Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri **prof.ssa Donatella Sciuto**
prof. Gerardo Pelosi **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – martedì 31 agosto 2021

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 : 30 m

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

esercizio 1 (4 punti) _____

esercizio 2 (5 punti) _____

esercizio 3 (5 punti) _____

esercizio 4 (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, come anche il prefisso `pthread` delle funzioni di libreria NPTL):

```
pthread_mutex_t drum
sem_t noisy, still
int global = 0
```

```
void * player (void * arg) {
   mutex_lock (&drum)
   global = 1                                          /* statement A */
   sem_wait (&noisy)
   mutex_unlock (&drum)
   global = 2
   sem_wait (&still)
   return NULL
} /* end player */
```

```
void * public (void * arg) {
   sem_wait (&noisy)
   global = 3
   mutex_lock (&drum)
   sem_post (&noisy)
   global = 4                                          /* statement B */
   mutex_unlock (&drum)
   sem_post (&still)                                   /* statement C */
   return NULL
} /* end public */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&noisy, 0, 1)
   sem_init (&still, 0, 0)
   create (&th_1, NULL, player, NULL)
   create (&th_2, NULL, public, NULL)
   join (th_2, NULL)                                   /* statement D */
   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 – *player*	**th_2 – *public***
subito dopo stat. **A**	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **B**	*ESISTE*	*ESISTE*
subito dopo stat. **C**	*PUÒ ESISTERE*	*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 supponga che il mutex valga 1 se occupato, e valga 0 se libero

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**			
	drum	*noisy*	*still*	*global*
subito dopo stat. **A**	*1*	*0 / 1*	*0 / 1*	*1 / 3*
subito dopo stat. **B**	*1*	*1*	*0*	*4*
subito dopo stat. **D**	*0 / 1*	*0 / 1*	*0 / 1*	*1 / 2 / 4*

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 – *player***	**th_2 – *public***	*global*
1	`wait noisy`	`lock drum`	*1 / 3*
2	`wait still`	`wait noisy`	*2*
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma esercizio.c
int main ( ) { char vett [300]                    // codice eseguito da P
   fd = open ("/acso/esame", ORDWR)
   pidQ = fork ( )
   if (pidQ == 0) {                               // codice eseguito da Q
      execl ("/acso/nuovo", "nuovo", NULL)
      exit (-1)
   } else {                                       // codice eseguito da P
      read (stdin, vett, 10)
      pidQ = wait (&status)
   } /* if */
   exit (0)
} /* esercizio */
```

```
// programma nuovo.c
pthread_mutex_t CHECK = PTHREAD_MUTEX_INITIALIZER
sem_t SEM
```

```
void * first (void * arg) {
     pthread_mutex_lock (&CHECK)
     sem_post (&SEM)
     pthread_mutex_unlock (&CHECK)
     sem_wait (&SEM)
     return NULL
} /* first */
```

```
void * second (void * arg) {
     pthread_mutex_lock (&CHECK)
     sem_wait (&SEM)
     pthread_mutex_unlock (&CHECK)
     sem_wait (&SEM)
     sem_post (&SEM)
     return NULL
} /* second */
```

```
int main ( ) { // codice eseguito da Q
   pthread_t TH_1, TH_2
   sem_init (&SEM, 0, 1)
   pthread_create (&TH_2, NULL, second, NULL)
   pthread_create (&TH_1, NULL, first, NULL)
   pthread_join (TH_2, NULL)
   pthread_join (TH_1, NULL)
   exit (1)
} /* main */
```

Un processo ***P*** esegue il programma ***esercizio.c*** e crea il processo ***Q***, il quale esegue una mutazione di codice che va a buon fine. Nel codice mutato, il processo ***Q*** crea i thread ***TH2*** e ***TH1***.

Si simuli l'esecuzione dei processi (normali e thread) completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale e dagli eventi indicati, facendo l'ipotesi che il processo ***Q*** abbia **già eseguito** la mutazione di codice, ma **non abbia ancora creato** nessun thread.

