



Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri
prof. Gerardo Pelosi

prof.ssa Donatella Sciuto
prof.ssa Cristina Silvano

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – lunedì 17 luglio 2023

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 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 rough, smooth
sem_t wavy
int global = 0
```

```
void * plane (void * arg) {
    mutex_lock (&smooth)
    sem_wait (&wavy)
    global = 1
    mutex_unlock (&smooth)
```

```
    global = 2                                     /* statement A */
```

```
    mutex_lock (&rough)
    sem_post (&wavy)
    mutex_unlock (&rough)
    return NULL
```

```
} /* end plane */
```

```
void * edge (void * arg) {
    mutex_lock (&smooth)
    sem_post (&wavy)
    mutex_unlock (&smooth)
```

```
    global = 3                                     /* statement B */
```

```
    mutex_lock (&rough)
    sem_wait (&wavy)
```

```
    global = 4                                     /* statement C */
```

```
    mutex_unlock (&rough)
    return NULL
```

```
} /* end edge */
```

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&wavy, 0, 0)
    create (&th_2, NULL, edge, NULL)
    create (&th_1, NULL, plane, NULL)
    join (th_2, NULL)
```

```
    join (th_1, NULL)                             /* statement D */
```

```
    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 – plane	th_2 – edge
subito dopo stat. A	<i>ESISTE</i>	<i>ESISTE</i> (già creato e non terminato)
subito dopo stat. B	<i>PUÒ ESISTERE</i>	<i>ESISTE</i>
subito dopo stat. C	<i>PUÒ ESISTERE</i>	<i>ESISTE</i>
subito dopo stat. D	<i>NON ESISTE</i>	<i>NON ESISTE</i>

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		
	<i>rough</i>	<i>smooth</i>	<i>wavy</i>
subito dopo stat. A	<i>0 / 1</i>	<i>0</i>	<i>0</i>
subito dopo stat. B	<i>0 / 1</i>	<i>0 / 1</i>	<i>0 / 1</i>
subito dopo stat. C	<i>1</i>	<i>0 / 1</i>	<i>0</i>

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

caso	th_1 – plane	th_2 – edge	<i>global</i>
1	<i>wait wavy</i>	<i>lock smooth</i>	<i>0</i>
2	<i>lock rough</i>	<i>wait wavy</i>	<i>2 / 3</i>
