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) _____

CON SOLUZIONI (in rosso e 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 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**	*ESISTE*	*ESISTE*
subito dopo stat. **B**	*ESISTE*	*ESISTE*
subito dopo stat. **C**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **D**	*PUÒ ESISTERE*	*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**			
	deep	*shallow*	*water*	*global*
subito dopo stat. **A**	*1*	*1*	*1*	*1*
subito dopo stat. **B**	*1*	*0*	*0*	*2 / 3*
subito dopo stat. **C**	*1*	*0*	*0 / 1*	*2 / 4*
subito dopo stat. **D**	*0*	*0*	*0 / 1*	*2 / 5*

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	`lock shallow`	`wait water`	*0*
2	`lock deep`	`lock shallow`	*0*
3	`----`	`wait water`	*2*

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	*3*	*4*	*5*
	TGID	1	2	*2*	*2*	*2*
P – pthread_create TH_1	**0**	**pronto**	**esec**	**pronto**	**NE**	**NE**
P – pthread_create TH_2	**1**	*pronto*	*esec*	*pronto*	*pronto*	*NE*
P – pthread_create TH_3	**2**	*pronto*	*esec*	*pronto*	*pronto*	*pronto*
P – pthread_join TH_1	**3**	*pronto*	*attesa (TH_1)*	*esec*	*pronto*	*pronto*
TH_1 – sem_post (pronti)	**4**	*pronto*	*attesa (TH_1)*	*esec*	*pronto*	*pronto*
TH_1 – sem_wait (flag)	**5**	*pronto*	*attesa (TH_1)*	*attesa (flag)*	*esec*	*pronto*
TH_2 – sem_post (pronti)	**6**	*pronto*	*attesa (TH_1)*	*attesa (flag)*	*esec*	*pronto*
TH_2 – sem_wait (flag)	**7**	*pronto*	*attesa (TH_1)*	*attesa (flag)*	*attesa (flag)*	*esec*
TH_3 – 1a sem_wait (pronti)	**8**	*pronto*	*attesa (TH_1)*	*attesa (flag)*	*attesa (flag)*	*esec*
TH_3 – 2a sem_wait (pronti)	**9**	*pronto*	*attesa (TH_1)*	*attesa (flag)*	*attesa (flag)*	*esec*

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**	*pronto*	*attesa (TH_1)*	*esec*	*attesa (flag)*	*pronto*
TH_1 – write (stdout)	**11**	*pronto*	*attesa (TH_1)*	*attesa (write)*	*attesa (flag)*	*esec*
interrupt da stdout, sette caratteri inviati	**12**	**pronto**	*attesa (TH_1)*	*esec*	*attesa (flag)*	**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**	*pronto*	*attesa (TH_1)*	*pronto*	*esec*	*pronto*
TH_2 – read (stdin)	**11**	*pronto*	*attesa (TH_1)*	*esec*	*attesa (read)*	**pronto**
TH_1 – write (stdout)	**12**	*pronto*	*attesa (TH_1)*	*attesa (write)*	*attesa (read)*	*esec*
interrupt da stdin, cinque caratteri ricevuti	**13**	*pronto*	*attesa (TH_1)*	*attesa (write)*	**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**		
		0,5	*CLONE*	*T1*	*falso*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*5*	*T1*	*100,5*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T1*	*1*	*0,20*	*1,2*	*1*	*20,50*	*100,5*
RB	*T3*	*1*	*0,20*	*1,2*	*1*	*0*	*101,7*
	T2	*3*	*0,60*	*3,6*	*0,33*	*30*	*102*
WAITING							

EVENTO 2		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
		1	*WAIT*	*T1*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	*4*	*T3*	*101*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T3*	*1*	*0,25*	*1,5*	*1*	*0*	*101,7*
RB	*T2*	*3*	*0,75*	*4,5*	*0,33*	*30*	*102*
WAITING	*T1*	*1*				*21*	*101*

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
		2,5	Q_scade	T3	vero		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	4	T2	102		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	3	0,75	4,5	0,33	30	102
RB	T3	1	0,25	1,5	1	1,5	103,2
WAITING	T1	1				21	101

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
		4	WUP	T2	vero		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	5	T1	102,5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	1	0,2	1,2	1	21	101
RB	T2	3	0,6	3,6	0,33	31,5	102,5
	T3	1	0,2	1,2	1	1,5	103,2
WAITING							

