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 – mercoledì 8 febbraio 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 mars, venus
sem_t sun
int global = 0
```

```
void * war (void * arg) {
    sem_wait (&sun)
    mutex_lock (&venus)
    sem_post (&sun)
    global = 1                                          /* statement A */
    mutex_unlock (&venus)
    global = 2
    mutex_lock (&mars)
    sem_wait (&sun)
    mutex_unlock (&mars)
    return (void *) 3
} /* end war */
```

```
void * peace (void * arg) {
    mutex_lock (&venus)
    sem_wait (&sun)
    mutex_unlock (&venus)
    global = 4                                          /* statement B */
    sem_post (&sun)
    sem_wait (&sun)
    mutex_lock (&mars)
    global = 5
    sem_post (&sun)
    mutex_unlock (&mars)                                /* statement C */
    return NULL
} /* end peace */
```

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&sun, 0, 1)
    create (&th_1, NULL, war, NULL)
    create (&th_2, NULL, peace, NULL)
    join (th_1, &global)                                /* statement D */
    join (th_2, 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	***thread***	
	th_1 – *war*	**th_2 – *peace***
subito dopo stat. **A**	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **B**	*ESISTE*	*ESISTE*
subito dopo stat. **C**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **D**	*NON ESISTE*	*PUÒ ESISTERE*

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**		
	mars	*venus*	*sun*
subito dopo stat. **B**	*0 / 1*	*0*	*0*
subito dopo stat. **C**	*0 / 1*	*0 / 1*	*0 / 1*
subito dopo stat. **D**	*0*	*0 / 1*	*0*

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in (almeno) **QUATTRO 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*:

caso	**th_1 – *war***	**th_2 – *peace***	*global*
1	`lock venus`	`prima wait sun`	*0*
2	`seconda wait sun`	`lock mars`	*2, 4*
3	`--`	`prima wait sun`	*2, 3*
4	`--`	`seconda wait sun`	*2, 3, 4*

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma main.c
sem_t second
char strP [9] = "P-hello!"
```

`void * cat (void * arg) {`	`void * dog (void * arg) {`
`char str [4] = {0}`	`char str [4] = "dog"`
`sem_wait` `(&second)`	**`sem_wait`** `(&second)`
`read` `(stdin, str, 3)`	**`write`** `(stderr, str, 3)`
`return NULL`	`return NULL`
`} // end`	`} // end`

```
int main ( ) { // codice eseguito da P
   sem_init (&second, 0, 0)
   pid_t pidQ = fork ( )
   if (pid == 0) {                  // codice eseguito da Q
      pthread_t TH_1, TH_2
      pthread_create (&TH_1, NULL, cat, NULL)
      pthread_create (&TH_2, NULL, dog, NULL)
      pthread_join (TH_1, NULL)
      sem_post (&second)
      pthread_join (TH_2, NULL)
      exit (1)
   } else {                         // codice eseguito da P
      sem_post (&second)
      write (stdout, strP, 8)
      sem_post (&second)
      exit (0)
   } // end_if pid
} // end main
```

Un processo **P** esegue il programma **`main.c`** e crea un processo figlio **Q**, che a sua volta crea i due thread **TH_1** e **TH_2**. 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 task **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	Q	TH_1	TH_2
evento oppure processo-chiamata	PID	1	2	*3*	*4*	*5*
	TGID	1	2	*3*	*3*	*3*
P – fork	**0**	**pronto**	**esec**	**pronto**	**NE**	**NE**
P – sem_post	**1**	*pronto*	*esec*	*pronto*	*NE*	*NE*
P – write	**2**	*pronto*	*attesa (write)*	*esec*	*NE*	*NE*
Q – pthread_create TH_1	**3**	*pronto*	*attesa (write)*	*esec*	*pronto*	*NE*
Q – pthread_create TH_2	**4**	*pronto*	*attesa (write)*	*esec*	*pronto*	*pronto*
Q – pthread_join TH_1	**5**	*pronto*	*attesa (write)*	*attesa (TH1)*	*esec*	*pronto*
TH_1 – sem_wait	**6**	*pronto*	*attesa (write)*	*attesa (TH1)*	*esec*	*pronto*
TH_1 – read	**7**	*pronto*	*attesa (write)*	*attesa (TH1)*	*attesa (read)*	*esec*
TH_2 – wait	**8**	*esec*	*attesa (write)*	*attesa (TH1)*	*attesa (read)*	*attesa (semwait)*
interrupt da stdout, otto caratteri inviati	**9**	*pronto*	*esec*	*attesa (TH1)*	*attesa (read)*	*attesa (semwait)*
P – sem_post	**10**	*pronto*	**pronto**	*attesa (TH1)*	*attesa (read)*	**esec**

