open* | *A sem_wait* | *pronto* | *pronto* | |
| *S – pthread_join (TH2)* | 90 | *pronto* | *A join* | *pronto* | *A open* | *A sem_wait* | *esec* | *pronto* | |
| *R - read* | 100 | *pronto* | *A join* | *esec* | *A open* | *A sem_wait* | *A read* | *pronto* | |
| *P - nanosleep* | 110 | *pronto* | *A join* | *A timeout* | *A open* | *A sem_wait* | *A read* | *esec* | |
| *TH2 – mutex_lock* | 120 | *pronto* | *A join* | *A timeout* | *A open* | *A sem_wait* | *A read* | *esec* | |
| *TH2 – sem_post (num = 1)* | 130 | *pronto* | *A join* | *A timeout* | *A open* | *esec* | *A read* | *pronto* | |
| *TH1 – mutex_lock* | 140 | *pronto* | *A join* | *A timeout* | *A open* | *A lock* | *A read* | *esec* | |
| *TH2 – mutex_unlock* | 150 | *pronto* | *A join* | *A timeout* | *A open* | *esec* | *A 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:

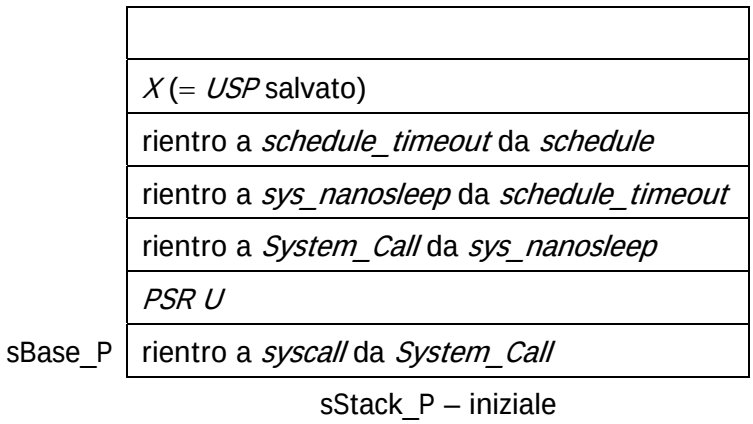
- senza invocare *System_Call*: *5*
- con invocazione di *System_Call*: *1, 8, 6, 7, 2*

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

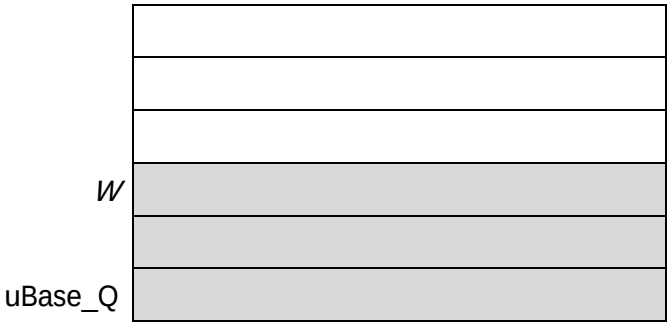
Sono dati due processi ***P*** e ***Q***. Lo stato iniziale delle pile di sistema e utente dei due processi è riportato qui sotto.



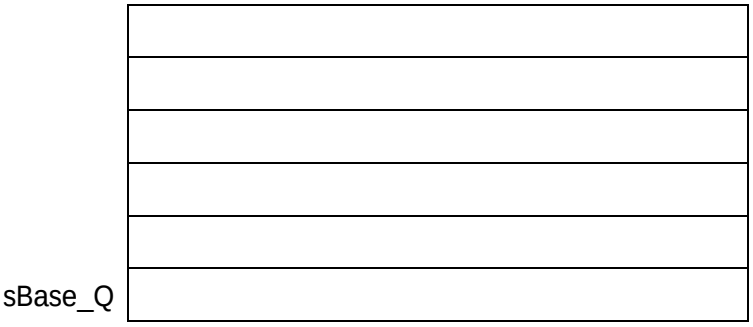

uStack_P – iniziale




sStack_P – iniziale




uStack_Q – iniziale




sStack_Q – iniziale

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

  ***P*** *in attesa da nanosleep (attesa scadenza timeout)*

  ***Q*** *in esecuzione modo U*

- 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.)

| ***processo*** | ***modo*** | ***modulo*** |
|---|---|---|
| *Q* | *U – S* | *> R_int (ck)* |
| *Q* | *S* | *> task_tick <* |
| *Q* | *S* | *> controlla_timer* |
| *Q* | *S* | *> wake_up_process* |
| *Q* | *S* | *> check_preemt_curr* |
| *Q* | *S* | *> resched (TNR = 1) <* |
| *Q* | *S* | *check_preemt_curr <* |
| *Q* | *S* | *wake_up_process <* |
| *Q* | *S* | *controlla_timer <* |
| *Q* | *S* | *> schedule* |
| *Q* | *S* | *> pick_next_task <* |
| *Q – P* | *S* | *context_switch* |
| *P* | *S* | *schedule <* |
| *P* | *S* | *schedule_timeout <* |
| *P* | *S* | *sys_nanosleep <* |
| *P* | *S* | *System_Call <* : *SYSRET* |
| *P* | *U* | *syscall <* |
| *P* | *U* | *nanosleep <* |
| *P* | *U* | *codice utente* |
| | | |
| | | |