Calcolo del VRT iniziale del **task T3** creato dalla **CLONE** *eseguita dal* ***task T1****:*

$T3.VRT\ (iniziale) = VMIN + T3.Q \times T3.VRTC = 100,5 + 1,2 \times 1 = 101,7$

Valutazione della cond. di rescheduling alla **CLONE** *eseguita dal* ***task T1****:*

$T3.VRT + WGR \times T3.LC < T1.VRT \Rightarrow 101,7 + 1 \times 0,2 = 101,9 < 100,5 \Rightarrow falso$

Valutazione della cond. di rescheduling alla **WAKEUP** *eseguita dal* ***task T2****:*

$T1.VRT + WGR \times T1.LC < T2.VRT \Rightarrow 101 + 1 \times 0,2 = 101,2 < 102,5 \Rightarrow vero$

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*

Legge la pagina Pc0, la pagina Pc0 viene allocata in pagina fisica 05, e il PC (codice corrente) del processo P ora punta a pagina c0. Scrive la pagina Pp0, in memoria. Le liste LRU vengono aggiornate (inserimenti e spostamenti – nota: le pagine di P scese in inactive si trovando davanti a quelle di Q pure scese in inactive, perché da TLB iniziale tali pagine di P risultano accedute, mentre quelle di Q non lo sono, dato che Q non è in esecuzione, pertanto le pagine di Q scendono in testa di inactive un passo prima di quelle di P e alla fine risultano in coda di inactive). Restano quattro pagine libere.

PT del processo: **P**				
c0: *5 R*	c1: *1 R*	d0: *3 R*	d1: *- -*	p0: *4 W*
p1: *- -*	p2: *- -*			

process P	NPV of **PC**: *c0*	NPV of **SP**: *p0*

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <CC, 1>
02: Qp0 D	03: Pd0 / Qd0
04: Pp0	05: *Pc0 / <CC, 0>*
06: *-----*	07: *-----*

08: -----	09: -----

LRU ACTIVE: *PC0, PP0*

LRU INACTIVE: *pd0, pc1, qd0, qp0, qc1,*

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

Viene definita la VMA M0 al NPV 0x 0300 0000 0, di due pagine, scrivibile, privata mappata su file FF con offset 0, e le pagine virtuali m00 e m01 vengono inserite nella tabella delle pagine del processo P, ma non allocate in memoria fisica.

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	*000030000*	*2*	*W*	*P*	*M*	*FF*	*0*

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

Legge le pagine Pc0 e Pd0, in memoria. Legge le pagine Pm00 e Pm01, alloca le pagine Pm00 e Pm01 nelle pagine fisiche 06 e 07, rispettivamente (tenendole mappate su file FF). Scrive la pagina Pp0, in memoria. Scrive la pagina Pp1, dunque si ha COW da ZP per la pagina Pp1, alloca la pagina Pp1 in pagina fisica 08. Scrive la pagina Pd1, dunque si ha COW da ZP per la pagina Pd1, per Pd1 si scende sotto minfree, invoca PFRA, reclama tre pagine, libera da lista inactive le tre pagine qp0, pc1/qc1 e pd0/qd0, ossia libera le tre pagine fisiche 02, 01 e 03, alloca la pagina Pd1 in pagina fisica 01; le pagine Qp0 e Pd0/Qd0 sono anonime, si ha swap_out di pagina Qp0 che va in swap slot s0 e di pagina Pd0/Qd0 che va in swap slot s1 (con map_counter = 2). Le liste LRU vengono aggiornate (inserimenti ed estrazioni). Restano tre pagine libere.

MEMORIA FISICA	
00: <ZP>	01: ~~*Pc1 / Qc1 / <CC, 1>*~~ *Pd1*
02: ~~*Qp0 D*~~ -----	03: ~~*Pd0 / Qd0*~~ -----
04: *Pp0*	05: *Pc0 / <CC, 0>*
06: *Pm00 / <FF, 0>*	07: *Pm01 / <FF, 1>*
08: *Pp1*	09: -----

SWAP FILE	
s0: *Qp0*	s1: *Pd0 / Qd0*
s2: -----	s3: -----

LRU ACTIVE: *PD1, PP1, PM01, PM00, PC0, PP0*

LRU INACTIVE:

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

Ci sono vari accessi a pagine (Pc0, Pp0 e Pp1), tutte in memoria, pertanto la memoria fisica non cambia. Le liste LRU vengono aggiornate (spostamenti da active a inactive), tenendo conto delle pagine accedute.

