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 – 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	***thread***	
	th_1 – *plane*	**th_2 – *edge***
subito dopo stat. **A**	*ESISTE*	*ESISTE* *(già creato e non terminato)*
subito dopo stat. **B**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **C**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **D**	*NON ESISTE*	*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**		
	rough	*smooth*	*wavy*
subito dopo stat. **A**	*0 / 1*	*0*	*0*
subito dopo stat. **B**	*0 / 1*	*0 / 1*	*0 / 1*
subito dopo stat. **C**	*1*	*0 / 1*	*0*

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***	*global*
1	*wait wavy*	*lock smooth*	*0*
2	*lock rough*	*wait wavy*	*2 / 3*
3	*wait wavy*	---	*4*
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**	**esec**	**pronto**	*pronto*			
P - pthread_create	**2**	*pronto*	*esec*	*pronto*	*pronto*	*pronto*		
P - pthread_join	**3**	*pronto*	*attesa (th_2)*	*esec*	*pronto*	*pronto*		
Q - fork	**4**	*pronto*	*attesa (th_2)*	*esec*	*pronto*	*pronto*	*pronto*	
Q - write	**5**	*pronto*	*attesa (th_2)*	*attesa (stdout)*	*esec*	*pronto*	*pronto*	
TH_2 - sem_wait	**6**	*pronto*	*attesa (th_2)*	*attesa (stdout)*	*attesa (palla)*	*esec*	*pronto*	
TH_1 - sem_post	**7**	*pronto*	*attesa (th_2)*	*attesa (stdout)*	*esec*	*pronto*	*pronto*	
16 interrupt da stdout (operazione comple-tata)	**8**	*pronto*	*attesa (th_2)*	*esec*	*pronto*	*pronto*	*pronto*	
Q - exit	**9**	*pronto*	*attesa (th_2)*	*NE*	*pronto*	*pronto*	*esec*	
R - execl	**10**	*pronto*	*attesa (th_2)*	*NE*	*pronto*	*pronto*	*esec*	
R - fork	**11**	*pronto*	*attesa (th_2)*	*NE*	*pronto*	*pronto*	*esec*	*pronto*
R - write	**12**	*pronto*	*attesa (th_2)*	*NE*	*pronto*	*esec*	*attesa (stdout)*	*pronto*
TH_1 - sem_wait	**13**	*pronto*	*attesa (th_2)*	*NE*	*pronto*	*esec*	*attesa (stdout)*	*pronto*
TH_1 - return	**14**	*pronto*	*attesa (th_2)*	*NE*	*esec*	*NE*	*attesa (stdout)*	*pronto*
TH_2 - sem_wait	**15**	*pronto*	*attesa (th_2)*	*NE*	*attesa (base)*	*NE*	*attesa (stdout)*	*esec*
16 interrupt da stdout (operazione completata)	**16**	**pronto**	**attesa (th_2)**	**NE**	**attesa (base)**	**NE**	**esec**	**pronto**
R - exit	**17**	*pronto*	*attesa (th_2)*	*NE*	*attesa (base)*	*NE*	*NE*	*esec*
S - execl	**18**	*pronto*	*attesa (th_2)*	*NE*	*attesa (base)*	*NE*	*NE*	*esec*

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:

X	rientro a nanosleep da *syscall*
	rientro a codice utente da *nanosleep*
uBase_A	

uStack_A – iniziale

	X (= *USP* salvato)
	rientro a s*chedule_timeout* da *schedule*
	rientro a *sys_nanosleep* da *sched-ule_timeout*
	rientro a *System_Call* da *sys_nanosleep*
	PSR U
sBase_A	rientro a *syscall* da *System_Call*

sStack_A – iniziale

Y	
uBase_B	

uStack_B – iniziale

	Y (= *USP* salvato)
	rientro a *R_int* (CK) da *schedule*
	PSR U
sBase_B	rientro a codice utente da *R_int (CK)*

sStack_B – iniziale

Z	
uBase_C	

uStack_C – iniziale

sBase_C

sStack_C – iniziale

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.)