caso 1: sequenza finale qualora *TH_2 <u>non</u> termini*

TH_2 – write	**11**	*pronto*	*esec*	*attesa (TH1)*	*attesa (read)*	*attesa (write)*
P – exit	**12**	*esec*	*NE*	*attesa (TH1)*	*attesa (read)*	*attesa (write)*

caso 2: sequenza finale qualora *TH_2 termini*

TH_2 – write	**11**	*pronto*	*esec*	*attesa (TH1)*	*attesa (read)*	*attesa (write)*
interrupt da stderr, tre caratteri inviati	**12**	*pronto*	*pronto*	*attesa (TH1)*	*attesa (read)*	*esec*
TH_2 – exit (terminazione implicita con "}")	**13**	*pronto*	*esec*	*attesa (TH1)*	*attesa (read)*	*NE*
P – exit	**14**	*esec*	*NE*	*attesa (TH1)*	*attesa (read)*	*NE*

seconda parte – moduli di kernel

Si consideri uno scenario dove sono presenti (oltre al processo **Idle**) tre processi **P**, **Q** e **R**. Il processo **P** ha già creato il processo figlio **Q** e ora si mette in attesa della terminazione di **Q**, con conseguente ripresa dell'esecuzione del processo **R**. Il processo **R** è in stato di pronto a seguito del risveglio per completamento di una **read** da *stdin*. Nel sistema non ci sono altri processi.

Mostrare le **invocazioni** di tutti i moduli (ed eventuali relativi ritorni) eseguiti nel contesto del processo **P** e del processo **R** (fino a quando il processo **R** sarà tornato in esecuzione a codice utente in modo U) per ottenere lo scenario descritto.

Si specifichino anche i **punti dove la variabile** globale di nucleo TIF_NEED_RESCHED viene **modificata** o **controllata**.

processo	**modo**	**modulo** (numero di righe non significativo)
P	**U**	> **waitpid**
P	*U – S*	*> syscall: SYSCALL*
P	*S*	*> System_Call*
P	*S*	*> sys_wait*
P	*S*	*> schedule*
P	*S*	*> pick_next_task <*
P – R	*S*	*schedule: context_switch*
R	*S*	*schedule (NEED_RESCHED viene posta a 0) <*
R	*S*	*wait_event <*
R	*S*	*sys_read <*
R	*S – U*	*System_Call: SYSRET (dato che NEED_RESCHED è 0) <*
R	*U*	*syscall <*
R	*U*	*read <*