Si completi la tabella riportando quanto segue:

- ⟨ PID, TGID ⟩ di ogni processo (normale o thread) che viene creato
- ⟨ evento oppure identificativo del processo-chiamata di sistema / libreria ⟩ nella prima colonna, dove necessario e in funzione del codice proposto (le istruzioni da considerare sono evidenziate in grassetto)
- 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

identificativo simbolico del processo		**IDLE**	**P**	**Q**	*TH_2*	*TH_1*
evento oppure processo-chiamata	**PID**	**1**	**2**	**3**	*4*	*5*
	TGID	**1**	**2**	**3**	*3*	*3*
Q – pthread_create TH2	**0**	**pronto**	**attesa (read)**	**esec**	*pronto*	
interrupt da RT_clock scadenza quanto di tempo	**1**	*pronto*	*attesa (read)*	*pronto*	*esec*	
TH_2 – mutex_lock CHECK	**2**	*pronto*	*attesa (read)*	*pronto*	*esec*	
TH_2 – sem_wait SEM	**3**	*pronto*	*attesa (read)*	*pronto*	*esec*	
TH_2 – mutex_unlock CHECK	**4**	*pronto*	*attesa (read)*	*pronto*	*esec*	
TH_2 – sem_wait SEM	**5**	*pronto*	*attesa (read)*	*esec*	*attesa (sem_wait)*	
interrupt da stdin tutti i caratteri trasferiti	**6**	*pronto*	*esec*	*pronto*	*attesa (sem_wait)*	
P – wait status	**7**	*pronto*	*attesa (wait)*	*esec*	*attesa (sem_wait)*	
Q – pthread_create TH1	**8**	*pronto*	*attesa (wait)*	*esec*	*attesa (sem_wait)*	*pronto*
Q – pthread_join TH2	**9**	*pronto*	*attesa (wait)*	*attesa (join TH2)*	*attesa (sem_wait)*	*esec*
TH_1 – mutex_lock CHECK	**10**	*pronto*	*attesa (wait)*	*attesa (join TH2)*	*attesa (sem_wait)*	*esec*
TH_1 – sem_post SEM	**11**	*pronto*	*attesa (wait)*	*attesa (join TH2)*	*esec*	*pronto*
TH_1 – sem_post SEM	**12**	*pronto*	*attesa (wait)*	*attesa (join TH2)*	*esec*	*pronto*
interrupt da RT_clock scadenza quanto di tempo	**13**	*pronto*	*attesa (wait)*	*attesa (join TH2)*	*pronto*	*esec*
TH_1 – mutex_unlock CHECK	**14**	*pronto*	*attesa (wait)*	*attesa (join TH2)*	*pronto*	*esec*
TH_1 – sem_wait SEM	**15**	*pronto*	*attesa (wait)*	*attesa (join TH2)*	*pronto*	*esec*
TH_1 – return	**16**	*pronto*	*attesa (wait)*	*pronto*	*esec*	*NE*

seconda parte – scheduling

Si consideri uno scheduler CFS con **due task** caratterizzato da queste condizioni iniziali (già complete):

CONDIZIONI INIZIALI (già complete)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	5	T1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T1	1	0,20	1,20	1,00	10	100
RB	T2	4	0,80	4,80	0,25	20	101

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

```
Events of task T1:  CLONE at 0.5;
Events of task T2:  WAIT at 2.0;   WAKEUP after 2.5;
```

Simulare l'evoluzione del sistema per **quattro eventi** riempiendo le seguenti tabelle (per indicare le condizioni di rescheduling di *clone* e *wakeup*, e altri calcoli eventualmente richiesti, si usino le tabelle finali):

EVENTO 1		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
		0,50	*CLONE*	*T1*	*falso*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*6*	*T1*	*100,50*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T1*	*1*	*0,17*	*1*	*1,00*	*10,50*	*100,50*
RB	*T2*	*4*	*0,67*	*4*	*0,25*	*20,00*	*101,00*
	T3	*1*	*0,17*	*1*	*1,00*	*0*	*101,50*
WAITING							

