POLITECNICO
MILANO 1863

Architettura dei calcolatori e sistemi operativi

Esercizi Memoria,
File System

requiredPages da parte di un processo

periodicamente

kswapd

(freePages - requiredPages) < minFree

Controlla_liste

freePages < maxFree

invoca **PFRA** che controlla la **LRU inactive** a partire dalla coda

toFree = maxFree - freePages + requiredPages

La funzione periodica **Controlla_liste** esegue le seguenti operazioni:

1. Scansione della **active list** dalla coda
   - se A=1
     - azzera A
     - se (ref=1) sposta P in testa alla active
     - se (ref=0) pone ref=1
   - se A=0
     - se (ref=1) pone ref = 0
     - se (ref=0) sposta P in testa alla inactive con ref=1

2. Scansione della **inactive list** dalla testa (escluse le pagine appena inserite provenienti dalla active)
   - se A=1
     - azzera A
     - se (ref=1) sposta P in coda alla active con ref=0
     - se (ref=0) pone ref=1
   - se A=0
     - se (ref=1) pone ref = 0
     - se (ref=0) sposta P in coda alla inactive

invocata da **kswapd**

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 = 2**

situazione iniziale (esiste un processo P)

```
PROCESSO: P  ********************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <X,0>
          S  000000600,  1 , W  ,  P  ,  M  , <X,2>
          D  000000601,  2 , W  ,  P  ,  A  , <-1,0>
          P  7FFFFFFFB,  4 , W  ,  P  ,  A  , <-1,0>

    PT: <c0 :- -> <c1 :1 R> <s0 :4 W> <d0 :- -> <d1 :- -> <p0 :s0 W>
        <p1 :5 W> <p2 :2 W> <p3 :- ->

process P - NPV of PC and SP:  c1, p2

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                 ||  01 : Pc1 / <X,1>          ||
      02 : Pp2                  ||  03 : ----                 ||
      04 : Ps0                  ||  05 : Pp1                  ||
      06 : ----                 ||  07 : ----                 ||
```

```
     STATO del TLB
_____________________________________________________________________
       Pc1 :  01 -  0: 1:    ||    Pp2 : 02 -  1: 0:     ||
       Ps0 :  04 -  1: 0:    ||    Pp1 : 05 -  1: 0:     ||
              -----          ||          -----           ||
              -----          ||          -----           ||

SWAP FILE:       Pp0  , ----, ----, ----, ----, ----
LRU ACTIVE:      PC1
LRU INACTIVE:    pp2, pp1, ps0
```

evento 1 e 1 bis: ***read*** **(Pd0),** ***write*** **(Pp3, Pd1)**

evento 2: ***sbrk*** **(2)**

evento 3: ***read*** **(Pc1) – 4** ***kswapd***

evento 4: ***write*** **(Pd2, Pd3)**

evento 5: ***read*** **(Pp2)**

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo: **P**				
s0: 4 W	d0:	d1:	c1 : 1 R	
p0: s0 W	p1: 5 W	p2: 2 W	p3:	p4:

MEMORIA FISICA	
00: <ZP>	01: Pc1 / <X, 1>
02: Pp2	03:
04: Ps0	05: Pp1
06:	07:

SWAP FILE	
s0: Pp0	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: PC1

LRU INACTIVE: pp2, pp1, ps0

evento 1 e 1 bis: ***read*** **(Pd0),** ***write*** **(Pp3, Pd1)**

PT del proc		P
s0: 4 W	d0:	d1:
p0: s0 W	p1: 5 W	p2: 2

Pd0 è anonima e viene solo letta, risulta quindi mappata in <ZP> in read-only, con eventuale COW futuro

MEMOR	
00: <ZP>	
02: Pp2	
04: Ps0	05: Pp1
06:	07:

SWAP FILE	
s0: Pp0	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: PC1

LRU INACTIVE: pp2, pp1, ps0

Scrittura su VMA non mappate su file (ANONYMOUS)

- Aree di tipo **ANONYMOUS** non hanno un file associato
  - pila, dati dinamici
- La definizione di un'area anonima **non alloca** memoria fisica
- Le pagine virtuali sono tutte mappate sulla **ZeroPage** (pagina fisica di tutti 0 mantenuta dal sistema operativo)
- Operazioni di lettura trovano una pagina inizializzata a 0 e non alloca pagina fisica
- Operazioni di scrittura: si attiva Copy-on-Write con allocazione di pagina fisica
- Pagine anonime non hanno un backing store
  - Pagine di VMA di tipo ANONYMOUS
  - Pagine di aree PRIVATE duplicate per scrittura
- Le pagine anonime se devono essere scaricate dalla memoria vengono salvate in un file chiamato **SWAP FILE**

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo: **P**				
s0: 4 W	d0: 0 **R**	d1:	c1 : 1 R	
p0: s0 W	p1: 5 W	p2: 2 W	p3:	p4:

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03:
04: Ps0	05: Pp1
06:	07:

SWAP FILE	
s0: Pp0	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: PD0, PC1

LRU INACTIVE: pp2, pp1, ps0

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo			
s0: 4 W	d0: 0 R	d1:	:1 R
p0: s0 W	p1: 5 W	p2: 2 W	p4:

MEMOI	
00: <ZP> / Pd0	
02: Pp2	
04: Ps0	
06:	

Pp3 è la pagina di growsdown che modifica la VMA P e alloca una pagina in memoria fisica