3	<i>wait wavy</i>	---	<i>4</i>
4			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

// programma ola.c		
int main () {		
pid1 = fork ()	// creazione del processo R e S	
if (pid1 == 0) {	// codice eseguito da R e S	
execl ("/acso/ola", "ola", NULL)		
exit (-1)		
} else {	// codice eseguito da Q e R(dopo mutazione)	
write (stdout, "Alzo le braccia", 16)		
} /* if */		
exit (0)		
}		
// programma gioco.c		
sem_t palla, base		
void * battitore (void * arg) {		void * ricevitore (void * arg) {
sem_post (&palla)		sem_wait (&palla)
sem_wait (&base)		sem_wait (&base)
return NULL		return NULL
}		}
int main () { // codice eseguito da P		
pthread_t TH_1, TH_2		
sem_init (&palla, 0, 0)		
sem_init (&base, 0, 1)		
pthread_create (&TH_2, NULL, ricevitore, NULL)		
pthread_create (&TH_1, NULL, battitore, NULL)		
pthread_join (TH_2, NULL)		
pthread_join (TH_1, NULL)		
exit(1)		
}		

Un processo **Q** esegue il programma `ola.c`. Nella simulazione considerata per l'esercizio, vengono creati i processi **R** (da **Q**) e **S** (da **R**) che eseguono con successo una mutazione di codice (allo stesso codice).

Il processo **P** esegue il programma `gioco.c` e crea i thread **TH_1** e **TH_2**.

Nella situazione iniziale il processo **P** e il processo **Q** esistono, ma non hanno ancora effettuato nessuna chiamata a sistema o di libreria.

Nell'istante iniziale il processo **P** è in esecuzione, mentre il processo **Q** è in stato di pronto da più tempo.

Si simuli l'esecuzione dei processi così come risulta dal codice dato, dagli eventi indicati.

Si completi la tabella riportando quanto segue:

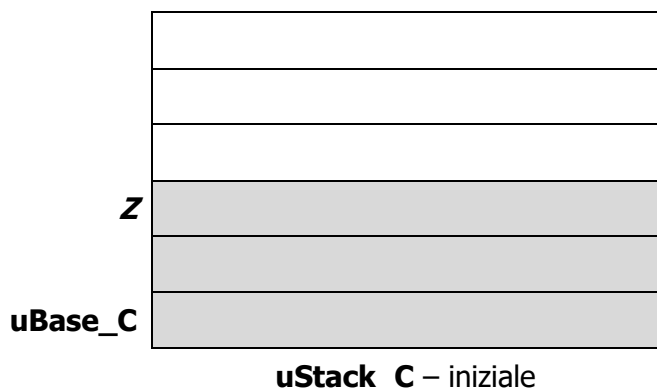
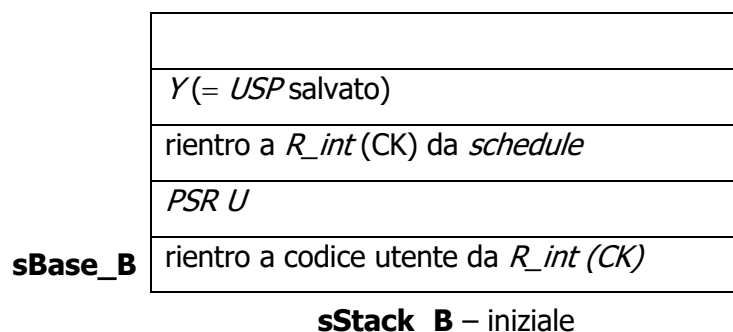
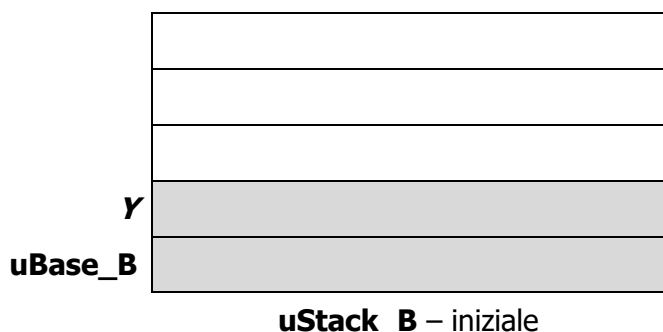
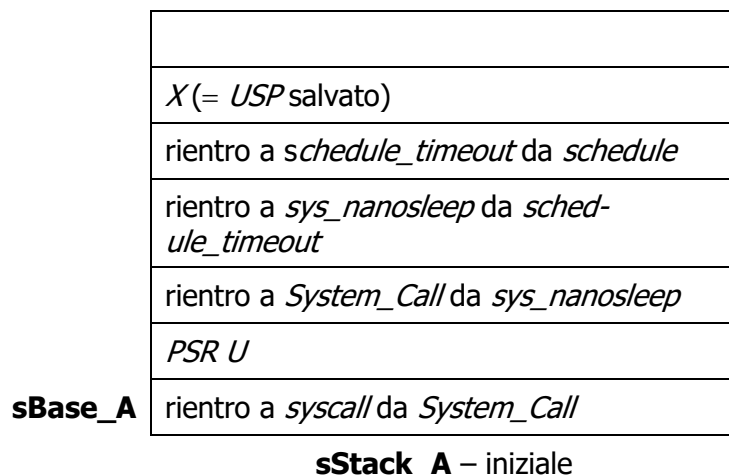
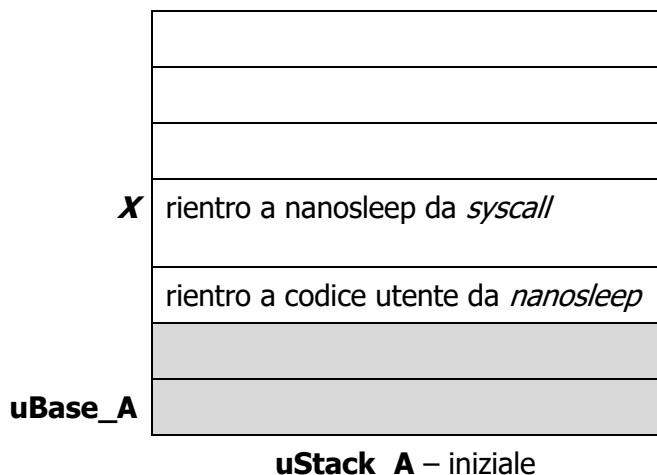
- PID e TGID di ogni processo che viene creato
- identificativo del processo-chiamata di sistema / libreria nella prima colonna, dove necessario
- in funzione del codice proposto in ciascuna riga, lo stato dei processi **al termine del tempo indicato**

TABELLA DA COMPILARE

identificativo simbolico del processo		IDLE	P	Q	TH_2	TH_1	R	S