EVENTO 2		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
		1,00	*Q_scade*	*T1*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*6*	*T2*	*101,00*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T2*	*4*	*0,67*	*4*	*0,25*	*20,00*	*101,00*
RB	*T1*	*1*	*0,17*	*1*	*1,00*	*11,00*	*101,00*
	T3	*1*	*0,17*	*1*	*1,00*	*0*	*101,50*
WAITING							

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
		3,00	*WAIT*	*T2*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	*2*	*T1*	*101,00*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T1*	*1*	*0,50*	*3*	*1,00*	*11,00*	*101,00*
RB	*T3*	*1*	*0,50*	*3*	*1,00*	*0*	*101,50*
WAITING	*T2*	*4*				*22,00*	*101,50*

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
		5,50	*WUP*	*T1*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*6*	*T3*	*101,50*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T3*	*1*	*0,17*	*1*	*1,00*	*0*	*101,50*
RB	*T2*	*4*	*0,67*	*4*	*0,25*	*22,00*	*101,50*
	T1	*1*	*0,17*	*1*	*1,00*	*13,50*	*103,50*
WAITING							

Calcolo del *VRT iniziale* del nuovo task generato dalla **CLONE**:

$$T3.VRT = VMIN + T3.Q \times T3.VRTC = 100,50 + 1 \times 1 = 101,50$$

Valutazione della *condizione di rescheduling* alla **CLONE**:

$$T3.VRT + WGR \times T3.LC = 101,50 + 1,00 \times 0,17 = 101,67 < 100,50 = CURR.VRT \Rightarrow falso$$

Valutazione della *condizione di rescheduling* alla **WAKEUP**:

$$T2.VRT + WGR \times T2.LC = 101,50 + 1,00 \times 0,67 = 102,17 < 103,50 = CURR.VRT \Rightarrow vero$$

esercizio n. 3 – memoria e file system

prima parte – memoria virtuale

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

MAXFREE = **3** **MINFREE** = **1**

situazione iniziale (esistono un processo P e un processo Q)

```
PROCESSO: P  ********************************************************
    VMA : C  0000 0040 0,  1 , R , P , M , <X,  0>
          S  0000 0060 0,  2 , W , P , M , <X,  1>
          D  0000 0060 2,  2 , W , P , A , <-1, 0>
          P  7FFF FFFF B,  4 , W , P , A , <-1, 0>

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

    process P - NPV of PC and SP:  c0, p2

PROCESSO: Q  ********************************************************
    VMA : C  0000 0040 0,  1 , R , P , M , <X,  0>
          S  0000 0060 0,  2 , W , P , M , <X,  1>
          D  0000 0060 2,  2 , W , P , A , <-1, 0>
          P  7FFF FFFF C,  3 , W , P , A , <-1, 0>

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

    process Q - NPV of PC and SP:  c0, p0

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                ||  01 : Pc0 / Qc0 / <X, 0>  ||
      02 : Pp1                 ||  03 : Pp0                 ||
      04 : Pp2                 ||  05 : ----                ||
      06 : ----                ||  07 : ----                ||

____STATO del TLB_____________________________________________________
     Pc0 : 01 -  0: 1:         || Pp0 : 03 -  1: 0:         ||
     Pp2 : 04 -  1: 0:         ||       ----                ||
     Pp1 : 02 -  1: 0:         ||       ----                ||

SWAP FILE:    Qp0,  Ps0 / Qs0,  Pd0 / Qd0,  ----,  ----,  ----,
LRU ACTIVE:   PC0,
LRU INACTIVE: pp2,  pp1,  pp0,  qc0,
```

evento 1: *read* **(Ps0, Pd0)**

PT del processo: **P**				
s0: *5 R*	s1: - -	d0: *6 R*	d1: - -	p0: *3 W*
p1: *2 W*	p2: *4 W*	p3: - -		

process P	NPV of **PC**: *c0*	NPV of **SP**: *p2*

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: *Pp1*	03: *Pp0*
04: *Pp2*	05: *Ps0 / Qs0*
06: *Pd0 / Qd0*	07: *----*

