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 – martedì 12 settembre 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 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) _____

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 door
sem_t in, out
int global = 0
```

```
void * entry (void * arg) {
   mutex_lock (&door)
   sem_wait (&in)
   global = 1                                          /* statement A */
   mutex_unlock (&door)
   mutex_lock (&door)
   sem_post (&out)
   sem_post (&in)
   mutex_unlock (&door)
   global = 2                                          /* statement B */
   return NULL
} /* end entry */
```

```
void * exit (void * arg) {
   mutex_lock (&door)
   sem_wait (&in)
   mutex_unlock (&door)
   sem_wait (&out)
   mutex_lock (&door)
   global = 4                                          /* statement C */
   mutex_unlock (&door)
   return (void * 5)
} /* end exit */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&in, 0, 1)
   sem_init (&out, 0, 0)
   create (&th_2, NULL, exit, NULL)
   create (&th_1, NULL, entry, NULL)
   join (th_2, &global)                                /* statement D */
   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	***thread***	
	th_1 – *entry*	**th_2 – *exit***
subito dopo stat. **A**	*ESISTE*	*ESISTE*
subito dopo stat. **B**	*ESISTE*	*PUÒ ESISTERE* *(esiste o è già terminato)*
subito dopo stat. **C**	*PUÒ ESISTERE* *(esiste o è già terminato)*	*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**		
	door	*in*	*global*
subito dopo stat. **A**	*1*	*0*	*1*
subito dopo stat. **B**	*0 / 1*	*0 / 1*	*2 / 4 / 5*
subito dopo stat. **C**	*1*	*0*	*2 / 4*
subito dopo stat. **D**	*0*	*0*	*2 / 5*

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 – *entry***	**th_2 – *exit***	*global*
1	`wait in`	`wait out`	*0*
2	`2a lock door`	`wait in`	*1*
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma main.c
sem_t sem
char s[] = "Hello world!"  // nota: strlen(s) == 12
int onw = 0
int back = 12 - 2
```

```
void add (void * arg) {
  if (onw <= back) {
    sem_post (&sem)
    onw = onw + back
    write (stderr, &s[onw], 1)
  } else {
    sem_wait (&sem)
  } // end if
} // end
```

```
void sub (void * arg) {
  sem_wait (&sem)
  back = back - onw
  if (onw > back) {
    write (stderr, &s[back], 1)
    sem_post (&sem)
    return
  } // end if
} // end
```

```
int main ( ) { // codice eseguito da P
   pid_t pidP, pidQ
   pthread_t TH_1, TH_2
   sem_init (&sem, 0, 1)
   pidQ = fork ( )
   if (pidQ != 0) {                  // codice eseguito da P
      onw = own + 6
      pthread_create (&TH_1, NULL, sub, NULL)
      pthread_create (&TH_2, NULL, add, NULL)
      write (stdout, &s[onw], 6)
      pthread_join (TH_2, NULL)
      pthread_join (TH_1, NULL)
      pidP = wait (NULL)
      exit (1)
   } else {                          // codice eseguito da Q
      read (stdin, &s[onw], 6)
   } // end_if pid
} // end main
```

Un processo **P** esegue il programma **`main.c`**, tramite cui crea un processo figlio **Q** e 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.

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*	*2*	*2*
P – fork	**0**	**pronto**	**esec**	**pronto**	**NE**	**NE**
P – pthread_create TH_1	**1**	*pronto*	*esec*	*pronto*	*pronto*	*NE*
P – pthread_create TH_2	**2**	*pronto*	*esec*	*pronto*	*pronto*	*pronto*
P – write	**3**	*pronto*	*attesa (write stdout)*	*esec*	*pronto*	*pronto*
Q – read	**4**	*pronto*	*attesa (write stdout)*	*attesa (read stdin)*	*esec*	*pronto*
TH_1 – sem_wait	**5**	*pronto*	*attesa (write stdout)*	*attesa (read stdin)*	*esec*	*pronto*
TH_1 – write	**6**	*pronto*	*attesa (write stdout)*	*attesa (read stdin)*	*attesa (write stderr)*	*esec*
TH_2 – wait	**7**	*esec*	*attesa (write stdout)*	*attesa (read stdin)*	*attesa (write stderr)*	*attesa (sem_wait)*
interrupt da stderr 1 carattere inviato	**8**	*pronto*	*attesa (write stdout)*	*attesa (read stdin)*	**esec**	*attesa (sem_wait)*
TH_1 – sem_post	**9**	*pronto*	*attesa (write stdout)*	*attesa (read stdin)*	*pronto*	*esec*
TH_2 – exit	**10**	*pronto*	*attesa (write stdout)*	*attesa (read stdin)*	*esec*	*NE*
interrupt da stdout sei caratteri inviati	**11**	*pronto*	*esec*	*attesa (read stdin)*	*pronto*	*NE*
P – pthread_join TH_2	**12**	*pronto*	*esec*	*attesa (read stdin)*	*pronto*	*NE*
P – pthread_join TH_1	**13**	*pronto*	*pronto*	*attesa (read stdin)*	*esec*	*NE*
TH_1 – exit	**14**	*pronto*	*esec*	*attesa (read stdin)*	*NE*	*NE*
P – wait	**15**	*esec*	*attesa (wait)*	*attesa (read stdin)*	*NE*	*NE*
interrupt da stdin 6 caratteri inviati	**16**	*pronto*	*attesa (wait)*	*esec*	*NE*	*NE*
Q – exit	**17**	*pronto*	*pronto*	*NE*	*NE*	*NE*
P – exit	**18**	*esec*	*NE*	*NE*	*NE*	*NE*

seconda parte – scheduler CFS

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

CONDIZIONI INIZIALI (già complete)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	3	T1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T1	2	0,67	4	0,5	10	100
RB	T2	1	0,33	2	1	20	102

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