evento oppure processo-chiamata	PID	1	2	3	4	5	6	7
	TGID	1	2	3	2	2	6	7
P – pthread_create	1	pronto	exec	pronto	<i>pronto</i>			
<i>P - pthread_create</i>	2	<i>pronto</i>	<i>exec</i>	<i>pronto</i>	<i>pronto</i>	<i>pronto</i>		
<i>P - pthread_join</i>	3	<i>pronto</i>	<i>attesa (th_2)</i>	<i>exec</i>	<i>pronto</i>	<i>pronto</i>		
<i>Q - fork</i>	4	<i>pronto</i>	<i>attesa (th_2)</i>	<i>exec</i>	<i>pronto</i>	<i>pronto</i>	<i>pronto</i>	
<i>Q - write</i>	5	<i>pronto</i>	<i>attesa (th_2)</i>	<i>attesa (stdout)</i>	<i>exec</i>	<i>pronto</i>	<i>pronto</i>	
<i>TH_2 - sem_wait</i>	6	<i>pronto</i>	<i>attesa (th_2)</i>	<i>attesa (stdout)</i>	<i>attesa (palla)</i>	<i>exec</i>	<i>pronto</i>	
<i>TH_1 - sem_post</i>	7	<i>pronto</i>	<i>attesa (th_2)</i>	<i>attesa (stdout)</i>	<i>exec</i>	<i>pronto</i>	<i>pronto</i>	
16 interrupt da stdout (operazione comple- tata)	8	<i>pronto</i>	<i>attesa (th_2)</i>	<i>exec</i>	<i>pronto</i>	<i>pronto</i>	<i>pronto</i>	
<i>Q - exit</i>	9	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>pronto</i>	<i>pronto</i>	<i>exec</i>	
<i>R - execl</i>	10	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>pronto</i>	<i>pronto</i>	<i>exec</i>	
<i>R - fork</i>	11	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>pronto</i>	<i>pronto</i>	<i>exec</i>	<i>pronto</i>
<i>R - write</i>	12	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>pronto</i>	<i>exec</i>	<i>attesa (stdout)</i>	<i>pronto</i>
<i>TH_1 - sem_wait</i>	13	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>pronto</i>	<i>exec</i>	<i>attesa (stdout)</i>	<i>pronto</i>
<i>TH_1 - return</i>	14	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>exec</i>	<i>NE</i>	<i>attesa (stdout)</i>	<i>pronto</i>
<i>TH_2 - sem_wait</i>	15	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>attesa (base)</i>	<i>NE</i>	<i>attesa (stdout)</i>	<i>exec</i>
<i>16 interrupt da stdout (operazione completata)</i>	16	pronto	attesa (th_2)	NE	attesa (base)	NE	exec	pronto
<i>R - exit</i>	17	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>attesa (base)</i>	<i>NE</i>	<i>NE</i>	<i>exec</i>