LRU ACTIVE: *PP1, PC0, PP0*

LRU INACTIVE: *pd1, pm01, pm00*

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

Commutazione di contesto a processo Q, il processo Q ha PC (codice corrente) in pagina c1 e SP (testa pila) in pagina p0 (vedi situazione iniziale), entrambe le pagine vanno subito allocate in memoria fisica, dunque alloca la pagina Qc1 in pagina fisica 02, si ha swap_in di pagina Qp0, alloca la pagina Qp0 in pagina fisica 03 e la modifica (poiché è pagina di testa pila), pertanto libera lo swap slot s0. Poi legge la pagina Qc0, in memoria, e la pagina Qd0, dunque si ha swap_in di pagina (condivisa) Pd0/Qd0, per Pd0/Qd0 si scende sotto minfree, invoca PFRA, reclama tre pagine, libera da lista inactive le tre pagine pm00, pm01 e pd1, ossia libera le tre pagine fisiche 06, 07 e 01, si ha swap_out di pagina pd1 (anonima) che va in swap slot s0 (che è stato appena liberato) (invece le pagine pm01 e pm00 sono immodificate e ancora mappate su file FF, pertanto non vanno in swap file), la pagina Pd0/Qd0 va in pagina fisica 01 (e resta anche in swap slot s1 con map_counter = 2). Poi scrive la pagina Qp0, in memoria. Le liste LRU vengono aggiornate (inserimenti ed estrazioni). Restano tre pagine libere.

MEMORIA FISICA	
00: <ZP>	01: ~~Pd1 D~~ Pd0 / Qd0
02: Qc1 / <CC, 1>	03: Qp0
04: Pp0 D	05: Pc0 / Qc0 / <CC, 0>
06: ~~Pm00 / <FF, 0>~~ -----	07: ~~Pm01 / <FF, 1>~~ -----
08: Pp1 D	09: -----

SWAP FILE	
s0: ~~Qp0~~ Pd1	s1: Pd0 / Qd0
s2: -----	s3: -----

LRU ACTIVE: QD0, QC0, QP0, QC1, PP1, PC0, PP0_____

LRU INACTIVE: pd0 _____

evento 6: *write* (Qd0)

Scrive la pagina Qd0 (condivisa), in memoria, dunque si ha COW di pagina Qd0 (per separarla da Pd0), alloca la pagina Qd0 in pagina fisica 06, la pagina Pd0 (non più condivisa) resta in swap slot s1 (ma ora ha map_counter = 1), nella tabella delle pagine (PT) del processo P la pagina d0 figura (ancora) allocata in pagina fisica 01 e resta marcata R (read-only), mentre quella del processo Q figura allocata in pagina fisica 06 e diventa marcata W (writable). Restano due pagine libere.

PT del processo: **P**				
d0: 1 R				
PT del processo: **Q**				
d0: 6 W				

SWAP FILE	
s0: Pd1	s1: Pd0 / ~~Qd0~~
s2: -----	s3: -----

Nota: come spiegato prima, all'evento 5 il context switch alloca in memoria la pagina Qp0 di testa pila del processo Q che va in esecuzione, e la scrive (modifica); supponendo invece che tale context switch si limiti ad allocare in memoria la pagina Qp0 di Q (in pagina fisica 03), senza immediatamente scriverla (modificarla), all'evento 5 lo swap slot s0 non verrebbe liberato subito da Qp0, in quanto tale pagina per il momento sarebbe ancora immodificata; pertanto la pagina Pd1 verrebbe scaricata in swap slot s2; e solo dopo lo swap_out di Pd1 la pagina di pila Qp0 (che ormai è in memoria) verrebbe effettivamente scritta (modificata) da parte dell'operazione write (Qp0), e dunque lo swap slot s0 verrebbe liberato, restando infine vuoto all'evento 6; mentre nelle tabelle delle pagine di P e Q la pagina d0 resterebbe registrata

SWAP FILE	
s0: -----	s1: Pd0 / ~~Qd0~~
s2: Pd1	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)

