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

Prova di mercoledì 11 gennaio 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 **2 h : 00 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) _____

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 deep, shallow
sem_t water
int global = 0
```

```
void * deep (void * arg) {
   mutex_lock (&shallow)
   sem_post (&water)
   mutex_lock (&deep)
   global = 1                                          /* statement A */
   mutex_unlock (&shallow)
   mutex_unlock (&deep)
   global = 2
   sem_wait (&water)
   return (void *) 9
} /* end deep */
```

```
void * shallow (void * arg) {
   mutex_lock (&deep)
   mutex_lock (&shallow)
   sem_wait (&water)
   mutex_unlock (&shallow)
   global = 3                                          /* statement B */
   sem_post (&water)
   global = 4                                          /* statement C */
   mutex_unlock (&deep)
   return (void *) 5
} /* end shallow */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&water, 0, 0)
   create (&th_2, NULL, shallow, NULL)
   create (&th_1, NULL, deep, NULL)
   join (th_2, &global)                                /* statement D */
   join (th_1, &global)
   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 – *deep*	**th_2 – *shallow***
subito dopo stat. **A**		
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D**		

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**			
	deep	*shallow*	*water*	*global*
subito dopo stat. **A**				
subito dopo stat. **B**				
subito dopo stat. **C**				
subito dopo stat. **D**				

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 **due casi**, indicando gli statement dove avvengono i blocchi e i possibili valori della variabile *global*:

caso	**th_1 – *deep***	**th_2 – *shallow***	*global*
1			
2			
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma gara.c
sem_t pronti
sem_t flag
void * bianco (void * arg) {
    char str[7] = "bianco\n"
    sem_post (&pronti)
    sem_wait (&flag)
    write (stdout, str, 7)
    return NULL
} // end bianco
```

```
void * nero (void * arg) {
    char str[5] = "nero\n"
    sem_post (&pronti)
    sem_wait (&flag)
    read (stdin, str, 5)
    return NULL
} // end nero
```

```
void * arbitro (void *arg) {
    sem_wait (&pronti)
    sem_wait (&pronti)
    sem_post (&flag)
    return NULL
} // end arbitro
int main ( ) { // codice eseguito da P
    pthread_t TH_1, TH_2, TH_3
    sem_init (&pronti, 0, 0)
    sem_init (&flag, 0, 0)
    pthread_create (&TH_1, NULL, bianco, NULL)
    pthread_create (&TH_2, NULL, nero, NULL)
    pthread_create (&TH_3, NULL, arbitro, NULL)
    pthread_join (TH_1, NULL)
    pthread_join (TH_2, NULL)
    pthread_join (TH_3, NULL)
    exit (1)
} // end main
```

Un processo **P** esegue il programma **gara.c** e crea i tre thread **TH_1**, **TH_2** e **TH_3**. Si simuli l'esecuzione dei vari processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale e dagli eventi indicati. **NB: la parte finale della simulazione va sviluppata in due casi diversi.**

Si completi la tabella seguente riportando:

- ⟨ PID, TGID ⟩ di ciascun 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	TH_1	TH_2	TH_3
evento oppure processo-chiamata	PID	1	2			
	TGID	1	2			
P – pthread_create TH_1	0	pronto	esec	pronto	NE	NE
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					

caso 1: sequenza finale qualora lo stato di attesa sul semaforo *flag* sia di tipo ESCLUSIVO, pertanto i thread vengano risvegliati in ordine di ingresso nella coda di attesa

TH_3 – sem_post (flag)	10					
	11					
	12	pronto				pronto

caso 2: sequenza finale alternativa qualora lo stato di attesa sul semaforo *flag* sia di tipo NON ESCLUSIVO, pertanto i thread TH_1 e TH2 vengano risvegliati insieme, e lo scheduler scelga di rimettere in esecuzione TH_2 prima di TH_1

TH_3 – sem_post (flag)	10					
	11					pronto
	12					
	13				esec	pronto

seconda parte – scheduling dei processi

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

CONDIZIONI INIZIALI (già complete)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	4	T1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T1	1	0,25	1,5	1	20	100
RB	T2	3	0,75	4,5	0,33	30	102

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