<i>S - execl</i>	18	<i>pronto</i>	<i>attesa (th_2)</i>	<i>NE</i>	<i>attesa (base)</i>	<i>NE</i>	<i>NE</i>	<i>exec</i>

seconda parte – moduli di nucleo

Si considerino i task **A**, **B** e **C**. Lo stato delle loro pile di sistema e utente è il seguente:



domanda 1 - Si indichi lo stato di ciascun task, così come è deducibile dallo stato iniziale delle pile, specificando anche l'evento o la chiamata di sistema che ha portato il task in tale stato:

A: *attesa di un certo lasso di tempo, tramite la funzione nanosleep*

B: *pronto, ha subito preemption per scadenza quanto di tempo*

C: *è in esecuzione*

domanda 2 – A partire dallo stato iniziale descritto, si consideri l'evento sotto specificato. **Si mostrino** le invocazioni di tutti i **moduli** (e eventuali relativi ritorni) per la gestione dell'evento stesso (precisando processo e modo) e il **contenuto delle pile** utente e di sistema richieste.

NOTAZIONE da usare per i moduli: > (invocazione), nome_modulo (esecuzione), < (ritorno)

EVENTO: *interrupt* dal clock e **scadenza del timeout** (a seguito dell'evento il task **A** ha maggiori diritti di esecuzione di tutti gli altri task in *runqueue*). Completare la lista dei moduli fino a quando si ritorna a eseguire codice utente

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

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

<i>processo</i>	<i>modo</i>	<i>modulo</i>
C	U	codice utente
<i>C</i>	<i>U → S</i>	<i>>R_int_clock</i>
<i>C</i>	<i>S</i>	<i>>task_tick<</i>
<i>C</i>	<i>S</i>	<i>>Controlla_timer</i>
<i>C</i>	<i>S</i>	<i>>wake_up_process</i>
<i>C</i>	<i>S</i>	<i>>check_preempt_curr</i>
<i>C</i>	<i>S</i>	<i>>resched<</i>
<i>C</i>	<i>S</i>	<i>wake_up_process<</i>
<i>C</i>	<i>S</i>	<i>Controlla_timer<</i>
<i>C</i>	<i>S</i>	<i>(tornato in R_int_clock)</i>
<i>C</i>	<i>S</i>	<i>>schedule</i>
<i>C</i>	<i>S</i>	<i>>pick_next_task<</i>
<i>C → A</i>	<i>S</i>	<i>schedule: context_switch</i>
<i>A</i>	<i>S</i>	<i>schedule<</i>
<i>A</i>	<i>S</i>	<i>schedule_timeout<</i>
<i>A</i>	<i>S</i>	<i>sys_nanosleep<</i>
<i>A</i>	<i>S → U</i>	<i>system_call< : SYSRET</i>
<i>A</i>	<i>U</i>	<i>syscall<</i>