SWAP FILE	
s0: *Qp0*	s1: *Ps0 / Qs0*
s2: *Pd0 / Qd0*	s3:
s4:	s5:

LRU ACTIVE: *PD0, PS0, PC0,* _____

LRU INACTIVE: *pp2, pp1, pp0, qc0, qs0, qd0,* _____

evento 2: *context switch* **(Q)**

Flush del TLB. Deve caricare (e scrivere) QP0 dallo swap file. Scatta PFRA che libera 3 pagine scandendo LRU inactive. Tutte le pagine erano state scritte (vedi TLB precedente), quindi in swap file. QP0 caricata in 02.

PT del processo: **P**				
s0: *5 R*	s1: - -	d0: *6 R*	d1: - -	p0: *s3 W*
p1: *s4 W*	p2: *s5 W*	p3: - -		

process P	NPV of **PC**: *c0*	NPV of **SP**: *p2*

PT del processo: **Q**				
s0: *5 R*	s1: - -	d0: *6 R*	d1: - -	p0: *2 W*
p1: - -	p2: - -			

process Q	NPV of **PC**: *c0*	NPV of **SP**: *p0*

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: *Qp0*	03: *----*
04: *----*	05: *Ps0 / Qs0*
06: *Pd0 / Qd0*	07: *----*

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
Qc0 :	*01 -*	*0:*	*1:*	*Qp0 :*	*02 -*	*1:*	*1:*

----	----
----	----

SWAP FILE	
s0: ----	s1: *Ps0 / Qs0*
s2: *Pd0 / Qd0*	s3: *Pp0*
s4: *Pp1*	s5: *Pp2*

LRU ACTIVE: *QP0, PD0, PS0, PC0,* _____

LRU INACTIVE: *qc0, qs0, qd0,* _____

evento 3: *read* (Qs0, Qd0)

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
Qc0 : 01 - 0: 1:				*Qp0 : 02 - 1: 1:*			
Qs0 : 05 - 0: 1				*Qd0 : 06 - 0: 1:*			
----				----			

evento 4: *write* (Qs0, Qp1)

Scatta COW per Qs0, Qs0 allocata in nuova pagina 03 e scritta, tolta da swap file. Qp1 da allocare e scrivere in 04, LRU active aggiornata.

PT del processo: **Q**				
s0: *3 W*	s1: - -	d0: *6 R*	d1: - -	p0: *2 W*
p1: *4 W*	p2: - -			

process Q	NPV of **PC**: *c0*	NPV of **SP**: *p1*

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: *Qp0*	03: *Qs0*
04: *Qp1*	05: *Ps0*
06: *Pd0 /Qd0*	07: ----

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
Qc0 :	*01 -*	*0:*	*1:*	*Qp0 :*	*02 -*	*1:*	*1:*
Qs0 :	*03 -*	*1:*	*1*	*Qd0 :*	*06 -*	*0:*	*1:*
Qp1 :	*04 -*	*1:*	*1*	----			

SWAP FILE	
s0: ----	s1: *Ps0*
s2: *Pd0 / Qd0*	s3: *Pp0*
s4: *Pp1*	s5: *Pp2*

LRU ACTIVE: *QP1, QP0, PD0, PS0, PC0,* _____

LRU INACTIVE: *qc0, qs0, qd0,* _____

spazio libero per brutta copia o continuazione

seconda parte – file system

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

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

Si consideri la seguente **situazione iniziale**, con il processo **P** in esecuzione:

```
____MEMORIA FISICA____(pagine libere: 5)_____________________________
      00 : <ZP>                  ||  01 : Pc0 / <X, 0>          ||
      02 : Pp0                   ||  03 : ----                  ||
      04 : ----                  ||  05 : ----                  ||
      06 : ----                  ||  07 : ----                  ||

____STATO del TLB_____________________________________________________
     Pc0 : 01 -  0: 1:           || Pp0 : 02 -  1: 1:           ||
           ----                  ||        ----                 ||
           ----                  ||        ----                 ||
```

eventi 1 e 2: fd = *open* (F), *write* (fd, 4000)