```
Events of task t1:  CLONE at 0.5;  WAIT at 1.0  WKUP after 3.0
```

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

EVENTO 1		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
		6					
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT							
RB							
WAITING							

EVENTO 2		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
		6					
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT							
RB							
WAITING							

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
		6					
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
		6					
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

Calcolo del VRT iniziale del **task T3** creato dalla **CLONE**

Valutazione della cond. di rescheduling alla **CLONE**

Valutazione della cond. di rescheduling alla **WAKEUP**

esercizio n. 3 – memoria e file system

prima parte – gestione dello spazio di memoria

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  000000400,  2 , R  ,  P  ,  M  , <CC,0>
          D  000000600,  2 , W  ,  P  ,  A  , <-1,0>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
    PT: <c0 :- ->  <c1 :1 R>  <d0 :3 R>  <d1 :- ->
        <p0 :4 W>  <p1 :- ->  <p2 :- ->
    process P - NPV of PC and SP:  c1, p0

PROCESSO: Q  ************************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <CC,0>
          D  000000600,  2 , W  ,  P  ,  A  , <-1,0>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
    PT: <c0 :- ->    <c1 :1 R>  <d0 :3  R>  <d1 :- ->
        <p0 :2 D W>  <p1 :- ->  <p2 :- ->
    process Q - NPV of PC and SP:  c1, p0

____MEMORIA FISICA____(pagine libere: 5)_____________________________
      00 : <ZP>                ||  01 : Pc1 / Qc1 / <CC,1>  ||
      02 : Qp0  D              ||  03 : Pd0 / Qd0           ||
      04 : Pp0                 ||  05 : ----                ||
      06 : ----                ||  07 : ----                ||
      08 : ----                ||  09 : ----                ||

____STATO del TLB_____________________________________________________
      Pc1 : 01 -  0: 1:        ||   Pp0 : 04 -  1: 1:       ||
      Pd0 : 03 -  1: 1:        ||         -----             ||

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

LRU ACTIVE:    QD0,  QP0,  QC1,  PD0,  PP0,  PC1

LRU INACTIVE:
```

evento 1: *read* (Pc0) – *write* (Pp0) – 4 *kswapd*

PT del processo: **P**				
c0:	c1:	d0:	d1:	p0:
p1:	p2:			

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

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <CC, 1>
02: Qp0 D	03: Pd0 / Qd0
04: Pp0	05:
06:	07:
08:	09:

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 2: *mmap* (0x 0000 3000 0000, 2, W, P, M, FF, 0)

VMA del processo **P** (è da compilare solo la riga relativa alla VMA **M0**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
M0							

evento 3: *read* (Pc0, Pd0, Pm00, Pm01) – *write* (Pp0, Pp1, Pd1)

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

SWAP FILE	
s0:	s1:
s2:	s3:

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 4: *read* (Pc0, Pp0) – *write* (Pp1) – 4 *kswapd*

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 5: *context switch* (Q) – *read* (Qc1, Qc0, Qd0) – *write* (Qp0)

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

SWAP FILE	
s0:	s1:
s2:	s3:

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 6: *write* (Qd0)

PT del processo: **P**				
d0:				
PT del processo: **Q**				
d0:				

SWAP FILE	
s0:	s1:
s2:	s3:

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.**

```
PROCESSO: P ***************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <XX,0>
          S  000000600,  2 , W  ,  P  ,  M  , <XX,2>
          D  000000602,  2 , W  ,  P  ,  A  , <-1,0>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
    PT: <c0 :1 R>  <c1 :- ->  <s0 :4 W>  <s1 :- ->
        <d0 :- ->  <d1 :- ->
        <p0 :2  W> <p1 :- -> <p2 :- ->
    process P - NPV of PC and SP:  c0, p0

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                ||  01 : Pc0 / <XX, 0>       ||
      02 : Pp0                 ||  03 : <XX, 2>             ||
      04 : Ps0                 ||  05 : ----                ||
      06 : ----                ||  07 : ----                ||

____STATO del TLB_____________________________________________________
      Pc0 : 01 -  0: 1:        ||  Pp0 : 02 -  1: 1:        ||
      Ps0 : 04 -  1: 0:        ||        -----              ||

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

LRU ACTIVE:   PP0,  PC0

LRU INACTIVE: ps0
```

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	**F**	**0**	**1**	**0**	**0**

ATTENZIONE: **è presente la colonna "processo" dove va specificato il nome/i del/i processo/i a cui si riferiscono le informazioni "f_pos" e "f_count" (campi di struct file) relative al file indicato.**

Il processo **P** è in esecuzione. Il file **F** è stato aperto da **P** tramite chiamata **fd1** = ***open*** **(F).**

ATTENZIONE: il numero di pagine lette o scritte di un file è cumulativo, ossia è la somma delle pagine lette o scritte su quel file da tutti gli eventi precedenti oltre a quello considerato. Si ricorda inoltre che la primitiva *close* scrive le pagine dirty di un file solo se *f_count* diventa = 0.