Legge le pagine 0, 1 e 2 del file F. Carica la pagina di file <F, 0> in pagina fisica 05, carica la pagina di file <F, 1> in pagina fisica 06, per <F, 2> si scende sotto minfree, invoca PFRA, reclama due pagine, libera la pagina fisica 05 da disk cache e la pagina fisica 03 da page cache, carica la pagina di file <F, 2> in pagina fisica 03. Tre letture da disco (per il file F), nessuna scrittura su disco. Restano due pagine libere.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <XX, 0>
02: *Pp0*	03: ~~*<XX, 2>*~~ *<F, 2>*
04: *Ps0*	05: ~~*<F, 0>*~~
06: *<F, 1>*	07: *----*

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	**F**	*9000*	*1*	*3*	*0*

evento 2: *fork* (Q)

Crea il processo Q figlio del processo P. Poiché il processo P ha SP (testa pila) in pagina p0, si ha COW di pagina Pp0, alloca la pagina Qp0 in pagina fisica 02, marcandola modificata (dirty) poiché viene modificata (scrivendoci dentro il valore restituito al figlio Q da fork) ma non compare nel TLB (poiché per ora resta in esecuzione il processo P), e rialloca la pagina Pp0 in pagina fisica 05. Aggiorna le liste LRU (inserimenti). Nessuna operazione su disco. L'apertura del file F viene ereditata dal processo Q. Resta una pagina libera.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02: ~~*Pp0 /*~~ *Qp0 D*	03: *<F, 2>*
04: *Ps0 / Qs0*	05: *Pp0*
06: *<F, 1>*	07: *----*

LRU ACTIVE: *QP0, QC0, PP0, PC0* _____

LRU INACTIVE: *qs0, ps0* _____

evento 3: *write* (fd1, 4000)

Scrive le pagine 2 e 3 del file F. Scrive in pagina fisica 03 (<F, 2>), in memoria, marcandola modificata (dirty), per <F, 3> si scende sotto minfree, invoca PFRA, reclama due pagine, libera la pagina fisica 03 da disk cache e la scarica su disco (poiché è modificata), e libera la pagina fisica 06 da disk cache, carica la pagina di file <F, 3> in pagina fisica 03, scrive in pagina fisica 03 (ora contiene <F, 3>), in memoria, marcandola modificata. Una lettura da disco e una scrittura su disco (entrambe per il file F). Restano due pagine libere.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02: *Qp0 D*	03: ~~*<F, 2> D*~~ *<F, 3> D*
04: *Ps0 / Qs0*	05: *Pp0*
06: ~~*<F, 1>*~~ *----*	07: *----*

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	**F**	*13000*	*2*	*4*	*1*

continua sulla pagina successiva

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

Commutazione di contesto da processo P a processo Q. Il processo Q ha PC (codice corrente) in pagina c0 e SP (testa pila) in pagina p0, entrambe già in memoria, dunque la memoria è immutata. La pagina di processo Pp0 viene marcata modificata nel descrittore di pagina fisica, vedi il TLB iniziale, poiché non sarà più nel TLB corrente, e anche la pagina condivisa Ps0 / Qs0 viene marcata modificata nel descrittore di pagina fisica, vedi TLB iniziale, perché pur essendo condivisa con il processo Q, per il momento non andrà nel TLB dato che non viene acceduta. Il processo Q apre il file G. Poi scrive le pagine di processo Qp0 e Qp1. Scrive in pagina Qp0 (ossia 02), in memoria, si ha COW da ZP per la pagina Qp1, alloca la pagina Qp1 in pagina fisica 06. Aggiorna le liste LRU (inserimenti). Nessuna operazione su disco. Resta una pagina libera.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02: *Qp0 D*	03: *<F, 3> D*
04: *Ps0 / Qs0 D*	05: *Pp0 D*
06: *Qp1*	07: *----*

LRU ACTIVE: *QP1, QP0, QC0, PP0, PC0*_____

LRU INACTIVE: *qs0, ps0*_____

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	**F**	*13000*	*2*	*4*	*1*
Q	**G**	*0*	*1*	*0*	*0*

evento 5: *write* (fd2, 8000)