SWAP FILE	
s0: Pp0	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: PD0, PC1

LRU INACTIVE: pp2, pp1, ps0

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo: **P**				
s0: 4 W	d0: 0 **R**	d1:	c1 : 1 R	
p0: s0 W	p1: 5 W	p2: 2 W	p3: 3 W	p4:

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Ps0	05: Pp1
06:	07:

SWAP FILE	
s0: Pp0	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: PP3, PD0, PC1

LRU INACTIVE: pp2, pp1, ps0

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo:		
s0: 4 W	d0: 0 R	d1:
p0: s0 W	p1: 5 W	p2:

Pd1 attiva il PFRA

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Ps0	05: Pp1
06:	07:

SWAP FILE	
s0: Pp0	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: PP3, PD0, PC1

LRU INACTIVE: pp2, pp1, ps0

evento 1 e 1 bis: ***read*** **(Pd0),** ***write*** **(Pp3, Pd1)**

PT del processo: **P**				
s0: 4 W	d0: 0 **R**	d1:	c1 : 1 R	
p0: s0 W	p1: 5 W	p2: 2 W	p3: 3 W	p4:

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Ps0	05: Pp1
06:	07:

SWAP FILE

Pp3 non è in memoria…

Attenzione! **(freePages - requiredPages) < minFree**

2 - 1 < 2

…interviene **Page Frame Reclaiming Algorithm (PFRA)**

toFree = maxFree - freePages + requiredPages

toFree = 3 - 2 + 1 = 2

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo: **P**				
s0: 4 W	d0: 0 **R**	d1:	c1 : 1 R	
p0: s0 W	p1: 5 W	p2: 2 W	p3: 3 W	p4:

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Ps0	05: Pp1
06:	07:

SWAP FILE
s0: Pp0
s2:
s4:

PFRA: libera **ps0, pp1**…

LRU ACTIVE: PP3, PD0, PC1

LRU INACTIVE: pp2, pp1, ps0

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo: **P**				
s0: s1 W	d0: 0 **R**	d1:	c1 : 1 R	
p0: s0 W	p1: s2 W	p2: 2 W	p3: 3 W	p4:

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04:	05:
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3:
s4:	s5:

LRU ACTIVE: PP3, PD0, PC1

LRU INACTIVE: pp2,

PFRA: libera **ps0, pp1**…
che vanno in SWAP

evento 1 e 1 bis: *read* (Pd0), *write* (Pp3, Pd1)

PT del processo: **P**				
s0: s1 W	d0: 0 **R**	d1: 4 W	c1 : 1 R	
p0: s0 W	p1: s2 W	p2: 2 W	p3: 3 W	p4:

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05:
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3:
s4:	s5:

LRU ACTIVE: PD1, PP3, PD0, PC1

LRU INACTIVE: pp2,

Infine carico **Pd1** per la scrittura

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	2	W	P	A	-1	0
P	7FFFFFFFB	4	W	P	A	-1	0

evento 3: *read* (Pc1) – 4 *kswapd*

LRU ACTIVE: PD1, PP3, PD0, PC1

LRU INACTIVE: pp2,

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

Dalla lettura di **growsdown** dell'evento precedente

evento 3: *read* (Pc1) – 4 *kswapd*

LRU ACTIVE: PD1, PP3, PD0, PC1

LRU INACTIVE: pp2,

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

evento 3: *read* (Pc1) – 4 *kswapd*

Per 4 volte vengono eseguite **read + kswapd** , infine **read**

LRU ACTIVE: PD1, PP3, PD0, PC1

LRU INACTIVE: pp2,

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

read(Pc1) 1

evento 3: *read* (Pc1) – 4 *kswapd*

A=1 ref=1	A=1 ref=1	A=1 ref=1	A=1 ref=1

LRU ACTIVE: PD1, PP3, PD0, PC1

LRU INACTIVE: pp2,

A=0 ref=0

La funzione periodica **Controlla_liste** esegue le seguenti operazioni:

1. Scansione della **active list** dalla coda
   - se A=1
     - azzera A
     - se (ref=1) sposta P in testa alla active
     - se (ref=0) pone ref=1
   - se A=0
     - se (ref=1) pone ref = 0
     - se (ref=0) sposta P in testa alla inactive con ref=1

invocata da **kswapd**

2. Scansione della **inactive list** dalla testa (escluse le pagine appena inserite provenienti dalla active)
   - se A=1
     - azzera A
     - se (ref=1) sposta P in coda alla active con ref=0
     - se (ref=0) pone ref=1
   - se A=0
     - se (ref=1) pone ref = 0
     - se (ref=0) sposta P in coda alla inactive

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

evento 3: *read* (Pc1) – 4 *kswapd*

Controlla_liste 1

A=0 ref=1	A=0 ref=1	A=0 ref=1	A=0 ref=1

LRU ACTIVE: PD1, PP3, PD0, PC1

LRU INACTIVE: pp2,

A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

read(Pc1) 2

evento 3: *read* (Pc1) – 4 *kswapd*

A=0 ref=1	A=0 ref=1	A=0 ref=1	A=1 ref=1

LRU ACTIVE: PD1, PP3, PD0, PC1

LRU INACTIVE: pp2,

A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

evento 3: *read* (Pc1) – 4 *kswapd*

Controlla_liste 2

A=0 ref=1	A=0 ref=0	A=0 ref=0	A=0 ref=0

LRU ACTIVE: PC1, pd1, pp3, pd0,

LRU INACTIVE: pp2,

A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

***read*(Pc1) 3**

evento 3: *read* (Pc1) – 4 *kswapd*

A=1 ref=1	A=0 ref=0	A=0 ref=0	A=0 ref=0

LRU ACTIVE: PC1, pd1, pp3, pd0,

LRU INACTIVE: pp2,

A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

evento 3: *read* (Pc1) – 4 *kswapd*

Controlla_liste 3

A=0
ref=1

LRU ACTIVE: PC1,

LRU INACTIVE: PD1, PP3, PD0, pp2,

A=0 ref=1	A=0 ref=1	A=0 ref=1	A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