Per ciascuno degli eventi seguenti, compilare le tabelle richieste con i dati relativi al contenuto della memoria fisica, delle variabili del FS relative ai file aperti e al numero di accessi a disco in lettura e in scrittura.

evento 1: *read* **(fd1, 9000)**

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <XX, 0>
02:	03:
04:	05:
06:	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
	F				

evento 2: *fork* **(Q)**

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02:	03:
04:	05:
06:	07:

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 3: *write* **(fd1, 4000)**

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02:	03:
04:	05:
06:	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
	F				

continua sulla pagina successiva

evento 4: *context switch* **(Q) – fd2** = *open* **(G) –** *write* **(Qp0, Qp1)**

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02:	03:
04:	05:
06:	07:

LRU ACTIVE: _____

LRU INACTIVE: _____

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
	F				
	G				

evento 5: *write* **(fd2, 8000)**

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02:	03:
04:	05:
06:	07:

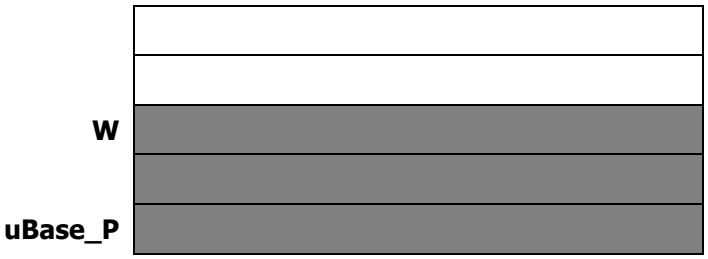
SWAP FILE	
s0:	s1:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
	F				
	G				

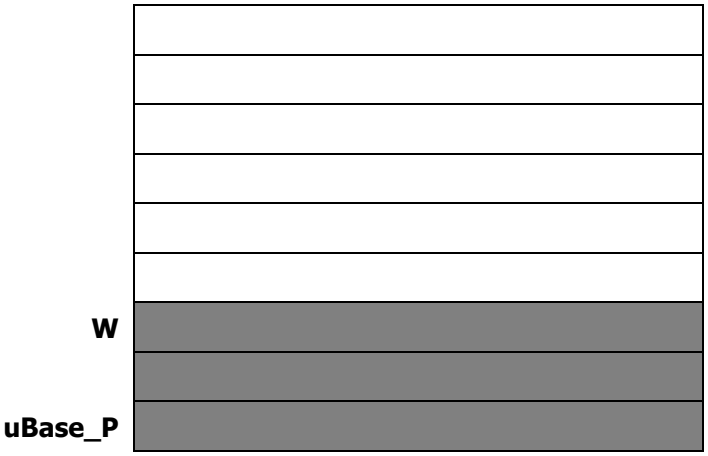
esercizio n. 4 – domande varie

prima domanda – moduli del SO

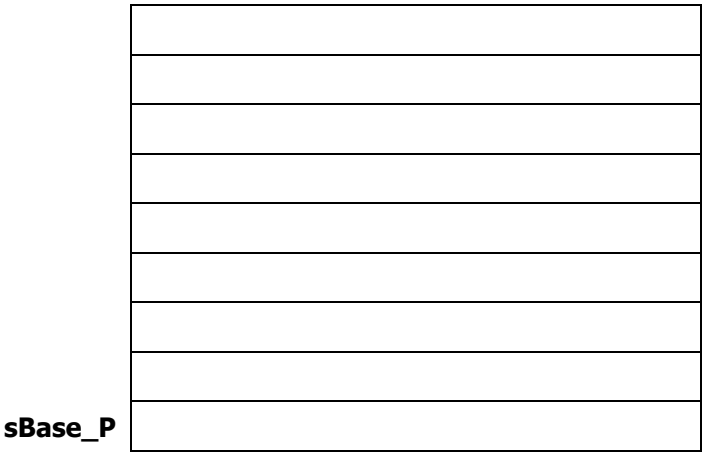
Un task **P** esegue la funzione **pthread_join (TH_Q, ...)** e si autosospende attendendo la terminazione di un task **Q** (di tipo thread), che al momento è in stato di **pronto** per scadenza di quanto di tempo. Si **mostrino** lo stato finale della **uPila** e della **sPila** di **P**, i contenuti finali delle **strutture dati** di **P**, e lo stato della **sPila** di **Q**.