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

contenuto della pila

Z	
uBase_C	

uStack_C

Z (= USP)
~~*Rientro a schedule da pick_next_task*~~
Rientro a R_int_clock da schedule
~~*Rientro a check_preempt_curr da resched*~~
~~*Rientro a wake_up da check_preempt_curr*~~
~~*Rientro a Controlla_timer da wake_up*~~
~~*Rientro a R_int_clock da Controlla_timer*~~
~~*Rientro a R_int_clock da task_tick*~~
PSR (U)
Rientro a codice utente da R_int_clock

sBase_C

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 liberate 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: *2 W*	p0: *s0 R*	*m00: – –*	*m01: – –*	*m02: – –*

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 i NPF 04 e scritta, mentre <F,2> rimane in 02.

PT del processo: **Q**				
s0: *s1 W*	p0: *s0 R*	m00: *4 W*	m01: – –	m02: *3 R*

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <X, 1>
02: ~~*Qs0 Qm00*~~*<F,2>*	03: *Qm02/<F,4>*
04: ~~*PS0 D*~~ *Qm00*	05: *Qp1 D*
06: *Pp1 D*	07:
08:	09: ---

SWAP FILE	
s0: *Pp0 / Qp0*	s1: *Ps0*
s2: *Qs0*	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*

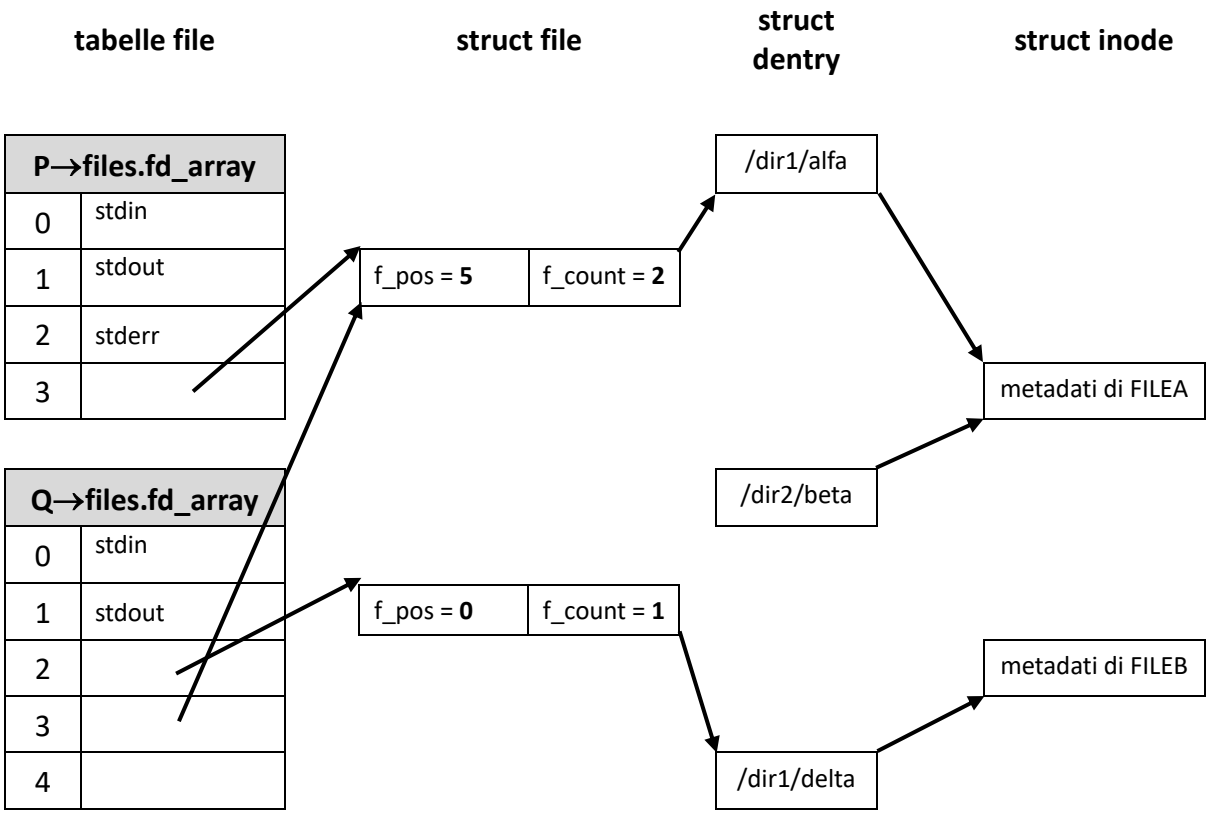
evento 5 – *close* (fd)