read(Pc1) 4

evento 3: *read* (Pc1) – 4 *kswapd*

A=1 ref=1

LRU ACTIVE: PC1, _____

LRU INACTIVE: PD1, PP3, PD0, pp2, _____

A=0 ref=1	A=0 ref=1	A=0 ref=1	A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

evento 3: *read* (Pc1) – 4 *kswapd*

Controlla_liste 4

A=0 ref=1

LRU ACTIVE: PC1, _____

LRU INACTIVE: pd1, pp3, pd0, pp2, _____

A=0 ref=0	A=0 ref=0	A=0 ref=0	A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

read(Pc1) 5

evento 3: *read* (Pc1) – 4 *kswapd*

A=1 ref=1

LRU ACTIVE: PC1, _____

LRU INACTIVE: pd1, pp3, pd0, pp2, _____

A=0 ref=0	A=0 ref=0	A=0 ref=0	A=0 ref=0

evento 2: *sbrk* (2)

VMA del processo **P** (compilare solo le righe relativa alle VMA **D** e **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	000000601	4	W	P	A	-1	0
P	7FFFFFFFA	5	W	P	A	-1	0

evento 3: *read* (Pc1) – 4 *kswapd*

LRU ACTIVE: PC1,

LRU INACTIVE: pd1, pp3, pd0, pp2,

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05:
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3:
s4:	s5:

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3:
s4:	s5:

evento 4: *write* (Pd2, Pd3)

Pd3 attiva il PFRA

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3:
s4:	s5:

evento 4: ***write*** **(Pd2, Pd3)**

Pd3 attiva il PFRA

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0

Pd3 non è in memoria…

Attenzione! **(freePages - requiredPages) < minFree**

2 - 1 < 2

…interviene **Page Frame Reclaiming Algorithm (PFRA)**

toFree = maxFree - freePages + requiredPages

toFree = 3 - 2 + 1 = 2

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3:
s4:	s5:

LRU ACTIVE: PD2, PC1, _____

LRU INACTIVE: pd1, pp3, pd0, pp2, ← _____

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3:
s4:	s5:

LRU ACTIVE: PD2, PC1,

LRU INACTIVE: pd1, pp3, pd0, pp2, ←

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	
s4:	

ATTENZIONE: mappata in <ZP>

LRU ACTIVE: PD2, PC1,

LRU INACTIVE: pd1, pp3, pd0, pp2,

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pp2	03: Pp3
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	
s4:	

PFRA: libera **pp2, pp3**…

LRU ACTIVE: PD2, PC1

LRU INACTIVE: pd1, pp3, pd0, pp2,

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02:	03:
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3: Pp2
s4: Pp3	s5:

LRU ACTIVE: PD2, PC1,

LRU INACTIVE: pd1, pd0,

PFRA: libera **pp2, pp3**…
che vanno in SWAP

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02:	03:
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3: Pp2
s4: Pp3	s5:

LRU ACTIVE: PD2, PC1,

LRU INACTIVE: pd1, pd0,

evento 4: *write* (Pd2, Pd3)

MEMORIA FISICA	
00: <ZP> / Pd0	01: Pc1 / <X, 1>
02: Pd3	03:
04: Pd1	05: Pd2
06:	07:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3: Pp2
s4: Pp3	s5:

LRU ACTIVE: PD3, PD2, PC1,

LRU INACTIVE: pd1, pd0,

evento 5: *read* (Pp2)

PT del processo: **P**				
p0: s0 W	p1: s2 W	p2: s3 W	p3: s4 W	p4:

process P – NPV of PC and SP:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3: Pp2
s4: Pp3	s5:

LRU ACTIVE: PD3, PD2, PC1,

LRU INACTIVE: pd1, pd0,

Swap_in

Le pagine che richiedono swapping sono solo le pagine **anonime** (consideriamo solo quelle **PRIVATE**)

1. Quando si esegue uno swap_in la pagina viene copiata in memoria fisica, la copia nella Swap Area non viene eliminata, ***la pagina letta in memoria è marcata in sola lettura (R)***
2. Nel Swap_Cache_Index viene inserito un descrittore che contiene i riferimenti alla pagina fisica e al Page Slot
3. Se la pagina fisica viene solo letta quando la pagina viene nuovamente scaricata non viene riscritta su disco
4. Se la pagina fisica viene scritta, si verifica un Page Fault per errore di protezione:
   - La pagina diventa PRIVATE e non appartiene più alla Swap Area
   - La sua protezione cambia in W
   - Il contatore swap_map_counter viene decrementato
   - Se swap_map_counter = 0 il Page Slot viene liberato nella Swap Area

evento 5: *read* (Pp2)

PT del processo: P				
p0: s0 W	p1: s2 W	p2: 3 R	p3: s4 W	p4:

process P – NPV of PC and SP:

Pronto per futura COW

SWAP	
s0: Pp0	s1: Ps0
s2: Pp1	s3: Pp2
s4: Pp3	s5:

LRU ACTIVE: PP2, PD3, PD2, PC1,

LRU INACTIVE: pd1, pd0,

evento 5: *read* (Pp2)

PT del processo: **P**				
p0: s0 W	p1: s2 W	p2: 3 **R**	p3: s4 W	p4:

process P – NPV of PC and SP:

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3: Pp2
s4: Pp3	s5:

LRU ACTIVE: PP2, PD3, PD2, PC1,

LRU INACTIVE: pd1, pd0,

evento 5: *read* (Pp2)

PT del processo: **P**				
p0: s0 W	p1: s2 W	p2: 3 **R**	p3: s4 W	p4:

process P – NPV of PC and SP: c1 , p2

SWAP FILE	
s0: Pp0	s1: Ps0
s2: Pp1	s3: Pp2
s4: Pp3	s5:

LRU ACTIVE: PP2, PD3, PD2, PC1,

LRU INACTIVE: pd1, pd0,

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**:

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                  ||  01 : Pc2 / <X,2>          ||
      02 : Pp0                   ||  03 : <G,2>                ||
      04 : Pm00                  ||  05 : ----                 ||
      06 : ----                  ||  07 : ----                 ||