PAGINA BIANCA DI ALLINEAMENTO – usabile per continuazione o brutta copia

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 (esistono un processo **P** e un processo **R**)

```
PROCESSO: P ***********************************************************
    VMA:  C  000000400,  2 , R  ,  P  ,  M  , <XX, 0>
          K  000000600,  1 , R  ,  P  ,  M  , <XX, 2>
          S  000000601,  1 , W  ,  P  ,  M  , <XX, 3>
          P  7FFFFFFF9,  6 , W  ,  P  ,  A  , <-1, 0>
    PT: <c0 :-  ->  <c1 :1 R>   <k0 :- ->  <s0 :- ->  <p0 :2 W>
        <p1 :7 R>   <p2 :s2 R>  <p3 :4 R>  <p4 :6 W>  <p5 :- ->
    process P - NPV of PC and SP:  c1, p1
PROCESSO: R  **********************************************************
    VMA:  C  000000400,  2 , R  ,  P  ,  M  , <XX, 0>
          K  000000600,  1 , R  ,  P  ,  M  , <XX, 2>
          S  000000601,  1 , W  ,  P  ,  M  , <XX, 3>
          P  7FFFFFFF9,  6 , W  ,  P  ,  A  , <-1, 0>
    PT: <c0 :- ->  <c1 :1 R>   <k0 :- ->  <s0 :- ->    <p0 :3 R>
        <p1 :7 R>  <p2 :s2 R>  <p3 :4 R>  <p4 :5 D W>  <p5 :- ->
    process R - NPV of PC and SP:  c1, p4
____MEMORIA FISICA____(pagine libere: 2)_____________________________
      00 : <ZP>                  ||  01 : Pc1 / Rc1 / <XX, 1>  ||
      02 : Pp0                   ||  03 : Rp0                  ||
      04 : Pp3 / Rp3             ||  05 : Rp4  D               ||
      06 : Pp4                   ||  07 : Pp1 / Rp1            ||
      08 : ----                  ||  09 : ----                 ||
____STATO del TLB_____________________________________________________
      Pc1 : 01 -  0: 1:          ||  Pp0 : 02 -  1: 1:         ||
      Pp1 : 07 -  0: 1:          ||        -----               ||
      Pp3 : 04 -  1: 0:          ||  Pp4 : 06 -  1: 0:         ||
            -----                ||        -----               ||
SWAP FILE:    Rp0,  Pp1/Rp1,  Pp2/Rp2,  ----,  ----,  ----,  ----
LRU ACTIVE:   PP1,  PP0,  PC1
LRU INACTIVE: pp4,  pp3,  rp4,  rp3,  rc1,  rp0,  rp1
```

evento 1: *read* (Pc1) – *write* (Pp2) – 4 *kswapd*

Legge la pagina Pc1, nessun problema. Scrive la pagina Pp2: ci sono due pagine libere, scatta PFRA che libera 03 (la pagina Rp0, che è già in swap file, slot s0, viene tolta da liste), e che libera 05 (la pagina Rp4 D viene scritta in swap file, slot s3, e viene tolta da liste). La pagina condivisa Pp2/Rp2 viene caricata in 03 e lasciata in swap file (slot s2), la pagina Pp2 va in testa active, la pagina Rp2 va in coda inactive, si ha COW per pagina Pp2, quindi Pp2 va in 05, Rp2 resta in 03 e in swap file (slot s2), e in swap file viene cancellato il riferimento a Pp2 (resta solo quello a Rp2). Ci sono due pagine libere. La prima KSWAPD attiva PFRA, perché free < maxfree, e libera una pagina, ossia la 03 (perché la pagina Rp2 è ultima in inactive), la pagina Rp2 è già in swap file (slot s2) e viene eliminata da lista. Poi tutto rimane come prima, tranne gli elementi in liste che vengono aggiornate.

PT del processo: **P**				
p0: *2 W*	p1: *7 R*	p2: *5 W*	p3: *4 R*	p4: *6 W*

process P	NPV of **PC**: *c1*	NPV of **SP**: *p2*

PT del processo: R				
p0: *s0 R*	p1: *7 R*	p2: *s2 R*	p3: *4 R*	p4: *s3 W*

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Rc1 / <XX, 1>
02: *Pp0*	03: ~~*Rp0 Rp2*~~ -----
04: *Pp3 / Rp3*	05: ~~*Rp4 D*~~ *Pp2*
06: *Pp4*	07: *Pp1 / Rp1*
08: -----	09: -----

SWAP FILE	
s0: Rp0	s1: Pp1 / Rp1
s2: ~~*Pp2 /*~~ *Rp2*	s3: *Rp4*
s4: ----	s5: ----