```
Events of task t1:  WAIT at 1.0;  WAKEUP after 2.5
Events of task t2:  CLONE at 2.0
```

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**		
	1	*WAIT*	*T1*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	1	6	*1*	*T2*	*100,50*	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T2*	*1*	*1*	*6*	*1*	*20*	*102*
RB							
WAITING	*T1*	*2*				*11*	*100,50*

EVENTO 2	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
	3	*CLONE*	*T2*	*falso*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	2	6	*2*	*T2*	*104*	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T2*	*1*	*0,50*	*3*	*1*	*22*	*104*
RB	*T3*	*1*	*0,50*	*3*	*1*	*0*	*107*
WAITING	*T1*	*2*				*11*	*100,50*

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
		3,5	*WUP*	*T2*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*4*	*T1*	*104,5*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T1*	*2*	*0,50*	*3*	*0,5*	*11*	*101,5*
RB	*T2*	*1*	*0,25*	*1,5*	*1*	*22,5*	*104,5*
	T3	*1*	*0,25*	*1,5*	*1*	*0*	*107*
WAITING							

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
		6,5	*Q_scade*	*T1*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*4*	*T1*	*104,5*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T1*	*2*	*0,50*	*3*	*0,5*	*14*	*103*
RB	*T2*	*1*	*0,25*	*1,5*	*1*	*22,5*	*104,5*
	T3	*1*	*0,25*	*1,5*	*1*	*0*	*107*
WAITING							

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

$T1.VRT + WGR \times T1.LC < T2.VRT \Rightarrow 101,5 + 1 \times 0,5 = 102 < 104,5 \Rightarrow$ *vero*

*Calcolo del VRT del **task T1** risvegliato della **WAKEUP**:*

$T1.VRT = max\ (T1.VRT,\ VMIN - LT / 2) = max\ (101,5,\ 104,5 - 6 / 2) = max\ (101,5,\ 100,5) = 101,5$

esercizio n. 3 – memoria virtuale e file system

prima parte – memoria virtuale

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = **2** **MINFREE** = **1**

situazione iniziale (esistono un processo **P** e un processo **Q**, il processo **P** è 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 :- ->  <c1 :1 R>  <s0 :5 R>  <s1 :- ->  <d0 :3 R>  <d1 :- ->
        <p0 :2 R>  <p1 :6 W>  <p2 :- ->

    process P - NPV of PC and SP:  c1, p1

PROCESSO: Q  ************************************************************
    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 :- ->  <c1 :1 R>    <s0 :5 R>  <s1 :- ->  <d0 :3 R>  <d1 :- ->
        <p0 :2 R>  <p1 :4 D W>  <p2 :- ->

    process Q - NPV of PC and SP:  c1, p1

____MEMORIA FISICA____(pagine libere: 1)_____________________________
      00 : <ZP>                  ||  01 : Pc1 / Qc1 / <X, 1>    ||
      02 : Pp0 / Qp0             ||  03 : Pd0 / Qd0             ||
      04 : Qp1  D                ||  05 : Ps0 / Qs0 / <X, 2>    ||
      06 : Pp1                   ||  07 : ----                  ||

____STATO del TLB_____________________________________________________
      Pc1 : 01 -  0: 1:          ||   Pp0 : 02 -  1: 0:         ||
      Pd0 : 03 -  1: 0:          ||   Pp1 : 06 -  1: 1:         ||
      Ps0 : 05 -  0: 1:          ||         -----               ||

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

LRU ACTIVE:   QS0,  QP1,  QC1,  PS0,  PP1,  PC1

LRU INACTIVE: qd0,  qp0,  pd0,  pp0
```

evento 1: *write* (Pd0)

COW per Pd0; una sola pagina libera, si attiva PFRA che libera pagine 02 e 03. Quindi Pp0 / Qp0 e Pd0 / Qd0 vanno in swap file (swap_out) e vengono tolte da INACTIVE aggiornando le PT di P e Q. Pd0 viene caricata in 02 (swap_in) e tolta da swap file, e viene aggiornata la PT di P. Aggiornata anche la ACTIVE.