____STATO del TLB_____________________________________________________
      Pc2  : 01 -  0: 1:    ||   Pp0 : 02 -  1: 1:     ||
      Pm00 : 04 -  1: 1:    ||         -----           ||
             -----          ||         -----           ||
```

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 in esecuzione il processo **P**.

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

eventi 1 e 2: fd = *open* ("F"), *write* (fd, 6000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05:
06:	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
`file F`				

eventi 1 e 2: fd = *open* ("F"), *write* (fd, 6000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>* *D*
06: *<F, 1>* *D*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*6000*	*1*	*2*	*0*

evento 3: *close* (fd)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0> D*
06: *<F, 1> D*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*6000*	*1*	*2*	*0*

evento 3: *close* (fd)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*--*	*0*	*2*	*2*

eventi 4 e 5: *fork* (“Q”), *context switch* (“Q”)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	--	*0*	*2*	*2*

Fork

Dopo una fork le pagine duplicate fisicamente sono solo quelle scritte durante l'esecuzione del servizio fork

`pid = fork ();` ➔ scrittura nella pagina contenente la variabile pid

In Linux nella **Copy on Write** **la pagina fisica originale è attribuita al processo figlio, mentre per il processo padre viene allocata una pagina fisica nuova**

Context switch

Context switch

- non ha effetti sulla memoria