**contenuto della pila**

uStack_Q

| | |
|---|---|
| | |
| | |
| | |
| | |
| | |
| *W* | *piena* |
| | *piena* |
| uBase_Q | *piena* |

sStack_Q

| | |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | *W* ( = *USP salvato)* |
| | *rientro a R_int (ck) da schedule* |
| | *PSR U* |
| sBase_Q | *rientro a codice utente da R_int (ck)* |

## 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  7FFFFFFC,  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: 01 R* | | | |
| *s0: 03 R* | | | | |
| *d0: 00 R* | *d1: 05 R* | *d2: 06 R* | | |
| *p0: 02 R* | *p1: 07 W* | *p2: - -* | | |
| **PT** del processo: **Q**<br>(indicare solo le PTE relative alle VMA D e P) | | | | |
| *d0: 00 R* | *d1: 05 R* | *d2: 06 R* | | |
| *p0: 02 R* | *p1: 04 D W* | *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: 01 R* | | | |
| *s0: 08 W* | | | | |
| *d0: 00 R* | *d1: 05 R* | *d2: 06 R* | | |
| *p0: 02 R* | *p1: 07 W* | *p2: - -* | | |

| **MEMORIA FISICA** | |
|---|---|
| 00: *Pd0 / Qd0 / <ZP>* | 01: *Pc1 / Qc1 / <X, 1>* |
| 02: *Pp0 / Qp0* | 03: *Qs0 / <X, 2>* |
| 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**<br>(compilare solo la riga relativa alla nuova VMA creata) | | | | | | | |
|---|---|---|---|---|---|---|---|
| AREA | NPV iniziale | dimensione | R/W | P/S | M/A | nome file | offset |
| *M0* | *0000 3000 0* | *2* | *W* | *P* | *M* | *F* | *2* |

| **PT** del processo: **P**<br>(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 / <X, 2>* |
| 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* | *0000 0040 0* | *4* | *R* | *P* | *M* | *Y* | *0* |
| *S* | *0000 0060 0* | *3* | *W* | *P* | *M* | *Y* | *4* |
| *D* | *0000 0060 3* | *1* | *W* | *P* | *A* | *-1* | *0* |
| *P* | *7FFF FFFF C* | *3* | *W* | *P* | *A* | *-1* | *0* |

| **PT** del processo **P** | | | | |
|---|---|---|---|---|
| *c0: - -* | *c1: - -* | *c2: - -* | *c3: 07 R* | |
| *s0: - -* | *s1: - -* | *s2: - -* | | |
| *d0: - -* | | | | |
| *p0: 08 W* | *p1: - -* | *p2: - -* | | |

processo **P** – **NPV** di **PC** e **SP**: *c3, p0* _____

| **MEMORIA FISICA** | |
|---|---|
| 00: *Qd0 / <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: *---* |

***Soluzione***

```
========== situazione iniziale ==========

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  7FFFFFFC,  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 : ----                 ||

========== 1: fork (Q) ==========

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

    PT: <c0 :- ->   <c1 :1  R>
        <s0 :3  R>
        <d0 :0  R>  <d1 :5  R>  <d2 :6  R>
        <p0 :2  R>  <p1 :7  W>  <p2 :- ->

PROCESSO: Q  ***************************************************

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

____MEMORIA FISICA____(pagine libere: 4)_____________________________
      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 : ----                 ||
```

========== 2: **write** (Ps0) ==========

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

PT: <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____(pagine libere: 3)_____

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

= 3: **mmap** (0x000030000000, 2, W, P, M, “F”, 2), **read** (Pm00) =

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

VMA: ...
**M0 000030000,  2 , W  ,  P  ,  M  , <F,2>**

PT: ...
**<m00:9  R>  <m01:- ->**

____MEMORIA FISICA____(pagine libere: 2)_____

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

========== 4: **exec** (4, 0, 3, 1 , 3, “Y”)  ==========

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

VMA:  **C  000000400,  4 , R  ,  P  ,  M  , <Y,0>**
**S  000000600,  3 , W  ,  P  ,  M  , <Y,4>**
**D  000000603,  1 , W  ,  P  ,  A  , <-1,0>**
**P  7FFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>**

PT: **<c0 :- ->  <c1 :- ->  <c2 :- ->  <c3 :7  R>**
**<s0 :- ->  <s1 :- ->  <s2 :- ->**
**<d0 :- ->**

**<p0 :8  W>  <p1 :- ->  <p2 :- ->**

process P - NPV of PC and SP: **c3**, **p0**

____MEMORIA FISICA____(pagine libere: 2)_____

| | |
|---|---|
| 00 : Qd0 / <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 : ---- |

Note: la pagina fisica 09 è mappata su file e resta in Page Cache; in memoria fisica (e volendo anche nella PT) hanno il bit D posto a 1 solo le pagine scritte che non sono in TLB.