<i>A</i>	<i>U</i>	<i>nanosleep<</i>
<i>A</i>	<i>U</i>	<i>codice utente</i>

contenuto della pila

Z	
uBase_C	

uStack_C

	<i>Z (= USP)</i>
	<i>Rientro a schedule da pick_next_task</i>
	<i>Rientro a R_int_clock da schedule</i>
	<i>Rientro a check_preempt_curr da resched</i>
	<i>Rientro a wake_up da check_preempt_curr</i>
	<i>Rientro a Controlla_timer da wake_up</i>
	<i>Rientro a R_int_clock da Controlla_timer</i>
	<i>Rientro a R_int_clock da task_tick</i>
	<i>PSR (U)</i>
sBase_C	<i>Rientro a codice utente da R_int_clock</i>

sStack_C

esercizio n. 3 – memoria virtuale e file system

prima parte – memoria virtuale

È dato un sistema di memoria caratterizzato dai seguenti parametri generali

MAXFREE = 4, MINFREE = 3

Situazione iniziale (esistono due processi P e Q)

PROCESSO: P *** (non di interesse) *******

PROCESSO: Q *****

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 :4 D R> <d0 :- -> <d1 :- -> <d2 :- ->
<p0 :2 D R> <p1 :5 D W> <p2 :- ->

process Q - NPV of PC and SP: c1, p1

MEMORIA FISICA (pagine libere: 3)

00 : <ZP>	01 : Pc1/Qc1/<X,1>
02 : Pp0/Qp0 D	03 : <X,2>
04 : Ps0/Qs0 D	05 : Qp1 D
06 : Pp1 D	07 : ----
08 : ----	09 : ----

STATO del TLB

Qc1 : 01 - 0: 1:	Qp1 : 05 - 1: 1:
----	----
----	----
----	----

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

LRU ACTIVE: QP1, QC1, PP1, PC1,

LRU INACTIVE: qs0, qp0, ps0, pp0,

evento 1: *write* (Qs0)

La scrittura di Ps0 causa un COW che richiede una pagina;

interviene PFRA con Required:1, Free:3, To Reclaim:2

la prima pagina liberata è NPF=3 (da Page Cache, pagina di file non utilizzata al momento)

la seconda pagina liberata è NPF=2, perché Qp0/Pp0 sono in inactive → scrittura in swap file perché Pp0/Qp0 è marcato D

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <X, 1>
02: Pp0/Qp0 D Qs0	03: <X, 2>
04: Ps0/ Qs0 D	05: Qp1 D
06: Pp1 D	07:
08:	09: ---

TLB

NPV	NPF	D	A	NPV	NPF	D	A
Qc1:	01	-	0: 1:	Qp1:	05	-	1: 1:
Qs0:	02	-	1: 1:				

SWAP FILE							
s0:	Pp0	/	Qp0	s1:			
s2:				s3:			

LRU INACTIVE: *qs0, ps0* _____

evento 2: *mmap* (0x 000050000000, 3, W, P, M, "F", 2),