- Causa svuotamento del TLB
- Copia il valore del dirty bit associato alle pagine del TLB nel descrittore della pagina fisica page_descriptor

eventi 4 e 5: *fork* ("Q"), *context switch* ("Q")

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	--	*0*	*2*	*2*

la **fork** prima fa coincidere ciascuna pagina del padre e del figlio

eventi 4 e 5: *fork* ("Q"), *context switch* ("Q")

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	--	*0*	*2*	*2*

la **fork** poi richiede una pagina per scrivere la cima della pila del padre

eventi 4 e 5: *fork* (“Q”), *context switch* (“Q”)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	--	0	2	2

La cima della pila viene scritta quando Q viene creato…

Attenzione! **(freePages - requiredPages) < minFree**

1 - 1 < 1

…interviene **Page Frame Reclaiming Algorithm (PFRA)**

toFree = maxFree - freePages + requiredPages

toFree = 2 - 1 + 1 = 2

eventi 4 e 5: *fork* (“Q”), *context switch* (“Q”)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03: *<G, 2>*
04: *Pm00*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count
file F	--	0

PFRA: libera **<G, 2>, <F, 0>** che sono in **Page Cache**

la **fork** poi richiede una pagina per scrivere la cima della pila del padre

eventi 4 e 5: *fork* (“Q”), *context switch* (“Q”)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / <X, 2>*
02: *Pp0*	03:
04: *Pm00*	05:
06: *<F, 1>*	07:

	f_pos	f_count
file F	--	*0*

PFRA: libera **<G, 2>, <F, 0>** che sono in **Page Cache**

la **fork** poi richiede una pagina per scrivere la cima della pila del padre