**spazio libero per brutta copia o continuazione**

## 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**<br>(compilare solo la riga relativa alla VMA della pila) | | | | | | | |
|---|---|---|---|---|---|---|---|
| AREA | NPV iniziale | dimensione | R/W | P/S | M/A | nome file | offset |
| P | *7FFF FFFF 9* | *6* | *W* | *P* | *A* | *-1* | *0* |

| **PT** del processo **P** | | | | |
|---|---|---|---|---|
| *c0: 01 R* | | | | |
| *s0: s1 R* | *s1: - -* | | | |
| *d0: s2 R* | *d1: - -* | | | |
| *p0: 02 R* | *p1: 07 W* | *p2: 03 W* | *p3: 08 W* | *p4: 04 W* |
| *p5: - -* | | | | |

| **PT** del processo **Q** | | | | |
|---|---|---|---|---|
| *c0: 01 R* | | | | |
| *s0: s1 R* | *s1: - -* | | | |
| *d0: s2 R* | *d1: - -* | | | |
| *p0: 02 R* | *p1: s0 W* | *p2: - -* | | |

| **MEMORIA FISICA** | |
|---|---|
| 00: *<ZP>* | 01: *Pc0 / Qc0 / <X, 0>* |
| 02: *Pp0 / Qp0* | 03: *Pp2* |
| 04: *Pp4* | 05: *---* |
| 06: *---* | 07: *Pp1* |
| 08: *Pp3* | 09: *---* |

| **TLB** | | | | | | | |
|---|---|---|---|---|---|---|---|
| NPV | NPF | D | A | NPV | NPF | D | A |
| *Pc0: 01 – 0: 1:* | | | | *Pp2: 02 – 1: 0:* | | | |
| *Pp4: 04 – 1: 1:* | | | | *---* | | | |
| *Pp1: 07 – 1: 1:* | | | | *Pp2: 03 – 1: 1:* | | | |
| *Pp3: 08 – 1: 1:* | | | | *---* | | | |
| | | | | | | | |

| **SWAP FILE** | |
|---|---|
| s0: *Qp1* | s1: *Ps0 / Qs0* |
| s2: *Pd0 / Qd0* | s3: |
| s4: | s5: |

LRU ACTIVE: *PP4*, *PP3*, *PP2, PC0*, *PP1* _____

LRU INACTIVE: *pp0*, *qp0*, *qc0* _____

***Soluzione***

```
========== 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
```

========== **write** (“Pp4”) ==========

Viene invocato PFRA - Required:1, Free:1, To Reclaim:3
Pagine liberate da inactive (in ordine): Qp1, Ps0 / Qs0, Pd0 / Qd0

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

VMA: ...
**P 7FFFFFF9, 6 , W , P , A , <-1,0>**

PT: <c0 :1 R>
<**s0 :s1 R**> <s1 :- ->
<**d0 :s2 R**> <d1 :- ->
<p0 :2 R> <p1 :7 W> <p2 :3 W> <p3 :8 W>
<**p4 :4 W**> <**p5 :- ->**

process P - NPV of PC and SP: c0, **p4**

PROCESSO: Q **********

PT: <c0 :1 R>
<**s0 :s1 R**> <s1 :- ->
<**d0 :s2 R**> <d1 :- ->
<p0 :2 R> <**p1 :s0 W**> <p2 :- ->

process Q - NPV of PC and SP: c0, p1

____MEMORIA FISICA____(pagine libere: 3)_____

| | |
|---|---|
| 00 : <ZP> | 01 : Pc0 / Qc0 / <X,0> |
| 02 : Pp0 / Qp0 | 03 : Pp2 |
| 04 : **Pp4** | 05 : **----** |
| 06 : **----** | 07 : Pp1 |
| 08 : Pp3 | 09 : ---- |

____STATO del TLB_____

| | |
|---|---|
| Pc0 : 01 - 0: 1: | Pp0 : 02 - 1: 0: |
| **Pp4** : **04** - **1**: **1**: | **-----** |
| Pp1 : 07 - 1: 1: | Pp2 : 03 - 1: 1: |
| Pp3 : 08 - 1: 1: | **-----** |

SWAP FILE: **Qp1**, **Ps0 / Qs0**, **Pd0 / Qd0**, ----, ----, ----

LRU ACTIVE: **PP4**, PP3, PP2, PC0, PP1

LRU INACTIVE: pp0, qp0, qc0

Si noti che la pila di P ha avuto growsdown da p4 a p5, e che lo SP corrente di P è diventata la pagina p4; nel TLB sono state riusate le righe appena cancellate per swap_out (ma il TLB è associativo è le righe vuote sono tutte equivalenti).

**spazio libero per brutta copia o continuazione**