Scrive le pagine 0 e 1 del file G. Per <G, 0> si scende sotto minfree, invoca PFRA, reclama due pagine, libera la pagina fisica 03 (ossia <F, 3>) da disk cache e la salva su disco (poiché è modificata, vedi evento 3), e libera la pagina fisica 04 (ossia ps0/qs0) da lista inactive, si ha swap_out della pagina (condivisa) ps0/qs0 (che è modificata e anonima dato che non è più mappata su file eseguibile XX, vedi situazione iniziale), carica le pagine di file <G, 0> e <G, 1> nelle pagine fisiche 03 e 04, rispettivamente, scrive in pagina fisica 03 e in pagina fisica 04, entrambe in memoria, marcandole entrambe modificate. Due letture da disco (per il file G), una scrittura su disco (per il file F). Resta una pagina libera.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <XX, 0>
02: *Qp0 D*	03: ~~*<F, 3> D*~~ *<G, 0> D*
04: ~~*Ps0 / Qs0 D*~~ *<G, 1> D*	05: *Pp0 D*
06: *Qp1*	07: *----*

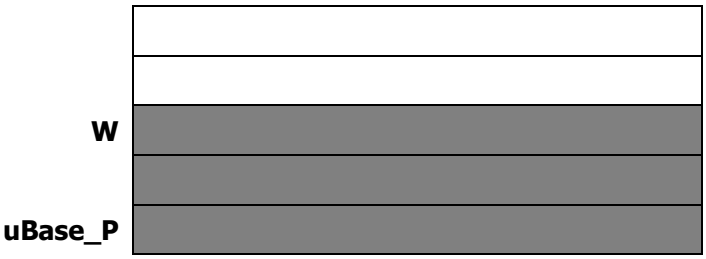
SWAP FILE	
s0: *Ps0 / Qs0*	s1: *----*

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	**F**	*13000*	*2*	*4*	*2*
Q	**G**	*8000*	*1*	*2*	*0*

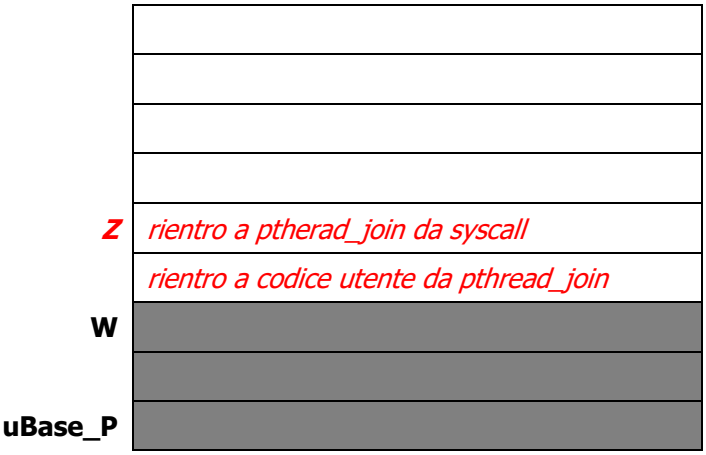
esercizio n. 4 – domande varie

prima domanda – moduli del SO

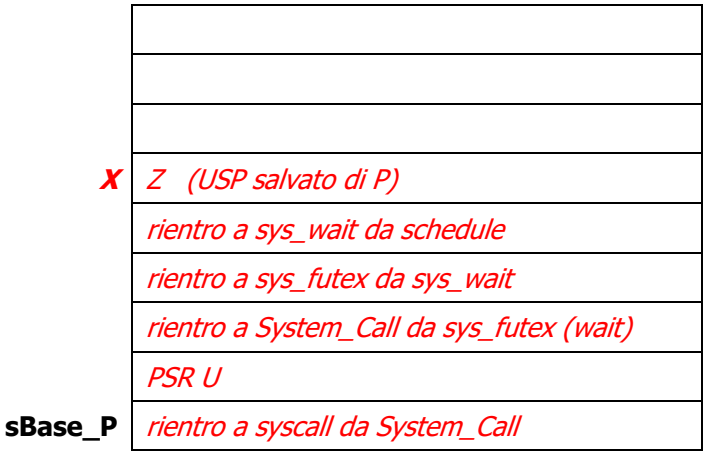
Un task **P** esegue la funzione **pthread_join (TH_Q, ...)** e si autosospende attendendo la terminazione di un task **Q** (di tipo thread), che al momento è in stato di **pronto** per scadenza di quanto di tempo. Si **mostrino** lo stato finale (cioè al momento del context switch) della **uPila** e 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)

