



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 – 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) _____

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
    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
    mutex_unlock (&drum)
    sem_post (&still)
    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)
    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	<i>thread</i>	
	th_1 – <i>player</i>	th_2 – <i>public</i>
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			
	<i>drum</i>	<i>noisy</i>	<i>still</i>	<i>global</i>
subito dopo stat. A				
subito dopo stat. B				
subito dopo stat. D				

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 – <i>player</i>	th_2 – <i>public</i>	<i>global</i>
1			
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) {	void * second (void * arg) {
pthread_mutex_lock (&CHECK)	pthread_mutex_lock (&CHECK)
sem_post (&SEM)	sem_wait (&SEM)
pthread_mutex_unlock (&CHECK)	pthread_mutex_unlock (&CHECK)
sem_wait (&SEM)	sem_wait (&SEM)
return NULL	sem_post (&SEM)
} /* first */	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

<i>identificativo simbolico del processo</i>		IDLE	P	Q		
evento oppure processo-chiamata	PID	1	2	3		
	TGID	1	2	3		
	0	pronto	attesa (read)	esec		
interrupt da RT_clock scadenza quanto di tempo	1					
	2					
	3					
	4					
	5					
interrupt da stdin tutti i caratteri trasferiti	6					
	7					
	8					
	9					
	10					
	11					
	12					
interrupt da RT_clock scadenza quanto di tempo	13					
	14					
	15					
	16					

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		
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 nuovo task generato dalla **CLONE**:

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

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

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:	s1:	d0:	d1:	p0:
p1:	p2:	p3:		

process P	NPV of PC :	NPV of SP :
------------------	--------------------	--------------------

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

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

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 2: *context switch* (Q)

PT del processo: P				
s0:	s1:	d0:	d1:	p0:
p1:	p2:	p3:		

process P	NPV of PC :	NPV of SP :
------------------	--------------------	--------------------

PT del processo: Q				
s0:	s1:	d0:	d1:	p0:
p1:	p2:			

process Q	NPV of PC :	NPV of SP :
------------------	--------------------	--------------------

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

TLB							
NPV	NPF	D	A	NPV	NPF	D	A

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

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 3: *read* (Qs0, Qd0)

TLB							
NPV	NPF		D	A	NPV	NPF	D A

evento 4: *write* (Qs0, Qp1)

PT del processo: Q				
s0:	s1:	d0:	d1:	p0:
p1:	p2:			

process Q	NPV of PC :	NPV of SP :
------------------	--------------------	--------------------

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

TLB							
NPV	NPF		D	A	NPV	NPF	D A

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

LRU ACTIVE: _____

LRU INACTIVE: _____

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:	03:
04:	05:
06:	07:

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

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

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02:	03:
04:	05:
06:	07:

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

evento 5: *write* (fd, 5000)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02:	03:
04:	05:
06:	07:

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

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				

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

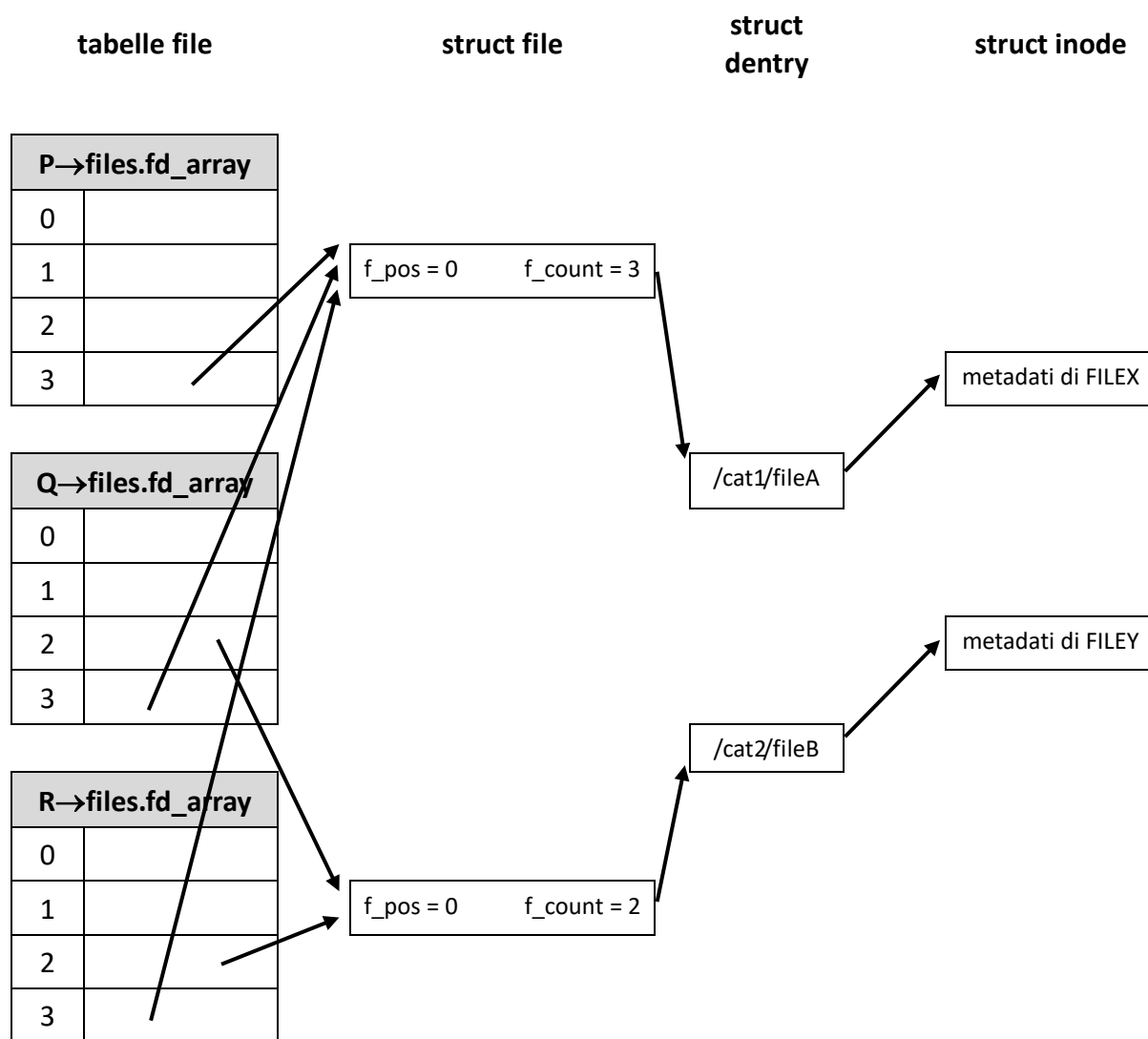
MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02:	03:
04:	05:
06:	07:

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

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.

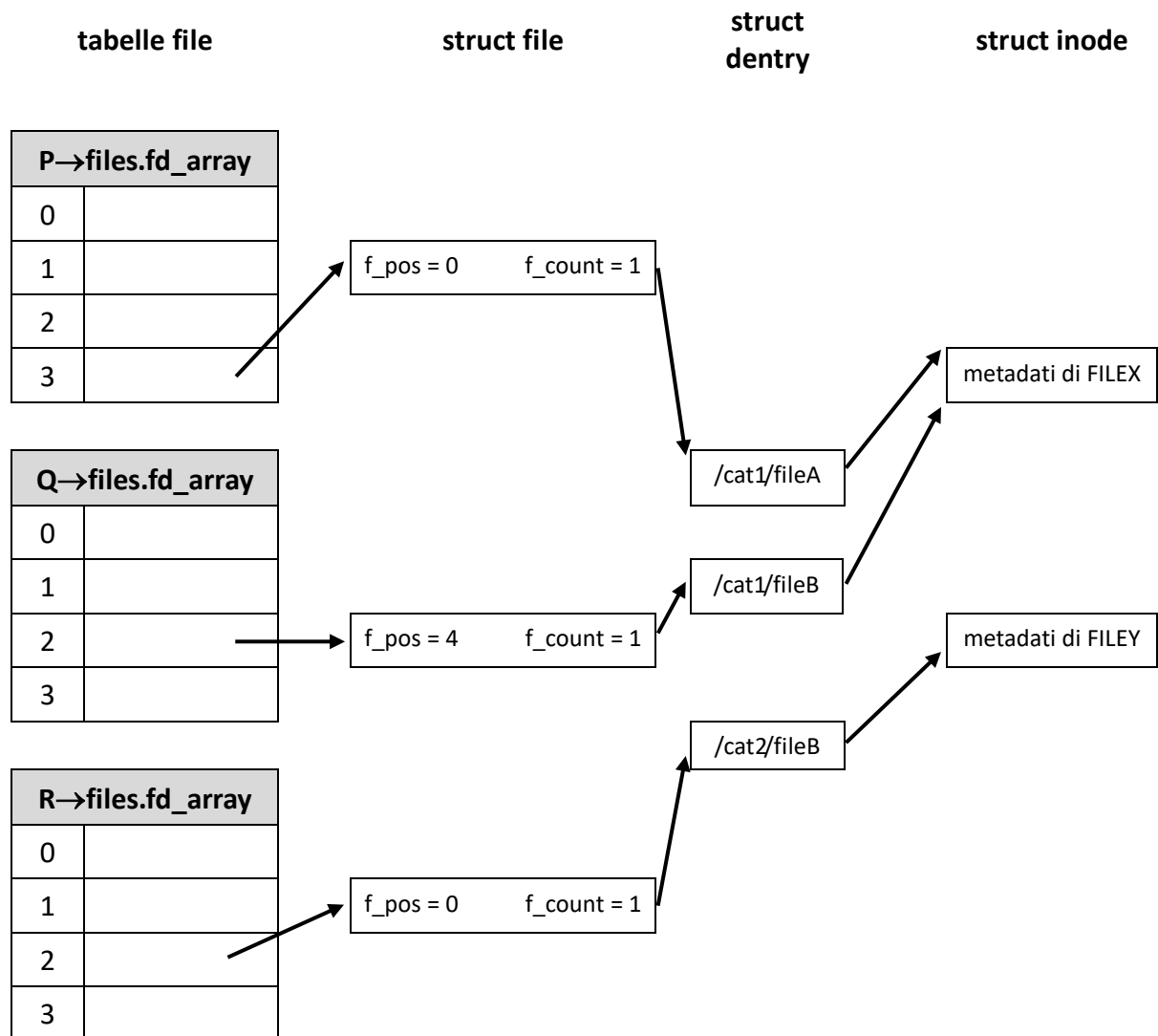


chiamate di sistema eseguite nell'ordine indicato

- 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**



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

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

spazio libero per brutta copia o continuazione