eventi 4 e 5: *fork* ("Q"), *context switch* ("Q")

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0*
04: *Pm00 / Qm00*	05:
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	--	*0*	*2*	*2*

eventi 4 e 5: *fork* ("Q"), *context switch* ("Q")

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05:
06: *<F, 1>*	07:

	f_pos	f_count	nur	tte
file F	--	0		

Le pagine di P vengono messe a **D** a causa del flush del **TLB** causato dal **context switch**

eventi 6 e 7: fd = *open* ("F"), *read* (7000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05:
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	--	*0*	*2*	*2*

eventi 6 e 7: fd = *open* ("F"), *read* (7000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*

eventi 8 e 9: fd1 = *open* (“H”), *write* (fd1, 3000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*

eventi 8 e 9: fd1 = *open* ("H"), *write* (fd1, 3000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*
file H	*0*	*1*		

eventi 8 e 9: fd1 = *open* ("H"), *write* (fd1, 3000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<F, 0>*
06: *<F, 1>*	07:

	f_pos	f_count
file F	7000	1
file H	0	1

PFRA: libera **<F, 0>, <F, 1>** che sono in **Page Cache**

eventi 8 e 9: fd1 = *open* (“H”), *write* (fd1, 3000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05:
06:	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*
file H	*0*	*1*		

eventi 8 e 9: fd1 = *open* (“H”), *write* (fd1, 3000)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<H, 0> D*
06:	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*
file H	*3000*	*1*	*1*	*0*