LRU ACTIVE: *PP2, PC1*_____

LRU INACTIVE: *pp1, pp0, pp4, pp3, rp3, rc1, rp1*_____

evento 2: *mmap* (0x 0000 0300 0000 0000, 3, W, S, M, F, 2)
mmap (0x 0000 0400 0000 0000, 3, W, P, M, G, 0)
read (Pm11, Pm02)

Viene creata la VMA M0 al NPV 0x 0300 0000 0, di tre pagine, scrivibile, shared (condivisa) e mappata su file F con offset 2. Viene creata la VMA M1 al NPV 0x 0400 0000 0, di tre pagine, scrivibile, privata e mappata su file G con offset 0. Le pagine virtuali relative vengono inserite nella tabella delle pagine del processo P, ma non (ancora) allocate in memoria fisica.

Legge la pagina Pm11: la pagina Pm11 (mappata su <G, 0 + 1 = 1>) viene allocata in 03, in testa ad active, e si aggiorna la PT (marcando Pm11 con R per predisporla a COW, essendo M1 un'area privata mappata su file). Legge la pagina Pm02: ci sono due pagine libere, si attiva PFRA che libera 04 (la pagina condivisa Pp3 / Rp3 viene scritta in swap file, slot s4, e tolta da liste), e che libera 06 (la pagina Pp4 viene scritta in swap file, slot s5, e tolta da liste). La pagina Pm02 (mappata su <F, 2 + 2 = 4>) viene caricata in 04 e inserita in active, e si aggiorna la PT (marcando Pm02 con W perché l'area M0 è shared non predisposta per COW).

VMA del processo **P** (è da compilare solo la riga relativa alle VMA **create**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
M0	*0300 0000 0*	*3*	*W*	*S*	*M*	*F*	*2*
M1	*0400 0000 0*	*3*	*W*	*P*	*M*	*G*	*0*

PT del processo: **P**				
p2: *5 W*	p3: *s4 R*	p4: *s5 W*	p5: - -	m00: - -
m01: - -	m02: *4 W*	m10: - -	m11: *3 R*	m12: - -

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Rc1 / <XX, 1>
02: *Pp0*	03: *Pm11 / <G, 1>*
04: ~~*Pp3/Rp3*~~ *Pm02 / <F, 4>*	05: *Pp2*
06: ~~*Pp4*~~ -----	07: *Pp1 / Rp1*
08: -----	09: -----

SWAP FILE	
s0: Rp0	s1: Pp1 / Rp1
s2: *Rp2*	s3: *Rp4*
s4: *Pp3 / Rp3*	s5: *Pp4*

LRU ACTIVE: *PM02, PM11, PP2, PC1* _____

LRU INACTIVE: *pp1, pp0, rc1, rp1* _____

(continua sulla prossima pagina)

evento 3: *read* (Pm11, Pp1, Pc1) – *write* (Pm02) – *4 kswapd*

Nessun problema con le letture. Nessun problema con la scrittura della pagina Pm02, perché l'area M0 è shared (niente COW). KSWAPD: solo aggiornamento delle liste.

process P	NPV of **PC**: *c1*	NPV of **SP**: *p1*

STATO DEL TLB	
Pc1: 01 – 0 : 1	Pp0: 02 – 1 : 0
Pp1: 07 – 0 : 1	Pp2: 05 – 1 : 0
Pm02: 04 – 1 : 1	-------
Pm11: 03 – 0 : 1	-------

LRU ACTIVE: *PM02, PM11, PC1, PP1* _____

LRU INACTIVE: *pp2, pp0, rc1, rp1* _____

evento 4: *write* (Pm11)

Scrive la pagina Pm11. la pagina Pm11 è in VMA M1, privata e mappata su file, quindi si ha COW (Pm11 era stata predisposta con protezione R) e va allocata una pagina, in questo caso la 06 che è libera, mentre la 03 rimane solo in page cache mappata su <G, 1>; non c'è nessun'altra modifica.