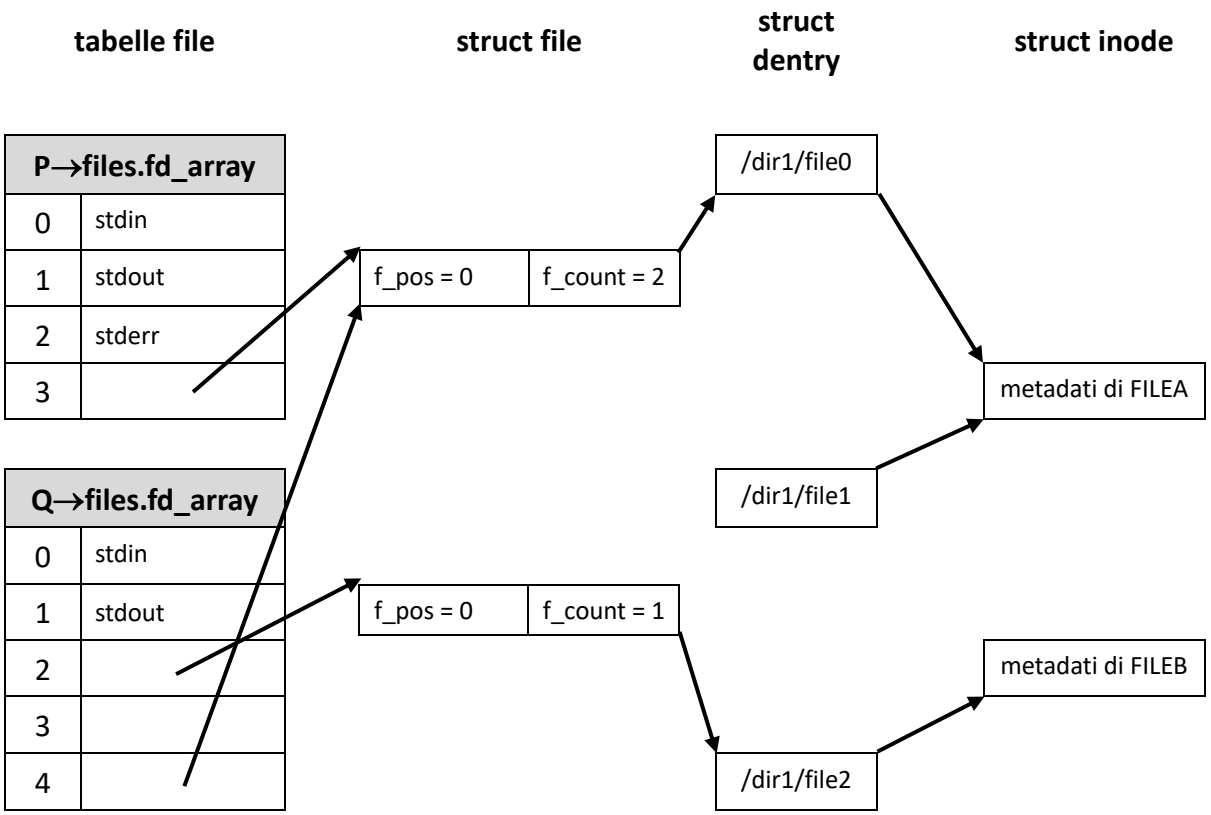
strutture dati finali del task P (da completare)	
registro PC	// non di interesse
registro SP	*X*
SSP	*sBase_P*
USP	*Z*
descrittore di P.stato	*ATTESA (per join)*

sStack_Q (da completare)
USP salvato di Q
~~*a schedule da pick_next_task*~~ *(pick_next_task has chosen task P as next)*
a R_int_CK da schedule
~~*a R_int_CK da Controlla_timer*~~ *(there are no timers, nothing happens)*
~~*a task tick da resched*~~ *(resched has set NEED_RESCHED to 1)*
~~*a R_int_CK da task_tick*~~
PSR (U)
a codice utente da R_int_CK

Nota: pthread_join utilizza il servizio futex (fast userspace mutex); se il descrittore di task TH_Q indica che il task Q è terminato, si rimane in modo U e si continua, altrimenti tramite meccanismo syscall si passa a modo S e si chiama il servizio sys_futex con parametro wait (come succede qui).