VMA del processo Q (compilare solo la riga relativa alla nuova VMA creata)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
M0	0000 5000 0	3	W	P	M	F	2

PT del processo: Q				
s0: <i>2 W</i>	p0: <i>s0 R</i>	m00: <i>- -</i>	m01: <i>- -</i>	m02: <i>- -</i>

evento 3: read (Qm02) write (Qm00)

Qm02 viene caricata in memoria in NPF 3 e con meccanismo COW senza altri effetti, poi la richiesta di scrittura di Qm00 causa l'intervento di PFRA con Required:1, Free:3, To Reclaim:2; vengono liberate da inactive Ps0 (NPF=4) e Qs0 (NPF=2)

Ambedue devono essere scritte su SWAP file (Ps0 è marcata dirty nella TP e Qs0 è dirty nel TLB). Ci sono 5 pagine libere e Qm00/<F,2> viene caricata in NPF 02 e messa R in TP per far scattare COW. Quando si esegue la scrittura si attiva COW, ci sono 4 pagine libere e Qm00 viene allocata in NPF 04 e scritta, mentre <F,2> rimane in 02.

PT del processo: Q				
s0: <i>s1 W</i>	p0: <i>s0 R</i>	m00: <i>4 W</i>	m01: <i>- -</i>	m02: <i>3 R</i>

MEMORIA FISICA	
00: <i><ZP></i>	01: <i>Pc1 / Qc1 / <X, 1></i>
02: <i>Qs0</i> <i><F, 2></i>	03: <i>Qm02 / <F, 2></i>
04: <i>Ps0</i> <i>D</i> <i>Qm00</i>	05: <i>Qp1 D</i>
06: <i>Pp1 D</i>	07:
08:	09: <i>---</i>

SWAP FILE	
s0: <i>Pp0 / Qp0</i>	s1: <i>Ps0</i>
s2: <i>Qs0</i>	s3:

LRU ACTIVE: *QM00, QM02, P1, QC1, PP1, PC1,* _____

LRU INACTIVE: _____

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: 3)			
00 : <ZP>		01 : Pc0 / Qc0 / <X, 0>	
02 : Pp0 / Qp0		03 : Qp1 D	
04 : Pp1		05 : ----	
06 : ----		07 : ----	

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 sempre in esecuzione il processo **P**.

ATTENZIONE: il numero di pagine lette o scritte è cumulativo, quindi è la somma delle pagine lette o scritte da tutti gli eventi precedenti oltre a quello considerato.

evento 1 e 2 – **fd = open (F) read (fd, 6500)**

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Qp1 D
04: Pp1	05: <F, 0>
06: <F, 1>	07: ----

f_pos	f_count	numero pagine lette	numero pagine scritte
6500	1	2	0

evento 3 – *write* (fd, 2000)

per poter caricare in memoria e scrivere <F,2> interviene PFRA (Required:1, Free:1, To Reclaim:2) e libera da Page Cache le pagine con NPF = 5 e 6 e <F,2> viene allocata e scritta in 05