uStack_P – iniziale



uStack_P – finale (da completare)



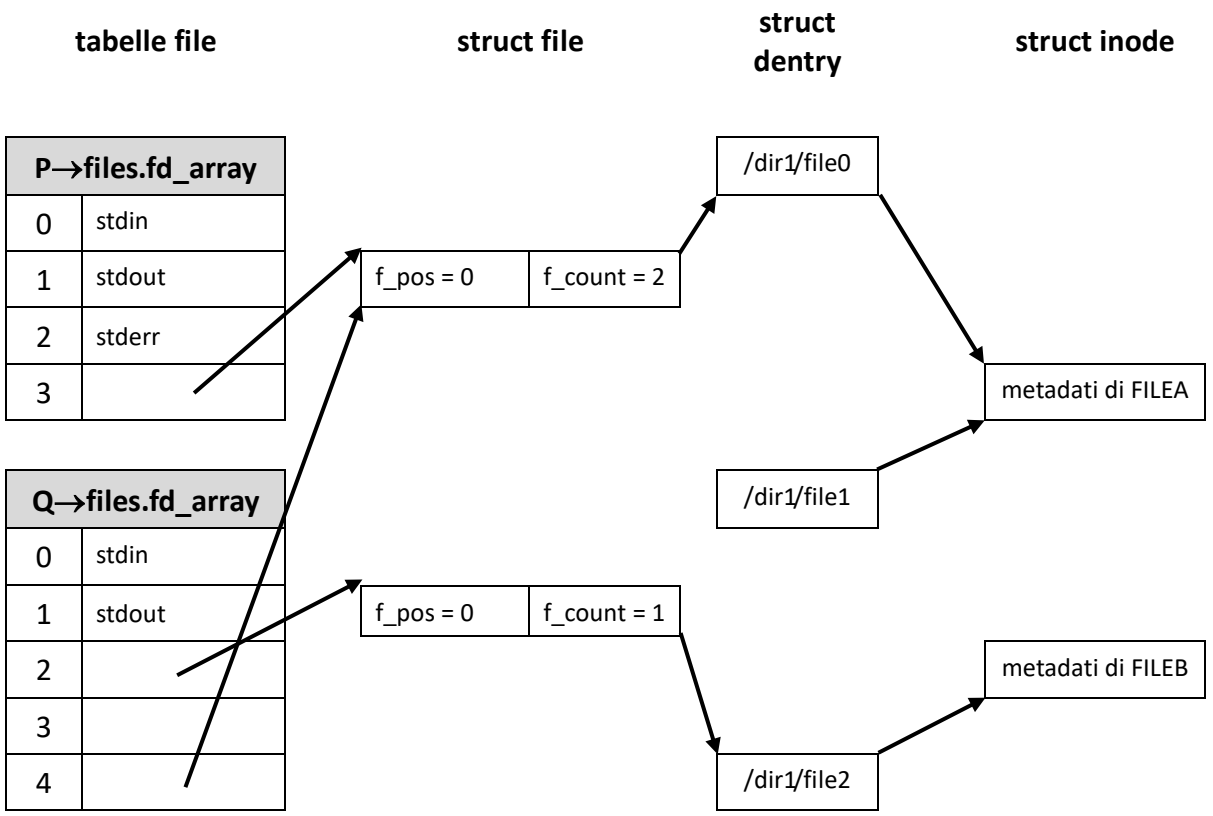
sStack_P – finale (da completare)

strutture dati finali del task P (da completare)	
registro PC	// non di interesse
registro SP	
SSP	
USP	
descrittore di P.stato	

sStack_Q (da completare)

seconda domanda – strutture del FS

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):