MEMORIA FISICA	
00: <ZP>	01: Pc0 /<X, 0>
02: *Pp0*	03: *<F, 0> D*
04: *----*	05: *----*
06: *----*	07: *----*

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	*4000*	*1*	*1*	*0*

eventi 3 e 4: *fork* (Q), *fork* (R), *context switch* (R)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02: *Qp0 D*	03: *<F, 0> D*
04: *Rp0 D*	05: *Pp0 D*
06: *----*	07: *----*

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	*4000*	*3*	*1*	*0*

evento 5: *write* **(fd, 5000)**

R scrive in F0, F1 e F2. F1 caricata in 06 e scritta, per F2 scatta PFRA che libera NPF 3 e NPF 6 (quindi due scritture su disco). F2 caricata e scritta in NPF 3.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02: *Qp0 D*	03: *<F, 2> D*
04: *Rp0 D*	05: *Pp0 D*
06: *----*	07: *----*

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	*9000*	*3*	*3*	*2*

eventi 6, 7 e 8: *close* **(fd),** *context switch* **(Q),** *lseek* **(fd, –6000)** // offset neg.

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	*3000*	*2*	*3*	*2*

eventi 9 e 10: *read* **(fd, 10),** *close* **(fd)**

Q legge 10 byte nella pagina F0 che deve essere caricata e poi chiude il file.

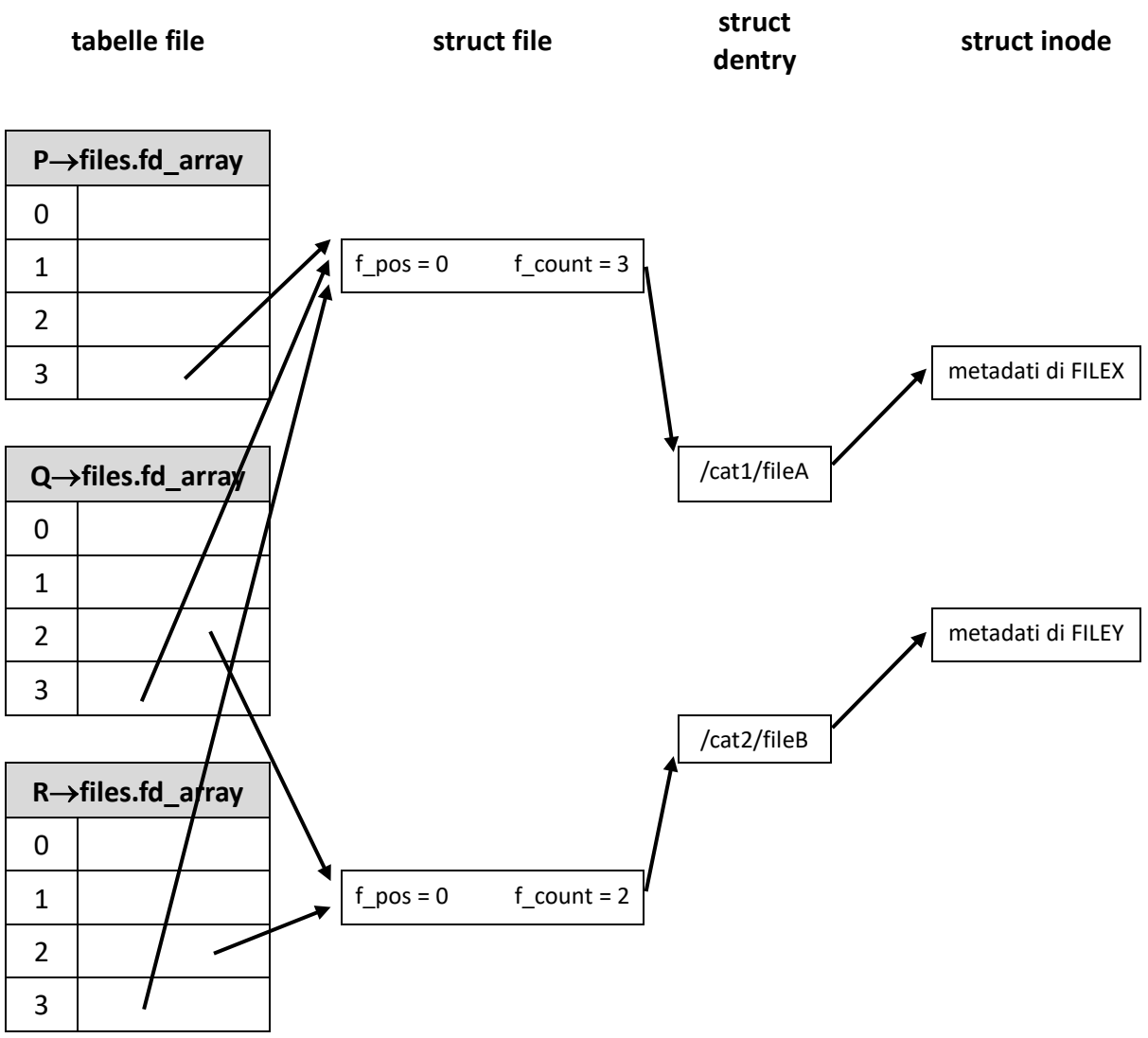
MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02: *Qp0 D*	03: *<F, 2> D*
04: *Rp0 D*	05: *Pp0 D*
06: *<F, 0>*	07: *----*