PT del processo: **P**				
m01: – –	m02: *4 W*	m10: – –	m11: *6 W*	m12: – –

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Rc1 / <XX, 1>
02: *Pp0*	03: ~~*Pm11 /* <G, 1>~~
04: *Pm02 / <F, 4>*	05: *Pp2*
06: *Pm11*	07: *Pp1 / Rp1*
08: -----	09: -----

PAGINA BIANCA DI ALLINEAMENTO – usabile per continuazione o brutta copia

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:** è presente un unico processo **P**, in esecuzione

```
PROCESSO: P ************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <X, 0>
          S  000000600,  2 , W  ,  P  ,  M  , <X, 2>
          D  000000602,  2 , W  ,  P  ,  A  , <-1, 0>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1, 0>
    PT: <c0 :1 R>  <c1 :- ->  <s0 :- ->  <s1 :- ->
        <d0 :- ->  <d1 :- ->
        <p0 :2 W>  <p1 :- ->  <p2 :- ->
    process P - NPV of PC and SP:  c0, p0
____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:        ||
      ------                   ||        -----              ||
LRU ACTIVE:   PP0,  PC0

LRU INACTIVE:
```

Per le informazioni relative a **apertura/accesso/chiusura dei file** si utilizzano le usuali tabelle.

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

ATTENZIONE: **è presente la colonna "processo" dove specificare 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.**

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: fd1 = *open* (F) – *read* (fd1, 13000)

*Il processo P apre il file F, legge le **pagine 0, 1, 2 e 3** del file e le carica ordinatamente nelle pagine libere 03, 04, 05 e 06 di memoria. Quattro letture da disco (per il file F), nessuna scrittura su disco. Resta una pagina libera.*

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <X, 0>
02: *Pp0*	03: *<F, 0>*
04: *<F, 1>*	05: *<F, 2>*
06: *<F, 3>*	07: ----

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

evento 2: fd1 = *write* (fd1, 7000) – *fork* (Q) – *context switch* (Q)

Il processo P scrive le pagine file ***<F, 3> e <F, 4>.*** *Scrittura in <F, 3> in memoria già allocata e marcata D. Per <F, 4> va allocata una pagina.* ***Scatta PFRA che libera 03 (<F, 0>) e 04 (<F, 1>).*** *La pagina <F, 4> viene allocata in 03 e marcata D. Una lettura da disco (file F), nessuna scrittura su disco.*

Esecuzione di fork: le pagine di P vengono condivise con Q e si devono separare le pagine di testa pila Qp0 e Pp0, allocando una pagina per Pp0. La pagina Qp0 rimane in 02 e risulterà scritta. La pagina Pp0 è allocata in 04 e viene marcata D in TLB. Il processo Q eredita l'apertura del file F (f_count = 2). Context switch: flush del TLB, la pagina 04 (Pp0) viene marcata D (da TLB), e il processo Q va in esecuzione.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: ~~*Pp0 / Qp0 D*~~	03: ~~*<F, 0>*~~ *<F, 4> D*
04: ~~*<F, 1> Pp0 D*~~	05: *<F, 2>*
06: *<F, 3> D*	07: ----

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	*F*	*20000*	*2*	*5*	*0*

evento 3: fd2 = *open* (G) – *write* (fd2, 8500)

Il processo Q apre il file G e scrive ***tre pagine, 0, 1 e 2.*** *Pagine libere = 1, quindi scatta PFRA che libera 03 (pagina <F, 4> con scrittura su disco) e 05 (pagina <F, 2>). La pagina <G, 0> viene caricata in 03 e scritta in memoria; la pagina <G, 1>. viene caricata in 05 e scritta in memoria. Scatta di nuovo PFRA che libera (di nuovo) 03 e 05 con relative scritture su disco., poi la pagina <G, 2> viene caricata in 03 e scritta in memoria.*

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: *Qp0 D*	03: ~~*<F, 4> D <G, 0> D*~~ *<G, 2> D*
04: *Pp0 D*	05: ~~*<F, 2><G, 1> D*~~
06: *<F, 3> D*	07: ----

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	*F*	*20000*	*2*	*5*	*1*
Q	*G*	*8500*	*1*	*3*	*2*

evento 4: *write* (fd1, 3000)

Il processo Q scrive le due pagine di file ***<F, 4> e <F, 5>.***

Carica la pagina <F, 4> in 05 e la scrive in memoria.