eventi 10 e 11: *close* (fd), *close* (fd1)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<H, 0> D*
06:	07:

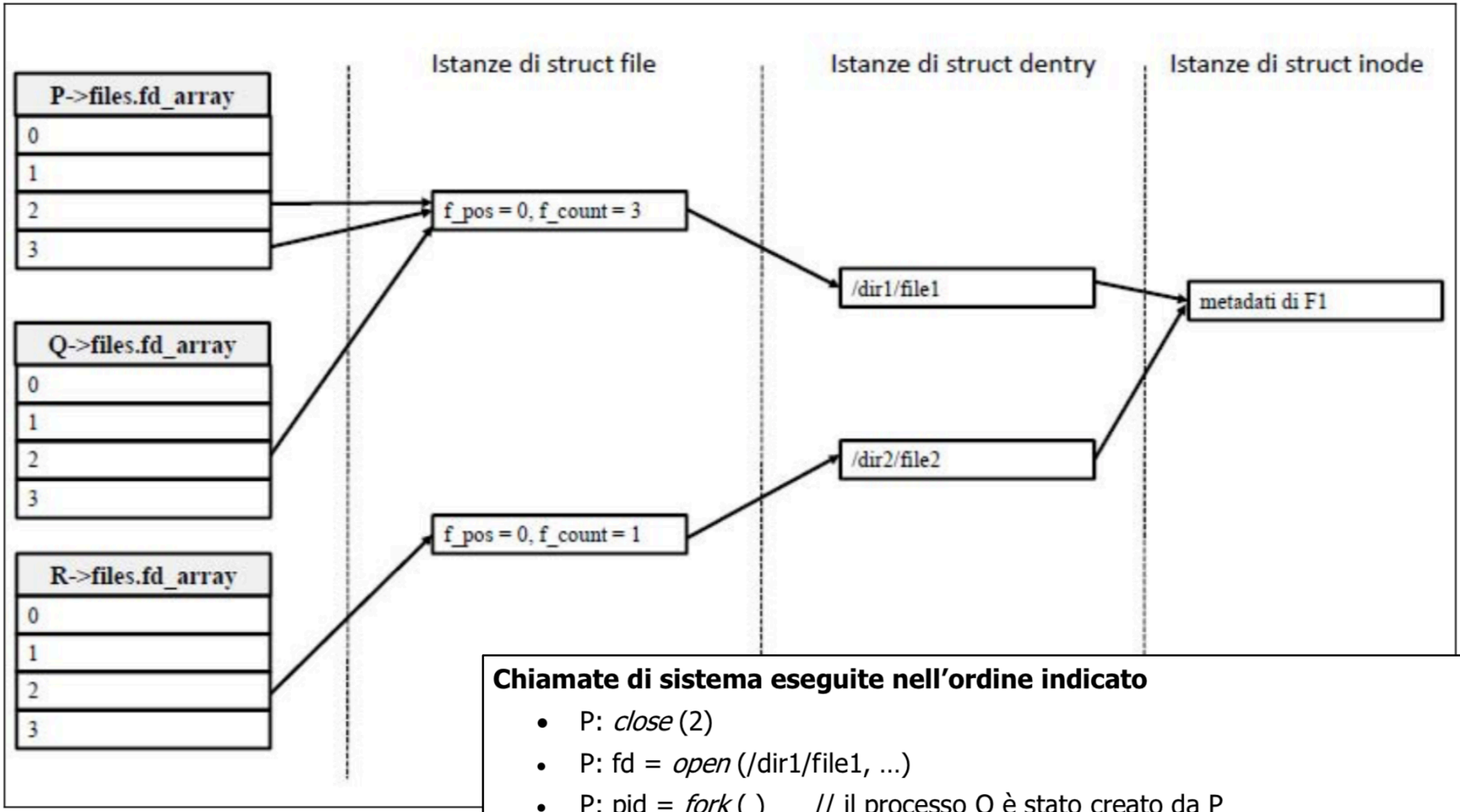
	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	*7000*	*1*	*3*	*2*
file H	*3000*	*1*	*1*	*0*

eventi 10 e 11: *close* (fd), *close* (fd1)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 /Qc2 / <X, 2>*
02: *Qp0 D*	03: *Pp0 D*
04: *Pm00 / Qm00 D*	05: *<H, 0>*
06:	07:

	f_pos	f_count	numero pag. lette	numero pag. scritte
file F	--	0	*3*	*2*
file H	--	0	*1*	1

esercizio n. 4 – virtual file system (VFS)

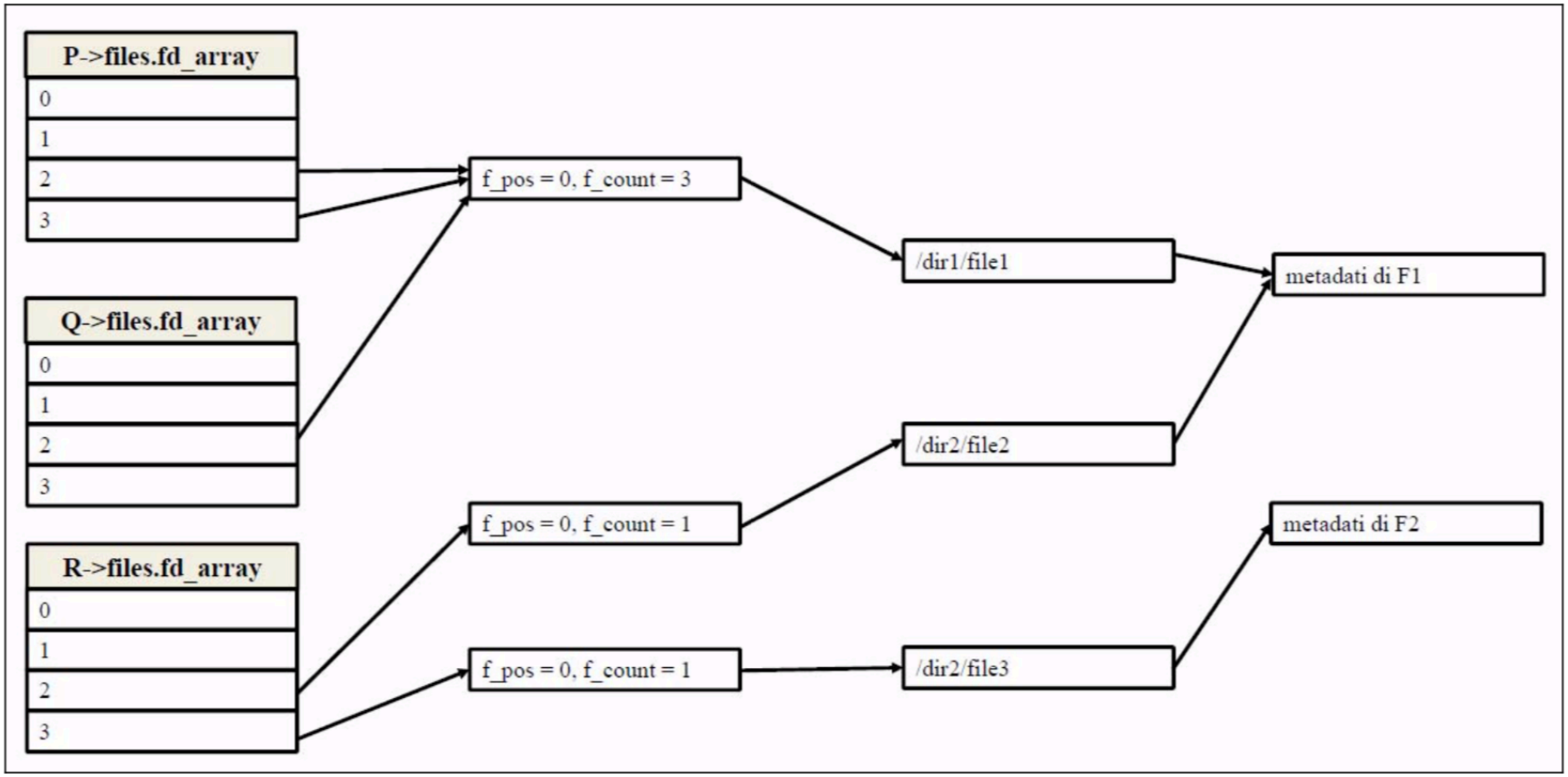


Chiamate di sistema eseguite nell'ordine indicato

- P: *close* (2)
- P: fd = *open* (/dir1/file1, …)
- P: pid = *fork* ( ) // il processo Q è stato creato da P
- P: fd1 = *dup* (fd)
- Il processo R è stato creato da altro processo non di interesse nell'esercizio
- R: *link* (/dir1/file1, /dir2/file2)
- R: *close* (2)
- R: fd = *open* (/dir2/file2)

Le sole tipologie di chiamate da considerare sono: `open()`, `close()`, `read()`, `dup()`, `link()`

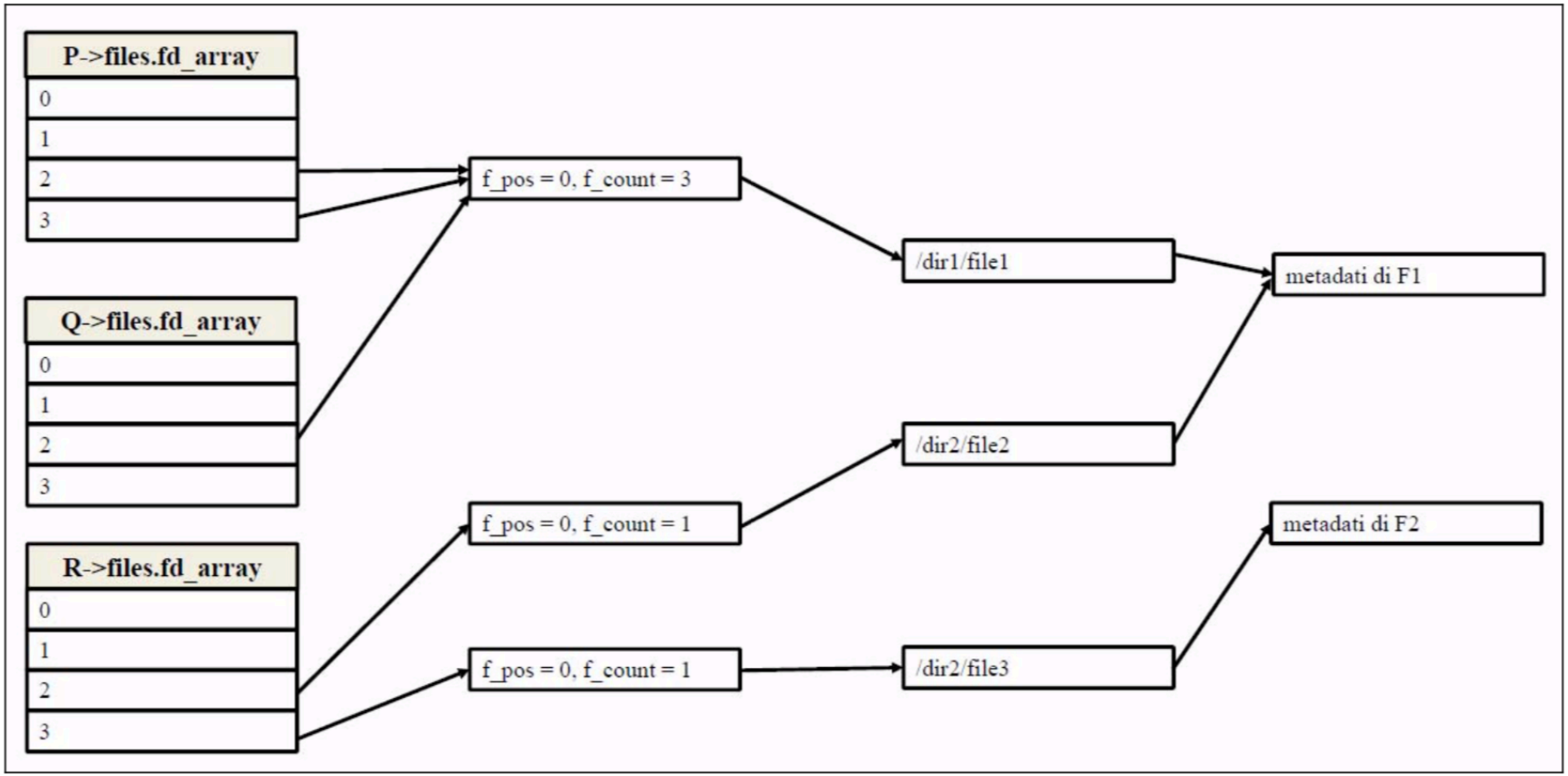
Domanda 1



Chiamata/e di sistema

Le sole tipologie di chiamate da considerare sono: `open()`, `close()`, `read()`, `dup()`, `link()`

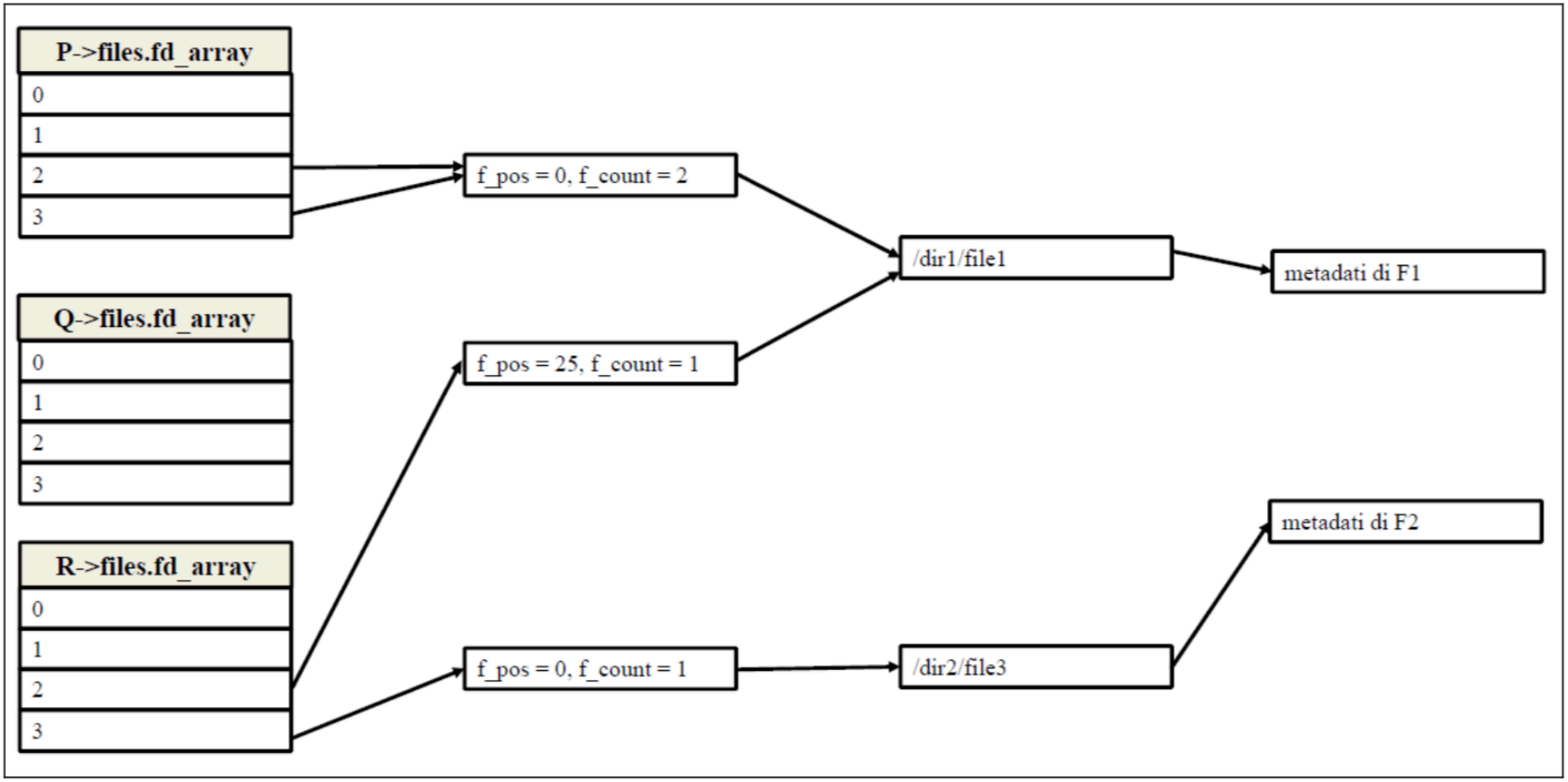
Domanda 1



Chiamata/e di sistema

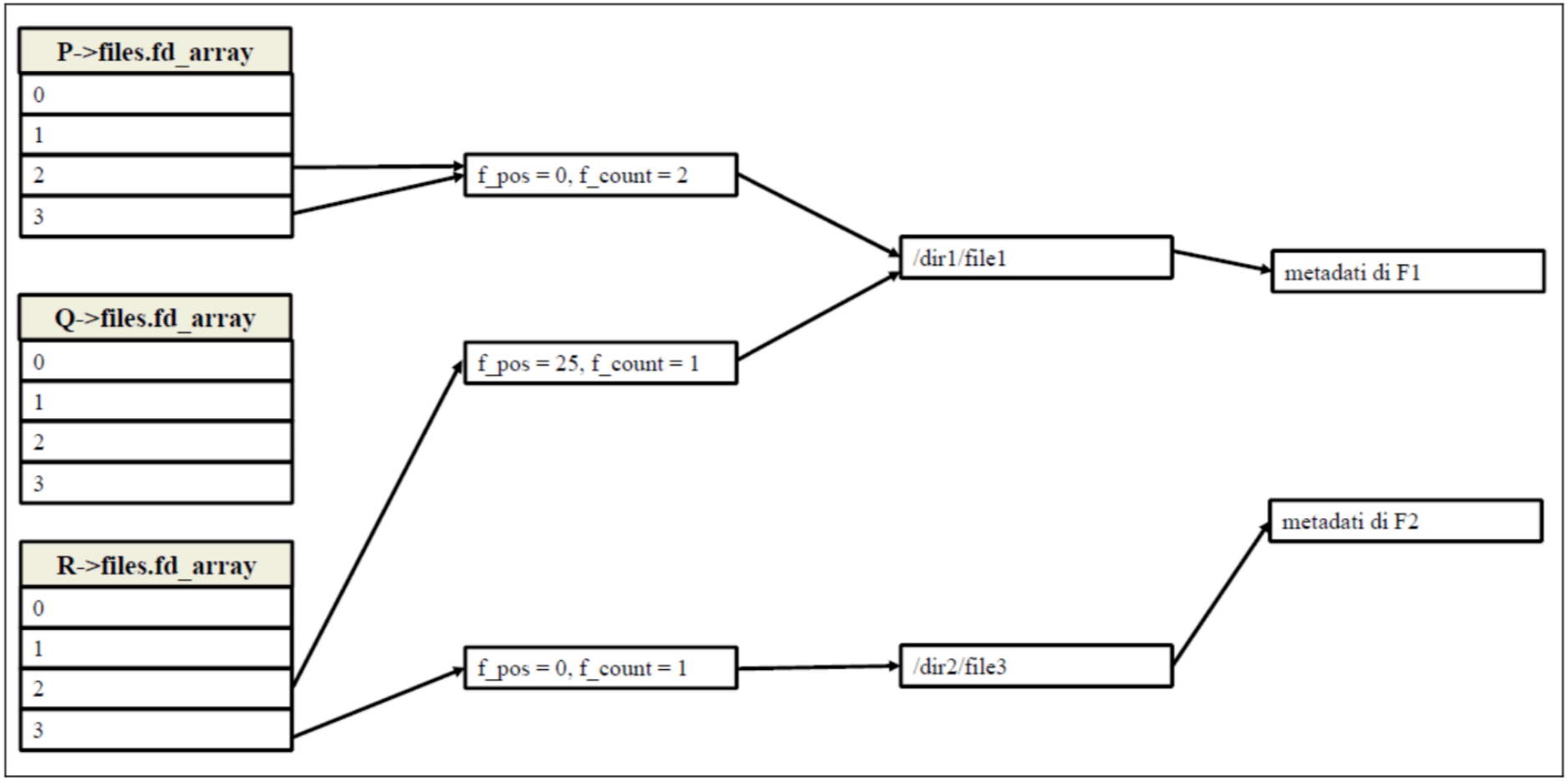
R: fd1 = open (/dir2/file3)

Domanda 2



Chiamata/e di sistema

Domanda 2



Chiamata/e di sistema

Q: close (2) (N.B.: può essere posizionata ovunque in questa sequenza)
R: close (2)
R: fd2 = open (/dir1/file1)
R: read (fd2, 25)