PT del processo: **P**				
d0: *2 W*	d1: – –	p0: *s0 R*	p1: *6 W*	

PT del processo: **Q**				
d0: *s1 R*	d1: – –	p0: *s0 R*	p1: *4 D W*	

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <X, 1>
02: ~~*Pp0 / Qp0*~~ *Pd0*	03: ~~*Pd0 / Qd0*~~
04: *Qp1 D*	05: *Ps0 / Qs0 / <X, 2>*
06: *Pp1*	07:

SWAP FILE	
s0: *Pp0 / Qp0*	s1: *Qd0*
s2:	s3:

LRU ACTIVE: *PD0, QS0, QP1, QC1, PS0, PP1, PC1*_____

LRU INACTIVE: _____

evento 2: *read* (Pc1) – 4 *kswapd*

LRU ACTIVE: *PC1* _____

LRU INACTIVE: *pd0, ps0, pp1, qs0, qp1, qc1* _____

evento 3: *read* (Ps1) – *write* (Pd1)

Carica Ps1 in 03 e aggiorna PT di P e liste. Per caricare Pd1 si attiva PFRA, che libera da INACTIVE 04 (Qp1) e 06 (Pp1). Entrambe le pagine vengono scritte in swap file (swap_out) per marca D e per stato del TLB. Viene aggiornata PT di P e INACTIVE. Pd1 viene caricata in 04, e vengono aggiornate PT e ACTIVE.

PT del processo: **P**				
s0: *5 R*	s1: *3 R*			
d0: *2 W*	d1: *4 W*	p0: *s0 R*	p1: *s3 W*	

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <X, 1>
02: *Pd0*	03: *Ps1 / <X, 3>*
04: ~~*Qp1 D*~~ *Pd1*	05: *Ps0 / Qs0 / <X, 2>*
06: ~~*Pp1*~~	07:

SWAP FILE	
s0: *Pp0 / Qp0*	s1: *Qd0*
s2: *Qp1*	s3: *Pp1*

LRU ACTIVE: *PD1, PS1, PC1* _____

LRU INACTIVE: *pd0, ps0, qs0, qc1*_____

seconda parte – file system

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = 2 MINFREE = 1

Si consideri l**a seguente situazione iniziale.**

```
PROCESSO: P  ************************************************************
    VMA : C  000000400,  1,  R,  P,  M,  <X,  0>
          S  000000600,  2,  W,  P,  M,  <X,  1>
          D  000000602,  2,  W,  P,  A,  <-1, 0>
          P  7FFFFFFC,  3,  W,  P,  A,  <-1, 0>

    PT: <c0 :1 R>  <s0 :- ->  <s1 :- ->  <d0 :- -> <d1 :- ->  <p0 :3 W>
        <p1 :- ->  <p2 :- ->

    process P - NPV of PC and SP:  c0, p0

PROCESSO: Q  ************************************************************
    VMA : C  000000400,  1,  R,  P,  M,  <X,  0>
          S  000000600,  2,  W,  P,  M,  <X,  1>
          D  000000602,  2,  W,  P,  A,  <-1, 0>
          P  7FFFFFFC,  3,  W,  P,  A,  <-1, 0>

    PT: <c0 :1 R>  <s0 :- ->  <s1 :- ->  <d0 :- ->  <d1 :- ->  <p0 :2 D W>

        <p1 :- ->  <p2 :- ->

    process Q - NPV of PC and SP:  c0, p0

____MEMORIA FISICA____(pagine libere: 2)_____________________________
      00 : <ZP>                    ||  01 : Pc0 / Qc0 / <X, 0>       ||
      02 : Qp0  D                  ||  03 : Pp0                      ||
      04 : <F, 0>  D               ||  05 : <F, 1>  D                ||
      06 : ----                    ||  07 : ----                     ||

____STATO del TLB_____________________________________________________
      Pc0 : 01 -  0: 1:            ||   Pp0 : 03 -  1: 1:            ||
            -----                  ||         -----                  ||
            -----                  ||         -----                  ||

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

LRU ACTIVE:   QP0,  QC0,  PP0,  PC0

LRU INACTIVE:
```

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
PQ	F	5500	2	2	0

ATTENZIONE: **nella colonna "processo" 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.**

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.