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X,0>
02: Pp0 / Qp0	03: Qp1 D
04: Pp1	05: <F,0> <F,2>, D
06: <F,1>	07: ----

f_pos	f_count	numero pagine lette	numero pagine scritte
8500	1	3	1

evento 4 – *write* (fd, 5000)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X,0>
02: Pp0 / Qp0	03: Qp1 D
04: Pp1	05: <F,2>, D
06: <F,3> D	07: ----

f_pos	f_count	numero pagine lette	numero pagine scritte
13500	1	4	1

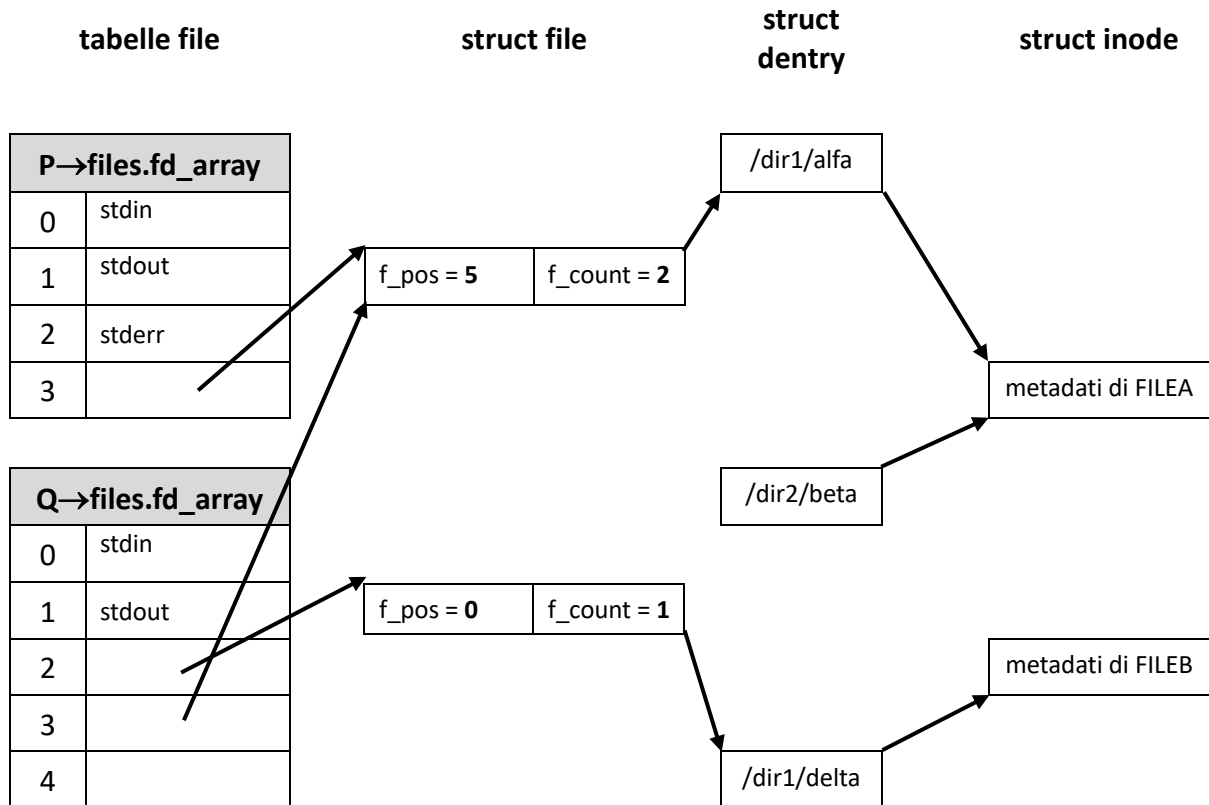
evento 5 – *close* (fd)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X,0>
02: Pp0 / Qp0	03: Qp1 D
04: Pp1	05: <F,2> D
06: <F,3> D	07: ----

f_pos	f_count	numero pagine lette	numero pagine scritte
		4	3

esercizio n. 4 – 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 (non riportate):

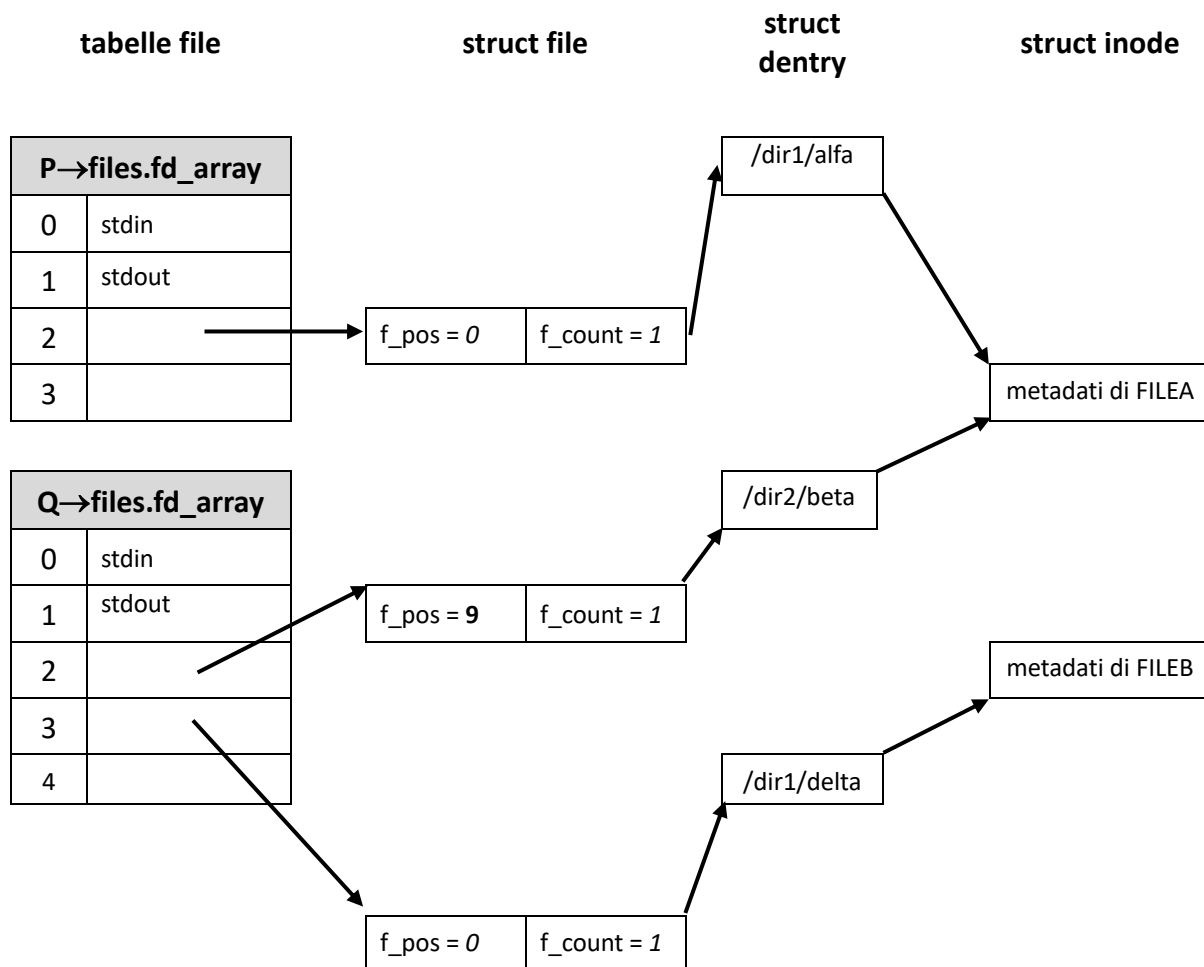


Il task (normale) **P** ha creato il task (normale) figlio **Q**.

Ora si supponga di partire dallo stato del VFS mostrato nella figura iniziale e si risponda alla **domanda** alla pagina seguente, riportando nella tabella finale **già parzialmente compilata, una possibile sequenza di chiamate di sistema** che può generare la nuova situazione di VFS mostrata nella figura successiva; si indichino anche i **valori mancanti dei campi f_pos e f_count**. Il numero di eventi da riportare è esattamente 7 (in aggiunta a quello già dato) ed essi sono eseguiti, nell'ordine, dai task indicati.

Le sole chiamate di sistema usabili sono: *open* (nomefile, ...), *read* (numfd, numchar), *close* (numfd).

domanda: si scriva la sequenza di chiamate che produce la nuova situazione qui sotto, e si indichino anche i valori mancanti di f_pos e f_count



sequenza di chiamate di sistema – UN'OPERAZIONE È GIÀ DATA

#	processo	chiamata di sistema
1	P	<i>close (3)</i>
2	P	<i>close (2)</i>
3	P	<i>open ("/dir1/alpha", ...)</i>
4	Q	<i>close (2)</i>
5	Q	<i>open ("/dir2/beta", ...)</i>
6	Q	<i>read (2, 9)</i>
7	Q	<i>close (3)</i>
8	Q	<i>open ("/dir1/delta", ...)</i>

Nota: le prime due close di P sono interscambiabili.

spazio libero per brutta copia o continuazione