Pagine libere = 2, dunque per la pagina <F, 5> scatta PFRA che libera 03 con scrittura di pagina <G, 2> su disco, e che libera 05 con scrittura di pagina<F, 4> su disco. Infine la pagina <F, 5> viene caricata in 03 e scritta in memoria.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: *Qp0 D*	03: ~~*<G, 2> D*~~ *<F, 5> D*
04: *Pp0 D*	05: ~~*<F, 4 > D*~~
06: *<F, 3> D*	07: ----

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	*F*	*23000*	*2*	*7*	*2*
Q	*G*	*8500*	*1*	*3*	*3*

evento 5: *close* (fd1) – *close* (fd2)

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, ~~*Q*~~	*F*	*23000*	*1*	*7*	*2*
~~*Q*~~	~~*G*~~	~~*8500*~~	~~*0*~~	~~*3*~~	~~*3*~~

esercizio n. 4 – tabella delle pagine

Date le VMA di un processo P sotto riportate, definire:

1. la scomposizione degli indirizzi virtuali dello NPV iniziale di ogni area secondo la notazione **PGD : PUD : PMD : PT**
2. il numero di pagine necessarie in ogni livello della gerarchia e il numero totale di pagine necessarie per rappresentare la Tabella delle Pagine (TP) del processo
3. il numero di pagine virtuali occupate dal processo
4. il rapporto tra l'occupazione della TP e la dimensione virtuale del processo in pagine
5. la dimensione virtuale massima del processo in pagine, senza dovere modificare la dimensione della TP
6. il rapporto relativo

VMA del processo **P**							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	000000400	3	R	P	M	X	0
K	000000600	2	R	P	M	X	3
S	000000602	6	W	P	M	X	5
D	000000608	4	W	P	A	-1	0
M0	000010000	2	W	S	M	G	0
M1	000020000	1	R	S	M	G	4
M2	000030000	1	W	P	M	F	2
M3	000050000	1	W	P	A	-1	0
T2	7FFFF77F8	2	W	P	A	-1	0
T1	7FFFF77FB	2	W	P	A	-1	0
T0	7FFFF77FE	2	W	P	A	-1	0
P	7FFFFFFFC	3	W	P	A	-1	0

1. Scomposizione degli indirizzi virtuali:

		PGD :	**PUD :**	**PMD :**	**PT**
C	000000400	0	0	2	0
K	000000600	0	0	3	0
S	000000602	*0*	*0*	*3*	*2*
D	000000608	*0*	*0*	*3*	*8*
M0	000010000	0	0	128	0
M1	000020000	*0*	*0*	*256*	*0*
M2	000030000	*0*	*0*	*384*	*0*
M3	000050000	0	1	128	0
T2	7FFFF77F8	*255*	*511*	*443*	*504*
T1	7FFFF77FB	*255*	*511*	*443*	*507*
T0	7FFFF77FE	255	511	443	510
P	7FFFFFFFC	*255*	*511*	*511*	*508*

2. Numero di pagine necessarie:

pag PGD: *1*

pag PUD: *2*

pag PMD: *3*

pag PT: *8*

pag totali: *14*

3. Numero di pagine virtuali occupate dal processo: *29*

4. Rapporto di occupazione: *14 / 29 ≈ 0,48 (circa 48 %)*

5. Dimensione massima del processo in pagine virtuali:

   con la stessa dimensione di TP il processo può crescere fino a 8 × 512 = 4096 pagine virtuali

6. Rapporto di occupazione con dimensione massima: *14 / 4096 ≈ 0,0034 (circa 3,4 millesimi)*

spazio libero per brutta copia o continuazione