ATTENZIONE: A ulteriore chiarimento della situazione iniziale, si precisa che il processo **P** è in esecuzione e che lo stato iniziale riportato deriva dalla seguente successione di chiamate di sistema eseguite da **P**:

- apertura del file **F** con **fd1 = *open* (F)**
- creazione del processo figlio **Q** tramite ***fork***
- accesso in scrittura al file **F** tramite ***write*** **(fd1, 5500)**

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 effettuati in lettura e scrittura.

evento 1: fd2 = *open* (G) – *write* (fd2, 1000)

Il processo P apre il file G, aggiorna posizione corrente e numero processi; P scrive in pagina 0 del file G; carica da disco la pagina G0 e la alloca in pagina fisica 6 (marcata dirty); resta una pagina libera; per il file G in totale una lettura da disco e nessuna scrittura su disco.

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

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P Q	*F*	*5500*	*2*	*2*	*0*
P	*G*	*1000*	*1*	*1*	*0*

evento 2: *context switch* (Q)

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

TLB							
NPV	NPF		D	A	NPV	NPF	D A
Qc0 :	*01 –*	*0: 1:*			*Qp0 :*	*02 –*	*1: 1:*

evento 3: fd3 = *open* (H) – *write* (fd3, 1000)

Bisogna scrivere la pagina 0 di file H, ma numero di pagine libere è uguale a 1, quindi si attiva PFRA che libera due pagine di memoria, la 4 e la 5 (pagine di disk cache), e scrive le relative pagine del file F su disco in quanto sono dirty. Quindi alloca la pagina 0 di di H in pagina 4 di memoria.

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

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P Q	*F*	*5500*	*2*	*2*	*2*
P	G	1000	1	1	0
Q	*H*	*1000*	*1*	*1*	*0*

evento 4: *write* (fd1, 2000)

Il processo Q carica (legge) in 05 la pagina 1 di F e quindi scrive in memoria marcandola D.

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

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P Q	*F*	*7500*	*2*	*3*	*2*
P	G	1000	1	1	0
Q	*H*	*1000*	*1*	*1*	*0*

evento 5: *close* (fd1) – *close* (fd3)

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

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P ~~Q~~	*F*	*7500*	*1*	*3*	*2*
P	G	1000	1	1	0
---	*H*	*----*	*0*	*1*	*1*

evento 6: *context switch* (P) – *read* (fd1, 1000)

Il context switch fa il flush del TLB. Il processo P legge dalla pagina 1 di F (in memoria), mentre per leggere dalla pagina 2 di F (non in memoria) deve chiamare PFRA. PFRA libera le due pagine di disk cache 4 e 5. Per la 4 (ossia <H, 0>) non c'è nulla da fare perché la versione in memoria è già allineata a quella del disco, invece per la 5 (ossia <F, 1> D) va fatta la scrittura su disco.

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

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	*8500*	*1*	*4*	*3*
P	G	1000	1	1	0

PAGINA DI ALLINEAMENTO – spazio libero per continuazione o brutta copia

esercizio n. 4 – domande su argomenti vari

tabella delle pagine

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

1. la scomposizione degli indirizzi virtuali dell'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 a 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

VMA del processo **P**							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	0000 0040 0	2	R	P	M	FF	0
K	0000 0060 0	3	R	P	M	FF	2
S	0000 0060 3	4	W	P	M	FF	5
D	0000 0060 7	128	W	P	A	-1	0
M1	0001 3000 0	2	W	S	M	AA	2
M2	0001 3000 F	3	W	P	M	BB	0
P	7FFF FFFF 2	13	W	P	A	-1	0

1. Scomposizione degli indirizzi virtuali

		PGD :	**PUD :**	**PMD :**	**PT**
C	0000 0040 0	*0*	*0*	*2*	*0*
K	0000 0060 0	*0*	*0*	*3*	*0*
S	0000 0060 3	*0*	*0*	*3*	*3*
D	0000 0060 7	*0*	*0*	*3*	*7*
M1	0001 3000 0	*0*	*4*	*384*	*0*
M2	0001 3000 F	*0*	*4*	*384*	*15*
P	7FFF FFFF 2	*255*	*511*	*511*	*498*

2. Numero di pagine necessarie

pag PGD: *1* # pag PUD: *2*

pag PMD: *3* # pag PT: *4*

pag totali: *10*

3. Numero di pagine virtuali occupate dal processo: *155*
4. Rapporto di occupazione: *10 / 155 = 0.064 = 6,4 %*
5. Dimensione massima del processo in pagine virtuali:

   Con la stessa dimensione di TP il processo può crescere fino a 4 x 512 = 2048 pagine virtuali

spazio libero per brutta copia o continuazione

spazio libero per brutta copia o continuazione