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	*3010*	*1*	*4*	*2*

esercizio n. 4 – domande su argomenti vari

strutture dati del file system

La figura sottostante è una rappresentazione dello stato del VFS raggiunto dopo l'esecuzione in sequenza di un certo numero di chiamate di sistema sotto riportate.



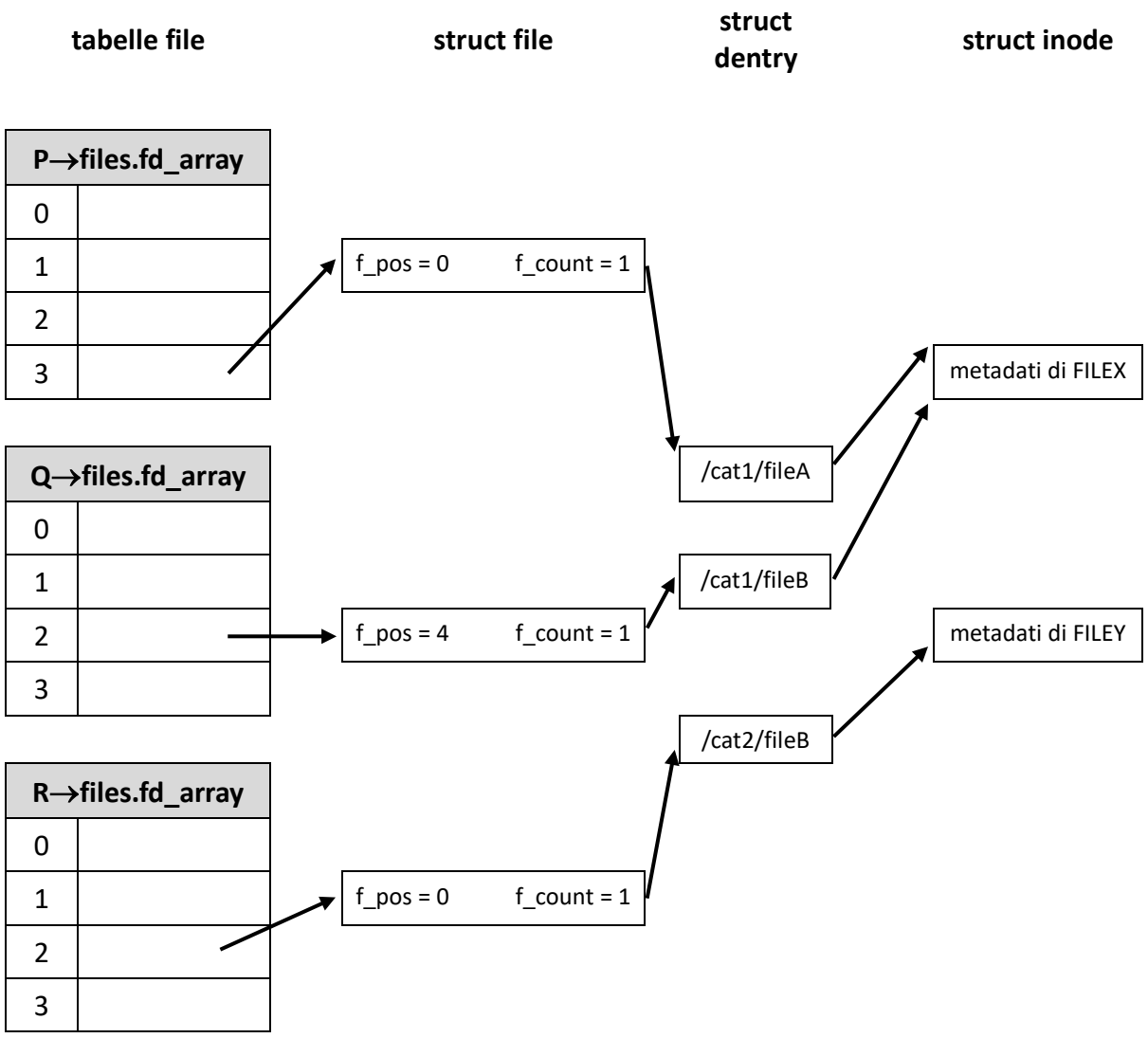
chiamate di sistema eseguite nell'ordine indicato

1) **P** fd1 = *open* ("/cat1/fileA", ...)
2) **P** *close (2)*
3) **P** pid = *fork* ( ) // il processo P crea il processo figlio Q
4) **Q** fd2 = *open* ("/cat2/fileB", …)
5) **Q** pid = *fork* ( ) // il processo Q crea il processo figlio R

Ora si supponga di partire dallo stato del VFS mostrato nella figura iniziale e si risponda alla **domanda** alla pagina seguente, riportando la **sequenza di chiamate di sistema** che può avere generato la nuova situazione di VFS mostrata nella figura successiva. Valgono questi vincoli:

- i soli tipi di **chiamata** da considerare sono: **`close, link, open, read`**
- lo **scheduler** mette in esecuzione i processi in questo ordine: **Q**, **R**

domanda



sequenza di chiamate di sistema (numero di righe non significativo)

#	processo	chiamata di sistema
1	*Q*	*close (fd1) // close (3)*
2	*Q*	*close (fd2) // close (2)*
3	*Q*	*link ("/cat1/fileA", "/cat1/fileB")*
4	*Q*	*fd3 = open ("/cat1/fileB", ...)*
5	*Q*	*read (fd3, 4)*
6	*R*	*close (fd1) // close (3)*
7		
8		

Le tre chiamate alle righe 1, 2 e 3 sono commutabili; l'ordine di esecuzione dei processi è deciso dallo scheduler ed è specificato nel testo dell'esercizio, e le tre variabili fd (i descrittori di file) sono indipendenti per i tre processi P, Q e R (i quali infatti sono processi normali che non condividono variabili).

spazio libero per brutta copia o continuazione