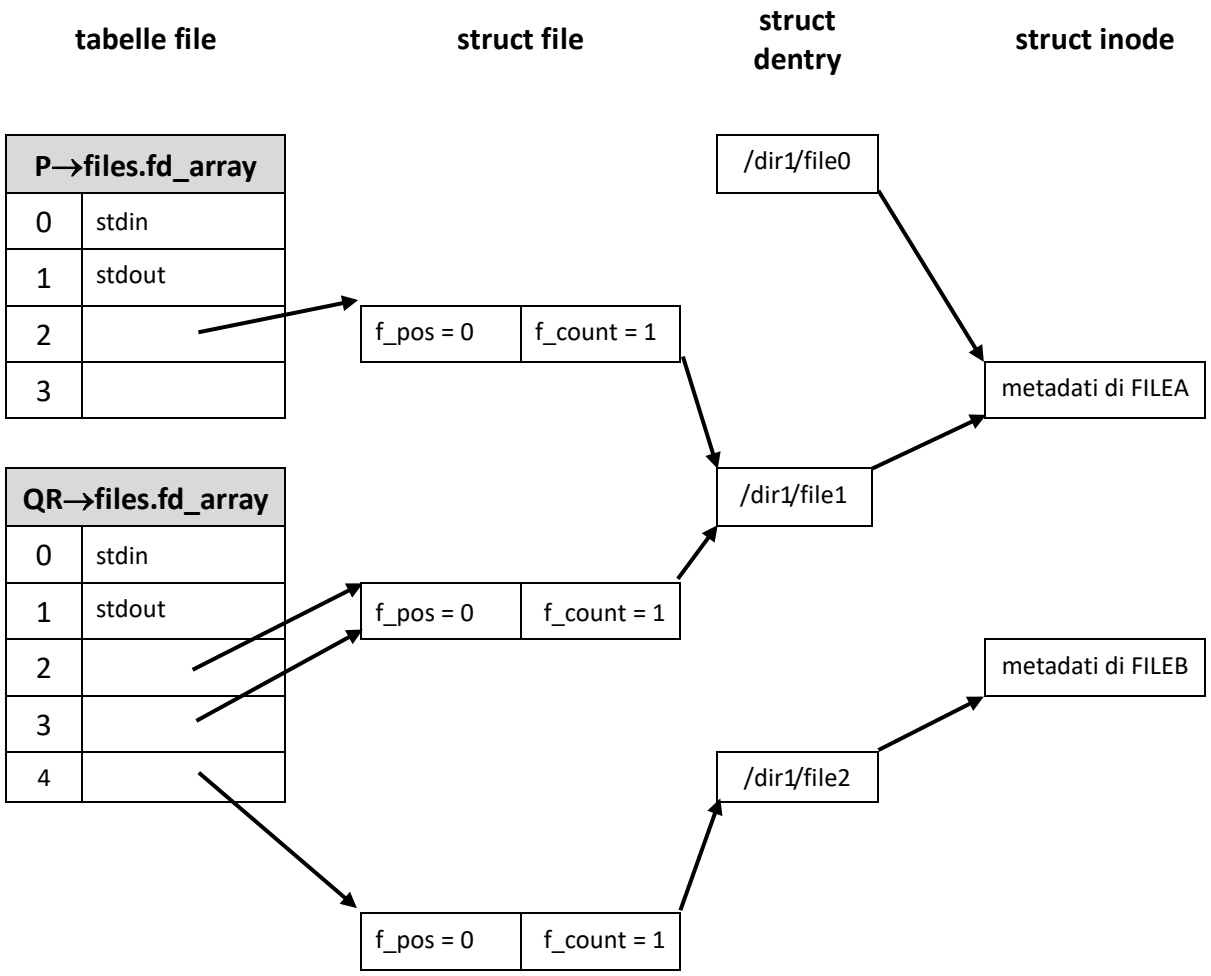


Il task (normale) **P** ha creato il task (normale) figlio **Q**.

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. Il numero di eventi da riportare è esattamente 6 ed essi sono eseguiti, nell'ordine, dai task indicati.

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

domanda – il task **Q** crea il task **R**, di tipo thread



sequenza di chiamate di sistema

#	processo	chiamata di sistema
1	P	close (3)
2	P	
3	P	
4	Q	
5	Q	
6	R	dup (3)
7	R	
8	R	

spazio libero per appunti correzioni o continuazioni (se necessario)