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):

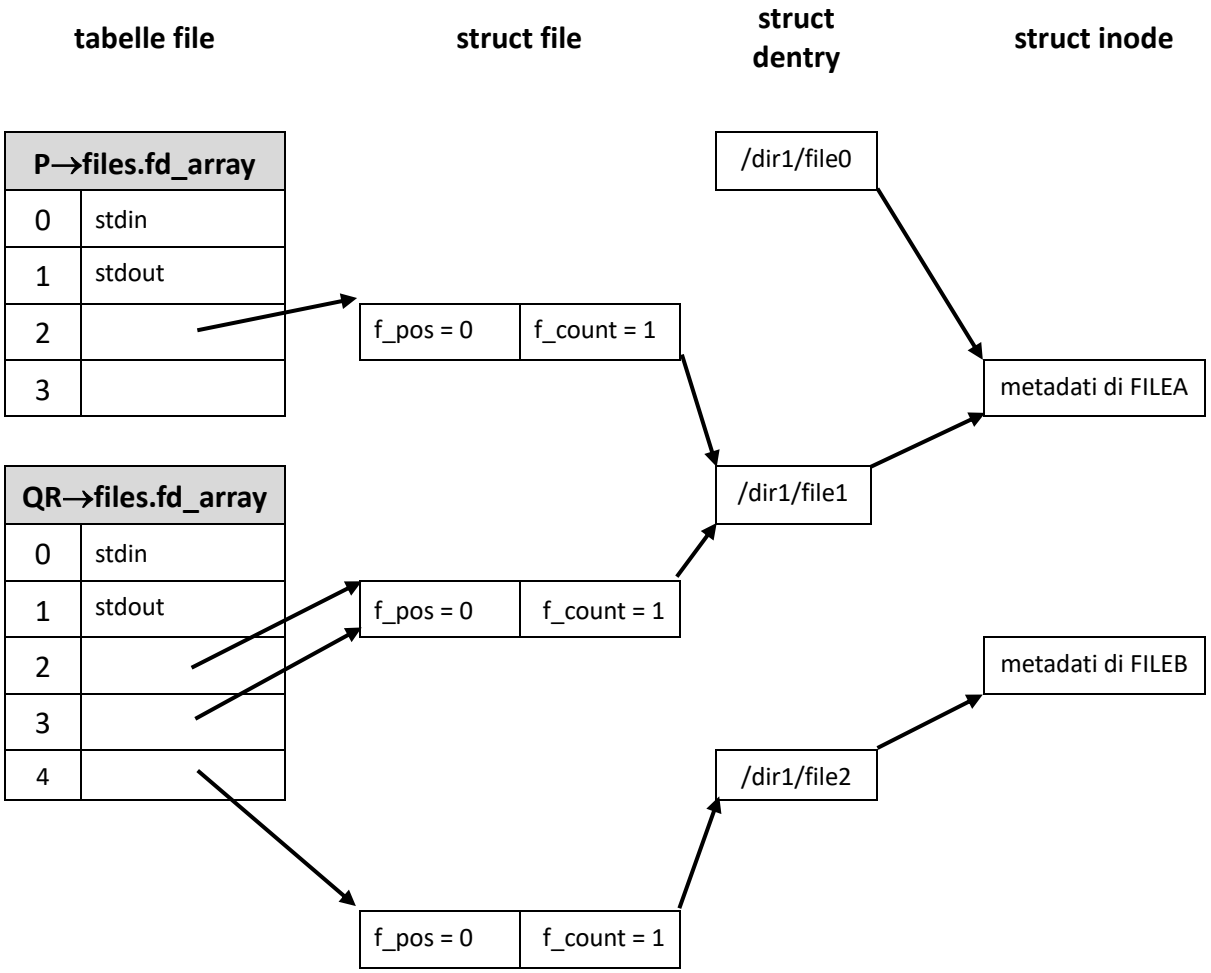


Il task (normale) **P** ha creato il task (normale) figlio **Q**.

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 (in aggiunta ai due già dati) 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	*close (2)*
3	P	*open ("/dir1/file1", ...)*
4	Q	*close (2)*
5	Q	*open ("/dir1/file1", ...)*
6	R	dup (3)
7	R	*close (4)*
8	R	*open ("/dir1/file2", ...)*

Nota: l'ordine delle operazioni è deterministico, e si ricordi che i task di un medesimo thread group condividono la tabella dei file aperti.