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):

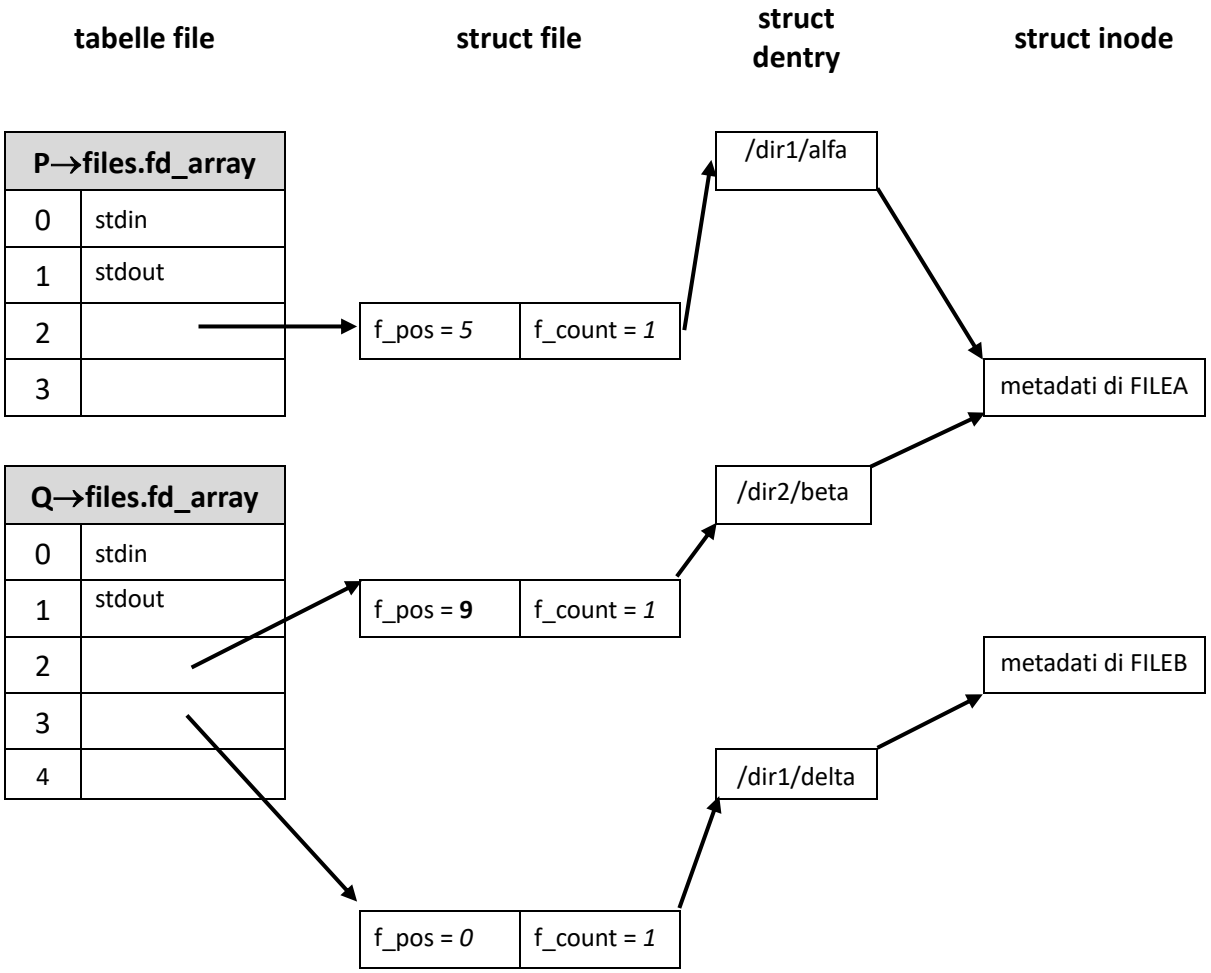


Il task (normale) **P** ha creato il task (normale) figlio **Q**.

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	*close (3)*
2	P	*close (2)*
3	P	*open ("/dir1/alpha", ...)*
4	Q	*close (2)*
5	Q	*open ("/dir2/beta", ...)*
6	Q	*read (2, 9)*
7	Q	close (3)
8	Q	*open ("/dir1/delta", ...)*

Nota: le prime due close di P sono interscambiabili.